Computers and Electrical Engineering 40 (2014) 252–269

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Software Survey

Distributed job scheduling based on Swarm Intelligence: A survey ☆

Elina Pacini [a,c], Cristian Mateos [b,c,*], Carlos García Garino [a,d]

[a] ITIC, UNCuyo University, Mendoza, Argentina
[b] ISISTAN Research Institute, UNICEN University, Campus Universitario, Tandil, Argentina
[c] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina
[d] Facultad de Ingeniería, UNCuyo University, Mendoza, Argentina

ARTICLE INFO

Article history:
Available online 7 December 2013

ABSTRACT

Scientists and engineers need computational power to satisfy the increasing resource intensive nature of their simulations. For example, running Parameter Sweep Experiments (PSE) involve processing many independent jobs, given by multiple initial configurations (input parameter values) against the same program code. Hence, paradigms like Grid Computing and Cloud Computing are employed for gaining scalability. However, job scheduling in Grid and Cloud environments represents a difficult issue since it is basically NP-complete. Thus, many variants based on approximation techniques, specially those from Swarm Intelligence (SI), have been proposed. These techniques have the ability of searching for problem solutions in a very efficient way. This paper surveys SI-based job scheduling algorithms for bag-of-tasks applications (such as PSEs) on distributed computing environments, and uniformly compares them based on a derived comparison framework. We also discuss open problems and future research in the area.

1. Introduction and problem definition

Scientists and engineers, who usually rely on CPU-intensive simulations to perform their experiments, require a computing infrastructure – a High-Throughput Computing (HTC) environment – delivering large amounts of computational power. In HTC, jobs are dispatched to run independently on multiple computers in parallel, whose sub-results are joined later to obtain a general simulation result. In terms of their anatomy, these simulation are often organized as *bag-of-tasks* applications.

For example, PSEs are a popular way of conducting such simulations through which the same application code is run several times with different input parameters, which results in different output data. Running PSEs involves managing many independent jobs, since the experiments are executed under multiple initial configurations (input parameter values) many times, to locate points in the parameter space fulfilling some user-provided criteria. However, to deal with these problems, it is necessary large amounts of CPU cycles, and to have efficient scheduling strategies to appropriately allocate the workload and reduce the associated computation time. Here, the term "scheduling" represents the mechanism by which jobs are allocated to run on the machines of a distributed environment, since typically, there are many more running jobs than available machines. However, job scheduling is known to be NP-complete.

When scheduling *bag-of-tasks* applications, to minimize makespan it is essential to assign jobs correctly so that computer loads and communication overheads will be well balanced. Makespan is the finishing time of the last job in the system. On

☆ Reviews processed and approved for publication by Editor-in-Chief Dr. Manu Malek.

* Corresponding author at: ISISTAN Research Institute, UNICEN University, Campus Universitario, Tandil, Argentina.
E-mail addresses: epacini@itu.uncu.edu.ar (E. Pacini), cmateos@conicet.gov.ar (C. Mateos), cgarcia@itu.uncu.edu.ar (C. García Garino).

0045-7906/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compeleceng.2013.11.023

the other hand, load balancing refers to distributing workload between several machines to obtain as good throughput and resource utilization as possible.

Recently, SI, which refers to the collective behavior that emerges from social insects swarms [1], has been receiving attention among researchers. Swarms are able to solve complex problems that exceed the capabilities of their individual insects without central supervision. Then, researchers have proposed algorithms exploiting this idea for solving combinatorial optimization problems. As job scheduling is also a combinatorial optimization problem, many SI-based schedulers have been proposed.

We have conducted a literature review of SI-based works aiming at making job scheduling more efficient at a higher level of abstraction, according to several objective functions such as makespan and load balancing levels. Then, our focus is on how the proposed works enhanced/combined existing SI and traditional optimization techniques to derive distributed job schedulers, without paying attention to deployment and implementation issues, mostly because the number of available implementation technologies and platforms for Grids and Clouds is vast.

This work is organized as follows. Section 2 lists related surveys and explains how our work differs from them. Section 3 gives an overview of distributed computing infrastructures, particularly Grids and Clouds, and explains the job scheduling problem in distributed environments in the context of SI. Section 4 reviews these job schedulers. Section 5 identifies common characteristics and open issues. Appendix A explains the SI techniques exploited by the surveyed job schedulers.

2. Related work

The last decade has witnessed an astonishing amount of research in SI from both a theoretical and a practical perspective. As a consequence, many works have been proposed, which have been at the same time summarized in a number of surveys that can be considered as related to our survey. It is worth noting that we do not aim here at considering all possible SI surveys in the literature, but only those that are somewhat recent and relate to our work the most. These surveys can be used as a starting point to get insight into related issues and solutions in the area not covered in this paper.

Certainly, SI techniques have been extensively applied in optimization problems from several domains. For example, [2] reviews several algorithms exploiting Ant Colony Optimization (ACO) to solve diverse engineering problems. The areas covered are classical combinatorial problems (e.g., traveling salesman), network-related algorithms (e.g., [3]) and electrical engineering (e.g., efficient power dispatch). Another survey [4] discusses ACO-based approaches for classical industrial scheduling problems. In contrast, we are concerned with distributed job scheduling, in which jobs are executed in a Grid, Cloud or a computer cluster. However, job scheduling benefits PSEs, which are used to approximate problems from diverse disciplines and domains of Science and Engineering.

With regard to surveys analyzing existing job schedulers based on metaheuristics, a survey that deserves mention is [5], which covers job schedulers roughly grouped into three main categories, i.e., those exploiting Hill Climbing, Simulated Annealing (SA) and Tabu Search (TS); those exploiting Evolutionary Algorithms, ACO and Particle Swarm Optimization (PSO); and hybrid heuristic approaches. The survey is not completely focused on SI-based schedulers and does not consider newer SI techniques such as Artificial Bee Colony (ABC) and Artificial Fish Swarm Algorithm (AFSA). The same applies to [6], which reviews job schedulers based on traditional direct acyclic graphs and metaheuristics algorithms. Both surveys analyze efforts addressing job scheduling in Grids only, but we also consider computer clusters and Clouds.

Likewise, as SI algorithms are usually used to approximate difficult problems with medium to large-sized inputs, the resulting running times represent a threat to applicability. [7] analyzes solutions to increase the performance of such algorithms when dealing with large input data, for example when solving job scheduling problems in the presence of a sheer number of jobs or machines. Unlike our work, which reviews SI-based algorithms to make job scheduling more effective in terms of common scheduling metrics, [7] reviews approaches to boost the performance of the SI algorithms themselves to gain scalability. Besides, [7] reviews ACO-based approaches only, but we cover more SI techniques (PSO, ABC and AFSA).

Finally, building SI algorithms for minimizing or maximizing a set of metrics when solving an optimization problem, or *multiobjectivity*, is a hot topic in the area. In other words, multiobjectivity is the ability of an optimization technique of coping with several objectives simultaneously. Particularly, we are interested in SI-based schedulers dealing with the optimization of one or more scheduling metrics common in distributed environments, such as makespan and load balancing.

3. Background

Due to the fact that Grid Computing and Cloud Computing are nowadays the most used infrastructures to execute scientific applications compared to mainframes and conventional supercomputers, an overview of each one is provided next.

3.1. Grid Computing

Grid Computing [8] can be defined as a type of parallel and distributed infrastructure that enables the sharing, selection and aggregation of geographically distributed autonomous and heterogeneous resources dynamically depending on their availability, capability, performance, cost, and user's quality-of-service requirements.

A Grid, is a form of distributed computing whereby a "super virtual computer" is composed of many networked, loosely coupled computers acting together to execute very large jobs. As such, a Grid is a shared environment implemented via the deployment of a persistent, standards-based service infrastructure that supports the creation of, and resource sharing within, distributed communities. Resources can be computers, storage space, instruments, software applications, network interfaces and data, all connected to a network (private/public or local/the Internet) through a middleware that provides basic services for security, monitoring, resource management, and so forth. Resources owned by various administrative organizations are shared under locally defined policies that specify what is shared, who is allowed to access what, and under what conditions.

Grids provide the means for offering information technology as an utility for users, with those users paying only for what they use, as with electricity or water. Despite the widespread use of Grid technologies in scientific computing, some issues still make the access to this technology not easy for disciplinary or domain users. For example, operationally, some Grids are bureaucratic, since researchers groups have to submit a proposal describing the type of research they want to carry out to a central coordinator prior to executing their experiments.

3.2. Cloud Computing

While the bureaucratic issues of Grid can be a minor problem, the technical ones could constitute a fundamental obstacle for scientific computing. Cloud Computing [8] has been proposed to address the aforementioned problems. By means of virtualization technologies, Cloud Computing offers to end-users a variety of services covering the entire computing stack, from the hardware to the application level, by charging them on a pay per use basis, i.e., cycles consumed and bytes transferred during computations. The term "Cloud" is used to denote the infrastructure in which these services are hosted, which are commonly accessed by users from anywhere in the world on demand. Within a Cloud, *services* that represent computing resources, platforms or applications are provided across (sometimes geographically dispersed) organizations. This makes the spectrum of options available to scientists wide enough to cover any specific need from their research. Another important feature, is the ability to scale up and down the computing infrastructure according to the application requirements and the user's budgets.

As suggested, central to any Cloud is the concept of *virtualization*, i.e., the capability of a software system of emulating various operating systems on a single machine. By means of this support, users exploit Clouds by requesting them Virtual Machines (VM) that emulate any operating system on top of several physical machines, which in turn run a host operating system. Particularly, for scientific applications, the use of virtualization has shown to provide many useful benefits, including user-customization of system software and services, check-pointing and migration, better reproducibility of scientific analyzes, and enhanced support for legacy applications.

3.3. Job scheduling basics

In the aforementioned environments, job management is a key concern that must be addressed. Particularly, scheduling algorithms for distributed environments have the goal of managing a single computation as several jobs and submitting these latter to many resources, while maximizing resources utilization and minimizing the makespan. Considering that job scheduling is NP-complete, many heuristics have already entered the scene.

