Applied Soft Computing 11 (2011) 5181–5197

Contents lists available at ScienceDirect

Applied Soft Computing

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

A survey on parallel ant colony optimization

Martín Pedemonte, Sergio Nesmachnow*, Héctor Cancela

Facultad de Ingeniería, Universidad de la República, Uruguay

ARTICLE INFO

Article history:
Received 8 November 2010
Received in revised form 26 April 2011
Accepted 23 May 2011
Available online 31 May 2011

Keywords:
Ant colony optimization
Parallel implementations
Taxonomy

ABSTRACT

Ant colony optimization (ACO) is a well-known swarm intelligence method, inspired in the social behavior of ant colonies for solving optimization problems. When facing large and complex problem instances, parallel computing techniques are usually applied to improve the efficiency, allowing ACO algorithms to achieve high quality results in reasonable execution times, even when tackling hard-to-solve optimization problems. This work introduces a new taxonomy for classifying software-based parallel ACO algorithms and also presents a systematic and comprehensive survey of the current state-of-the-art on parallel ACO implementations. Each parallel model reviewed is categorized in the new taxonomy proposed, and an insight on trends and perspectives in the field of parallel ACO implementations is provided.

1. Introduction

In the last twenty years, the research community has been searching for new optimization techniques that are able to improve over the traditional exact ones, whose large computational requirements often make them useless for solving complex real-life optimization problems in acceptable times. In this context, nature-inspired metaheuristic methods have emerged as flexible and robust tools for solving NP-hard optimization problems, exploiting their ability to compute accurate solutions in moderate execution times [13,49]. Ant colony optimization (ACO) is a swarm intelligence population-based metaheuristic inspired in the social behavior of ant colonies, which applies the key concepts of distributed collaboration, self-organization, adaptation, and distribution found in ant communities, in order to efficiently solve real-life optimization problems [41].

Parallel implementations became popular in the last decade in order to improve the efficiency of population-based metaheuristics. By splitting the population into several processing elements, parallel implementations of metaheuristics allow reaching high quality results in a reasonable execution time, even when facing hard-to-solve optimization problems [2]. Parallel algorithms not only take benefit of using several computing elements to speed up the search, they also introduce a new exploration pattern that is often useful to improve over the result quality of the sequential implementations.

Many papers can be found in the related literature stating that parallel implementations are useful to improve the ACO exploration pattern; Fig. 1 shows the number of publications per year in this area. However, researchers often lack a generalized point of view, since they usually tackle a unique implementation to solve a specific problem.

Dorigo [39,40] first suggested the application of parallel computing techniques to enhance both the ACO search and its computational efficiency, while Randall and Lewis [84] proposed the first classification of ACO parallelization strategies. The book chapter by Janson et al. [57] and the article by Ellabib et al. [46] are the only previous works that have collected bibliography of published papers proposing parallel ACO implementations. Janson et al. reviewed parallel ACO proposals published up to 2002, focusing on comparing "parallelized" standard ACO algorithms, specific parallel ACO methods, and hardware parallelization; although they did not include an explicit algorithmic taxonomy. Ellabib et al. briefly commented parallel ACO implementations up to 2004, focusing in describing the applications, and they only distinguished between coarse-grain and fine-grain models for parallel ACO.

The classic proposals of parallel ACOs focused on traditional supercomputers and clusters of workstations. Nowadays, the novel emergent parallel computing architectures such as multicore processors, graphics processing units (GPUs), and grid environments provide new opportunities to apply parallel computing techniques to improve the ACO search results and to lower the required computation times.

In this line of work, the main contributions of this article are: (i) to introduce a new taxonomy to classify software-based parallel ACO algorithms, (ii) to present a systematic and comprehensive survey of the current state-of-the-art on parallel ACO implementations, and (iii) to provide an insight of the current trends and perspectives in the field. The survey focuses mainly on the parallel models, addressing the principal characteristics of each proposal, the experimental analysis – including the optimization problems

* Corresponding author.
E-mail address: sergion@fing.edu.uy (S. Nesmachnow).

1568-4946/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2011.05.042

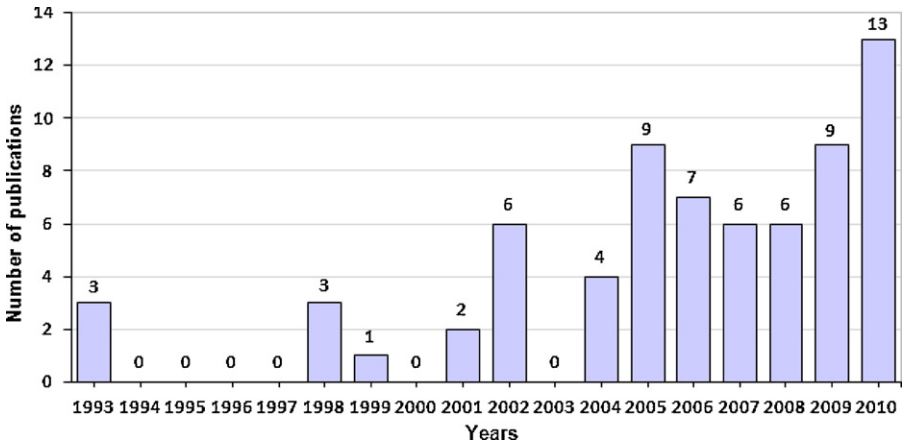

Fig. 1. Number of reviewed publications by year.

faced, the test cases and the parallel platform used in the experiments, and the reported results –, and the main contributions of the reviewed works. Each parallel ACO proposal is categorized in the new taxonomy proposed.

The manuscript is structured as follows. Next section describes the research methodology used in the review. Section 3 describes the main features of the ACO technique and briefly introduces the most popular ACO variants. Section 4 presents the generic concepts of the strategies for ACO parallelization and comments previous classification criteria. It also describes the new taxonomy proposed in this work to categorize parallel ACOs. Section 5 reviews the previous work on parallel ACO implementations and categorizes each proposal using the new taxonomy. A comparative analysis regarding the computational efficiency and quality of results is offered in Section 6. Section 7 presents the trends and perspectives in the field of parallel ACO implementations, before stating the conclusions of the survey in Section 8.

