Self-Optimization module for Scheduling using Case-based Reasoning

I. Pereira, A. Madureira

A B S T R A C T

Metaheuristics performance is highly dependent of the respective parameters which need to be tuned. Parameter tuning may allow a larger flexibility and robustness but requires a careful initialization. The process of defining which parameters setting should be used is not obvious. The values for parameters depend mainly on the problem, the instance to be solved, the search time available to spend in solving the problem, and the required quality of solution.

This paper presents a learning module proposal for an autonomous parameterization of Metaheuristics, integrated on a Multi-Agent System for the resolution of Dynamic Scheduling problems.

The proposed learning module is inspired on Autonomic Computing Self-Optimization concept, defining that systems must continuously and proactively improve their performance. For the learning implementation it is used Case-based Reasoning, which uses previous similar data to solve new cases. In the use of Case-based Reasoning it is assumed that similar cases have similar solutions.

After a literature review on topics used, both AutoDynAgents system and Self-Optimization module are described. Finally, a computational study is presented where the proposed module is evaluated, obtained results are compared with previous ones, some conclusions are reached, and some future work is referred.

It is expected that this proposal can be a great contribution for the self-parameterization of Metaheuristics and for the resolution of scheduling problems on dynamic environments.

Keywords:
Autonomic Computing
Case-based Reasoning
Learning
Meta-heuristics
Multi-Agent Systems
Scheduling

1. Introduction

The exponential growth of computational capacity together with the appearance of network communication, mainly Internet, led companies to invest significantly in infrastructures and computational applications. These systems are subject to failures, dynamism, overloads, and others, due to exponential growth of its complexity. Generally, organizations invest more in maintenance than in development. With the prediction of a "breaking point", in October 2001, Paul Horn, vice-president of IBM Research, gave some visibility to this problem launching an IBM challenge named Autonomic Computing (AC) [1].

An autonomous system is viewed as a system with the ability of self-management. The objective of these systems is to free users from repetitive tasks, enabling a full-time execution, and providing enough intelligence to be possible to take decisions in order to reach an objective [2].

In recent years, there has been a growing interest in decentralized approaches for the resolution of complex real world problems, like Scheduling, as the number of proposed solutions and successful implementations is increasing. It is possible to highlight Multi-Agent Systems (MAS), which concern with behaviors' coordination of a set of agents, in order to share knowledge, abilities, and objectives, in the resolution of complex problems. Due to the exponential growing of system's complexity, it is important that MAS become more autonomous to deal with dynamism, overloads and failures recovery.

MAS typically operate in open, complex, dynamic, and unpredictable environments. It is not possible to predict every situation that an agent can find so it is necessary that agents have the ability to adapt to new situations. Since intelligence implies a certain degree of autonomy, requiring the capacity of taking decisions autonomously, agents must have the appropriate tools to take such decisions. Therefore learning becomes, many times, indispensable [3].

Biological and natural processes have been a source of inspiration for computer science and information technology, which led to the development of techniques that converge, in general, to satisfactory solutions in an effective and efficient way, generally named Meta-heuristics (MH) [4]. MH have often been shown to be effective for difficult combinatorial optimization problems appearing in several industrial, economic, and scientific domains [5–10].

Since MH parameterization revealed to be a hard task, requiring expertise knowledge about the application domain and which

techniques and parameters should be used, it is important to turn systems capable of autonomously parameterize themselves. To validate this self-parameterization, we use learning about past experience, with Case-based Reasoning (CBR) revealing to be a promising approach. Using CBR, systems can remember past effectively solved cases and autonomously decide which MH and parameters to use for the resolution of a new similar problem [11].

With this, we propose using MAS, MH, AC, together with learning capabilities, to improve the resolution of the Dynamic Scheduling problem, with the objective of maximizing the quality of solutions, minimizing the computational time spent, and also reducing the human intervention.

The remaining sections are organized as follows. Section 2 describes the Dynamic Scheduling problem and Section 3 presents a literature review of AC, MH, MAS, and CBR applications for Scheduling. In Section 4 AutoDynAgents system is presented and the Self-Optimizing Module is described in Section 5, which was integrated in AutoDynAgents. The computational study is presented in Section 6 and, finally, the paper presents some conclusions and puts forward some ideas for future work.

1.1. Dynamic Scheduling problem

Scheduling problems arise in a diverse set of domains, ranging from manufacturing to transports, hospitals settings, computer and space environments, amongst others, most of them characterized by a great amount of uncertainty that leads to significant dynamism in the system. Such dynamic scheduling is getting increased attention between researchers and practitioners [12,13].

The dynamism can arise from requirements of a new user or from the evolution of the external environment. In a more general way, dynamic changes can be seen as a set of inserted and cancelled constraints.

Dynamic optimization problems environments are often impossible to avoid in practice. For these, the optimization algorithm must continuously find the optimum in dynamic environments, or find a robust solution that is capable of operate optimally in the occurrence of perturbations, instead of simply locate the global optimum solution [14].

In spite of all research made so far, the scheduling problem is still known to be NP-complete, even for static environments [12]. This fact presents serious challenges to conventional algorithms and incites researchers to explore new directions. Multi-Agent technology has been considered an important approach for developing industrial distributed systems [15].

The scheduling problem treated in this work has some major extensions and differences compared to the classic Job-Shop Scheduling Problem (JSSP), named Extended Job-Shop Scheduling Problem (EJSSP), proposed by Madureira [16]. The main elements of EJSSP problem can be modeled as:

(1) a *set* of multi-operation jobs $J_1, \ldots, J_n$ has to be scheduled on a set of machines $M_1, \ldots, M_n$.
(2) d_j represents the due date of job J_j.
(3) t_j represents the initial processing time of job J_j.
(4) r_j represents the release time of job J_j.

The existence of operations on the same job, on different parts and components, processed simultaneously on different machines, followed by components assembly operations (multi-level jobs).

Furthermore EJSSP should meet the following restricted conditions:

(a) The existence of different job release dates r_j and due dates d_j.
(b) The possibility of job priorities definition, reflecting the importance of satisfying their due dates, being similar to the weight assigned to jobs in scheduling theory.
(c) Precedence constraints among operations of the different jobs.
(d) New jobs can arrive at unpredictable intervals. Jobs can be cancelled. Changes in task attributes can occur: processing times, due dates release dates and priorities.
(e) Each operation O_{ijkl} must be processed on one machine of the set M_i, where p_{ijkl} is the processing time of operation O_{ijkl} on machine M_i.
(f) A machine can process more than one operation of the same job (recirculation).
(g) The existence of alternative machines, identical or not.

The methods for the resolution of NP-hard combinatorial optimization problems, where Scheduling is included, can be divided in exact and approximation algorithms [4,13]. In the first, an exhaustive solutions space search is made and the optimal solution obtaining is ensured. The second type of algorithms has the objective to find a good solution in an acceptable period of time, but do not guarantee the optimal solution. MH are a more representative class of approximation algorithms, used in this work.

2. Literature review

This section will discuss some efforts related to literature on applications of different paradigms to the Dynamic Scheduling problem resolution, such as AC, MH, MAS, and CBR.

2.1. Autonomic Computing applications for Scheduling

Autonomic Computing (AC) is an IBM Grand Challenge proposed in 2001 by Paul Horn, Senior Vice-President of IBM Research [1]. Horn argues that the Information Technology (IT) industry is on constant expansion and will soon reach its breaking point. This can happen because massive data centers are built in organic, ad hoc ways, resulting in a heterogeneous composition where maintenance costs in terms of qualified staff, time and capital can soon exceed corporate capabilities [2].

AC represents a new computation paradigm in which IT systems have managing mechanisms embedded in the application, with the objective to automate the management. So, applications must be able to adapt, accommodate and protect themselves to the changes in the environment and in the objectives.

AC proposes a broad new field of research related to the automation of IT management processes, drawing inspiration from the human autonomous nervous system, since many essential functions to the welfare and regulation of living beings are not consciously triggered, e.g., heart beating, digestive system, etc. However, without an autonomous system to manage these mechanisms, the body would stop working or the concentration in other life aspects would not be possible.

From its inception, the AC concept involves four properties, generally referred as self-* properties, in which research efforts may be categorized [2,17]:

- **Self-configuration** deals with installation, configuration and integration of IT systems. When a new component is intended to insert the system, it is autonomously incorporated, like a new cell incorporates the human body, or even when a person incorporates a population. The installation procedures work by gathering information about the environment, figuring out the dependencies among needed tasks and also optimizing performance measures, and finally executing the tasks to perform changes.

- **Self-optimization** deals with performance improving of running systems by leveraging alternative opportunities. This concept emerges allowing autonomous systems to continuously and proactively seek for opportunities to improve its execution, through identification and use of opportunities to become more efficient in its performance and cost. The system is capable of monitoring, experimenting, planning and tuning itself, based on the acquired knowledge of the environment.
- **Self-healing** deals with identification of problematic situations and recovering from them, minimizing the consequences of failures or even making a total recovery of the system. It requires the system to reason how recovering activities can be performed, how diagnostic information is produced and how new changes can be affected with minimal cost and maximum benefit.
- **Self-protection** deals with monitoring the environment for threats and responding to them. This can be made in two ways: first, protects the system in large scale against malicious and intrusive attacks and also protects the system against possible failures resulting from self-healing; second, self-protection must be able to, based on sensors reports, predict problems that may occur, to avoid or attenuate them.

Some AC applications for the resolution of scheduling problems can be found in, e.g. [18–21].

2.2. Meta-heuristics applications for Scheduling

The adaption of ideas from different research areas, inspired on nature, led to the development of Meta-heuristics (MH), which are heuristic methods to solve complex generic problems of combinatorial optimization. These techniques have the objective of guiding and improving the search process in a way to overcome the local optimal solutions and obtain solutions with satisfactory quality, very close to the optimal solution [4,22].

To solve a problem, it is necessary to choose a MH, which is a considered difficult task, requiring a study about the problem type and about the working of the chosen technique. It is also very difficult to choose a MH to be the best among all, since it depends on the problem to solve and each MH has its advantages and drawbacks on some scenarios. With this, it is possible to realize that this choice should be made based on the results to be obtained, which may be efficiency, effectiveness, robustness, etc. It is also necessary to define the parameters of the chosen MH, which is also a very hard task, since it also depends on the problem to solve and requires some expertise knowledge.