In distributed scheduling, from the point of view of solution quality, any scheduling algorithm can be classified into optimal or sub-optimal. The former characterizes scheduling algorithms that, based on complete information regarding the state of the distributed environment (e.g., hardware capabilities and load) and resource needs (e.g., job length), carry out optimal job-resource mappings. When this information is not available, or the time to compute a solution is unfeasible, sub-optimal algorithms are used instead. Sub-optimal algorithms are further classified into heuristic or approximate. First, heuristic algorithms are those that make as few assumptions as possible about resource load or job duration prior to perform scheduling. Approximate schedulers are based on the same input information and formal computational model as optimal schedulers but they try to reduce the solution space to cope with the NP-completeness of optimal scheduling algorithms. However, having this information again presents problems in practice.

All in all, in general heuristic algorithms are preferred in practice. One of the aspects that particularly makes SI techniques interesting for distributed scheduling is that they perform well in approximating optimization problems without requiring too much information on the problem beforehand. From the scheduling perspective, SI-based job schedulers can be conceptually viewed as hybrid scheduling algorithms, i.e., heuristic schedulers that partially behave as approximate ones.

4. Job scheduling based on Swarm Intelligence

SI finds its niche in routing applications and in specialized job scheduling algorithms. Not surprisingly, these two applications correlate very well with two fundamental traits of SI, i.e., positive feedback or reinforcing good solutions present in the system, and labor division. Moreover, social insects collectively solve complex problems, which are beyond their individual capabilities, in an intelligent and decentralized way. As a result, these collective, intelligent and decentralized behaviors of insects have become a model for solving job scheduling problems.

In recent years, several researchers have proposed algorithms based on ACO (Section 4.1), PSO (Section 4.2), ABC (Section 4.3) and AFSA (Section 4.4) for job scheduling problems in distributed environments, particularly Grids and Clouds. In turn, within each algorithm group, the associated job scheduling techniques are organized according to the main objective they are designed for. Finally, paragraphs associated to reviewed works have been built by digesting the associated paper(s) and contain the necessary details to understand the scheduling policy proposed by the authors from an algorithmic standpoint. Technical and implementation issues are left out of the scope of the paper.

4.1. Job Scheduling based on ACO

In this Section, approaches for job scheduling based on ACO are discussed. To date, the ACO algorithm has been applied to minimize the makespan, achieve a good load balancing in resources, minimize flowtime, minimize monetary cost, or different combinations of these.

4.1.1. Approaches minimizing makespan

In [9] have proposed an ACO-based scheduler for dynamic job scheduling in Grids where the availability of resources is constantly changing and jobs arrive to be executed at different times. Each processor executes only one job per unit time and each job is independent of each other. The authors have defined the Completion Time (CT) in which a machine finishes executing each job, measured as clock time. The CT of a job is computed by relating its arrival and release time, and the time the job spends in a machine. The scheduler includes four steps. First, pheromone initialization: the algorithm creates a number of artificial ants that start to work with a job. Second, state transition rule: each ant performs a move according to the pheromone trails and a heuristic that is used to determine the desirability of moving a job from one machine to another. Third, a local update rule is used by ants while constructing solutions to modify the pheromone level. Finally, a global update rule is applied and only the ant holding the best solution can leave pheromone on the path. At the end of each iteration, jobs will be moved among machines in the global best schedule.

In the ACO algorithm proposed in [10] a modified pheromone updating rule has been developed, which handles scheduling in Grids more effectively. The basic pheromone updating rule $\tau_{ij}(t)_{new} \leftarrow \rho \cdot \tau_{ij}(t)_{old} + \Delta\tau_{ij}(t)$ of the original ACO algorithm [1] has been changed to $\tau_{ij}(t)_{new} = (\rho \cdot \tau_{ij}(t)_{old}) + (\rho/(\rho+1).\ \Delta\tau_{ij}(t))$, where τ_{ij} is the trail intensity of the path (i,j) (j is a job and i is the machine assigned to the job j), ρ is a pheromone evaporation rate and $\Delta\tau_{ij}$ is an additional pheromone that is added by the scheduler when a job is moved to a machine. The modifications introduced to the pheromone update rule have improved the algorithm making it to perform more efficiently compared to the original ACO in terms of makespan.

Moreover, in [11] an ACO scheduler was introduced to address job scheduling within a Cloud. The proposed method is aimed to maximize scheduling throughput to handle all the diversified job requests according to different resources available in a Cloud, and minimize the makespan of a pool of jobs. In the algorithm each path between machines (r,s) has an associated distance or cost $\delta(r,s)$ and a pheromone concentration level $\tau(r,s)$. The pheromone updating rule is calculated considering a pheromone evaporation factor, and the cost $(\Delta\tau_k(r,s))$ incurred by ant k when (r,s) is its path. Then, every time a job request is processed on a machine, the pheromone concentration is updated for all the paths between machines by adding an evaporation factor within all machines. The whole heuristic is divided in two operation modes: online and batch modes. When the online mode is used, an arriving job request is immediately allocated to the first free resource. In batch mode, all jobs requests are first collected and the scheduler considers the approximate execution time for each job before making scheduling decisions.

The work proposed in [12] describes an ACO algorithm that has been complemented with Local Search (LS) and Tabu Search (TS), to find better schedules than other similar techniques. In this algorithm the authors have assumed that the expected execution time of the jobs on each machine is available beforehand. This information is held in an *nxm* matrix where a row represents the execution time on each available machine of each input job.

The authors have defined different types of matrices for simulating several heterogeneous scheduling scenarios in a realistic way based on three metrics: job heterogeneity, machine heterogeneity and consistency. Due to the fact that each job executes at a different speed on a different processor, this information is used to save information about which processors are suitable for each job. Therefore, the pheromone value is used by the scheduler in order to determine how desirable assigning a particular job into a particular processor is. The information encoded in the pheromone trail is used by an ant through a heuristic to build a solution. Furthermore, to leave a pheromone trail the authors have used the Max–Min Ant System (MMAS) described in [13]. Finally, the authors have applied a LS technique to exhaustively search jobs in the neighborhood and choose the swap that best reduces the schedule length.

4.1.2. Approaches maximizing load balancing

The work proposed in [14] focuses on an improved ACO for job scheduling in Grids. To work properly, the scheduler needs to have some initial information about resources in the Grid, i.e., the processors number, processing capability, communication ability, etc. These parameters are used to initialize the pheromone trail intensity. The way in which the pheromone trail is updated with respect to the classical ACO was modified by adding a load balancing factor. This factor indicates the finishing rate of a job in a resource, which makes the finishing rate of jobs in different resources similar, enhancing in turn the overall load balancing levels. The more the jobs completed, the greater the intensity of the pheromone trail. Contrarily, if the jobs are not finished then the pheromone trail decreases.

In [15] have proposed a job scheduling algorithm for Grids that uses a function *free*(i) to report when a machine i is released. When another job is assigned to machine i the new value of *free* is the release time associated to i plus the expected execution time of the submitted job. This algorithm uses a heuristic that allows to determine when a machine is released before. If the machine is released earlier, it will be more desirable in SI terms. The objective function is computed as the maximum value of the free function on the solution that each ant constructs. Furthermore, the additional pheromone value added to a trail by an ant includes an evaporation factor.

Furthermore, in [16] have proposed an algorithm that includes a mechanism for load balancing based on ACO and complex network theory. The algorithm was designed for Open Cloud Computing Federation (OCCF). A OCCF includes multiple Cloud providers devoted to create a uniform resource interface to users. A complex network is defined as a graph, which has topological characteristics that are not present in simple networks, i.e., lattices or random graphs that often can occur in real graphs. In the algorithm four steps are carried out. First, an ant is periodically sent from an underloaded machine to load balance the OCCF and update the pheromone on each machine. Second, an ant is sent out by a machine when this latter cannot properly handle its current workload. Then, the algorithm executes the previous step except when the machine of the ant is the machine with maximum workload. Third, once load balancing is performed the ant goes backward throughout the road that has followed and updates the pheromone on it. Each machine has a table that contains the pheromone trails with links to its neighbor machines, and stores values that represent the pheromone on the paths. Pheromone update includes both increase and evaporation. Finally, the complex network structure evolves to adapt to the changes in the workload after the pheromone has been updated. A complex network with the aforementioned characteristics is obtained through local behaviors of ants, since these characteristics are useful for the load balancing process in the proposed ACO algorithm.

4.1.3. Approaches minimizing makespan and maximizing load balancing

In the work proposed in [17] the authors have modified the classical ACO algorithm to address job scheduling in Grids. The modified ACO exploits LS and considers the available time of machines and the running time of jobs to improve machine usage and increase scheduler efficiency. The used LS technique is useful to define a solution neighborhood.

In the algorithm authors use a matrix of *NxM* entries, being *N* the input independent jobs and *M* the available resources. Each row represents the estimated execution time on each resource for each job. In this proposal, the *free* function introduced in [15] is exploited by a heuristic to find out the resource that is freed earlier. The pheromone level is updated by adding an evaporation factor and an additional pheromone value. To move a job among resources the algorithm calculates a probability to make the move. The probability takes into account the attractiveness computed by some heuristic, the level of pheromone trail of the moving and a new variable that represents the time a job takes to execute in a machine. An individual scheduling result in the modified ACO algorithm has four values (job, machine, starting time, completion time). These values are added to an output list which is then passed to an algorithm that uses the LS technique to further reduce the overall makespan.

In [18] have proposed the Balanced ACO (BACO) algorithm for job scheduling in Grids. In the proposal the pheromone level on a path stands for the *weight* of a machine. When a machine has the highest weight value means that the machine has the best processing power. For materializing the BACO algorithm the authors assume that each ant represents a job and the algorithm submits the ants to look for machines. BACO considers the load of each machine and modifies the pheromone in accordance to load across resources through local and global update functions. The local function modifies the status of resources after each job assignment round. The global update function, conversely, changes the status of resources upon any job finalization. The pheromone value in each resource is determined by adding the estimated job transfer and execution times upon sending a job to a resource for all jobs. The greater the pheromone level, the greater the efficiency of the resource for executing a job.