2. Methodology

The research methodology used in the review involved searching and reviewing papers from soft computing/computational intelligence conferences, journals, and books, where the application of parallel processing techniques to ACO have been proposed.

2.1. Sources and search methods

A comprehensive search was performed in conferences, journals and books about metaheuristics and parallelism. The databases searched in this study include ScienceDirect, Scopus, Thomson Reuters (formerly ISI) Web of Knowledge, ACM Digital Library, IEEE Explore, Elsevier, SpringerLink, Citeseer, as well as many others Open Access Publishing databases. The reviewed papers come out from leading conferences and journals about soft computing, such as International Conference on Parallel Problem Solving from Nature, Conference on Genetic and Evolutionary Computation, Conference on Evolutionary Computation, Journal of Heuristics, Information Sciences, Lecture Notes in Computing Science, IEEE Transactions on Evolutionary Computation, Applied Soft Computing, Future Generation Computer Systems, and Journal of Artificial Intelligence Research, among others.

The search of related papers in specific databases was done using a group of keywords that include ant colony optimization, parallel, distributed, parallelism, and soft computing. Additionally, the reference section of each paper found was reviewed to locate additional studies of interest. As a result, the final references consist of **69** papers: 19 published in journals, 44 in referred conferences, 3 in books, and 3 M.Sc./Ph.D thesis. The analysis of related works was mainly focused on the features of the parallel models, describing the distinctive characteristics of each parallel ACO proposal, the optimization problem tackled, the test cases and the parallel platform used in the experiments, and the efficiency and quality results reported. Each parallel ACO proposal is categorized in the new taxonomy proposed in this work. In order to study the recent contributions about parallel ant colony optimization, those papers published in the last five years (2005–2010) were further studied to analyze the main trends and perspectives about parallel ACO implementations.

2.2. Scope

The review focuses in papers that have proposed explicitly parallel implementations of ACO, disregarding those proposals using implicit parallelism or a distributed-agent-based search. A complete description of the implicit-parallel and other ACO categories that have been left apart from the review are summarized in Section 4.3.

The reviewed works face optimization problems from a large spectrum of application domains. Only single-objective, static optimization problems are covered in this survey, since they are the large class of problems frequently solved using parallel ACO. The algorithmic structure of ACO to solve multi-objective and dynamic optimization problems is different to the traditional ACO algorithm, so they have not been included in the scope of this review

3. Ant colony optimization

Ant Colony Optimization [45] is a population-based metaheuristic for solving optimization problems, originally proposed by Dorigo and Di Caro [41]. ACO uses artificial ants to construct solutions by incrementally adding components that are chosen considering heuristic information of the problem and pheromone trails that reflect the acquired search experience.

Algorithm 1 presents the skeleton of an ACO algorithm applied to a combinatorial optimization problem for minimizing one objective function. At first, the ACO sets the initial pheromone trails values (T) and the heuristic value of the solution components (H, known as *visibility*). After that, the algorithm iterates until a given stop condition is reached. Every iteration step is divided in four stages. First, each ant of the colony concurrently, independently, and asynchronously constructs a solution by selecting components using a probabilistic rule that considers both the experience acquired during the search (through the trace of pheromone deposited) and heuristic information of the considered components

(through the visibility). The next stage optionally applies a local search method to improve the solutions. In the third stage, the pheromone trails are updated: the trace values are decreased by evaporation and increased by depositing pheromone in the components used to construct solutions; the net change in the pheromone value depends on the contributions of these two update processes. Finally, in the last stage, the best solution found since the start of the algorithm (the *best-so-far* solution) is updated if a better solution has been found. ACO returns the best-so-far solution, when the stop criteria is accomplished.

Algorithm 1.

ACO applied to a static combinatorial optimization problem

```
T = initializePheromoneTrails()
H = initilizeVisibilities()
s_best = s | f(s) = +∞
  While not stopCriteria () do
  pop = constructAntsSolutions(T,H)
  pop' = applyLocalSearch(pop)    %   optional
  T = updatePheromones(T,pop')
  s = selectBestOfPopulation(pop')
  if f(s) < f(s_best) then % update best-so-far solution
    s_best = s
  end if
end while
return s_best
```

The scientific community has proposed multiple variants that instantiate the general scheme shown in Algorithm 1. Some of the most popular variants include:

- *Ant System* (AS) [44], the classic method that uses a random proportional state transition rule, while the pheromone is deposited by all ants proportionally to their solution quality and is evaporated in all the components.
- *Ant Colony System* (ACS) [43], which employs a pseudo-random state transition rule, and the pheromone is only deposited and evaporated on the components of the best solution. ACS incorporates a local pheromone update during the solution construction, allowing the exploration of unused components.
- $\mathcal{MAX}-\mathcal{MIN}$ *Ant System* ($\mathcal{MM}$AS) [91] that includes explicit lower and upper limits on the pheromone, which is only deposited on the components of the best solution.

ACO methods have also been proposed for solving multi-objective and dynamic optimization problems. However, these methods have particular features (such as Pareto-based evaluation of solutions, elite populations or colonies, alternative pheromone updating and evaporation rules, multiple pheromones, etc.) that make them different from the traditional ACO schema presented in Algorithm 1, so they have not been included in the scope of this review.

4. ACO parallelization strategies

The systematic study of the application of parallel computing techniques to ACO algorithms is recent. Several authors agree that exhaustive work still needs to be done in this subject [45,57]. There are few articles that specifically discuss possible strategies to implement parallel ACO; and no recent work features a complete state-of-the-art review on this subject. This section briefly introduces the metrics to evaluate the performance of parallel algorithms, which will be used in the literature review in Section 5. Later, it presents different strategies to implement parallel ACO algorithms. The currently proposed classifications for software-based parallel ACO are discussed before introducing a new taxonomy, which aims to extend the previous ones and to overcome some of their shortcomings and omissions.