In the literature, it is possible to find different approaches using MH [4,22], mainly Tabu Search, Simulated Annealing, Genetic Algorithms, Ant Colony Optimization, Particle Swarm Optimization, Artificial Bee Colony, amongst others.

Tabu Search was introduced by Glover [23] and consists in a Local Search strategy with the main objective to escape from local minimal values. It presents some similarities with human behavior since past influences, in a determinant way, the process of searching new solutions, as the past experience of a human being defines the future decision making. This MH was applied to the resolution of scheduling problems in, e.g. [21,24–27].

Simulated Annealing is a classic algorithm, proposed in the beginning of the 80s by Kirkpatrick et al. [28] and Cerny [29]. The original motivation is based on the process in which molten metal is slowly cooled, with a tendency to solidify in a structure of minimum energy. This technique has a statistical basis and is based on permitting the movement to a worst solution, relatively to current solution, with the objective to escape from local optimum. Recent applications of this MH to the resolution of scheduling problem can be found in [21,30–33].

In beginning 70s, John Holland, together with his students and colleagues, developed researches and studies based on natural selection of species, reaching a formal model designated by Genetic Algorithms [34]. These studies were based on genetics and on the idea presented in Charles Darwin book about the theory of species origin [35]. In his book, Darwin reports a competition between individuals of each animal species, refers about the species evolution, and also points that only the fittest ones can survive, with the nature being responsible for the natural selection. In the 80s, David Goldberg, a Holland's student, implemented the first well successful application of these algorithms [36]. Since then, Genetic Algorithms were applied with success in the several optimization problems [21,24,37–39]. However, they shown to be not very suitable for scheduling problems because they are not very efficient finding a near-optimal solution in reasonable time, for example when compared with Tabu Search and Simulated Annealing, which work with a single configuration and not with an entire population of individuals [40].

Ant Colony Optimization is based on ants behavior, i.e., on a behavior that allows ants to find the shortest path between a food source and the respective colony [41]. This phenomenon occurs because ants deposit in the path a substance named pheromone, and when choosing a path, they opt, with a greater probability, by the one that have more quantity of pheromone, because that probably is the shortest, i.e., the trajectory that the higher number of ants have made. The first system based in this MH was introduced by Dorigo et al., in 1991, with the designation of Ant System [42]. Ant Colony Optimization was applied to scheduling problem resolution in, e.g. [21,43–47].

Particle Swarm Optimization is an evolutionary technique based on populations, developed by James Kennedy and Russell Eberhart [48], with the objective to simulate a simplified social system. Initially, the basic idea was to demonstrate the behavior that flocks of birds or schools of fishes assume in their random local trajectories, but globally certain. Flocks of birds or schools of fishes make coordinated and synchronized movements as a way of finding food or as a mechanism of self-defense. This MH was applied to the resolution of scheduling problems in, e.g. [21,49–53].

Artificial Bee Colony [54] was proposed by Karaboga in 2005 inspired in the foraging behavior of the bees. Real bees are social insects living in organized group called hive. In a hive, bees have some specific tasks performed by specialized individuals. The goal of this organization is to maximize the amount of nectar in the colony getting the utmost of the food sources. The basis of this model are three types of specialized bees: Employed, Onlooker and Scout, representing a minimal model of the real swarm intelligent forage selection. Recent Artificial Bee Colony applications on the resolution of scheduling problems can be found in, e.g. [55–58].

2.3. Multi-Agent Systems applications for Scheduling

Multi-Agent Systems (MAS) derive from Distributed IA and highlight the agents' common behaviors, with some degree of autonomy, and the complexity arising from agents' interactions. MAS concern with the coordination of behaviors of a certain community of agents, in order to share knowledge, capacities, and objectives for the resolution of complex problems [59].

MAS have come to prominence in the last two decades as an interesting paradigm for modeling and implementation of some applications on dynamic and unpredictable environments [60]. Luck et al. [61] have referred that these systems can be the next great step in computation evolution, like Object Oriented was.

Multi-agent paradigm is emerging for solutions development of very hard distributed computational problems. This paradigm is based both on "intelligent" agents activity, which perform complex functionalities, either on the exploitation of a large number of

simple agents that can produce an overall intelligent behavior leading to the solution of supposed intractable problems [59].

Considering the inherent complexity of manufacturing systems, dynamic scheduling is considered a good candidate to apply agent-based technology. In many implementations of MAS for manufacturing scheduling, agents model the system resources and the scheduling of tasks is done in a distributed manner by means of cooperation and coordination amongst agents [15]. When in response to disturbances, the distributed nature of MAS can also be a benefit to the rescheduling algorithm by involving only the agents directly affected, without disturbing the rest of the community which can continue with their work.

There are different implementations of MAS architectures for the resolution of Scheduling problem. According to Shen and Norrie [62], it has traditionally been implemented three kinds of MAS architectures: hierarchical, federations, and autonomous. Ouelhadj and Petrovic [40] have referred two MAS architectures for Dynamic Scheduling resolution: autonomous and mediator architectures. In Autonomous Architectures, agents represent manufacturing entities as jobs and resources, and can, autonomously, define their scheduling plans, react to changes and cooperate among them. Mediator Architectures have mediator agents to coordinate agents' behaviors, in order to obtain global scheduling plans.

Some work in Production Systems and Scheduling can be found in [21,47,63–67].

2.4. Case-based Reasoning applications for Scheduling

Case-based Reasoning (CBR) is an AI methodology which aims to solve new problems by using information about the solutions to previous similar problems [68]. CBR operates under the premise that similar problems can require similar solutions [11].

In CBR, previous solved cases and its solutions are memorized as cases, saved in a repository (casebase), in order to be reused in the future [11]. Instead of defining a set of rules or general lines, a CBR system solves a new problem by reusing similar cases that were previously solved [69]. CBR consists in a cycle (Fig. 1), usually described as the '4 Rs' cycle [11,70]:

(1) Retrieve the most similar case or cases;
(2) Reuse the retrieved information and knowledge;
(3) Revise the proposed solution;
(4) Retain the revised solution to be useful for future problem solving.

The initial description of a problem (new case) is used to retrieve a case from the casebase. In the Reusing phase, the retrieved case is analyzed in order to origin a suggested solution for the new case. In the Revising phase, this suggested solution is tested, for example, by executing it in the system, and repaired if it fails. In the Retaining phase, the useful experience is retained for future use, and the casebase is updated with the new learned case (or by modifying some existing cases).

The Retrieving phase starts with a partial problem's description, and ends when the most similar previous case (or cases) is discovered. A very important aspect of this phase is the similarity measure, which is usually defined by a formula to calculate the similarity between previous cases and the new case [71]. The correct definition of similarity measures for real world problems is one of the greatest challenges in this research area.

In the Reusing phase, it is possible to reuse a solution or a method. In solution reuse, the past solution is not directly copied to the new case, but there is some knowledge allowing the previous solution to be transformed into the new case solution. In case of method reuse, it is observed how the problem was solved in the retrieved case, which has information about the method used for the problem resolution, including an explanation about the used operators, sub-objectives considered, generated alternatives, failures, etc. The retrieved method is then reused to the new problem resolution, in the new context.

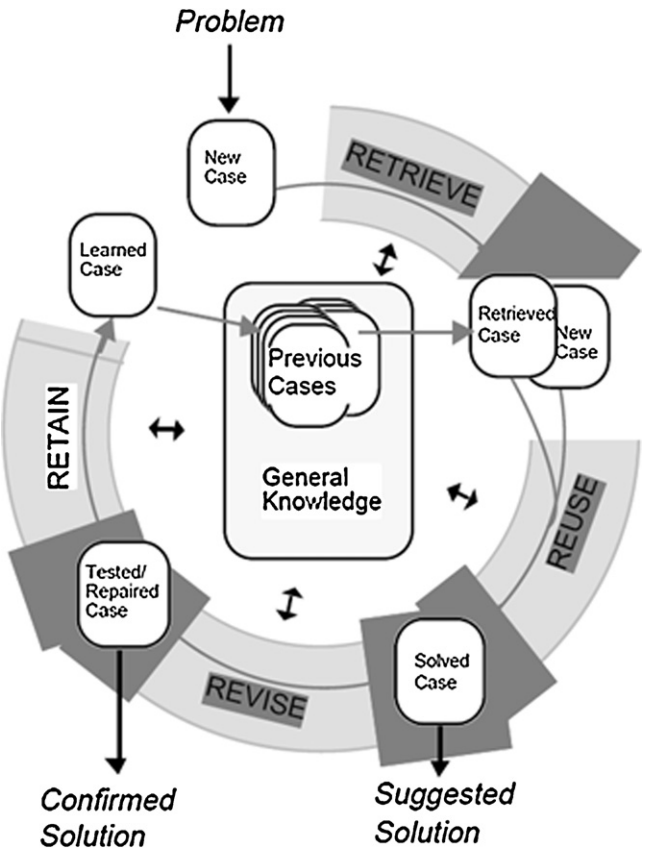

Fig. 1. CBR cycle [70].

The objective of Revising phase is to evaluate the retrieved solution. If this solution is well succeeded it is possible to learn about the success, otherwise the solution is repaired using some problem domain's specific knowledge. The evaluating task applies the proposed solution in an execution environment and the result is evaluated. This is usually a step outside the CBR, once the problem may be executed in an application.

Finally, the Retaining phase consists in the integration of the useful information about the new case resolution into the casebase. It is necessary to know which information is important to retain, how to retain it, how to index the case for a future retrieve, and how to integrate the new case in the memory structure.

Burke et al. [71] have referred that CBR is an appropriate approach for scheduling systems with expertise knowledge, and emphasize a research potential in dynamic scheduling. CBR applications for scheduling domain can be found in [11,21,69,71–73].