In the work presented in [19], the authors propose an extended ACO algorithm that exploits historical information regarding job execution within a Grid. Once the authors have results for n machines, they can compute the results for $n + m$ or $n - m$ machines very quickly by basing on the former results, being both n and m integer values representing a certain number of machines. When a resource enters the Grid, it submits its performance parameters, i.e., the number of CPUs, processing power, etc. These parameters are used for validating and initializing pheromone links. Later, the pheromone value changes every time a resource fails, or a job is assigned or there is some job results available/returned. Finally, the probability of assigning a job to a machine is calculated taking into account the pheromone intensity on the path to the resource, the initial pheromone value, and two parameter values that correspond to the pheromone importance and the innate attributes importance of the machine.

In the work proposed in [20], two distributed algorithms, one based on ACO (AntZ) and another one based on PSO, are presented. In the algorithm the jobs and ants are strongly linked. Each time a job is submitted for execution, an ant is created for finding the best machine to assign the job. Once this is done, the assigning ant stores information of the machines that has visited – load information – as a trail of pheromone in a load information table. The load information table contains information of load across all machines. Loads are stored in the table each time ants visit the machines with the aim of guiding other ants to select better paths. Authors have added to the proposed algorithm two rates: Decay Rate (DR) and Mutation Rate (MR). These rates are used when an ant moves from one machine to another. In this context, an ant can choose one of two possibilities. One possibility is to move towards a random machine according to the probability given by MR. Another alternative is to use the load information table in the machine to select the next destination. As time passes MR is decreased based on DR so the ant is more dependent on the load information table and not on random choice.

4.1.4. Approaches minimizing makespan, maximizing load balancing and minimizing monetary cost

In [21], the authors have presented a dynamic scheduling strategy that enhances [19] by considering the processing requirements of jobs, the capacity and the current load of the available resources, and the processing cost of those resources. The *possibility* of allocating a job to a resource is determined by calculating pheromone intensity on the path to the resource. Possibility refers to a numeric value that indicates how good a resource is for a given job. The pheromone intensity is moreover computed via a simple formula that combines two factors representing pheromone weight and resource weight, respectively.

The proposed algorithm arose in an attempt to improve the algorithm published in [19] where a job can be scheduled to a resource with low possibility even if the resources with high possibility are free. To avoid this problem, Sathish and Reddy have proposed that if the difference between the possibility of the resource selected for executing a job using the ant algorithm proposed in [19] and the possibility of the resource with the highest possibility is less than a certain threshold, then the job will be scheduled to the resource selected according to [19]. Otherwise, the scheduler selects another resource and the above procedure is repeated.

4.1.5. Approaches minimizing makespan, maximizing load balancing and minimizing flowtime

The work presented in [22] proposes a mechanism based on ACO to carry out efficient resource management on Grids. In this algorithm each job is carried by an ant that searches the less loaded machines and transfers the jobs to these machines for execution. When an ant assigns the job, the ant leaves some pheromone to tag a detected solution in a matrix. For each job-machine pair, the matrix has a single entry. Moreover, a parameter to define the pheromone evaporation rate is used. Then, to build a solution each ant uses heuristic information to guide its search. The heuristic value is an inverse lineal function of the minimum makespan for each job on the best available machine. Finally, the fitness function used for allocating and dealing with load balancing is the inverse of the makespan plus the mean flowtime of a solution. A factor is also used to weight and prioritize makespan, since it is the variable of utmost importance according to the authors' goal.

4.2. Job Scheduling based on PSO

In this section, the most representative approaches for job scheduling exploiting PSO are discussed. The goals pursued by the authors have been basically minimizing the makespan or the monetary cost of running jobs, or alternatively, minimizing the makespan while achieving good load balancing or low flowtime. Finally, another line aims at dealing with load balancing and monetary cost.

4.2.1. Approaches minimizing makespan

A heuristic approach proposed in [23] relying on the PSO technique was built for dealing with job scheduling in Grid environments. The authors have modified the classical PSO algorithm by varying the inertia weight used in the inertia term of the velocity equation. The inertia weight is varied by applying a scheme where the weight decreases over the whole run. The decrease rate depends on a start value and an end value of the weight given. The inertia term decreases linearly in order to facilitate exploitation over exploration in later phases of the search. Exploration is a wider search among alternatives, and exploitation is the refinement of a chosen alternative. In this algorithm each particle holds a potential solution. The dimension of a particle is associated to the number of jobs, while each dimension represents a job. The position vectors associated to particles are periodically transformed in order to appropriately change the continuous particle positions. To this end, a *smallest position value* approach helps in the process of finding a fitting permutation of the continuous positions.

Another heuristic approach proposed in [24] based on PSO was adapted to solve job scheduling problems in Grids. Again, each particle is encoded as in the previous algorithm [23]. Also, similar to [23], the authors have also employed a small position value approach. Each particle and its elements – position, velocity, fitness value – represent a possible solution. Furthermore, a sequence of jobs represents the order in which jobs are executed. In the proposed PSO, an LS algorithm is applied for permutation purposes. Moreover, this scheduling heuristic starts by considering a feasible initial solution and iteratively moving to a neighbor one. Usually, candidate solutions have many neighbor solutions. The neighbor solution chosen to move to in each step depends only on information extracted from the neighborhood of the current solution.

In [25] the authors propose a novel approach to PSO-based job scheduling in Grid environments. The way the position and velocity are represented in the conventional PSO is extended so as to move from vectors to fuzzy matrices. In this way, a brand new model for particles is constructed. Given a set G comprising machines and a set J of jobs to be executed, a generic fuzzy relation S between each machine and each job is established. For each cell in the matrix S, a membership function is applied and each cell of S represents the degree of membership exhibited by a machine in a feasible schedule when processing a job. To map the problem solution into particles, the authors have assumed that jobs and machines are arranged and ordered based on job lengths and machine processing speeds. Since the fuzzy matrix S represents the potential scheduling solution, the matrix should be decoded to get a feasible solution.

Further, to optimize makespan, the authors have proposed to swap alternatively the usage of two heuristics. When the overall job count does not exceed the number of machines, job allocation is carried out with a first come, first served policy and by complementary using an heuristic called Longest Job on the Fastest Node. Moreover, in the case of having more jobs to execute than available machines, jobs are assigned to machines via a heuristic called Shortest Job on the Fastest Node. In this way the machines will be released more quickly for each job.

The work in [26] introduced a PSO-based scheduler to assign jobs in heterogeneous computing systems and Grids. The algorithm updates particles in a discrete domain, and specifically the authors proposed a position update mechanism that takes into account the characteristics of discrete variables. The proposed PSO algorithm iteratively performs the following steps: First, each particle is associated to a vector containing N items (i.e., jobs), where each item is an integer number indicating the unique identifier of the machine to which a job is assigned. Second, the fitness value of a particle, i.e., makespan, is calculated. Smaller fitness values mean better particle quality. Third, to update the position of a particle the authors have defined a Perturbation Factor which decreases as iterations increase. Four, a Variable Neighborhood Search (VND) technique is used to complement the whole scheduling algorithm so as to increase convergence. Finally, a migration phase is used in the algorithm to rebuild a new population of individuals to more exhaustively explore the search space and prevent the algorithm as much as possible from selecting moderately-sized or small populations. The new individuals are obtained by using non-uniform random choices and by basing on the best individual. This steps are applied iteratively until a marginal improvement in the current best fitness value or an upper bound for the iterations is reached.

In the work presented in [27] the authors have introduced a discrete PSO for executing jobs in computer clusters. In the algorithm, a particle is defined as an array with as many elements as jobs to execute exist, and each dimension can be viewed as the machine where a job is allocated. Moreover, the process of updating velocities and positions has been modified introducing three new operators. A Subtract operator checks two position arrays and returns a new array where each value indicates whether the former array (current position) differs from the second one (desired position) or not. A Multiply operator generates binary vectors from two vectors and carrying out their multiplication. Finally, an Add operator represents a crossover operator usually employed in Genetic Algorithms (GAs). It interchanges structural information built during the search. To augment the effectiveness of the discrete PSO scheduler, [27] has been "hybridized" with a modified efficient LS algorithm. Based on a scheduling solution obtained from the discrete PSO, the LS algorithm reduces the makespan through suitable pairwise swapping of jobs between machines.

4.2.2. Approaches minimizing monetary cost

In [28] have proposed a heuristic based on PSO to schedule jobs to resources in a Cloud that considers both job computation costs and job data transfer costs. [28] dynamically optimizes the monetary cost of a job-resource mapping combination by basing on the solution obtained via the classical PSO algorithm. The optimization method relies on two components, namely the scheduling heuristic itself and the standard PSO steps for obtaining optimal job-resource combinations. In this context, one particle represents a resource-job mapping. The initial step of the heuristic computes the mapping for all jobs, which may have dependencies between them. The algorithm, to validate the various job dependencies, allocates the *ready* jobs to resources based on the output pairs as suggested by PSO. Ready jobs are those jobs whose parent jobs have already finished their execution and computed the input data for executing the child job. As jobs finish their execution, the ready list is updated. Then, the average latencies and bandwidth for transferring data between machines according to the current network usage are updated. In other words, since communication costs change over time, the PSO mapping is recomputed. Precisely, this periodic re-computation makes the heuristic to consider at runtime alternative mappings for jobs (online scheduling). This process repeats until all the jobs in the application are scheduled.

4.2.3. Approaches minimizing makespan and maximizing load balancing

In [29] the authors present a PSO algorithm for handling job scheduling in Grids. Specifically, a discrete variant of the PSO algorithm (i.e., DPSO) is proposed. The main difference introduced by DPSO with respect to the classical PSO algorithm is that the former deals with discrete variables and as such every component of a particle is represented as an integer number. Particularly, particle velocity and position are updated using the update rules from the conventional PSO algorithm. The results from the conventional position and velocity update rules are fixed to maximum values, so the current solution falls within a certain range in which it has a correct meaning in terms of job scheduling. This forces particles to move towards suitable areas in the search space, which means ensuring that each position is greater or equal than 1 and less or equal than the number of executing machines.