4.1. Parallel performance metrics

Several metrics have been proposed to evaluate the performance of parallel algorithms. The most common metrics used by the research community are the *speedup* and the *efficiency*. The *speedup* evaluates how much faster a parallel algorithm is than a corresponding sequential algorithm, and it is computed as the ratio between the execution time of the sequential algorithm (T_1) and the execution time of the parallel version using m processors (T_m) (Eq. (1)). When evaluating the performance of non-deterministic algorithms, the speedup should compare the *mean* values of the sequential and parallel execution times (Eq. (2)) [1]. The previous definition allows distinguishing among *sublinear* speedup ($S_m < m$), *linear* speedup ($S_m = m$), and *superlinear* speedup ($S_m > m$). The ideal case for a parallel algorithm is to achieve linear speedup, although the most common situation is to achieve sublinear speedup values due to the times required to communicate and synchronize the parallel processes. When linear or almost-linear speedup is achieved, the parallel algorithm is said to have a good *scalability* behavior (i.e. the time required to perform it diminishes proportionally with the number of processing elements used). The *computational efficiency* (simply named *efficiency*) is the normalized value of the speedup, regarding the number of processors used to execute a parallel algorithm (Eq. (3)). The efficiency metric allows to compare different algorithms, eventually executed in non-identical computing platforms. The linear speedup corresponds to $e_m = 1$, and in the most common situations $e_m < 1$.

$$S_m = \frac{T_1}{T_m} \tag{1}$$

$$S_m = \frac{E[T_1]}{E[T_m]} \tag{2}$$

$$e_m = \frac{S_m}{m} \tag{3}$$

According to the Amdahl's law [10], the performance of any parallel application is limited by the sequential part of the code, that depends on the choice of the parallelization strategy. Amdahl's law can be used in parallel computing to predict the theoretical maximum speedup when using multiple computing resources. Far away from a pessimistic view of parallel computing, the Gustafson's Law [52] indicates that parallel implementation are useful for solving larger problem instances in a reasonable amount of time, while achieving almost linear speedup values. In fact, in the optimization field, parallel implementations of metaheuristics are known to been able to efficiently solve hard problems, even achieving superlinear ($S_m > m$) in some specific situations, by taking advantage of particular hardware or algorithmic design issues [7].

4.2. Previous parallel ACO classifications

Although the research community has proposed many parallel ACO implementations, there do not exist standardized taxonomies for classifying the parallelization strategies applied to ACO. Researchers usually merely present one or a few parallel ACO implementations, but they often do not put much effort in following a methodology to systematically categorize the parallel approaches. The most usual criterion when classifying parallel ACO simply distinguishes two wide categories: fine-grained and coarse-grained models [46], also called ant-based and colony-based models [76].

The proposal by Randall and Lewis in 2002 [84] has been the unique attempt to provide a classification of parallel ACO approaches. This categorization distinguished five parallel ACO models. Three of them used a hierarchical master-slave paradigm and the other two categories follow an indepen-

dent executions model and a synchronous cooperative model, respectively.

Several aspects to consider when implementing parallel ACO algorithms were discussed by Janson et al. [57]. The work did not provide a comprehensive taxonomy, but used two criteria to classify parallel ACOs. The first criterion differentiates "*parallelized*" standard ACO, designed to decrease the execution time without changing the sequential algorithmic model, and *specifically designed* parallel ACO, aimed at improving the results quality and the computational efficiency, following a different algorithmic behavior. The second criterion distinguishes between centralized and decentralized parallel ACO models, regarding whether a central process that collects both the solutions and the pheromone information exists or not.

Other high-level taxonomies for parallel metaheuristics have been sporadically used to classify parallel ACO approaches. The parallel metaheuristics classification by Crainic and Nourredine [28] distinguishes three categorization levels, regarding the search control cardinality, control and communications, and differentiation; but it is a generic classification that has not gained popularity. In the taxonomy for parallel metaheuristics by Talbi [92], three hierarchical levels (*algorithmic*, *iteration*, and *solution*) are used to classify parallel implementations of metaheuristics, regarding the granularity of the parallel approach. The parallel ACO model that uses several colonies is introduced as an example of algorithmic level parallelization, but other models for parallel ACO do not get a mention. Another approach by Cung et al. [31] identified the *single walk* and the *multiple walk* – with independent and cooperative search threads – parallel strategies of metaheuristics. Up to now, this classification has been only used in the survey of pioneering parallel ACO proposals presented in that same paper.

The research on parallel metaheuristics has significantly advanced in the last decade, thus the taxonomy presented by Randall and Lewis in 2002 is no longer accurate to describe and classify the parallel ACO models proposed by the research community. Only ten proposals of parallel ACO implementations were proposed up to 2002, and the bulk of works about parallel ACO has been done in the period from 2005 to 2010, so a new taxonomy is required to capture the main features of nowadays parallel ACO proposals. On the other hand, the generic classifications for parallel metaheuristics have not been proposed to consider the ACO features, and they often provide an abstract view that do not help to understand the specific details of parallel ACO. To overcome this lack of a standardized taxonomy for the classification of parallel ACO algorithms, this work introduces a new taxonomy for software-based parallel ACO, conceived to take into account the particular features of all the proposed strategies for ACO parallelization.

Next subsection presents the proposal of a comprehensive and specific new taxonomy for categorizing parallel ACO implementations.

4.3. A new taxonomy for parallel ACO

This subsection presents a new taxonomic proposal for parallel ACO algorithms. The categorization takes some basic concepts identified by Randall and Lewis [84], but it expands the classification in order to introduce some missing categories, to extend other ones, and also to include general ideas from the work by Janson et al. [57] and from the evolutionary algorithms literature. Two main criteria related to the population organization are used to discriminate the categories in the taxonomy: the number of colonies and the cooperation. The amount of work that is performed in parallel is used to refine the classification within the master-slave category.