3. AutoDynAgents system

AutoDynAgents [21,67] is a MAS for the resolution of Dynamic Scheduling problems with autonomic capacities, in which a community of agents models a real manufacturing system subject to perturbations. The system is able to find optimal or near optimal solutions through the use of MH, deal with dynamism (arriving of new jobs, cancelled jobs, changing jobs attributes, etc.), change/adapt the parameters of the algorithm according to the current situation, to switch from one MH to another, and perform a

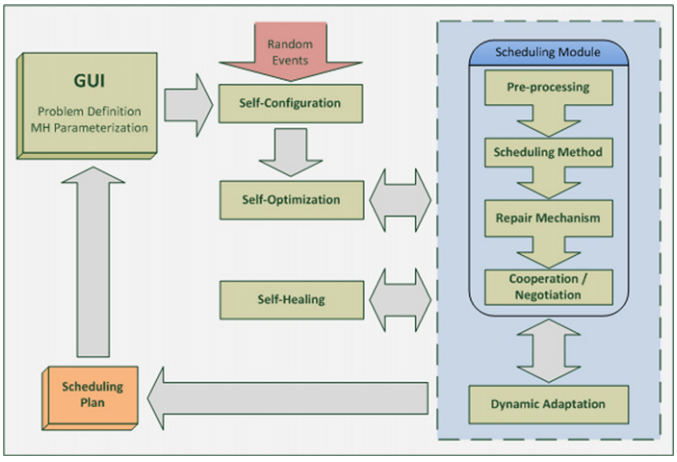

Fig. 2. AutoDynAgents global architecture.

coordination between agents through cooperation or negotiation mechanisms.

The approach used by AutoDynAgents for the resolution of Scheduling problems is rather different from the ones found in the literature. The EJSSP problem defined in Section 2 is decomposed into Single Machine Scheduling Problems (SMSP) [12,13]. In this system, there is a group of Resource Agents, each one responsible for optimizing the corresponding machine plan through the application of MH, obtaining local solutions. Another agent (UI Agent) is responsible for gathering those local solutions into a global schedule, applying a coordination mechanism.

The coordination between agents can be made in different ways. First, it is applied a repair mechanism to shift some operations till a feasible solution can be obtained. Then, a coordination is established between related agents in the process, in order to pursuit common objective, through a cooperation or negotiation mechanism. These coordination mechanisms are prepared to accept agents subject to dynamism. More details about the cooperation mechanism can be found in [74] and the negotiation mechanism is described in [75].

There are two classes of dynamic events that can occur: partial events, which imply variability in the attributes of jobs/operations, such as processing times, due dates or release times; and total events, which imply variability in the structure of neighborhood/population, resulting from new jobs arrivals, jobs cancellations, machines breakdown, etc.

AutoDynAgents architecture (Figs. 2 and 3) is based on six different types of agents. This is a hybrid autonomic architecture, since agents can define their scheduling plans, react to changes and cooperate among them. According to Horling and Lesser [76], it can represent a Team Model, if using the cooperation mechanism (Resource Agents cooperate in order to reach a common objective), or a Market Model, if using the negotiation mechanism (Resource Agents negotiate among themselves using market rules).

In order to allow a consistent communication with the user, a User Interface Agent (**UI Agent**) was implemented. This agent is responsible for the user interface and also for the dynamic generation of the necessary Job and Resource agents, according to the number of jobs and machines comprising the scheduling problem, and assigns each task to the respective Job Agent. In the end, this agent applies the repair mechanism and starts the cooperation/negotiation mechanism.

Job agents process the necessary information about each job. They are responsible for the generation of the earliest and latest processing times on the job and automatically separate each job's operation for the respective Resource Agent.

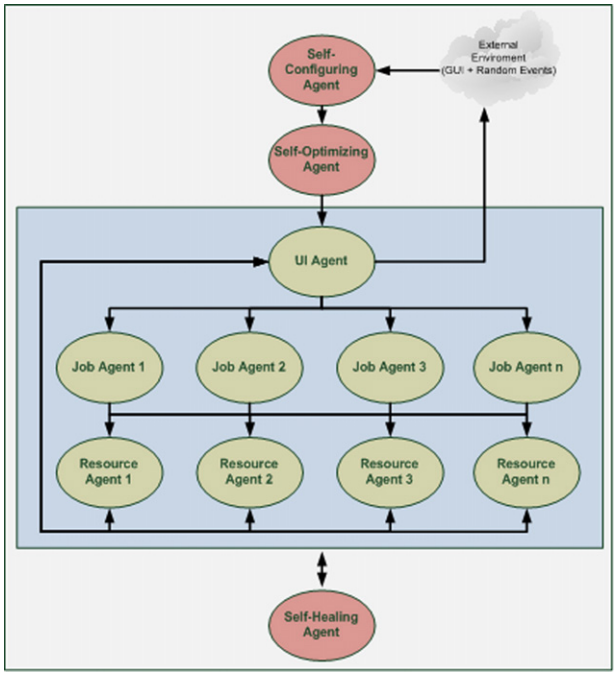

Fig. 3. AutoDynAgents agents architecture.

Resource agents are responsible for scheduling the operations that require processing in the machine supervised by the agent. These agents implement MH in order to find the best possible singe-machine schedules/plans of operations and communicate those solutions to the UI Agent.

Respectively to self-* agents [21], **Self-Configuring Agent** is responsible for monitoring the system, with the objective to detect changes occurred in the schedule, allowing the system to perform a dynamic adaptation. With this agent, the system is prepared to automatically handle with dynamism, by adapting the solutions to external perturbations. While, on one hand, partial events only require a redefinition of job attributes and re-evaluation of the objective function, on other hand, total events require changes on the structure and size of solutions. Therefore, under total events, the modification of the current solution is imperative, through job arrival integration mechanisms (when a new job arrives to be processed), job elimination mechanisms (when a job is cancelled) and regeneration mechanisms, in order to ensure a dynamic adaptation of population/neighborhood.

Self-Optimizing Agent is responsible for autonomously tuning the MH parameters, according to problem's characteristics and system's behavior. This agent receives the initial problem (or the changes detected by the Self-Configuring Agent) and automatically chooses the MH and parameters to use. If some dynamism occurs, the MH and respective parameters may change in run-time. This tuning of parameters is made through learning and experience, using a CBR module, each time a new problem (case) appears. When the new case is solved, it is stored for later use. This Self-Optimization module is better described in the next section.

Finally, **Self-Healing Agent** gives to the system the capacity for diagnosing deviations from normal conditions and proactively takes actions to normalize them and avoid service disruptions. This agent monitors other agents in order to provide overall self-healing capabilities. Since agents may crash for some reason, self-healing provides one or more agents backup registries in order to grant storage for the reactivation of lost or stuck scheduling agents with meaningful results, thus enabling the system to restart from a previous checkpoint as opposed to a complete reset. With this

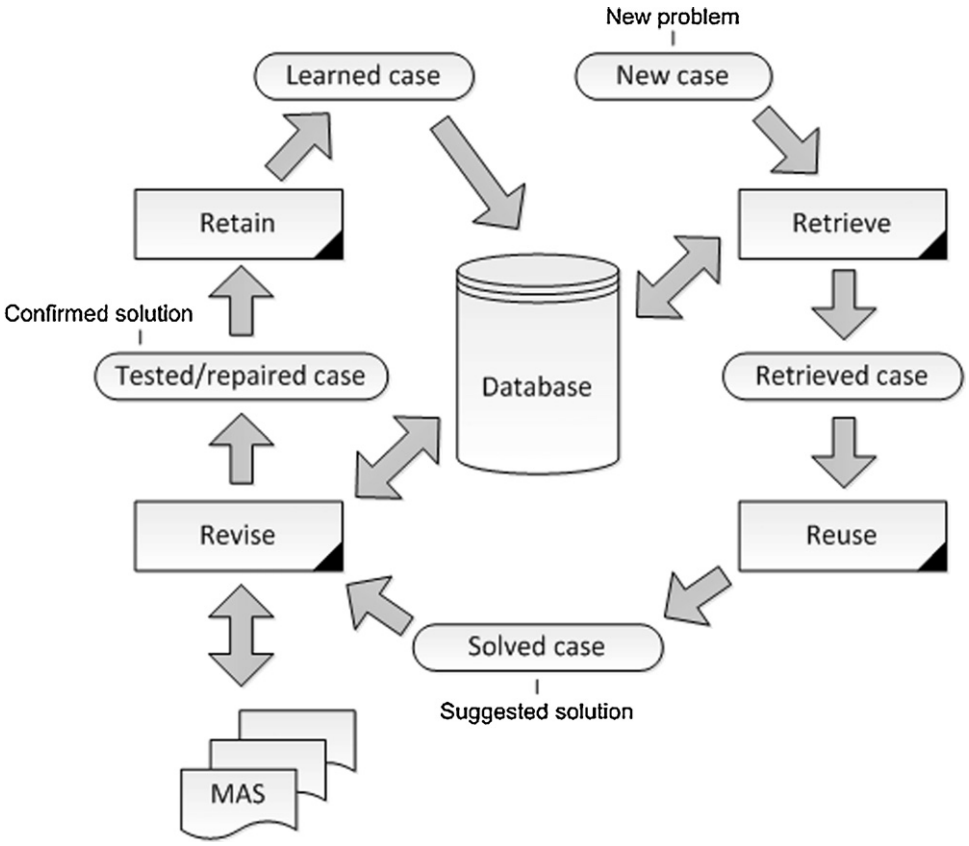

Fig. 4. Self-Optimization module – CBR architecture.

agent, the system becomes stable, even if some deadlocks or crashes occur.

4. Self-Optimization module

As previously mentioned, MH are very effective for obtaining good solutions, and sometimes even optimal solutions, in reasonable execution times. But, to be possible to obtain optimal or near-optimal solutions, it is required the appropriate tuning of the parameters, which has revealed to be a hard task, since it needs some expertise knowledge about the MH in use and from the problem to solve. Sometimes it is necessary to use the trial-error method, and the difficulties increase when exist more than one MH that can be used, because it requires an a priori choice of it and only then the tuning of its parameters.

Given this, the objective is that AutoDynAgents adopts and provides self-parameterization of MH respectively to the problem to be solved, with the possibility that parameters can change in runtime. The system must be able to define which MH must be used and also define the respective parameters according to the current situation being treated. It is even possible to change from one MH to another, according to current state and previous information, through learning and experience.

In order to provide the system with capacities of self-parameterization of MH, it is presented, in this section, an AC Self-Optimization module. This module is capable of monitoring the system and autonomously tuning the parameters of different MH, taking into account each new single problem emerging in the system.

Since it is impossible to predict each problem to treat, this module must be capable of learning about its experience during lifetime, e.g., as humans do. To perform this learning mechanism, it is proposed the use of CBR.

The CBR application on this module consists in retrieving the most similar case of the new problem, regardless the MH to use. So, it is retrieved the case containing the MH and respective parameters to use. With this approach it will be possible to know, for example, which MH is more appropriate for the resolution of a particular type of problem.

The use of CBR in Self-Optimization module, embedded on an agent, represents a MAS Team Learning approach, as described by Panait and Luke [59], since there is only an apprentice involved but with the objective to discover a subset of behaviors for a team of agents.