Finally, in the work proposed in [20] a distributed algorithm exploiting PSO (ParticleZ) for scheduling in Grids has been presented. In the ParticleZ algorithm, each machine is considered as a particle in a flock (i.e., the environment). The position of each machine in the flock is determined by its load. Particle velocity and position are defined in terms of the load difference that a machine registers with respect to its surrounding machines. Since particles try to balance the overall load, they move towards each other based on their position changes (i.e., load). These changes are achieved by exchanging jobs between machines. This is done locally in each machine, and gradually results in moving towards the global optimal load.

4.2.4. Approaches maximizing load balancing and minimizing monetary cost

In the work presented in [30] the authors model a scheduling problem for applications with dependencies between jobs by formulating and modeling the problem through a PSO-based approach. The authors base on a search space comprising n dimensions where every dimension of a particle position is mapped to one job, while the position value indicates a machine to which a job is allocated. Due to the fact that the particle position represents a potential schedule, it must be decipher back to obtain a solution. To minimize the monetary cost authors have introduced a function that optimizes both the makespan and the overall sum of the completion times, weighted by two non-negative weights whose sum is equal to 1.

4.2.5. Approaches minimizing makespan and minimizing flowtime

In [31] a version of a discrete PSO is introduced for addressing job scheduling in Grids. Under this algorithm, the particles are modeled to represent the order in which jobs are assigned to available machines. Moreover, the solutions are held in an $m \times n$ *position matrix*, being m the number of machines and n the number of jobs. All cells in the matrix of each particle have the value of 0 or 1, and in each column only one cell is set to 1. Columns then represent job assignations and rows represent assigned jobs within a machine. In the proposed PSO, the particle velocity, and the most suitable (or best) individual and global positions are also redefined as a matrix of $m \times n$ dimension. Here, the individual most suitable position is the best position that a particle has ever visited. The global best position is on the other hand the best position that all particles have ever visited.

4.3. Job Scheduling based on ABC

In this section, the most representative job scheduling approaches exploiting ABC are discussed. The goals pursued by the authors have been minimizing makespan, achieve a good load balancing in resources, minimize monetary cost, or combinations of these.

4.3.1. Approaches minimizing makespan

Ref. [32] proposes an Efficient Binary Artificial Bee Colony (EBABC), an extension of BABC [33]. The algorithm was proposed for solving job scheduling problems in Grids.

A weakness of the BABC algorithm is that the new randomly generated solutions by the scout bees in general have poorer quality compared with existing solution groups identified by the employee and onlooker bees. To solve this problem and to generate new solutions whose quality is not significantly different from existing solutions the authors have modified BABC by incorporating a Flexible Ranking Strategy (FRS), which is used to generate and employ new solutions for diversified search in early generations and to speed up convergence in latter generations. FRS step is introduced between the steps performed by onlooker bees and scout bees. In the FRS step the food sources of the employed bee population are arranged in order of ascending nectar value (makespan). Then the worst solution in the population is removed and a new solution is generated probabilistically as a combination of the best N solutions in the population. Moreover, two variants are introduced to minimize the makespan. In the first variant – called EBABC1 – a fixed number of best solutions is employed with the FRS, while in the second variant – EBABC2 – the number of the best solutions is reduced with each new generation. The advantage of EBABC2 is that in early generations the new food source is generated using all food sources in the swarm thereby improving the diversity of the search, while in later generations only the few best solutions are used improving the convergence of the algorithm.

4.3.2. Approaches minimizing makespan and maximizing load balancing

In [34], the authors have presented an ABC algorithm named Honey Bee Behavior inspired Load Balancing (HBB-LB), which aims to achieve well balanced load across VMs and minimize the makespan in a Cloud infrastructure. In the algorithm the VMs are grouped based on their loads in three sets: overloaded VMs, underloaded VMs and balanced VMs. Each set contains the number of VMs. Jobs removed from an overloaded VM have to a make decision to get placed in one of the underloaded VMs. A job is considered as a honey bee and the VMs with low load are considered as the destination of the honey bees. The information that bees update are load on a VM, load on all VMs, number of jobs in each VM, and the number of VMs in each set. Once the jobs switching process is over, the balanced VMs are included into the balanced VM set. Once this set has all the VMs, the load balancing process ends.

In [35], the authors have extended the classical ABC algorithm adding an additional step including a mutation operator after the process performed by the employed bees in ABC. The mutation operator is applied after the employed bees have explored the solution space. The selection of the food source is done in a random manner and the mutation operator is performed if a mutation probability is satisfied. Through mutation, there is a chance of changing the local best position, and the algorithm may not be trapped into local optima. When applying the mutation operator new food sources are produced. Accordingly, the new generated FSs replace the older if their fitness value are better.

4.3.3. Approaches minimizing makespan and minimizing monetary cost

In [36] the authors propose a modified ABC algorithm to deal with job scheduling in Grids. The algorithm was modified to implement a multi-objective version, called Multi-Objective Artificial Bee Colony (MOABC), that optimizes time and cost requirements. MOABC only requires two parameters: population size and mutation probability. Population size indicates the number of bees that are going to be maintained in each iteration or the number of food sources. Bees are represented by two vectors called allocation and order vector. The allocation vector denotes the current job allocation in the available Grid resources. The order vector indicates the order for the jobs execution.

In the multi-objective exploitation process, employed bees and onlookers search the best solutions from their last experience and the experience of their fellows. The employed bees generate a neighbor regarding to the mutation probability parameter per each vector. There are two types of mutations regarding to the two vectors: replacing mutation and reordering mutation. Moreover, in the exploration process, scout bees generate their allocation and order vectors from a random basis using information from the problem but without the experience of their fellow bees. Solutions selection is made when

the three types of bees are in the population. The solution of the previous iteration with the current are compared by means of a classification operator again. The new solution is saved as the solution for the current iteration and it will be compared with the solution from the next iteration. The new population is selected by the classification operator too, keeping the number of employed bees beginning from the first solution.

The work presented by [37] proposes an algorithm to schedule jobs on Grids via various additional techniques. In the proposed method three techniques are applied: Single Shift Neighborhood (SSN), Double Shift Neighborhood (DSN) and Ejection Chain Neighborhood (ECN). These techniques consist of moving jobs (nectar amount) between resources (food sources) in order to minimize the makespan and cost. The algorithm performs the following procedure: first, the employed bees are constructed guided by fitness values based on makespan and cost, i.e., makespan and cost represent the nectar amount in the food sources. Moreover, a food source with minimal makespan and cost has the greater probability to be chosen by a bee. Then, SSN and DSN are applied for each employed bee according to the fitness value of their food sources. SSN is a type of neighbor obtained from an original solution – food source – by moving the assignment of one job. Moreover, with DSN two jobs are moved. Second, when constructing a solution, the onlooker bees are assigned to employed bees according to a probability value associated with the food source. Third, ECN is applied for each employed bee based on the probability value. To apply ECN a new food source is obtained by performing multiple moves of jobs, where the number of moves is specified as length chain.

4.4. Job Scheduling based on AFSA

Due to the fact that AFSA is more difficult to implement, there are fewer studies using AFSA to job scheduling. The AFSA algorithm has been applied to maximize load balancing in resources and minimize the makespan and flowtime.

4.4.1. Approaches maximizing load balancing

In the work proposed in [38], the authors have presented an algorithm based on AFSA and GA for distributed job scheduling in wireless clusters named Fish Swarm Genetic with Survival Mechanism Algorithm (FSGSMA). FSGSMA is essentially a classical AFSA algorithm (see Appendix A) that includes a survival index comprising a food energy value related to an AF's current position, a consumption factor λ representing the energy consumed per unit time, and the life cycle of the AF. The optimization efficiency of the algorithm is enhanced using these survival information. A GA also operates upon each iteration of AFSA with the survival mechanism. In the algorithm, an iteration of the fish swarm represents a population in GA terms. In the solution set of the GA, AFs which can remain alive will be selected by analyzing their survival index.

4.4.2. Approaches minimizing makespan and minimizing flowtime

In [39] a Modified AFSA (MAFSA) algorithm for addressing job scheduling was proposed. The optimization criteria are stored by their importance, i.e., makespan first and flowtime second. In the structure of an AF, achievable solutions are stored in a vector where each element value represents the resource in which the scheduler assigns a job. AFSA bases on the idea of imitating the basic behavior of fishes (i.e., prey, swarm, follow) usually with LS of fish individuals, in order to reach a global optimum. Moreover, in AFSA, several parameters impact on the optimization result. With the objective of achieving higher levels of global convergence, the author has modified the conventional AFSA from two angles. First, the swarm behavior and the following behavior, in some degree, are local behaviors. When in AFSA the value of the objective function does not change after some iterations, the algorithm may converge to a local minimum. If the algorithm keeps iterating, each AF result will be similar and the probability to leap out of a local optimum becomes increasingly smaller. Precisely, avoiding local optimum and achieving a global optimum in MAFSA are ensured by means of a new leaping behavior added to AFs, which helps in properly balancing convergence rate and solution precision.

4.5. Analysis of the reviewed scheduling approaches

Table 2 summarizes the reviewed schedulers. Its columns are described below (a "–" cell value means that either a column does not apply to the work or the authors did not provide information):

- Algorithm type/environment: Is the base SI technique and the kind of distributed environment supported. SI technique may be "ACO", "PSO", "ABC" o "AFSA". On the other hand, "Grid" is an abbreviation for "Grid Computing" and "Cloud" refers to "Cloud Computing". Likewise, "Cluster" refers to conventional computer clusters.
- Objectives: Lists the objective variables to be minimized and/or maximized by the scheduler.
- Paper: Contains a reference to the paper in which the authors describe the proposed work.
- Additional technique: Indicates whether the authors have combined the proposed SI technique with another technique not strictly belonging to the SI area.
- Algorithm evaluation: Refers to the type of environment in which the scheduler was evaluated. Possible values for the environment are a real execution platform or middleware, a simulated environment (when details about the simulation tools are not given in the paper), or a specific ad-hoc or third-party simulation toolkit.