The main contributions of the new taxonomy, which have not been proposed in previous attempts to classify parallel ACO implementations, are:

- Three subcategories were included in the master-slave model, regarding the amount of work that is performed in parallel: coarse-grain, medium-grain, and fine-grain. The medium-grain master-slave is an original problem decomposition-based new category that includes those works that apply a hierarchical master-slave model using a domain decomposition approach [36,38,76].
- A new *cellular model* category – where a single colony is structured in small neighborhoods with limited interactions– is included, based on the similar class found in the most widely accepted taxonomies of parallel evolutionary algorithms [7,17]. This model does not appear in previous parallel ACO classifications, and one implementation of this new model has been recently proposed [78].
- A wider category – far more comprehensive than those previously used in other taxonomies – was adopted for cooperative parallel ACO methods that use more than one colony (the *multicolony* model). This category allows grouping a larger number of parallel ACO proposals than other previously defined multicolony classes.
- The taxonomy also incorporates a category including *hybrid* models, which comprehends those proposals that feature characteristics of more than one parallel model.

The full proposal of a new taxonomy of strategies for parallel implementations of ACO includes the following categories:

- **Master-slave model**. This category applies a hierarchical parallel model, where a master process manages the global information (i.e. pheromone matrix, best-so-far solution, etc.) and it also controls a group of slave processes that perform subordinated tasks, related to the ACO search space exploration. The model includes three distinguished subcategories regarding the *granularity* (i.e., the amount of work performed by each slave process):

Coarse-grain master-slave model. The master manages the pheromone matrix and the interaction with the slaves is based on complete solutions. The tasks delegated to the slaves may correspond to one or more ants, and they comprise building, improving and/or evaluating one or more full solutions, and communicating back the result to the master. This subcategory is more comprehensive than the *parallel ants* model by Randall and Lewis [84], since it allows grouping several ants in the same slave process.
Medium-grain master-slave model. A domain decomposition of the problem is applied. The slave processes solve each subproblem independently, whereas the master process manages the overall problem information and constructs a complete solution from the partial solutions reported by the slaves.
Fine-grain master-slave model. The slaves perform minimum granularity tasks, such as processing single *components* used to construct solutions, and frequent communications between the master and the slaves are usually required. The model includes the *parallel evaluation of solution elements* category originally proposed by Randall and Lewis [84], but it also incorporates other proposals that frequently communicate components or information about the components.

- **Cellular model**. A single colony is structured in small neighborhoods, each one with its own pheromone matrix. Each ant is placed in a *cell* in a toroidal grid, and the trail pheromone update in each matrix considers only the solutions constructed by the ants in its neighborhood. The model uses overlapping neighborhoods, so the effect of finding high-quality solutions

		#colonies	
		one	many
cooperation	yes	cellular	multicolony
	no	master-slave (coarse/medium/fine)	parallel independent runs

Fig. 2. Main categories in the new taxonomy for parallel ACO.

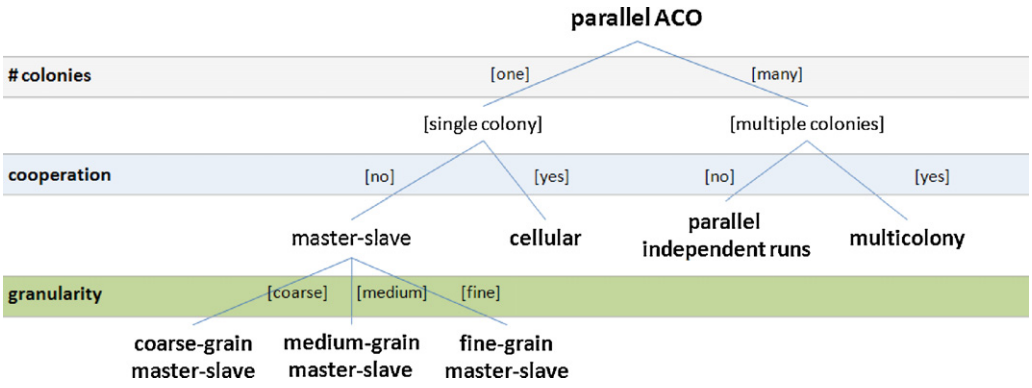

Fig. 3. A hierarchical view of the new taxonomy for parallel ACO.

gradually spreads to other neighborhoods using the *diffusion* model employed in cellular evolutionary algorithms [3,81].

- **Parallel independent runs model**. Several sequential ACO, using identical or different parameters, are concurrently executed on a set of processors. The executions are completely independent, without communication among the ACOs, therefore the model does not consider cooperation between colonies.
- **Multicolony model**. In this model, several colonies explore the search space using their own pheromone matrices. The cooperation is achieved by periodically exchanging information among the colonies. The *parallel interacting ant colonies* model previously defined by Randall and Lewis [84] is a particular case of multicolony that communicates the full pheromone matrix among colonies, thus it is comprised in this category.
- **Hybrid models**. This category includes those proposals that feature characteristics from more than one parallel model. The category *parallel combination of ants and evaluation of solution elements* by Randall and Lewis [84] is a special case of hybrid that combines two master-slave models. However, several other models are also included in this category.

Fig. 2 presents the main categories in the new taxonomy for parallel ACO, and Fig. 3 shows a hierarchical view of the categories regarding the criteria considered in the classification.

The proposed taxonomy focuses on explicitly parallel ACO algorithms, disregarding those approaches that use implicit parallelism or a distributed-agent-based search. The following methods have been left apart from the categorization:

1. *implicit-parallel ACOs*: the ACO construction process is inherently parallel, since ants build solutions in a concurrent and independent way. Some ACO variants use inherent parallel features without a parallel implementation, e.g.: several colonies, each one with its own pheromone matrix [9,32,53,60]; one colony with two types of ants and a single pheromone matrix [61]; and one colony with several pheromone matrices [12,77],
2. *distributed-agent-based ACOs*: this kind of (non-parallel) distributed ACO is usually employed to solve distributed problems such as dynamic network routing (e.g: AntNet [18]), but they are not specifically designed to take advantage of parallel computing architectures,
3. *Ant Based Optimization* (ABO): despite the name similarity, ABO has a different behavior than ACO (it uses ants to reduce the search space by identifying areas that potentially contain a good solution). Few parallel ABO implementations have been proposed, though a recent paper discussed strategies for parallel ABO in shared memory computers [15],
4. *hardware-parallel ACOs*: this kind of ACO implemented in hardware [57,72,88] are platform-dependent (e.g., they depend on the flow of information between the hardware processing units), and so their algorithmic behavior differs from the traditional ACO. Thus, hardware-parallel ACOs are not classifiable in a taxonomy for software-based ACO, such as the one proposed in this work, and they have been also left apart from the categorization.