4.1. CBR architecture

The CBR based architecture is presented in Fig. 4. Every new problem or perturbations occurred lead to a new case in the system, and the previous most similar cases are retrieved from the database. Then, the better case is reused, becoming in a suggested solution. After the revision of the solution, the case is executed in the MAS. This revision is made to be possible to escape from local optimal solutions and stagnation, since it is used some disturbance in the parameters of the proposed solution. After the conclusion of the MAS execution, the case is confirmed as a good solution, being retained on the database as a new learned case, for future use.

4.2. Casebase

The CBR casebase is presented in Fig. 5, with six tables, one for each MH in use and another for saving the attributes of each case.

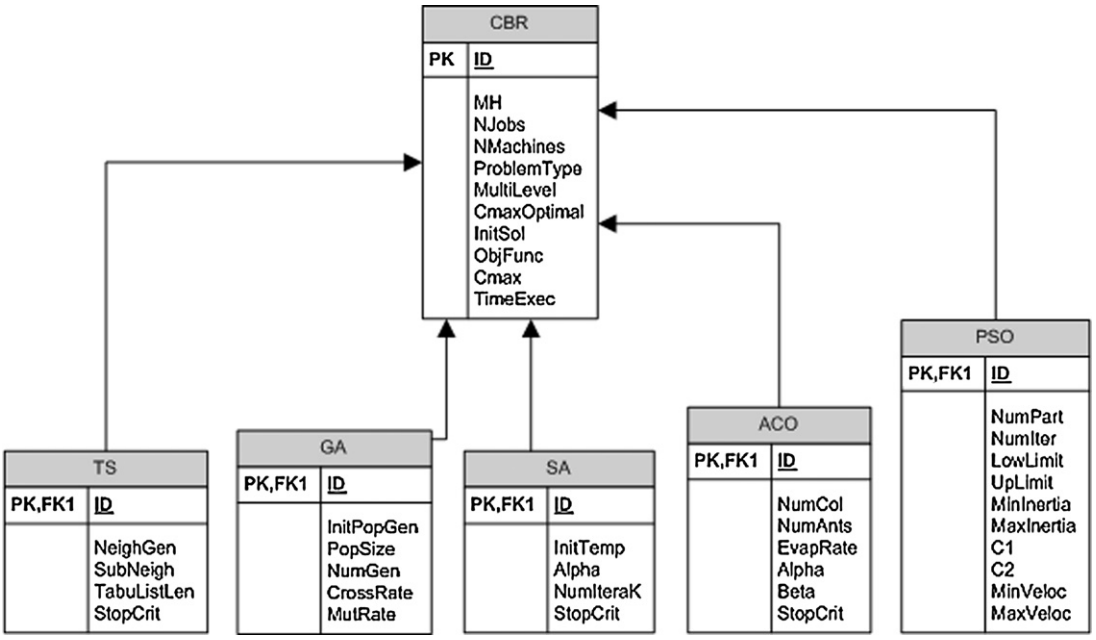

Fig. 5. Self-Optimization module – CBR casebase.

Table 1
Casebase – CBR table fields description.

Field	Data type	Description
ID	Integer	Primary key of the case, auto-incremented
MH	String	Name of the used MH
NJobs	Integer	Number of jobs of the problem (e.g., 10)
NMachines	Integer	Number of machines of the problem (e.g., 5)
ProblemType	String	Problem type ("single-machine", "open-shop", "flow-shop", or "job-shop")
MultiLevel	Boolean	Indicates if the problem has operations with more than one precedent
CmaxOptimal	Integer	Optimal makespan value known, if available
InitSol	String	Heuristic used to calculate the initial solution (EDD (Earliest Due Date), SPT (Shortest Processing Time), or SeqNivel (increasing ordering of levels of operations))
ObjFunc	String	Objective function used (Cmax (makespan minimization) or WT (Weighted Tardiness minimization))
Cmax	Integer	Makespan value obtained from the case
TimeExec	Double	Execution time, in seconds

Table 2
Casebase – TS table fields description.

Field	Data type	Description
NeighGen	Double	Neighborhood generation percentage
SubNeigh	Double	Sub-neighborhood percentage
TabuListLen	Integer	Tabu list length
StopCrit	Integer	Stopping criteria, number of iterations

MH tables are TS, GA, SA, ACO, and PSO, representing Tabu Search, Genetic Algorithms, Simulated Annealing, Ant Colony Optimization, and Particle Swarm Optimization respectively. Each one of these tables contains the cases parameters for tuning the respective technique.

CBR table, as referred before, stores the attributes of each single case in the CBR system. The fields are described in Table 1.

The description of each MH field is also presented in Tables 2–6, corresponding to each parameter defined by the CBR system.

Table 3
Casebase – GA table fields description.

Field	Data type	Description
InitPopGen	Double	Initial population generation percentage
PopSize	Double	Population size percentage
NumGen	Integer	Number of generations
CrossRate	Double	Crossover rate
MutRate	Double	Mutation rate

Table 4
Casebase – SA table fields description.

Field	Data type	Description
InitTemp	Double	Initial temperature
Alpha	Double	Alpha factor of temperature reduction
NumIteraK	Integer	Number of iterations at the same temperature
StopCrit	Integer	Stopping criteria, number of iterations

Table 5
Casebase – ACO table fields description.

Field	Data type	Description
NumCol	Integer	Number of colonies
NumAnts	Integer	Number of ants per colony
EvapRate	Double	Pheromone evaporation rate
Alpha	Double	Heuristic value importance
Beta	Double	Pheromone importance
StopCrit	Integer	Stopping criteria, number of iterations

Table 6
Casebase – PSO table fields description.

Field	Data type	Description
NumPart	Integer	Number of particles
NumItera	Integer	Number of iterations
LowLimit	Integer	Lower limit
UpLimit	Integer	Upper limit
MinInertia	Double	Minimum inertia
MaxInertia	Double	Maximum inertia
C1	Double	Cognitive component
C2	Double	Social component
MinVeloc	Double	Minimum velocity
MaxVeloc	Double	Maximum velocity

4.3. Retrieving phase

The objective of Retrieving phase is to search the casebase, finding the most similar previous cases, and retrieving them for analysis, in order to select one and reuse it in the next phase.

A pre-selection of cases is made in order to only select cases that can be considered similar enough, i.e., with a similarity over 75%, in order to ignore cases that are not very similar with the new one. After this pre-selection, the cases are analyzed one by one, in order to select cases that have enough similarity with the new case, by calculating its similarity measure.

The considered attributes in the proposed similarity measure are some fields from the CBR table (Table 1), namely *NJobs*, *NMachines*, *ProblemType*, *MultiLevel*, and *CmaxOptimal*, which have different weights (Eq. (1)). The proposed similarity measure is a value between zero and one ($\in[0,1]$), with zero (0) corresponding to non-similar cases and one (1) corresponding to equal cases.

$$Sim = 0.5 \times Sim_{Njobs} + 0.25 \times Sim_{Nmachines} + 0.15 \times Sim_{ProbType} + 0.05 \times Sim_{MultiLevel} + 0.05 \times Sim_{\mathrm{Cmax}Opt} \quad (1)$$

It is given a greater importance to the number of jobs and to the number of machines, since those attributes define the problem dimension, which is a characteristic for MH parameterization. Some parameters are directly related to the problem dimension and its complexity, e.g., stopping criteria.

The proposed similarities of number of jobs and number of machines are calculated similarly, corresponding to the quotient of the lower value by the higher value (Eqs. (2) and (3)). Similarities values are uniformly distributed in the interval [0,1].

$$Sim_{Njobs} = \frac{\min(Njobs_1, Njobs_2)}{\max(Njobs_1, Njobs_2)} \quad (2)$$

$$Sim_{Nmachines} = \frac{\min(Nmachines_1, Nmachines_2)}{\max(Nmachines_1, Nmachines_2)} \quad (3)$$

The proposed similarities of problem type and multi-level value are binary (0 or 1), if the attributes are the same or not, respectively (Eqs. (4) and (5)).

$$Sim_{ProbType} = \begin{cases} 0, ProbType_1 \neq ProbType_2 \\ 1, ProbType_1 = ProbType_2 \end{cases} \quad (4)$$

$$Sim_{MultiLevel} = \begin{cases} 0, MultiLevel_1 \neq MultiLevel_2 \\ 1, MultiLevel_1 = MultiLevel_2 \end{cases} \quad (5)$$

The proposed similarity of the optimal makespan value, if known, is calculated similarly to the number of jobs or number of machines, if the values of both cases are positive (Eq. (6)). If any optimal makespan value is negative that means that the value is unknown. In those cases the optimal makespan value similarity is equal to zero.

$$Sim_{\mathrm{Cmax}Opt} = \begin{cases} \dfrac{\min(\mathrm{Cmax}Opt_1, \mathrm{Cmax}Opt_2)}{\max(\mathrm{Cmax}Opt_1, \mathrm{Cmax}Opt_2)} \\ \mathrm{Cmax}Opt_1 \geq 0 \; and \; \mathrm{Cmax}Opt_2 \geq 0 \\ 0, \mathrm{Cmax}Opt_1 < 0 \; or \; \mathrm{Cmax}Opt_2 < 0 \end{cases} \quad (6)$$

In the end, a list of most similar cases is retrieved, in order to work as a input to the Reusing phase.

4.4. Reusing phase

In the Reusing phase, a case from the list of most similar cases returned by the Retrieving phase is selected. The respective suggested solution works as a candidate solution to solve the new case. With this, the MH and respective parameters are returned to be used in the resolution of the new case.

First, it is checked if the list of retrieved cases is empty or not. If it is empty, it means that there are not cases similar enough with the new case (minimum similarity of 75%), being used the parameters pre-defined by the user/expert in the graphical interface. These pre-defined parameters are a starting point for the resolution of future cases similar with the new case.

If the list is not empty, it is selected the best cases considering effectiveness/efficiency criteria, or the most similar case, if there are not good enough cases. When there are cases very similar with each other (Eq. (7)), it is necessary to calculate the ratio between *Cmax-Opt* and *Cmax* (Eq. (8)), in order to detect the best effective/efficient cases. When there are not cases very similar with each other, it is selected the most similar case among all.

$$\frac{\min(Sim_{Case1}, Sim_{Case2})}{\max(Sim_{Case1}, Sim_{Case2})} > 0.95 \quad (7)$$

$$RatioCase_i = \frac{\mathrm{Cmax}Opt_{Casei}}{Cmax_{Casei}} \quad (8)$$

After the selection of the best cases considering effectiveness/efficiency criteria, it is randomly selected a case. With this, it is granted the selection of a good case, instead of selecting the best case among all. This is important, since it avoids stagnation and also the choice of the same case often (cycle), because in that way the system could not evolve. If the best case among all is selected, it would be selected always the same, unless new cases always obtain best results, which does not happen due to the randomness underlying MH.