Table 1
Reference variables used across surveyed SI-based algorithms.

ACO	PSO
– α [0.1–50], importance of trail intensity – β [0.5–50], importance of resource – ρ [0.5–0.8], permanence of pheromone trail – P [0.1–0.99], overhead incurred in resource – τ_0 {0.01}, initial pheromone – c_e [0.003–1.1], encouragement factor – c_p [0.002–0.8], punishment factor – c {0.4}, factor for load balancing – *Ants* [1–30], number of ants in the colony – *Cost p/sec* [1–7], the processing cost per second of machines – *Mutation* {0.5}, probability to move an ant to a random machine – *Decay* [0–0.5], factor that causes *mutation* rate to decrease – *Steps* [1–10], maximum number of steps that performs an ant – *Threshold* [0.1–0.5], tells whether to assign a job to a resource	– *Swarm* [10–41], number of particles in a swarm – c_1 [0–2], self-recognition factor – c_2 [1,2], social factor – w [0–1.5], inertia factor – minVel {−0.4}, minimum velocity reached by the particles – maxVel [0.4–100], maximum velocity reached by the particles – *cost p/hs* [1.1–1.3], the processing cost per hour of machines – *link* {149}, number of connections between machines – λ [0–1] fitness function factor to prioritize makespan over flowtime
ABC	**AFSA**
– FS [20–40], number of food sources – Limit [100–20000], number of trials after which a food source is assumed to be abandoned	– *Visual* {3}, visual distance of a fish – *Step* {2.6}, moving step length – *Fishes* {200}, number of fishes in a swarm – δ {0.8}, crowd factor – η {0.9}, priority factor – *en* [168–438], computing energy – *cc* [7–25], communication cost

- Input variables: Here are the SI-specific variables used in the algorithms. Table 1 lists variable names and summarizes the ranges in which the control variable values lie across surveyed works. $\{v\}$ means that a fixed value was used, whereas $[v_a–v_b]$ means a range was used. For example, in Table 1, the range of values [20–40] for the food source variable in ABC represents the lowest food source value among all ABC related works, which comes from the work proposed in [32], and the highest value, which was used in the work proposed in [35]. A second example is the range of values [0.1–50] for the α variable in ACO. These values arise from the lowest α value from the proposed work in [9] and the highest α value, which was used in the work [12]. On the other hand, not all works rely on the same set of variables, as evidenced from Table 2.
- Experiment size: Describes the number of jobs and machines used in the performed experiments. This gives a hint on the extent to which scalability was assessed.
- Reproducibility: Indicates whether the authors have provided all the information needed to reproduce the experiments, which broadly includes variables (the values of all SI-related variables used in the algorithm), jobs (# of executed jobs, # of instructions in each case, and the input/output file sizes), and machines (the number of machines and processing cores, the processing power of each core, storage capacity, memory, and bandwidth). We have used three categories – high, medium and low – depending on the amount of information provided in each work to reproduce the experiments. For example, when all the information listed above is provided the category is considered "high". Reproducibility "medium" is considered when no more than two elements in each item from the list are missing. Finally, jobs are categorized as "low" when the authors do not provide enough information to reproduce the experiments.
- Resource allocation: Is the time at which jobs are allocated to resources. *Static* means that when the allocation takes place, the scheduler has complete information in advance of both the jobs and the resources. *Dynamic* supports scheduling of jobs arriving at different times, and moreover operates well even when resource availability changes over time. A *hybrid* resource allocation is when some of the jobs are known in advance, and other jobs arrive at different times to be scheduled for execution and/or details of them are not available until they are received.
- Application awareness: Tells whether the proposed works have considered in the algorithm both the computational burden of moving the data – i.e., input data and code – associated to the jobs to be processed and executed (awareness = "full"), or only the computational cycles of the job (awareness = "partial"). For example, works in the former category manage job allocation based on the underlying network as well as processing capabilities, whereas proposals in the latter typically consider processing power only.
- Compared with: Refers to the algorithms and techniques against which the authors have compared their work in their experiments. Scheduling algorithms found include:
  - Simple algorithms such as Random, Min–Min, Max–Min, FCFS (First Come First Served), FPLTF (Fastest Processor to Largest Task First) and RR (Round Robin) in which time slices are assigned to each job in equal portions in a circular order.

Table 2
Summary of the analyzed approaches.

Algorithm type/ environment	Objective variables	Paper	Additional technique	Algorithm evaluation	Values of variables	Experiment size	Reproducibility	Resource allocation	App. awareness	Compared with
ACO/Grid	Makespan	[9]	–	GridSim toolkit	$< (ants, 30), (\alpha, 0.1), (\beta, 2.0), (P, 0.1) >$	3000 jobs10–20 machines	High	Dynamic	Partial	MTEDD, MTERD, FCFS
		[10]	–	GridSim toolkit	–	10–100 jobs 5 machines	Low	Static	Partial	Classical ACO
		[12]	LS and TS	Simulated Grid	$< (ants, 10), (\alpha, [1–50]), (\beta, [1–50]), (\rho, 0.75), (\tau_0, 0.01) >$	512 jobs 16 machines	Medium	Static	Partial	GA, Min–Min
	Load bal.	[14]	–	Simulated Grid	$< (\rho, 0.99), (\alpha, 0.5), (\beta, 0.5), (c_e, 0.003), (c_p, 0.002), (c, 0.4) >$	1000 jobs 10 machines	High	Dynamic	Partial	Classical ACO
		[15]	–	Simulated Grid	$< (\tau_0, 0.01), (\rho, 0.5), (ants, 1) >$	20 jobs 5 machines	Medium	Hybrid	Partial	FFM
	Makespan, load bal.	[17]	LS	Simulated Grid	–	512 jobs 16 machines	Medium	Dynamic	Partial	Min–Min
		[18]	–	Real Grid (Taiwan UniGrid + GT4 Globus toolkit)	$< (P, 0.99), (\alpha, 0.5), (\beta, 0.5), (c_e, 0.003), (c_p, 0.002), (c, 0.4) >$	1000 jobs 25 machines	High	Dynamic	Full	ACO [14], FPLTF, Random
		[19]	–	Simulated Grid	$< (\rho, 0.8), (\alpha, 0.5), (\beta, 0.5), (c_e, 1.1), (c_p, 0.8) >$	20 jobs 10 machines	High	Dynamic	Partial	Classical ACO
		[20]	–	GridSim toolkit	<(*mutation*,0.5), (decay,0-0.5), (*Steps*,[1–10])>	1000 jobs 100 machines	High	Dynamic	Full	SBA, Random, PSO (ParticleZ) [20]
ACO/Grid	Makespan, load bal.	[22]	–	Simulated Grid	$< (\rho, 0.8), (\alpha, 15), (\beta, 10) >$	512–4096 jobs 32–256 machines	Medium	Dynamic	Partial	TS
	Makespan, load bal., cost ($)	[21]	–	GridSim toolkit	<(*cost p/sec*, [1–7]), (*threshold*, [0.1–0.5])>	40 jobs 20 machines	High	Dynamic	Partial	Classical ACO
ACO/Cloud	Makespan	[11]	–	Simulated Cloud	$< (ants, 25), (\tau_0, 0.01), (\rho, 0.5) >$	25 service requests 5 machines	Medium	Dynamic	Full	Classical ACO
	Load bal.	[16]	–	Third-party Java simulation toolkit	–	1000 jobs 100 machines	Medium	Dynamic	Partial	Messor algorithm
PSO/Grid	Makespan	[23]	–	Alea toolkit	<(minVel, -0.4), (maxVel, 0.4), (w, [0.3–1.5])>	9 jobs 3 machines	High	Dynamic	Partial	Classical PSO
		[24]	LS	Simulated Grid	<(*swarm*, 30), $(c_1, 2), (c_2, 2), (w, [0.4 - 0.9])$, (*maxVel*, 100)>	100–200 jobs 5–20 machines	High	Dynamic	Partial	GA
		[25]	–	Simulated Grid	<(*swarm*, 20), $(c_1, 1.49), (c_2, 1.49), (w, [0.1 - 0.9]) >$	13–100 jobs 3–10 machines	High	Dynamic	Partial	GA, SA
		[26]	VND	Mathlab	$< (swarm, 32), (w, [0 - 0.3]) >$	512 jobs 16 machines	High	Dynamic	Partial	GA, SA, TS
	Makespan, load bal.	[29]	–	Mathlab	$< (swarm, 60), (c_1, 2), (c_2, 2), (w, [0 - -1]) >$	60–80 jobs 10–20 machines	Medium	Dynamic	Partial	Max–Min
		[20]	–	GridSim toolkit	$< (link, 149), (c_1, 0), (c_2, 1), (w, 0) >$	1000 jobs 100 machines	High	Dynamic	Full	SBA, Random, ACO (AntZ) [20]
PSO/Grid	Load bal., cost ($)	[30]	–	Simulated Grid	<(*swarm*, 20), $(c_1, 1.49), (c_2, 1.49), (w, [0.1 - 0.9]) >$	7 jobs 3 machines	High	Static	Partial	GA