Table 1 summarizes the main features of the parallel ACO models identified in the new taxonomy (for the hybrids category, D/P stands for "depends on the proposal"). The data in Table 1

Table 1
Characteristics of the models in the new taxonomy.

Model	Population organization	# Colonies	# Pheromone matrices	Communication frequency
Coarse-grain master-slave	Hierarchical, non-cooperative	One	One	Medium
Medium-grain master-slave	Hierarchical, non-cooperative	One	One	Medium-high
Fine-grain master-slave	Hierarchical, non-cooperative	One	One	High
Cellular	Structured, cooperative	One	Many	Medium
Parallel independent runs	Distributed, non-cooperative	Several	Several	Zero
Multicolony	Distributed, cooperative	Several	Several	Low
Hybrids	Hierarchical	D/P	D/P	D/P

correspond to *pure* implementations of the models, although the boundary between some models may be diffuse. For example, a coarse-grain master-slave with several ants per slave could use local pheromone matrices at each slave to reduce the communications, therefore improving the computational efficiency. Such a coarse-grain master-slave implementation is quite similar to a multicolony model in which pheromone matrices are periodically synchronized.

The previously presented taxonomy was conceived to comprehend all the parallel ACO proposals found in the related literature. The next section reviews the works that have presented parallel ACO implementations and it also classifies each proposal in the corresponding category of the new taxonomy.

5. Categorizing parallel ACO implementations

This section presents a comprehensive review of parallel ACO proposals in the related literature, describing the main features of the parallel implementation and the details of the experimental analysis. The first subsection introduces the pioneering works on parallel ACO and the next subsections categorize the works proposed since 1998, following the new taxonomy. When a specific work proposes two or more parallel ACO implementations, it is classified in the category that corresponds to the method which obtained the best results, regarding both the solution quality and the performance. Only a single review is included for cellular parallel ACO, since this model has been recently presented by two of the authors of the present work.

5.1. Pioneering works

The pioneering works on parallel ACO implementations date from the early 1990s. The primary proposals were mostly focused in investigating the benefits of the available parallel architectures for speeding up the ACO search. It is hard to classify them in the proposed taxonomy, since the basic concepts on parallel ACO models were not formulated at that time.

In his Ph.D. thesis, Dorigo [40] first suggested using a parallel version of ACO in order to improve the quality of the search and its computational efficiency. The first implementation of a parallel AS is attributed to Bolondi and Bondaza [14], who presented a very fine-grain implementation that placed one ant in each available processor of a Connection Machine (CM-2) supercomputer to solve the Traveling Salesman Problem (TSP). Although it was an innovative proposal, the fine grain parallel ACO did not scale due to the high communication overhead required in the synchronization and the pheromone updating phases [42,75].

Bolondi and Bondaza achieved better results when solving the TSP using a hierarchical panmictic parallel ACO implemented in a network of transputers. Several groups of ants executed a standard AS algorithm – each one in an available processor –, and a synchronous update of the pheromone trails was performed after hierarchically broadcasting all the information. The parallel ACO showed good scalability: almost linear speedup was achieved when increasing the number of processors, no matter the size of the problem instances faced [42].

Bullnheimer et al. [16] studied the communications in synchronous and partially asynchronous master-slave parallel ACOs. In the synchronous version, each slave locally updated its pheromone matrix independently and the pheromone trials were globally synchronized with a given frequency. No certain conclusions can be drawn from the experimental analysis, since it did not solve any concrete problem: it only simulated the communications on TSP instances with up to 500 cities. The asynchronous version had better speedup and efficiency values than the synchronous version, but the improvements diminished for the largest instances.

5.2. Master-slave model

Master-slave parallel ACO implementations have been quite popular in the research community, mainly due to the fact that this model is conceptually simple and easy to implement.

5.2.1. Coarse-grain master-slave

The standard implementation of coarse-grain master-slave ACO assigns one ant to a slave process that is executed on an available processor. The master process globally manages the global information (i.e. the pheromone matrix, the best-so-far solution, etc.), and each slave builds, optionally applies the local search, and evaluates a single solution. The communication between the master and the slaves usually follows a synchronous model.

The first proposal of this implementation was ANTabu, a method combining ACO and Tabu Search (TS), applied to solve the Quadratic Assignment Problem (QAP) by Talbi et al. [93]. The TS method was used as a local search to improve the solutions in each slave. The parallel version was compared against a sequential ACO, a parallel TS, genetic algorithms (GA) and variable neighborhood search. Regarding the solution quality, parallel ANTabu was one of the two best methods studied when solving QAPLIB instances with up to 256 locations in a cluster of 10 SGI Indy workstations. The computational efficiency of the parallel methods was not reported.

Synchronous and asynchronous versions of a coarse-grain master-slave AS-like method were studied by Catalano and Malucelli [20] to solve the Set Covering Problem (SCP). Both versions had similar speedup behavior and efficiency values when solving OR-Library instances with up to 400 elements and 650 subsets in a Cray T3D with 64 processors, although the quality of solutions was not studied in the scalability analysis.

Delisle et al. [35] solved an industrial scheduling problem in an aluminum casting center, using a multithread coarse-grain master-slave ACO implemented with OpenMP on a Silicon Graphics Origin 2000 computer. The master spawns several threads (one for each ant), which generate and evaluate the solutions. Specific considerations about load balancing and information update were presented. The experimental analysis solved problem instances with 50 and 80 jobs using up to 16 processors. Significant speedups were obtained (up to 5.45 when using 16 processors), but the computational efficiency values degraded as the number of processors grew.

Two applications following the standard implementation were presented by Peng et al. [79,80] to solve a packing problem and the image registration problem, respectively. The analysis were mainly focused on the solution quality: in both cases the parallel ACO quickly converged to better solutions than evolutionary algorithms, but the computational efficiency was not studied.