If there are not cases with enough effectiveness/efficiency ratio, it is selected the most similar case (the case with higher similarity).

4.5. Revising phase

In this phase, the suggested solution is adapted, since the direct use of solutions leads the system to stagnation and it could not evolve to better results. So, to avoid local optimal solutions and system's stagnation, we propose an algorithm that applies some diversity and perturbation to the suggested parameters, using a global credit approach to assign different credit to the different parameters.

If the reused case has a similarity more than 95%, then the global credit is assigned with a minimum value of 15 (Eq. (9)). If not, the global credit is inversely proportional to the similarity of the reused case. This means that the less similar a case is, more perturbation should be included in the suggested MH parameters.

$$GlobalCredit = \begin{cases} 15, Sim_{Case_i} \geq 0.95 \\ 10 + (1 - SimCase_i) \times 100, Sim_{Case_i} < 0.95 \end{cases} \quad (9)$$

After initializing the global credit, it is randomly distributed to each parameter, according to the importance of the parameter. Usually, the parameter specifying the number of iterations is the most important, but it depends on the MH in use. After this distribution, the parameters are updated and the revised MH is returned.

The parameters update is made like illustrated in Eqs. (10) and (11), for integer and float values, respectively. Integer values are rounded to units and float values are rounded to the second decimal unit.

$$Param_i = Param_i + round\left(\left(Param_i \times \frac{Credit_{Param_i}}{100}\right), 0\right) \quad (10)$$

$$Param_i = Param_i + round\left(\left(Param_i \times \frac{Credit_{Param_i}}{100}\right), 2\right) \quad (11)$$

With this procedure, the system is able to introducing perturbation in the suggested solutions, being possible to escape from stagnation and evolve to better results.

4.6. Retaining phase

The objective of Retaining phase is to store the new solved case in the casebase. This retention is made in two phases. First, the makespan and execution time values are collected from the MAS. After that, a new record in the CBR table is created, with the case data, being also created a new record on the MH table, used to solve the problem. These two records are related by their primary key.

With this phase, the CBR cycle is concluded. In the next execution, the solved case will be available to be used in the resolution of a new case.

5. Computational study

The computational study pretends to analyze if the Self-Optimization module improves the effectiveness of AutoDynAgents system and how good is its contribution. It is not analyzed the efficiency because it is not expected to obtain better results, since a new module is incorporated in the system, with increasing computational effort.

For this study, it were used all instances from OR-Library Job-Shop Scheduling problems [77] (a total of 82 instances), proposed by Adams, Balas and Zawack [78], Fisher and Thompson [79], Lawrence [80], Applegate and Cook [81], Storer, Wu and Vaccari [82], and Yamada and Nakano [83]. These instances cover problems with 10, 20, 30, and 50 jobs, processed by 5, 10, 15, and 20 machines.

AutoDynAgents system, including Self-Optimizing module, was implemented using Java, with the Hibernate framework to work with the database developed in HSQLDB. The machine used for the computational study is a HP Z400 Workstation, with the following main characteristics: Intel® Xeon® CPU W3565 @ 3.20 GHz, 6GB RAM, Samsung HD103SJ disk with 1TB, and Windows 7 64-bit.

The MH used for the implementation and computational study were Tabu Search, Genetic Algorithms, Simulated Annealing, Ant Colony Optimization, and Particle Swarm Optimization.

The computational study is divided in three main phases. In the first phase were obtained results of 5 executions (runs) for each instance, before the introduction of Self-Optimization module, with a default parameterization for each MH. The two best cases and the best average value for each instance were retrieved, from the obtained results, in order to initialize the CBR casebase, working as a starting point for the next phase. In the second phase, with the objective to analyze the CBR evolution, were obtained results of 30 executions for each instance, after integrating the Self-Optimization module. Finally, in third and last phase, and similarly to the first phase, results from 5 executions for each instance were obtained, in order to compare them with the previous obtained results from the first phase.

At this point, three main objectives arise. First, understand how the proposed Self-Optimization module can evolve in its lifetime. Second, understand if CBR usage is worth or not, analyzing the comparison between obtained results before and after the integration of Self-Optimization module. Finally, this computational study also permits to understand how the system works in the presence of perturbations, and so, a dynamic EJSS problem resolution is presented.

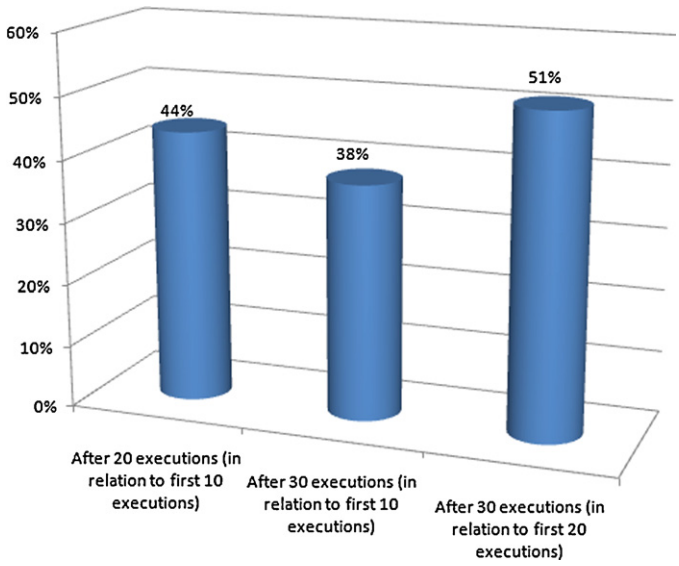

Fig. 6. Computational study – percentage of average results evolution.

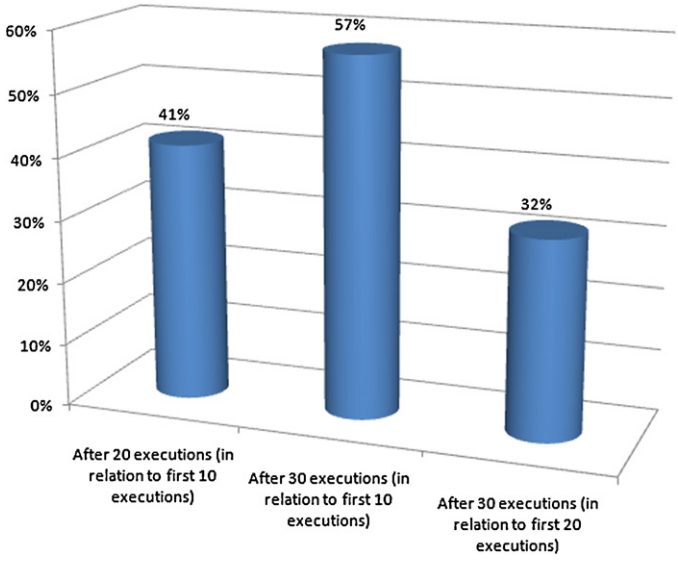

Fig. 7. Computational study – percentage of best results evolution.

5.1. Self-Optimization module evolution

To evaluate the evolution of the proposed Self-Optimization module, two main aspects were analyzed: the evolution of **average makespan** values and the evolution of **best makespan** values.

As previously explained, in this phase the system was executed 30 runs for each instance, resulting in a total of 2460 executions/cases. For each instance were obtained the results after 10, 20 and 30 executions.

According to the evolution of average results (Fig. 6), it is possible to understand that, after 20 executions, 44% of results were better than compared with the results obtained in the first 10 executions. It already represents a signal of evolution of the module. After 30 executions, the average results were improved by 38% and 51% when compared with the first 10 and 20 executions respectively. It might be strange since the percentage compared with 10 executions is lower than the percentage compared with 20 executions, but the explanation is that the average results can be deteriorated during lifetime. However it represents an interesting

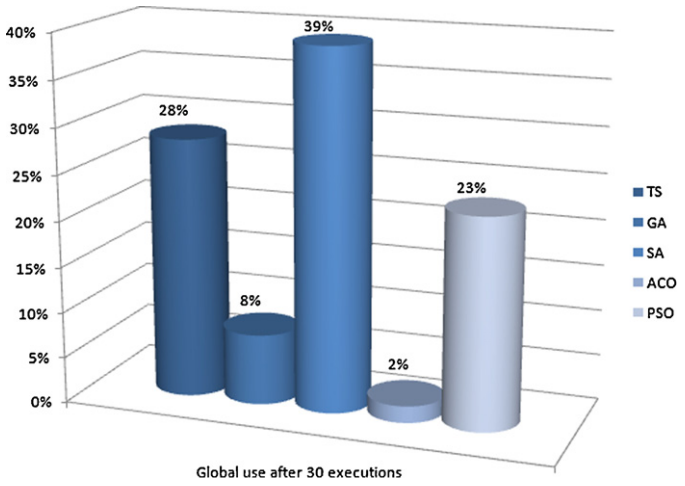

Fig. 8. Computational study – percentage of Meta-heuristics global use.

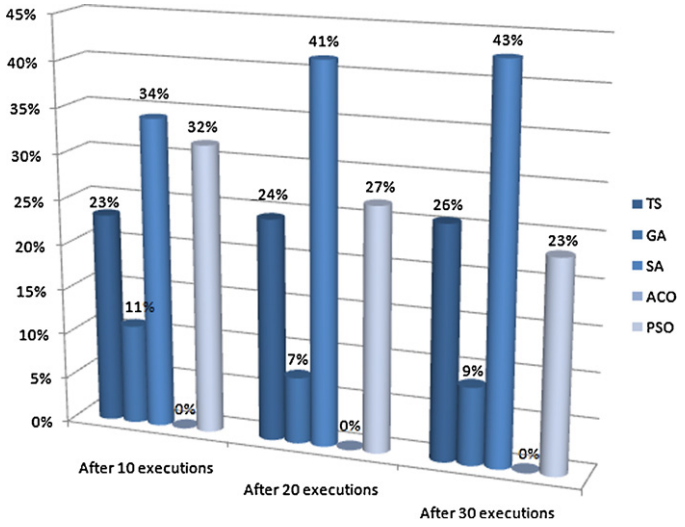

Fig. 9. Computational study – percentage of Meta-heuristics use obtaining the best results.

signal of system evolution, since in only 30 executions the results were improved by 50% in average.