	Makespan, flowtime	[31]	–	Ad-hoc VC++ toolkit	<(*swarm*, 50), (c_1, 1.5), (c_2, 1.5), (*w*, [0.1 − −0.9]), (λ, [0 − 1]), (*maxVel*, 30)>	512 jobs 16 machines	Medium	Dynamic	Partial	Min–Min, GA, ACO [12], PSO [25]
PSO/Cluster	Makespan	[27]	LS	Mathlab	<(*swarm*, 10)>	10–100 jobs 2–10 machines	High	Static	Partial	SA
PSO/Cloud	Cost ($)	[28]	–	Jswarm toolkit	<(*swarm*, 25), (*cost p*/*hs*, [1.1–1.3])>	5 jobs 3 machines	Medium	Static	Full	BRS
ABC/Grid	Makespan	[32]	FRS	Simulated Grid	<(FS, 20),(Limit, [100–8000])>	13–1000 jobs 3–100 machines	Medium	Dynamic	Partial	BABC
	Makespan, load bal.	[35]	–	Simulated Grid	<(FS, 40), (Limit, 20,000)>	17–30 jobs 5–12 machines	Low	Dynamic	Partial	GA
	Makespan, cost ($)	[36]	–	GridSim toolkit	–	60 jobs 11 machines	Low	Dynamic	Partial	DBC WMS
		[37]	SSN, DSN, ECN	Simulated Grid	–	128–1024 jobs 8–64 machines	Medium	Dynamic	Partial	ACO
ABC/Cloud	Makespan, load bal.	[34]	–	CloudSim toolkit	–	10–40 jobs 3–7 machines	Low	Dynamic	Partial	FCFS. RR,DLB
AFSA/Grid	Makespan, flowtime	[39]	–	Simulated Grid	<(*visual*, 3), (*step*, 2.6), (*fishes*, 200), (δ, 0.8), (η, 0.9) >	13 jobs 3 machines	Medium	Dynamic	Partial	GA, SA
AFSA/Cluster	Load bal.	[38]	GA	Mathlab	<(en, [168–438]), (cc, [7–25])>	7 jobs 4 machines	Medium	Static	Full	Classical AFSA

- Time-based heuristics, particularly MTEDD (Minimum Time Earliest Due Date), MTEDD (Minimum Time Earliest Release Date), DBC (Deadline Budget Constraint) and WMS (Workload Management System). MTEDD orders the sequence of jobs to be serviced from the job with the earliest due date to the job with the latest due date. MTEDD prioritizes jobs with earliest release dates. DBC tries to keep the deadline and budget (cost) of a specific experiment within certain limits. Finally, WMS considers hardware or software requirements, such as storage capacity, processing performance or operating system.
- Greedy algorithms, particularly First free machine (self-described), BRS (Best Resource Selection) and DBL (Dynamic Load Balancing). BRS maps a job to the resource that is able to deliver the minimum finish time, i.e., a resource with good computing power and low load. DBL allocates/reallocates resources at runtime based on no-a-priori job information, which may determine when and which of their jobs can be migrated.
- Algorithms based on traditional metaheuristics, i.e., approaches exploiting GAs, techniques based on SA, or TS.
- Algorithms either based on modified versions of any of the previous algorithms, or miscellaneous algorithms. The latter group includes the Messor algorithm and SBA (State Broadcast Algorithm). Messor is an ant-inspired algorithm that is based on two complementary processes: SearchMax and SearchMin (an ant explores the network to find an overloaded and an underloaded machine, respectively). Under SBA, whenever a job arrives to or departs from a resource, i.e., this latter broadcasts a status message informing the situation.
- Algorithms strictly based on unmodified (classical) versions of the SI techniques.

In the ACO algorithms included in Table 2, pheromone trail modeling and modification has been subject of great attention to achieve the proposed objectives. For example, in [14,18] a load balancing factor has been added to the pheromone trail. With this factor the resources have similar completion rates, and thus the ability of load balancing the overall system is improved. Many authors [9–11,22] included in the pheromone update rules a pheromone evaporation rate or a pheromone permanence rate. The pheromone evaporation rate is used to prevent that other ants choose those paths. The pheromone permanence rate strengthens the more interesting paths, i.e., the paths more frequently used by ants.

Specifically, [18] changes the pheromone update rules (local and global) to achieve better load balancing. The local update rule refreshes a selected resource status after job allocation. Moreover, the global update rule refreshes the status of resources after each job finishes. Thus, the scheduler keeps updated information of all resources in every allocation step. Moreover, in [12,17] the authors have combined the classical ACO algorithm with other techniques or algorithms, such as LS or TS. These algorithms have been helpful for researchers to obtain better results than classical (raw) SI approaches. However, just by looking at the Table 2 it seems that the community is still influenced by the idea of developing pure SI-based approaches.

Within the surveyed PSO algorithms, some authors have proposed to modify the way in which particle position and particle velocity are represented. Some of these modifications involved the discretization of the vectors associated to these two variables [24,29,31] and the application of fuzzy logic [25]. Other authors [20] added or modified some terms used to calculate the position or velocity of a particle. In [28] have used two matrices to incorporate the computation and network communication costs that are then minimized. Instead, in [23] have modified an inertia parameter that causes a particle to move always in the same direction.

From the studied literature follows that most of the works do not take into account jobs dependencies, with the exception of [28,30]. In most simulation-based experiments, job dependency is not nevertheless a necessary feature as they are based most of the time on *independent* jobs.

In the works based on ABC, authors have introduced different changes to the classical ABC to improve performance. For example, in the works [32,37] the authors have combined the ABC with other techniques. In [32] authors have incorporated a FRS strategy with the aim to generate and use new solutions for diversified search in early generations and to speed up convergence in latter generations. On the other hand, in [37] the authors have applied different neighborhood techniques (SSN, DSN and ECN). In [34], the authors have presented an ABC algorithm in which machines are grouped according to their loads in order to achieve a good balance among them. In the algorithm each job is considered as a honey bee and the machines with low load are considered as the destination of these bees. Moreover, in [35,36] the authors have modified the classical ABC by adding additional steps using a mutation operator to explore new areas of the solution space.

With respect to AFSA, in [39] the MAFSA has been proposed, through which the authors calculate food concentration and model the structure of an artificial fish in a novel way. On the other hand, in [38] have proposed to include in the conventional AFSA algorithm a survival index to improve efficiency. Here, the modified AFSA algorithm is combined with GA to achieve faster convergence and good load balancing.

A general remark is that most works have been validated in simulated environments, with a number of used jobs that do not exceed 1000 jobs. As an evaluation approach, many works rely on simulation and particularly using the GridSim simulation toolkit (http://www.buyya.com/gridsim). Within the distributed computing community, it is broadly accepted to establish simulated experimental scenarios due to the inherent difficulty of performing tests in real environments. Nevertheless, in real scientific experiments, the number of jobs can far exceed that amount. It would then be interesting to consider how these algorithms respond to situations of greater stress on the machines, at least in simulated scenarios.

Regarding the reproducibility of the experiments, not all the works surveyed can be completely replicated. The authors have not always provided the information necessary to reproduce the experiments they performed. It is desirable, however, to provide complete information of all experiments settings to corroborate through their replication the efficiency of the

algorithms. This is important on an area like SI, where many algorithms and combinations of these are useful to approach a given optimization problem.

A crucial issue to achieve good performance in distributed environments is resource allocation, which can be either static or dynamic. Table 2 shows that some researchers [10,12] have proposed static job scheduling algorithms. Since both Grids and Clouds are environments where the availability of resources is by nature highly dynamic and jobs arrive over time, applying static resource allocation is unrealistic in practice. On the other hand, in real systems available today it is difficult to obtain complete information about the jobs and resources in advance. Then, hybrid resource allocation schemes have been also proposed to provide a balance to this trade-off.

Another feature among the works is the application awareness considered. Table 2 evidences that most authors have considered in their experiments only the computational cycles for executing jobs, i.e., partial "awareness", while only six authors have also considered the computational burden of moving the data of jobs to be processed, i.e., "full awareness". Although it is important to consider in a real application the extra time that demands the computational load due to data transfer, in some applications input and output parameter sizes (e.g., data files) are not large enough to alter the execution time. The presence of data-intensive scientific applications, conversely, renders partial awareness algorithms not applicable.

On the upside, each one of the changes introduced by the authors to the raw SI algorithms has made the resulting algorithms more efficient according to their objectives. Table 2 evidences, however, that most of these algorithms are focused on minimizing makespan and achieve a proper load balancing, but they do not deal with other interesting and important metrics such as energy use. Precisely, efforts in distributed and parallel computing have traditionally focused on improving speedup, i.e., minimizing execution times of the applications. Due to the large growth of scientific applications, more and more resources are necessary for processing, and energy consumption has become a crucial problem due to high electricity costs and CO_2 emissions. This has gave birth to a new field called Green Computing. It is then important to minimize energy consumption within a distributed environment as well as the energy consumed to move data to/from the environment. We believe that since ACO-based job scheduling techniques are highly efficient in optimization problems they are also good candidates to address this problem. Energy consumption could be then a variable within the objectives of a new algorithm. Indeed, a search for recent journal papers regarding distributed SI-based job schedulers that consider energy consumption, performed at the time of writing this work, resulted in few papers, which shows the undeveloped nature of the topic.

Another feature among the works is the type of environment in which experiments have been performed. For job schedulers for Grids and traditional clusters, the machines used to run the experiments were always heterogeneous, i.e., the machines had different number of processing elements and computing power. In works focused on Cloud Computing, the authors have used homogeneous machines, i.e., each machine has the same processing power. This fact occurs because in Clouds is the user who decides the number and type of resources that need to run applications, as the infrastructure is much more tailorable, and hence homogeneity is often preferred. For this reason, resources are homogeneous and have the same or similar specifications, i.e., the same computing power, available memory, and so on. This eliminates the necessity of elaborated SI schedulers, but as Clouds evolve from location-centric computing environments to *federated Clouds* arranging several local Clouds, resource heterogeneity will increase, thus requiring more complex SI-based solutions.

Finally, a distinctive feature of the surveyed works not shown in Table 2 is that they do not consider job priority. Particularly, for running PSEs, this is a very important aspect. For example, when designing a PSE as sets of N_s jobs, where every job in a set s is associated a particular value for the ith variable of the model being simulated by the PSE, job running times between sets can be very different. This is since running the same PSE code or solver (i.e., job) varies according to the variable and mathematical model being explored and executed. Sometimes important variations may occur between jobs in the same set as well. These situations are very undesirable since the user can not process/visualize the outputs until all jobs within a PSE finish. Therefore, giving higher priority to individuals carrying jobs that are supposed to take longer to finish may help in reducing makespan, and hence improve output processing at the user end.