Li et al. [64] applied a standard coarse-grain master-slave ACS to the vector quantization codebook design. The method obtained high efficiency values (>0.8) when working on 2–16 processors of a DeepSuper-21C cluster (P4 Xeon), and it computed better solutions than both a sequential ACS and a parallel independent runs. Grouping four ants on each processor and using a modified pheromone updating rule improved the efficiency of the parallel model.

Ibri et al. [55] proposed an hybrid ACS/TS to solve a dispatching and covering problem for emergency vehicle fleets. The standard coarse-grain master-slave parallelization was used for the ACS, where the slaves (implemented by threads) constructs the solutions. A second parallel stage is applied to perform the TS operator and the neighborhood evaluation. Problem instances with up to 100 vehicles, 23 stations, 20 zones and 30 emergencies were solved on a Intel Core2 Duo, comparing synchronization strategies between the parallel processes and the impact of the information exchange on

the computational efficiency. The parallel implementations computed better solutions than the sequential one. Sublinear speedup values were found when using more than two threads, and the asynchronous method significantly reduced the execution time when compared with the synchronous implementation.

Some researchers have proposed assigning several ants per processor in coarse-grain master-slave ACO implementations.

Doerner et al. [37] used this idea in a synchronous master-slave implementation of AS_{rank} (a variant of AS where the pheromone deposit is weighted according to a rank of the best solutions) to solve the Vehicle Routing Problem (VRP). Several classic VRP instances with up to 199 clients were solved in a Beowulf cluster with up to 32 processors. A sub-linear speedup behavior was detected. The best values of computational efficiency (0.7) were obtained when using 8 processors, and then the efficiency decreased when the number of used processors increased.

Lv et al. [70] proposed a parallel ACO using P groups of ants distributed in P processors, which shared one pheromone matrix in a symmetric multiprocessing computer. This approach corresponds to the coarse-grain master-slave model, where the shared memory acts like an implicit master and each slave holds a group of ants. Parallel versions of $\mathcal{MM}$AS and ACS were studied to solve TSPLIB instances with up to 15,915 cities on an IBM p-server with two Power5 processors. The parallel algorithms achieved better solutions than the sequential versions, but their computational efficiency was not studied. The same idea was applied by Guo et al. [51] to the protein structure prediction problem, implementing each group of ants with a different thread and using an asynchronous access to the shared pheromone matrix. Similar results than a previous sequential ACO were obtained on a 8 CPU IBM pServer. Reductions between 10 and 50 times were reported in the execution time, but the comparison is unfair since the sequential ACO was executed on a slower computer and both methods used a different number of ants. The master-slave ACO was between 2 and 10 times faster when using more than one group of ants.

Chintalapati et al. [25] grouped several ants in a same processor in their coarse-grain master-slave ACO applied to the discovery of classification rules. Each group discovers rules and send them to the master, who manages the pheromone matrix. Standard cancer datasets with up to 168 features were solved in the experimental analysis performed in an heterogeneous cluster. The speedup values depended on the dataset features and the number of ants, and the best efficiency values (0.95) were obtained when using 128 ants executing on 8 CPUs to solve the largest problem instance studied.

Recently, multithreading programming have been applied to coarse-grain master-slave ACO, by assigning one thread to each ant, and executing several threads on the same processor. Tsutsui and Fujimoto [97] studied synchronous and asynchronous variants of parallel cunning AS (*c*AS) to solve the TSP. The experimental analysis performed on a i7 965 (4 cores, 3.2 GHz) showed that a *rough* asynchronous implementation obtained superlinear speedup by introducing a different algorithmic behavior than the synchronous *c*AS. In the parallel ACO by Gao et al. [48] to solve the Target Assignment Problem, the main task to execute in parallel is the construction of solutions. The experimental analysis compared multithreading implementations using OpenMP and Threading Building Block (TBB) in a Pentium Dual Core (2.4 GHz). Almost linear speedup values were obtained for problem instances with 100 targets, and the OpenMP variant was more efficient than the TBB implementation. The same idea was applied in the coarse-grain TBB implementation by Li et al. [63] to solve the TSP. Speedup values up to 1.72 were obtained when using 400 ants to solve a TSP instance with 500 cities on a Pentium Dual Core (3.0 GHz).

In other implementations the main tasks to perform in parallel are only related to the evaluation of solutions or the application of the local search, mainly due to the inherent complexity of the problem faced. These implementations are categorized within the coarse-grain master-slave model, considering the granularity of the work performed by each slave process.

Tsutsui [95] solved the QAP using a parallel *c*AS conceived to speed up a local search method performed by the slaves on solutions previously built by the master. A multithreading approach was adopted, and the communication overhead was reduced by using the shared memory paradigm. Several QAPLIB instances with up to 150 locations were solved using two quadcore PCs, and significant improvements in the execution time were obtained. Later [96], the coarse-grain master-slave outperformed both a synchronous multicolony and a parallel independent runs model in experiments performed in two dual-core Opteron machines. The optimal pump scheduling in water distribution networks was tackled with a coarse-grain master-slave ACO by López-Ibáñez et al. [68]. Multithreading programming techniques were used to implement the parallel evaluation of solutions, and a dynamic load balancing scheduling for assigning solutions to the threads was also included. The experimental analysis solved a well-known problem instance -already tackled with a sequential ACO- in a computer with 2 dual-core AMD64 Opteron processors. Accurate solutions were computed by the parallel ACO, which also obtained increasing speedup values when using a higher number of ants.

The parallel evaluation of solutions was also used in the ACS by Weis and Lewis [99], applied to the design of a radio frequency antenna structure. An ad-hoc grid computing approach was implemented in a cluster of 47 non-dedicated PCs, using an instant messaging protocol for the communications. The experimental results demonstrated that the parallel ACS obtained high speedup values with reduced overhead when increasing the size of the problem instances.

Working in a higher level of abstraction, Craus and Rudeanu [29] implemented a reusable framework for executing master-slave parallel applications. Checkpoints were used to optimize the communication between the master and the slaves, which asynchronously requested exchange information by turns. Then, the master only sent the modified information for each slave since its last checkpoint. A parallel ACO was used to test the framework by solving a TSPLIB instance with 229 cities on a Sun Fire 15K with 48 processors, achieving almost linear speedup when using up to 25 processors, but the efficiency decreased when using more resources. Later [30], the authors implemented a pyramidal framework following a master-slave model which includes submasters that managed the slaves under their control. The hierarchical organization allowed to reduce the communications, so it was able to achieve almost linear speedup values when using more than 25 processors.