Analyzing the best results evolution (Fig. 7) it is possible to conclude that, after 20 executions, the results were improved by 41% and after 30 executions were improved by 57% (comparing with the first 10 executions) and 32% (comparing with the first 20 executions). It also seems promising since the improvement of best obtained results after the 30 executions is better than 50% comparing with the first obtained results.

With this evaluation, it is also possible to conclude about the use of MH. Since the Self-Optimization module automatically choose and parameterize a MH according to the problem to solve, it might be interesting understand which MH were more used by the module. Two aspects could be analyzed: the percentage of MH global use (Fig. 8) and the usage percentage of each MH obtaining the best results (Fig. 9).

Fig. 8 represents the global use percentage of MH. The first thing to notice is the reduced use of ACO. The reason for this fact can be that ACO revealed to be very ineffective and it was used not many times. GA was also used in a very few times. The most used MH was SA, followed by TS and PSO.

Analyzing the percentage of obtaining best results, like shown in Fig. 9, it is possible to reach almost the same conclusions. ACO was

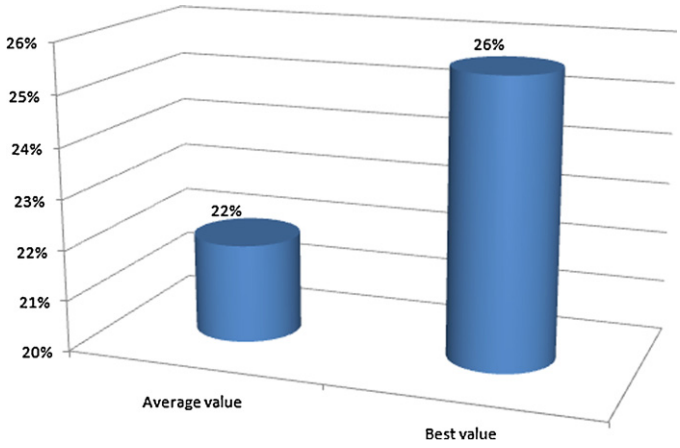

Fig. 10. Computational study – percentage of obtained results improvement.

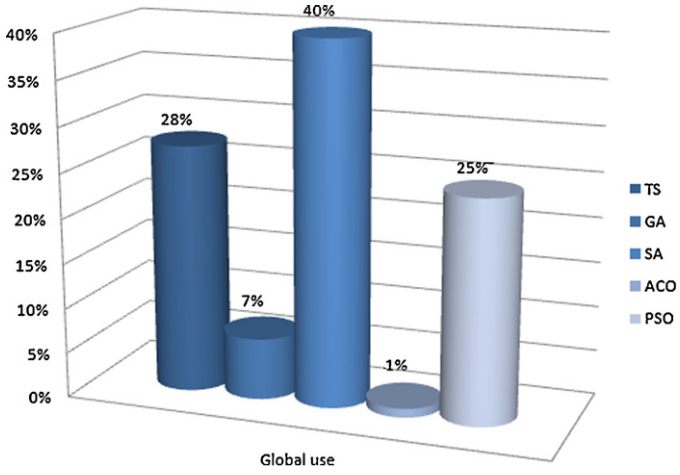

Fig. 11. Computational study – percentage of Meta-heuristics global use.

not selected anytime and GA was few times used. SA was the MH that obtained more best results, but a difference occurs after 10 and 20 executions. PSO is the second one, contrary to the conclusions reached in analyzing average results percentage.

5.2. Performance evaluation

In this subsection it is analyzed the comparison of results before and after the integration of Self-Optimization module. It is possible to analyze both makespan and execution times optimization criteria. Since computational times were deteriorated due to the higher processing charge (because a new module was introduced), it was only analyzed the evolution of makespan value (Fig. 10).

Fig. 10 represents the systematization of the percentage of improvement of average and best obtained results, after running the system by 5 executions (before and after the integration). So, the results were improved by 22% and 26% for average values and best values respectively. We consider it a very good evolution of the results, since the previous obtained results were considerate to be already very good. So, with the obtained improvements together with the ability of providing a self-parameterization of MH, we believe that the integration of Self-Optimization module is worth.

Similarly to the previous subsection, it is possible to understand the usage of MH. For the global use (Fig. 11), SA continuous to be the most used, followed by TS and PSO, and ACO the less used. GA is also very few times used.

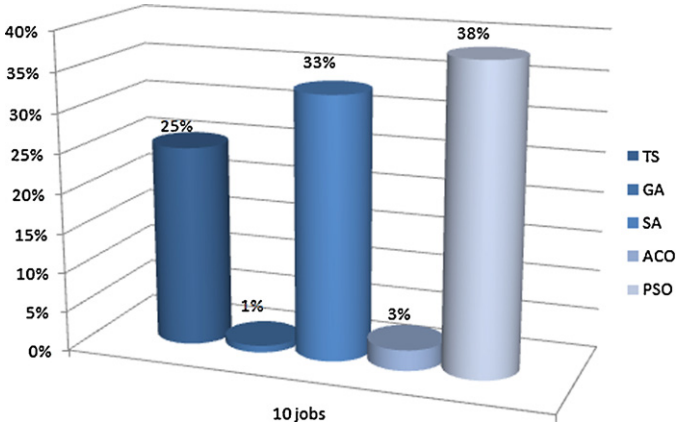

Fig. 12. Computational study – percentage of Meta-heuristics use for 10 jobs problem instances.

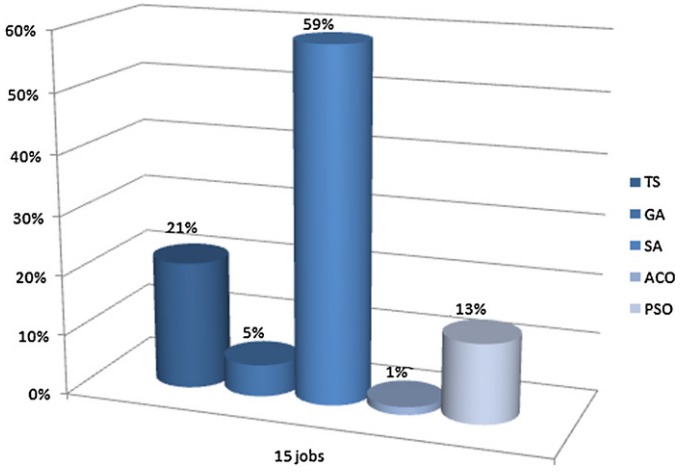

Fig. 13. Computational study – percentage of Meta-heuristics use for 15 jobs problem instances.

It is also possible to analyze the MH usage through classes of problems, related to their dimension. For 10 jobs problems (Fig. 12), PSO was the most used, followed by SA and TS, respectively. GA was the less used.

For 15 jobs problem instances (Fig. 13), SA was the most used, with 59% of usage, followed by TS and PSO. ACO was the less used, only with 1% of usage.

For instances with 20 jobs (Fig. 14), and similarly to 15 jobs instances, SA was the most used, followed by TS and PSO. ACO was not used at all.

For 30 jobs problem instances (Fig. 15), and similarly to 10 jobs instances, PSO was the most used, followed by SA and TS. The difference is in the usage of GA and ACO, since ACO was not used at all, like in 20 jobs instances.

For problem instances with 50 jobs (Fig. 16), and in opposite to other instances classes, TS was the most used, followed by SA, GA and PSO. ACO was not used, similarly to 20 and 30 jobs instances.

An interesting conclusion reached is about the using of GA. It is possible to observe that the GA use increased proportionally with problem dimension. Also, in problems with 50 jobs, GA was used in almost 25% of the cases, with TS being the most used MH, relegating SA to the second place.

Finally, analyzing the percentage of MH use obtaining the best results (Fig. 17), it is possible to compare before and after the use of CBR based module. Comparing the two moments, the results are very similar, with the main differences in the use of GA and ACO. With the integration of Self-Optimization module, ACO does not

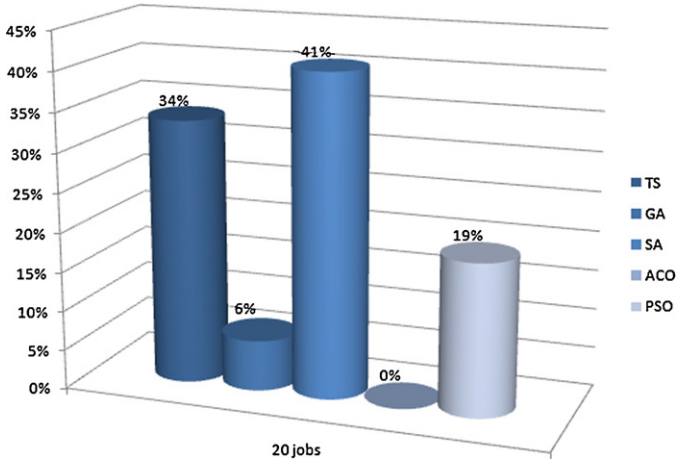

Fig. 14. Computational study – percentage of Meta-heuristics use for 20 jobs problem instances.

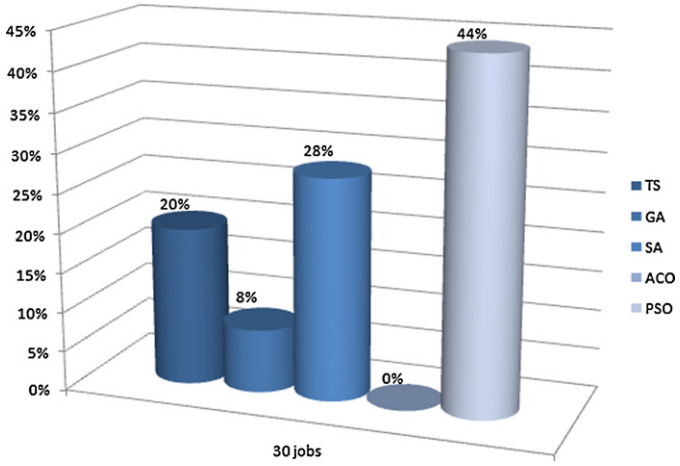

Fig. 15. Computational study – percentage of Meta-heuristics use for 30 jobs problem instances.