5. Conclusions

From the 30 analyzed works summarized in Table 2, it seems there are shortcomings concerning aspects such as algorithm evaluation, experiment size, reproducibility and comparison. Only one work was evaluated on a real distributed environment. Moreover, just a 48% of the works have been evaluated with a number of jobs in the order of 100 or 1000 jobs (18% and 30%, respectively), and only 5 works have used $\sim$ 100 machines. This does not mean the techniques cannot scale, but this should be assessed. Likewise, the entire set of experiments in each work can be fully reproduced in the 43.3% of the cases. At the exception of few works, cross-comparison of the proposed extended techniques has not been carried out yet. This reveals the immature nature of the area and evidence many future research opportunities.

Most efforts target Grid Computing, while few target Cloud Computing. This latter essentially offers the means for building easy-to-use parallel computing infrastructures. Although the use of Clouds finds its roots in IT environments, the idea is entering scientific and academic ones. Consequently, in general, few researchers have deeply evaluated the benefits of Clouds for scheduling and processing resource intensive scientific applications. In fact, most Cloud schedulers come from mere adaptations of job schedulers for Grids. Some efforts however have employed Clouds to run simulations using *pure* scheduling policies for Clouds [41]. Then, the line of research underneath such studies could greatly benefit from SI-based scheduling approaches. All in all, since there are less efforts devoted to job scheduling in Clouds, we aim at designing a

new SI-based scheduler to efficiently running scientific simulations in Clouds while addressing aspects not fully considered by existing efforts such as energy consumption, and job priorities [41]. Eventually, we will implement the scheduler on top of a real Cloud platform.

Acknowledgements

We thank the reviewers for their helpful suggestions. We also thank Pablo Godoy for helping us proof-reading the article. The first author acknowledges her Ph.D. scholarship granted by the CONICET and formerly by UNCuyo and ANPCYT through PRH project. This research is supported by the ANPCyT (PAE-PICT 2007 program, projects 2311 and 2312), and UNCuyo (Grant No. 06/B253).

Appendix A. Swarm Intelligence techniques

SI [1] is based on studying collective behaviors that emerge from interactions between individuals and the environment which they live to solve optimization problems. Examples of systems in which SI is inspired are ants colonies, fish schools, birds flocks, and herds of land animals, where the whole group of individuals perform a desired task (i.e., feeding), which may not be made individually. For example, an ant is relatively unintelligent, but when it is part of a colony, some behaviors emerge from the interactions between ants, such as searching for food.

According to Dorigo and Birattari [1], an SI system (a) it is composed of many individuals, (b) the individuals are relatively homogeneous, i.e., they are either all identical or they belong to a few typologies, (c) the interactions among the individuals are based on simple behavioral rules that exploit only local information that the individuals exchange directly or via the environment, and (d) the overall behavior of the system results from the interactions of individuals with each other and with their environment, i.e., the group behavior is self-organizes.

The following subsections briefly describe three SI techniques that are currently widely used in job scheduling problems described in this survey. The techniques are ACO, PSO, ABC and AFSA.

A.1. Ant colony optimization

The ACO algorithm [1] arises from the way real ants behave in nature. An interesting aspect of this behavior is how ants manage to locate short paths to reach a food source from their nest. The ACO algorithm can solve computational problems since the algorithm has the ability to reduce paths and precisely to find the shortest paths. In nature, ants move randomly from one place to another to search for food. On the return to its nest each ant leaves an hormone that lures other working ants to the same course. When more and more ants choose the same path, the pheromone trail is intensified and even more ants will further choose it. Over time, the shortest paths will be intensified by the pheromone faster. That is because the ants will both reach the food source and travel back to their nest at a faster rate. Furthermore, if over time ants do not follow a certain path, its pheromone trail evaporates. From an algorithmic point of view, the pheromone evaporation process is useful for preventing the convergence to a local optimum solution.

Fig. 1 shows two possible nest-food source paths. Fig. 1(a) shows that ants will move randomly at the beginning and choose one of the two paths. The ants that follow the faster path will naturally get to the objective before other ants, and in doing so the former group of ants will leave a pheromone trail. Moreover, the ants that perform the round-trip faster, strengthen more quickly the quantity of pheromone in the shorter path (see Fig. 1(b)). The ants that reach the food source through the slower path will find attractive to return to the nest using the faster path. Eventually, most ants will choose the left path as shown in Fig. 1(c).

ACO employs artificial pheromone trails that play the role of information that is dynamically updated by ants to reflect their accumulated experience in contributing to solve an entire problem. In practice, to optimize job scheduling problems, the ACO algorithm is mapped to graphical representations, usually graphs. A graph for example may include jobs and executing physical machines (nodes) and scheduling decisions (arcs). Each job can be carried out by an ant to search for machines with available computing resources.

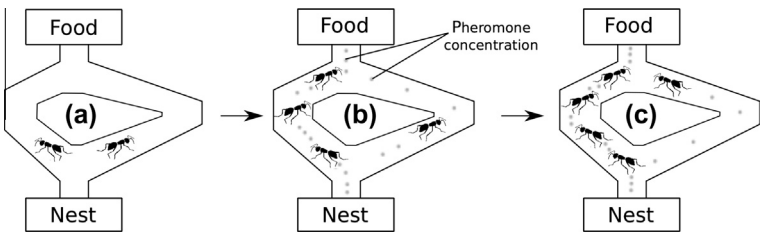

Fig. 1. Adaptive behavior of ants.

A.2. Particle swarm optimization

PSO [1] mimics the behavior of animals such as birds and insects such as bees. The general term "particle" is used to represent birds, bees or any other individuals who exhibit social and group behavior. Suppose a group of bees flies over the countryside looking for flowers. Their goal is to find as many flowers as possible. At the beginning, bees do not have knowledge of the field and fly to random locations with random velocities looking for flowers. Each bee has the capability of remember the places where it saw more flowers, and moreover, somehow knows the places where other bees have found a high density of flowers. These two pieces of information – *nostalgia* and *social knowledge* – are used by the bees to continually modify their trajectory, i.e., each bee alters its path between the two directions to fly somewhere between the two points and find a greater density of flowers. Occasionally, a bee may fly over a place with more flowers than any other place found. If this happens then the whole swarm is attracted towards this new direction. The process leads to having all bees in one single place of the field where the highest density of flowers is found.

To model job scheduling, PSO is instantiated with particles, each maintaining one potential solution to the entire scheduling problem. The design of the representation of a particle is different for each type of problem. An example of how to represent particles when modeling the job scheduling problem is placing the position of a particle in an n-dimensional search space. Each dimension i is the job to schedule, and the corresponding value represents the machine in which a job can be allocated. Furthermore, the global best position will indicate the best possible schedule.

A.3. Artificial bee colony

ABC is inspired by the intelligent foraging behavior of honey bees when searching for food (nectar), which operate by sharing the food sources (FSs) information between the bees in the nest. Each FS position is a solution. Bees are classified according to how they select the FS to exploit.

There are three types of bees in the foraging process: employed bees, onlookers bees and scouts bees. Employed bees are associated with a particular FS which they are currently exploiting and share this information with a certain probability by a waggle dance. Scout bees search the environment for new FS without any guidance. Onlookers bees observe the waggle dance and so are placed on the FS by using a probability based selection process. As the nectar amount of a FS increases, the probability value with which the FS is preferred by onlookers increases too.

In the initial population, a number of employed bees is generated. At the initialization stage, a set of FS positions are randomly selected by the employed bees and their nectar amounts are determined. The FS represents a candidate solution to the optimization problem, and the nectar amount – fitness value – is used to assign a quality to the FS regarding the associated solution in ABC algorithm. After initialization of random solutions, employed bees start their searching initially. Employed bees search new FS near to the current FS. If the generated new solution is better than the current solution then the new solution replaces the old one. The comparison of FS is done on the basis of fitness value or nectar amount in the FS.

After all employed bees complete the search process, they share the nectar information of FS (solutions) and their position information with onlooker bees waiting in the hive. The most profitable the FS, the longer the duration of the waggle dance. Onlookers watch the dances and choose FS depending on dances. After selecting its FS each onlooker bee seeks out one new FS within its neighborhood and moves to this FS if it has a higher nectar value. If a solution cannot be improved during a number of cycles, that source is abandoned, and its employed bee becomes a scout bee. A scout bee finds a new random solution to be replaced with the abandoned source. When a termination criteria is satisfied, the algorithm terminates.

A.4. Artificial fish swarm algorithm

AFSA essentially imitates the behavior of fishes in nature and allows to find a global optimum. In the water fishes can find the most nutritional area in two ways, i.e., by themselves or by following other fishes. Thus, the area in the water with the highest number of fishes is in general the one with the highest amount of food. According to this behavior, the AFSA algorithm relies on the concept of Artificial Fish (AF), and uses several AFs imitate the behavior of fish schools and perform the search of an optimal solution.

Each AF has its own behaviors and data. All AFs perceive information from the environment via their sense organs. In the algorithm the environment of an AF is the solution space. The basic behaviors that an AF can perform are *Prey, Swarm* and *Follow*. Through the *Prey* behavior an AF seeks water areas with high concentration of food and decides to move in that direction. The *Swarm* behavior allows fishes to join in groups to avoid dangers and assure the existence of the school. The *Follow* behavior is used to follow one or more fishes when they have found food. Moreover, there are two states for each AF – current and environmental – that define the next behavior of an AF. A state includes the solution quality of a fish obtained so far and the states of other fishes, i.e., the environmental state. Therefore, a behavior is influenced by the environment both by the own fish activities towards building a solution and the other fishes activities.

An AF (center of Fig. 2) can perceive the environment through its vision. Its visual reach is represented by X_v and its current state by X. If X_v is better than X, then the AF advances a *Step* towards that direction, and enters another state, i.e., X_{next}. Otherwise, the AF continues exploring within its vision area. The greater the exploring area, the more the knowledge the AF has about all possible next states to obtain a better location.