Recently, the novel GPU platforms have provide an efficient hardware to implement coarse-grain variants of master-slave ACO.

Catalá et al. [19] solved the Orienteering Problem (OP) with a coarse-grain master-slave in which the solutions were built on the GPU and the master executed in the CPU. The experimental analysis solved OP instances with up to 3000 nodes, comparing the GPU parallel ACO executed in a PC with a nVidia GeForce 6600 GT graphic card with 8 pixel shader processors against a domain decomposition method. Accurate solutions were achieved by the GPU implementation using few ants, but the quality of results did not further improve when using additional ants. The execution time for the GPU computing was linear with respect to the number of ants.

Zhu and Curry [106] implemented a GPU coarse-grain master-slave ACO with a local search to solve bound-constrained continuous optimization problems. The solutions were built, evaluated, and improved using a local search method on the GPU, while the remaining tasks were executed on the CPU. The experimental evaluation studied twelve benchmark functions on a PC with

a nVidia GeForce GTX 280 with 240 streaming processors. Given a fixed execution time, the GPU implementation obtained better solutions than a full CPU implementation. The authors reported speedup values ranging between 128 to 403 when using 15360 threads.

5.2.2. Medium-grain master-slave

The master-slave ACO proposed to solve the OP by Mocholí et al. [76] was conceived to execute in a grid environment. The master splits the problem in clusters, and the slaves find partial solutions for each cluster using independent groups of ants with non-overlapping pheromone matrices. Then, the master builds a complete solution for the problem by combining the partial solutions. The communication was provided by high level grid services implemented using web services. Several random generated instances with up to 10,000 nodes were solved in a cluster with 32 PCs, showing that when using up to 32 groups of ants (the number of available processors) the execution times exponentially decreased, and the quality of solutions improved.

A similar decomposition strategy was applied in the *D-ant* algorithm to solve the VRP by Doerner et al. [38]. D-ant splits the problem and uses several slaves to compute a partial solution for each subproblem. Then, the partial solutions are merged by a master process, which also performed the pheromone evaporation and deposit on components of the best-so-far solution. Unlike the previous work, D-ant globally manages the pheromone matrix for the whole problem, and each slave uses its own submatrix. D-ant was able to compute accurate solutions for classical VRP instances with up to 199 clients on a cluster with 16 processors, while reducing the execution time. However, the efficiency values deteriorated when using more than two processors, suggesting a poor scalability behavior. Later [36], an improved implementation of the medium-grain master-slave D-ant outperformed a coarse-grain master-slave, a multicolony, and an hybrid combining these two methods. Classic VRP instances with up to 480 clients were solved on a IBM 1350 cluster with 32 processors. D-ant obtained superior efficiency values (up to 0.75 with 8 processors) with a very small degradation in the quality of the obtained solutions.

5.2.3. Fine-grain master-slave

Randall and Lewis [84] tackled the TSP with a fine-grain master-slave ACS. The slave processes sent each new component included in the solution to the master, which performed the local update of pheromone traces. When solving TSPLIB instances with up to 657 cities in a cluster with 8 processors, the speedup was far below linear, and the maximum efficiency (0.83) was obtained when using only two processors. These poor results suggested that the fine-grain approach with local update of pheromone is not an efficient idea for ACO parallelization, due to the high frequency of communications. The quality of the obtained solutions were not reported.

An improved fine-grain master-slave implementation in a shared memory computer was proposed by Delisle et al. [33]. The master process was replaced by a global memory that stored the global pheromone matrix and the best-so-far solution, and critical regions were used to avoid the mutual update of global information. Each slave updated the local information periodically, but not in every iteration. TSPLIB instances with up to 657 cities were solved on a IBM/P 1600 NH2 with 16 Power3 processors. By spacing out the local information updates, the proposed method achieved better performance than the implementation by Randall and Lewis: it maintained the quality of solutions while avoiding the scalability degradation up to 8 processors. Increasing the number of ants per processor and reducing the number of iterations raised the performance, but reduced the quality of solutions. The experimental analysis was later extended to include a SGI Origin 3800 computer and also to use a Regatta node with Power 4 processors [34]. Better efficiency values were systematically obtained in the IBM Regatta machine, suggesting that technological evolution can improve the efficiency of parallel ACO.

A recent fine-grain implementation of $\mathcal{MM}$AS on GPU was proposed by Fu et al. [47] to solve the TSP. Unlike other approaches, the GPU holds the pheromone matrix, which is actualized after every step. So, the information of the master process is partly stored in CPU and partly in the global memory of the GPU. In each step, the GPU is used to generate random numbers and to compute the next city for each ant, and the CPU only manages small pieces of data (visited cities and routes). The experimental evaluation solved TSPLIB instances with up to 1000 cities on a i7 (3.3 Ghz) with a Nvidia Tesla C1060 (240 cores). The speedup values were up to 30 on the largest instances, and a bottleneck in the communication between CPU and GPU – which demanded more than 20% of the execution time- was detected.

5.2.4. Summary: master-slave parallel ACO

Master-slave implementations have been extensively used to design parallel ACOs (see Table 2 for a summary of the related publications). The model provides an easy and effective way to take benefit of the additional processing power of parallel computers for solving complex problems. Many proposals have used the coarse-grain submodel, since it supplies a conceptually simple schema that achieves good speedup and scalability behavior. The medium-grain submodel was incorporated in the taxonomy in order to include those works that propose a divide-and-conquer-like approach for master-slave parallelization. The first proposals of fine-grain models showed poor efficiency due to the large amount of communications required, so innovative implementations were devised in order to overcome this problem by exploiting fast communication paradigms such as shared memory parallel architectures.