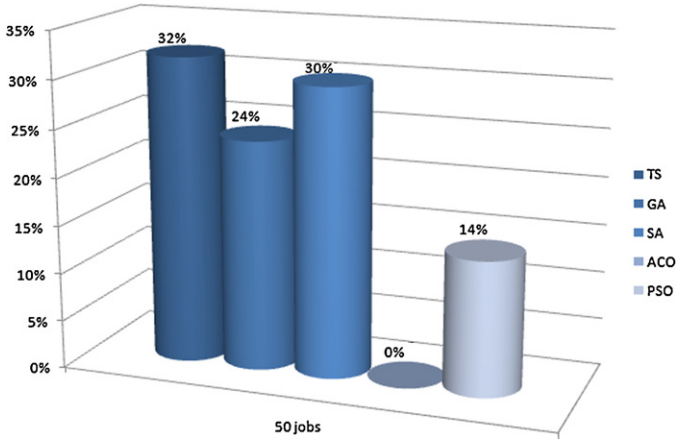

Fig. 16. Computational study – percentage of Meta-heuristics use for 50 jobs problem instances.

obtain any best result and GA only obtains 6%. TS, SA, and PSO are used in almost the same number of executions, so no differences are verified in this evaluation.

However, it can indicate two aspects. The first is that the parameterization of MH is becoming better, since the percentage of MH use is almost the same but the results were improved in 26% of

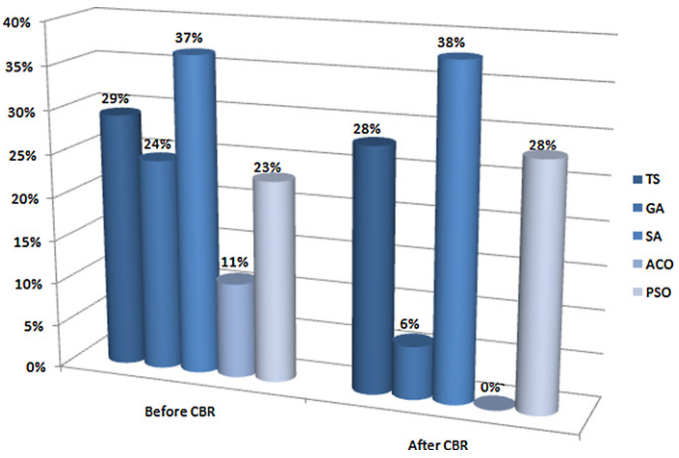

Fig. 17. Computational study – percentage of Meta-heuristics usage obtaining the best results, before and after CBR integration.

Table 7
Dynamic EJSSP – jobs data.

Jobs	Products	Priorities	Release date	Due date
J1	P1	3	0	55
J2	P2	2	2	70
J3	P3	2	1	85
J4	P4	1	5	65
J5	P5	3	0	72
J6	P1	1	6	85
J7	P3	1	6	98

Table 8
Dynamic EJSSP – occurred events.

Time	Event type	Job	Attributes
5	Job cancellation	J4	
12	Update of release and due date	J6	dt_rel = 28; dt_due = 88
15	Priority update	J7	prior = 3
30	Job arrival	J8	prior = 1; dt_rel = 32; dt_due = 87

the cases. Another aspect that can be happening is the more correct use of MH according to the problem characteristics, with the Self-Optimization module choosing and parameterizing a more effective technique.

It is possible to observe that SA is the most used MH, followed by TS and PSO, with a small difference of use about them. ACO and GA were the less used, which we can assume that they are the less effective MH in solving scheduling problems, in AutoDynAgents System.

5.3. Dynamic EJSS problem

In this subsection it is described the resolution of a dynamic EJSS problem using AutoDynAgents with the Self-Optimization module. This problem instance, which is an example of a dynamic problem in order to illustrate how the system works, is composed of 7 jobs and 8 machines and is better explained in [84].

Table 7 describes the data of the jobs processed in this dynamic problem. It is possible to notice different priorities and different release and due dates for each job, which are characteristics of EJSSP's.

Table 8 describes the list of occurred events in the simulation of dynamic problem. On time instant 5 it was made a cancellation of job J4. On time instant 12 release and due dates for J6 were changed. On time instant 15 it was changed the priority of J7. And in time instant 30 a new job arrives. Job J8 has the same structure than job J1.

Table 9
Dynamic EJSSP – obtained results.

	Meta-heuristic	Objective function	Cmax	Execution time
Initial Plan	ACO	Cmax	149	6.21
First Event	SA	Cmax	–	–
Second Event	SA	WT	–	–
Third Event	TS	WT	–	–
Final Plan	GA	Cmax	165	7.14

Initially it was made an off-line execution of the problem, obtaining an initial plan. The events were processed using this initial plan and a final plan was reached. The obtained results of the initial and final plans are presented on Table 9, also as the MH and objective functions used in the resolution of the events.

It is possible to notice that the Self-Optimizing module works similarly with and without dynamism. Whenever an event occurs, it represents a new case occurring in the system, allowing the MH swapping and respective parameterization, according to the current situation. With this module, the system becomes more robust when dealing with dynamic events.

6. Conclusions and future work

This paper envisaged the application of a learning module for the resolution of Scheduling problems. The proposed module is based on the AC Self-Optimization concept, with the objective to self-parameterize MH considering the importance of this issue in the process of designing and implementing those techniques. With this Self-Optimization module, the system is able to choose and parameterize MH autonomously, for the resolution of static or Dynamic Scheduling problems. The implementation of the learning process is based on CBR, which uses previous similar cases to solve new cases.

Analyzing the obtained results from the computational study, it was possible to reach some conclusions about results' improvement and MH usage. With the proposed Self-Optimization module, AutoDynAgents system's obtained results were improved by 22% and 26%, for average values and best values respectively. We consider it a very good evolution, since the previous obtained results were considerate to be already very good. It was also possible to conclude about the MH usage. Simulated Annealing, followed by Tabu Search and Particle Swarm Optimization, was the most used MH by the Self-Optimization module. Ant Colony Optimization was the less used and Genetic Algorithms was also few times used (except when solving 50 jobs instances). A curious conclusion reached was about the use of Genetic Algorithms, since it was possible to observe that their usage increased proportionally with problem dimension. We believe that the results can improve during the lifetime of the system and that the proposed module is a significant contribution for MH self-parameterization since the objective of this module is more about robustness than about finding the optimal solutions.

The proposed module can also be applied to the resolution of dynamic problems since each occurrence of perturbation corresponds to a new case. The system becomes more robust and effective in the resolution of scheduling problems with the existence of dynamic events, which represents an improvement of the system.

For future work we expect a extensive study of the proposed CBR module and the implementation of other learning techniques, to compare with this proposal.

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.