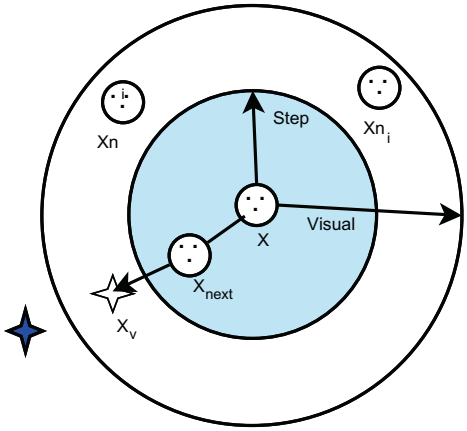

Fig. 2. Vision concept of an Artificial Fish.

In practice, to address job scheduling with AFSA, the structure of an AF and the food are the key issues to model. An important consideration is how to encode the scheduling solutions into AFs. While the way to encode the solutions depends on the specific design decisions of the algorithm, one possibility is to encode feasible solutions in a *fish vector*. The fish vector has a size of *N* fishes (or jobs) and each fish in the vector represents the machine to which a job is assigned by scheduler. Finally, the food concentration is a guide to search for the global optimal solution.

References

[1] Dorigo M, Birattari M. Swarm intelligence. Scholarpedia 2007;2(9):1462.
[2] Chandra B, Baskaran R. A survey: ant colony optimization based recent research and implementation on several engineering domain. Expert Syst Appl 2012;39(4):4618–27.
[3] Watcharasitthiwat K, Wardkein P. Reliability optimization of topology communication network design using an improved ant colony optimization. Comput Electr Eng 2009;35(5):730–47.
[4] Tavares Neto RF, Godinho Filho M. Literature review regarding ant colony optimization applied to scheduling problems: guidelines for implementation and directions for future research. Eng Appl Artif Int 2013;26(1):150–61.
[5] Xhafa F, Abraham A. Computational models and heuristic methods for Grid scheduling problems. Future Gener Comput Syst 2010;26(4):608–21.
[6] Tinghuai M, Qiaoqiao Y, Wenjie L, Donghai G, Sungyoung L. Grid task scheduling: algorithm review. IETE Techn Rev 2011;28(2):158–67.
[7] Pedemonte M, Nesmachnow S, Cancela H. A survey on parallel ant colony optimization. Appl Soft Comput 2011;11(8):5181–97.
[8] Foster I, Kesselman C. The history of the Grid in high performance computing: from Grids and Clouds to exascale. Advances in parallel computing, vol. 20. Amsterdam, Washington, DC: IOS Press; 2011.
[9] Lorpunmanee S, Sap M, Abdullah A, Chompooinwai C. An ant colony optimization for dynamic job scheduling in Grid environment. Int J Comput Inform Sci Eng 2007;1(4):207–14.
[10] Mathiyalagan P, Suriya S, Sivan S. Modified ant colony algorithm for Grid scheduling. Int J Comput Sci Eng 2010;2:132–9.
[11] Banerjee S, Mukherjee I, Mahanti P. Cloud computing initiative using modified ant colony framework. In: World academy of science, engineering and technology, WASET; 2009. p. 221–4.
[12] Ritchie G, Levine J. A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In: 23rd Workshop of the UK planning and scheduling special interest group; 2004.
[13] Stützle T, Hoos H. Max–min ant system. Future Gener Comput Syst 2000;16(8):889–914.
[14] Hui Y, Xue-Qin S, Xing L, Ming-Hui W. An improved ant algorithm for job scheduling in Grid computing. International conference on machine learning and cybernetics, vol. 5. IEEE Computer Society; 2005. p. 2957–61.
[15] Fidanova S, Durchova M. Ant algorithm for Grid scheduling problem. In: 5th International conference on large-scale scientific computing. Lecture notes in computer science, vol. 3743. Springer; 2006. p. 405–12.
[16] Zehua Z, Xuejie Z. A load balancing mechanism based on ant colony and complex network theory in open Cloud Computing federation. In: 2nd International conference on industrial mechatronics and automation. IEEE Computer Society; 2010. p. 240–3.
[17] Kousalya K, Balasubramanie P. To improve ant algorithm's Grid scheduling using local search. Int J Intell Inform Technol Appl 2009;2(2):71–9.
[18] Ruay-Shiung C, Jih-Sheng C, Po-Sheng L. An ant algorithm for balanced job scheduling in Grids. Future Gener Comput Syst 2009;25:20–7.
[19] Xu Z, Hou X, Jizhou S. Ant algorithm-based task scheduling in Grid computing. Canadian conference on electrical and computer engineering (CCECE 2003), vol. 2. Montreal: IEEE Computer Society; 2003. p. 1107–10.
[20] Ludwig S, Moallem A. Swarm intelligence approaches for Grid load balancing. J Grid Comput 2011;9(3):279–301.
[21] Sathish K, Reddy ARM. Enhanced ant algorithm based load balanced task scheduling in Grid computing. IJCSNS Int J Comput Sci Network Secur 2008;8(10):219–23.
[22] Palmieri F, Castagna D. Swarm-based distributed job scheduling in next-generation Grids. In: Advances and innovations in systems, computing sciences & software engineering. Netherlands: Springer; 2007. p. 137–43.
[23] Mathiyalagan P, Dhepthie U, Sivanandam S. Grid scheduling using enhanced PSO algorithm. Int J Comput Sci Eng 2010;02(02):140–5.
[24] Lei Z, Yuehui C, Runyuan S, Shan J, Bo Y. A task scheduling algorithm based on PSO for Grid computing. Int J Comput Intell Res 2008;4(1):37–43.
[25] Liu H, Abraham A, Hassanien A. Scheduling jobs on computational Grids using a fuzzy particle swarm optimization algorithm. Future Gener Comput Syst 2010;26(8):1336–43.
[26] Kang Q, He H. A novel discrete particle swarm optimization algorithm for meta-task assignment in heterogeneous computing systems. Microprocess Microsyst 2011;35(1):10–7.
[27] Kashan A, Karimi B. A discrete particle swarm optimization algorithm for scheduling parallel machines. Comput Indust Eng 2009;56(1):216–23.
[28] Pandey S, Wu L, Guru S, Buyya R. A particle swarm optimization-based heuristic for scheduling workflow applications in Cloud Computing environments. In: International conference on advanced information networking and applications. IEEE Computer Society; 2010. p. 400–7.

[29] Bu Y, Zhou W, Yu J. An improved PSO algorithm and its application to Grid scheduling problem. International symposium on computer science and computational technology, vol. 1. IEEE Computer Society; 2008. p. 352–5.
[30] Liu H, Sun S, Abraham A. Particle swarm approach to scheduling work-flow applications in distributed data-intensive computing environments. In: International conference on intelligent systems design and applications. ISDA '06, vol. 2. IEEE Computer Society; 2006. p. 661–6.
[31] Izakian H, Ladani B, Abraham A, Snasel V. A discrete particle swarm optimization approach for Grid job scheduling. Int J Innovat Comput Inform Contr 2010;6(9):4219–33.
[32] Kim S, Byeon J, Liu H, Abraham A, McLoone S. Optimal job scheduling in Grid computing using efficient binary artificial bee colony optimization. Soft Comput 2013;17(5):867–82.
[33] Pampara G, Engelbrecht A. Binary artificial bee colony optimization. Proceedings of 2011 IEEE symposium on swarm intelligence (SIS), vols. 1–8. Paris, France: IEEE Computer Society; 2011.
[34] Dhinesh Babu L, Venkata Krishna P. Honey bee behavior inspired load balancing of tasks in cloud computing environments. Appl Soft Comput 2013;13(5):2292–303.
[35] Gupta M, Sharma G. An efficient modified artificial bee colony algorithm for job scheduling problem. Int J Soft Comput Eng (IJSCE) 2012;1(6):291–6.
[36] Arsuaga Ríos M, Vega Rodríguez M, Prieto Castrillo F. Multi-objective artificial bee colony for scheduling in Grid environments. In: Symposium on swarm intelligence (SIS). Paris, France: IEEE Computer Society; 2011. p. 206–12.
[37] Vivekanandan K, Ramyachitra D, Anbu B. Artificial bee colony algorithm for Grid scheduling. J Converg Inform Technol 2011;6(7):328–39.
[38] Ruan D, Zhang X, Li H, Liu Y. The research of optimal algorithm for task scheduling underground wireless network based on distributed computing. In: 2010 International conference on manufacturing automation, ICMA 2010. Washington, DC, USA: IEEE Computer Society; 2010. p. 151–5.
[39] Farzi S. Efficient job scheduling in Grid computing with modified artificial fish swarm algorithm. Int J Comput Theory Eng 2009;1(1):13–8.
[40] Buyya R, Murshed M. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. Concurr Comput: Pract Exp (CCPE) 2002;14(13):1175–220.
[41] Mateos C, Pacini E, García Garino C. An ACO-inspired algorithm for minimizing weighted flowtime in Cloud-based parameter sweep experiments. Adv Eng Softw 2013;56:38–50.

Elina Pacini received a BSc. in Information Systems Engineering from the UTN-FRM, Argentina, in 2005. She is presently pursuing his Ph.D. thesis since 2010 under the supervision of Carlos García Garino and Cristian Mateos. Her thesis topic is Cloud scheduling for parameter sweep experiments based on Swarm Intelligence techniques.

Cristian Mateos received a Ph.D. degree in Computer Science from the UNICEN, in 2008, and his M.Sc. in Systems Engineering in 2005. He is a full time Teacher Assistant at the UNICEN and member of the ISISTAN and the CONICET. He is interested in parallel/distributed programming, Grid middlewares and Service-oriented Computing.

Carlos García Garino received his Ph.D. degree from Universidad Politécnica de Cataluña, Spain, in 1993, and the Civil Engineering degree from UBA, Argentina, in 1978. He is a full Professor at the UNCuyo, and director of the ITIC–UNCuyo. He is interested in Computer Networks, Distributed Computing and Computational Mechanics.