References

[1] P. Horn, Autonomic Computing IBM's Perspective on the State of Information Technology, IBM Research, 2001.
[2] J. Kephart, D. Chess, The vision of autonomic computing, Computer Magazine (January) (2003).
[3] E. Plaza, J.L. Arcos, F. Martin, Cooperative case-based reasoning, in: G. Weiss (Ed.), Distributed Artificial Intelligence Meets Machine Learning, Lecture Notes in Artificial Intelligence, Springer, 1996.
[4] T. Gonzalez, Handbook of Approximation Algorithms and Metaheuristics, Chapman&Hall/Crc Computer and Information Science Series, 2007.
[5] H.Md. Azamathulla, F.-C. Wu, A.Ab. Ghani, S.M. Narulkar, N.A. Zakaria, C.K. Chang, Comparison between genetic algorithm and linear programming approach for real time operation, Journal of Hydro-environment Research 2 (3) (2008) 172–181.
[6] H.Md. Azamathulla, A.Ab. Ghani, Genetic programming to predict river pipeline scour, ASCE, Journal of Pipeline System and Engineering Practice 1 (3) (2010) 127–132.
[7] Z. Ahmad, H.Md. Azamathulla, N.A. Zakaria, ANFIS-based approach for the estimation of transverse mixing coefficient, IWA-Water Science & Technology 63 (5) (2011) 1004–1009.
[8] K.Y. Chan, M.E. Aydin, T.C. Fogarty, Main effect fine-tuning of the mutation operator and the neighbourhood function for uncapacitated facility location problems, Soft Computing 10 (11) (2006).
[9] K.Y. Chan, C.K. Kwong, T.S. Dillon, Y.C. Tsim, Reducing overfitting in manufacturing process modeling using a backward elimination based genetic programming, Applied Soft Computing 11 (2) (2011) 1648–1656.
[10] K.Y. Chan, T.S. Dillon, C.K. Kwong, Modeling of a liquid epoxy molding process using a particle swarm optimization-based fuzzy regression approach, IEEE Transactions on Industrial Informatics 7 (1) (2011) 148–158.
[11] G. Beddoe, S. Petrovic, J. Li, A hybrid metaheuristic case-based reasoning system for nurse rostering, Journal of Scheduling 12 (2) (2009) 99–119.
[12] K.R. Baker, D. Trietsch, Principles of Sequencing and Scheduling, John Wiley & Sons, Inc., 2009.
[13] M. Pinedo, Scheduling: Theory, Algorithms, and Systems, Fourth edition, Springer, 2012.
[14] H. Aytug, M.A. Lawley, K. McKay, S. Mohan, R. Uzsoy, Executing production schedules in the face of uncertainties: a review and some future directions, European Journal of Operational Research 16 (1) (2005) 86–110.
[15] L. Monostori, J. Váncza, S. Kumara, Agent based systems for manufacturing, CIRP Annals-Manufacturing Technology 55 (2) (2006) 697–720.
[16] A. Madureira, Meta-heuristics application to scheduling in dynamic environments of discrete manufacturing, Ph.D. Dissertation, University of Minho, Braga, Portugal, 2003 (in Portuguese).
[17] M. Parashar, S. Hariri, Autonomic Computing: Concepts, Infrastructure, and Applications, CRC Press, 2006.
[18] S. Whiteson, P. Stone, Towards autonomic computing: adaptive job routing and scheduling, in: Proceedings of the 16th Conference on Innovative applications of Artificial Intelligence (IAAI'04), 2004, pp. 916–922.
[19] J. Abawajy, Autonomic job scheduling policy for grid computing, Lecture Notes in Computer Science 3516 (2005) 213–220.
[20] K. Ross, N. Bambos, Job scheduling for maximal throughput in autonomic computing systems, in: International Workshop on SelfOrganizing Systems, 2006.
[21] A. Madureira, I. Pereira, Self-Optimization for Dynamic Scheduling in Manufacturing Systems, Technological Developments in Networking, Education and Automation, Springer, Netherlands, 2010, pp. 421–426.
[22] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Computing Surveys 35 (2003) 268–308.
[23] F. Glover, Future paths for integer prog. and links to artificial intelligence, Computers & Operations Research 5 (1986) 533–549.
[24] G. Vilcot, J.-C. Billaut, A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem, European Journal of Operational Research 190 (2) (2008) 398–411.
[25] X. Wang, L. Tang, A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers, Computers & Operations Research 36 (3) (2009).
[26] J.-Q. Li, Q.-K. Pan, Y.-C. Liang, An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, Computers & Industrial Engineering 59 (4) (2010) 647–662.
[27] J.-Q. Li, Q.-K. Pan, N. Suganthan, T.J. 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 (2011) 683–697.
[28] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[29] V. Cerny, A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications 45 (1985) 41–51.
[30] S.-W. Lin, J. Gupta, K.-C. Ying, Z.-J. Lee, Using simulated annealing to schedule a flowshop manufacturing cell with sequence-dependent family setup times, International Journal of Production Research 47 (12) (2009) 3205–3217.
[31] P.-H. Chen, S.M. Shahandashti, Hybrid of genetic algorithm and simulated annealing for multiple project scheduling with multiple resource constraints, Automation in Construction 18 (4) (2009) 434–443.
[32] B. Naderi, R. Tavakkoli-Moghaddam, M. Khalili, Electromagnetism-like mechanism and simulated annealing algorithms for flowshop scheduling problems minimizing the total weighted tardiness and makespan, Knowledge-Based Systems 23 (2) (2010) 77–85.
[33] R. Zhang, C. Wu, A hybrid immune simulated annealing algorithm for the job shop scheduling problem, Applied Soft Computing 10 (2010) 79–89.
[34] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan, 1975.
[35] C. Darwin, On the Origins of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle of Life, 1859.
[36] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.
[37] H. Zhou, W. Cheung, L. Leung, Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm, European Journal of Operational Research 194 (3) (2009) 637–649.
[38] J.J.M. Mendes, J.F. Gonçalves, M.G.C. Resende, A random key based genetic algorithm for the resource constrained project scheduling problem, Computers & Operations Research 36 (2009) 92–109.
[39] L. De Giovanni, F. Pezzella, An improved genetic algorithm for the distributed and flexible job-shop scheduling problem, European Journal of Operational Research 200 (2) (2010) 395–408.
[40] D. Ouelhadj, S. Petrovic, A survey of dynamic scheduling in manufacturing systems, Journal of Scheduling (2008).
[41] V. Maniezzo, L.M. Gambardella, F. De Luigi, Ant colony optimization, in: G.C. Onwubolu, B.V. Babu (Eds.), New Optimization Techniques in Engineering, Springer, 2004, pp. 101–117.
[42] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: an autocatalytic optimizing, Technical Report, TR91-016, Politecnico di Milano, 1991.
[43] Z. Haipeng, G. Mitsuo, F. Shigeru, K.K. Woo, Hybrid ant colony optimization for job-shop scheduling problem, Faji Shisutemu Shinpojiumu Koen Ronbunshu 20 (2004) 304–305.
[44] M. Yoshikawa, H. Terai, A hybrid ant colony optimization technique for job-shop scheduling problems, in: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06), 2006.
[45] L.-N. Xing, Y.-W. Chen, P. Wang, Q.-S. Zhao, J. Xiong, A knowledge-based ant colony optimization for flexible job shop scheduling problems, Applied Soft Computing 10 (3) (2010) 888–896.
[46] B. Yagmahan, M. Yenisey, A multi-objective ant colony system algorithm for flow shop scheduling problem, Expert Systems with Applications 37 (2) (2010) 1361–1368.
[47] W. Xiang, H.P. Lee, Ant colony intelligence in multi-agent dynamic manufacturing scheduling, Engineering Applications of Artificial Intelligence 21 (2008) 73–85.
[48] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Procedures of the IEEE International Conference on Neural Networks, 1995.
[49] H. Liu, A. Abraham, O. Choi, S.H. Moon, Variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems, in: 6th international Conference, SEAL 2006, Springer, China, 2006.
[50] P. Pongchairerks, V. Kachitvichyanukul, A particle swarm optimization algorithm on job-shop scheduling problems with multi-purpose machines, Asia-Pacific Journal of Operational Research (APJOR) 26 (2) (2009) 161–184.
[51] D.Y. Sha, C. Hsu, A hybrid particle swarm optimization for job shop scheduling problem, Computers & Industrial Engineering 51 (4) (2006) 791–808.
[52] G.G. Yen, B. Ivers, Job shop scheduling optimization through multiple independent particle swarms, International Journal of Intelligent Computing and Cybernetics 2 (1) (2009) 5–33.
[53] H. Liu, A. Abraham, A.E. Hassanien, Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm, Future Generation Computer Systems 26 (8) (2010) 1336–1343.
[54] D. Karaboga, An Idea Based On Honey Bee Swarm For Numerical Optimization, Technical Report-TR06 (Erciyes University, Engineering Faculty, Computer Engineering Department), 2005.
[55] P. Pansuwan, N. Rukwong, P. Pongcharoen, Identifying Optimum Artificial Bee Colony (ABC) Algorithm's Parameters for Scheduling the Manufacture and Assembly of Complex Products, in: Second International Conference on Computer and Network Technology (ICCNT), 2010, pp. 339–343.
[56] R. Xiao, T. Chen, Enhancing ABC Optimization with Ai-Net Algorithm for solving project scheduling problem, in: Seventh International Conference on Natural Computation (ICNC), vol. 3, 2011, pp. 1284–1288.
[57] G. Zhou, L. Wang, Y. Xu, S. Wang, An effective artificial bee colony algorithm for multi-objective flexible job-shop scheduling problem, in: D.-S. Huang, Y. Gan, P. Gupta, M.M. Gromiha (Eds.), 'ICIC (2)', Springer, 2011, pp. 1–8.
[58] A. Banharnsakun, B. Sirinaovakul, T. Achalakul, Job shop scheduling with the best-so-far ABC, Engineering Applications of Artificial Intelligence (2011).
[59] L. Panait, S. Luke, Cooperative multi-agent learning: the state of the art, Autonomous Agents and Multi-Agent Systems (2005) 387–434.

[60] F. Zambonelli, H. Parunak, Towards a paradigm change in computer science and software engineering: a synthesis, The Knowledge Engineering Review 18 (4) (2004).
[61] M. Luck, P. McBurney, O. Shehory, S. Willmott, Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing), 2005.
[62] W. Shen, D.H. Norrie, Agent based systems for intelligent manufacturing: a state of the art survey, International Journal of Knowledge and Information Systems 1 (2) (1999) 129–156.
[63] W. Shen, D.H. Norrie, J.P.A. Barthes, Multi-agent Systems for Concurrent Intelligent Design and Manufacturing, Taylor & Francis, London, 2001.
[64] T.C.E. Cheng, C.T. Ng, J.J. Yuan, Multi-agent scheduling on a single machine with max-form criteria, European Journal of Operational Research 188 (2) (2008) 603–609.
[65] K. Lee, B.-C. Choi, J. Leung, M. Pinedo, Approximation algorithms for multi-agent scheduling to minimize total weighted completion time, Information Processing Letters 109 (16) (2009) 913–917.
[66] J. Leung, M. Pinedo, G. Wan, Competitive two-agent scheduling and its applications, Operations Research 58 (2) (2010) 458–469.
[67] A. Madureira, I. Pereira, N. Sousa, P. Ávila, J. Bastos, Scheduling a Cutting and Treatment Stainless Steel Sheet Line with Self-Management Capabilities, Computational Intelligence for Engineering Systems: Emergent Applications, Springer, 2011.
[68] J. Kolodner, Case-Based Reasoning, Morgan Kaufmann Publishers Inc, 1993.
[69] S. Petrovic, Y. Yang, M. Dror, Case-based selection of initialisation heuristics for metaheuristic examination timetabling, Expert Systems with Applications 33 (2007) 772–785.
[70] A. Aamodt, E. Plaza, Case-based reasoning: foundational issues, methodological variations, and system approaches, Artificial Intelligence Communications 7 (1994) 39–52.
[71] E.K. Burke, B.L. MacCarthy, S. Petrovic, R. Qu, Knowledge Discovery in a Hyper-Heuristic for Course Timetabling Using Case-Based Reasoning PATAT 2002, 2002.
[72] S. Oman, P. Cunningham, Using case retrieval to seed genetic algorithms, International Journal of Computational Intelligence and Applications (2001) 71–82.
[73] E. Xia, I. Jurisica, J. Waterhouse, V. Sloan, Runtime estimation using the case-based reasoning approach for scheduling in a grid environment, in: Proceedings of ICCBR'2010, 2010, pp. 525–539.
[74] A. Madureira, I. Pereira, N. Sousa, Collective intelligence on dynamic manufacturing scheduling optimization, in: Proceedings of the IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010), Liverpool, 2010, pp. 1693–1697.
[75] A. Madureira, N. Sousa, I. Pereira, Negotiation mechanism for self-organized scheduling system, in: Third World Congress on Nature and Biologically Inspired Computing (NaBIC), 2011, pp. 291–296.
[76] B. Horling, V. Lesser, A survey of multi-agent organizational paradigms, Knowledge Engineering Review 19 (4) (2004) 281–316.
[77] OR-Library, http://people.brunel.ac.uk/~mastjjb/jeb/info.html.
[78] J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling, Management Science 34 (1988) 391–401.
[79] H. Fisher, G.L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in: J.F. Muth, G.L. Thompson (Eds.), Industrial Scheduling, Prentice Hall, 1963, pp. 225–251.
[80] S. Lawrence, Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, 1984.
[81] D. Applegate, W. Cook, A computational study of the job-shop scheduling instance, ORSA Journal on Computing 3 (1991) 149–156.
[82] R.H. Storer, S.D. Wu, R. Vaccari, New search spaces for sequencing instances with application to job shop scheduling, Management Science 38 (1992) 1495–1509.
[83] 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.
[84] A. Madureira, I. Pereira, N. Sousa, Self-organization for Scheduling in Agile Manufacturing, in: Proceedings of the 10th IEEE International Conference on Cybernetic Intelligent Systems, London, 2011, pp. 38–43.