5.3. Cellular model

The cellular model for parallel ACO is a generic proposal following the cellular model for parallel evolutionary algorithms. So, it differs from previous proposals of cellular-like models such as the hardware-parallel ACOs by Middendorf et al. (using processor arrays [72] and FPGA [89]) and other so-called *cellular* implementations, mainly because they use a cellular automata model [8,103], but without proposing a parallel implementation.

The single one implementation of a parallel cellular ACO was presented by Pedemonte and Cancela [78], who proposed a distributed-memory implementation of the cellular model for solving a reliable network design problem. The cellular model was the best parallel method among the studied ones, improving over the results obtained using previous parallel evolutionary algorithms (up to 31% for a specific problem instance), and also showing high values of computational efficiency (over 0.9 using a cluster with four processors). However, the results slightly deteriorated with respect to those achieved by a sequential ACO. Several improvements are reported to be currently investigated in order to overcome the lose of quality when using the cellular model. Implementations using both multicore and GPU parallel architectures could further improve the computational efficiency of this model.

5.4. Parallel independent runs model

The parallel independent runs is a straightforward approach that applies a multi-start search using several processes that executes the same ACO algorithm. The model does not involve communications between processes, so its search mechanism is identical to the one resulting from applying several sequential ACOs.

Table 2
Summary of master-slave parallel ACO proposals.

Author	Year	Algorithm	Problem	Computational platform
Coarse-grain				
Talbi et al. [93]	2001	ANTabu	QAP	SGI Indy cluster
Catalano and Malucelli [20]	2001	AS	SCP	Cray T3D
Delisle et al. [35]	2001	ACO	Industrial scheduling	SGI Origin 1000
Craus and Rudeanu [29]	2004	ACO	TSP	Sun Fire 15K
Craus and Rudeanu [30]	2004	ACO	TSP	Sun Fire 15K
Doerner et al. [37]	2004	AS_{rank}	VRP	Cluster
Peng et al. [80]	2005	ACO	Packing problem	PC
Peng et al. [79]	2006	ACO	Image restoration	PC
Lv et al. [70]	2006	$\mathcal{MM}$AS,ACS	TSP	IBM p-server multiprocessor
Li et al. [64]	2007	ACS	Codebook design	DeepSuper-21C
Catalá et al. [19]	2007	ACO	OP	GeForce 6600GT GPU
Tsutsui [95]	2007	*c*AS	QAP	PCs (dualcore, quadcore)
Tsutsui [96]	2008	*c*AS	QAP	PCs (dualcore, quadcore)
López-Ibáñez et al. [68]	2009	ACO	Pump scheduling	Dualcore PC
Guo et al. [51]	2009	ACO	Protein structure prediction	IBM p-server multiprocessor
Zhu and Curry [106]	2009	ACO	Bound constrained optimization	GeForce GTX 280 GPU
Weis and Lewis [99]	2009	ACS	Antenna design	Non-dedicated cluster
Ibri et al. [55]	2010	ACS	Emergency fleet dispatching	Intel Core2 Duo
Chintalapati et al. [25]	2010	ACO	Classification rules discovery	Cluster
Tsutsui and Fujimoto [97]	2010	*c*AS	TSP	i7 965
Gao et al. [48]	2010	ACO	Target assignment	Pentium Dual Core
Li et al. [63]	2010	ACO	TSP	Pentium Dual Core
Medium-grain				
Mocholí et al. [76]	2005	ACO	OP	Cluster, grid
Doerner et al. [38]	2005	D-ant	VRP	Cluster
Doerner et al. [36]	2006	D-ant	VRP	IBM 1350 cluster
Fine-grain				
Randall and Lewis [84]	2002	ACS	TSP	IBM SP-2 cluster
Delisle et al. [33]	2005	ACS	TSP	IBM/P1600
Delisle et al. [34]	2005	ACS	TSP	SGI Origin 3800, IBM Regatta
Fu et al. [47]	2010	$\mathcal{MM}$AS	TSP	i7, Nvidia Tesla C1060

Stützle [90] studied the parallel independent execution of $\mathcal{MM}$AS with a local search for the TSP. The evaluation analyzed the benefits of using a parallel model with respect to a sequential ACO, comparing the quality of the solutions for TSPLIB instances with up to 1173 cities in a UltraSparc II workstation. The results showed that the parallel executions obtained better solutions than the sequential algorithm in all the studied instances.

Later, a parallel independent runs implementation of AS with a local search demanding high execution times was presented by Rahoual et al. [83] to tackle the SCP. The AS was compared against a coarse-grain master-slave parallel ACO for solving SCP instances with up to 500 elements and 5000 subsets in a cluster of 40 PCs. The parallel independent runs AS obtained near ideal efficiency values due to the negligible processes communication, and it also improved the solutions quality with respect to the sequential algorithm. On the other hand, the coarse-grain master-slave efficiency values strongly depended on the size of the problem instances, ranging from 0.21 for the smaller instances to 0.83 for the larger ones.

Alba et al. [4] compared three parallel ACS (parallel independent runs, multicolony, and coarse-grain master-slave) to solve the Minimum Tardy Task Problem (MTTP) instances in a cluster with only 3 PCs. The three methods achieved similar quality of solutions. Mixed results were reported when increasing the problem size using a fixed number of processors, but the parallel independent runs model generally obtained the highest efficiency values. Later [5], the parallel independent runs was compared against asynchronous multicolony models using star and unidirectional ring topologies in a cluster of 8 PCs. The models that involve communication found better solutions, but the parallel independent runs model had better efficiency values.

Bai et al. [11] implemented a parallel independent runs of $\mathcal{MM}$AS on GPU. Each thread executed one ant and each thread block was used for an independent execution. The main algorithm runs on GPU, while the CPU is only used to initialize the solutions and to control the iteration process. Six TSPLIB instances with up to 400 cities were solved in a PC with a nVidia GeForce 8800 GTX with 128 stream processors. Regarding the solutions quality, the GPU-parallel implementation outperformed three sequential $\mathcal{MM}$AS versions, while acceleration values between 2 and 32 were obtained.

Table 3 summarizes the proposals of parallel independent runs ACO implementations. The model has been seldom used. It has been employed in works mainly focused on achieving speedup and scalability improvements. The search mechanism is similar to the one employed by sequential ACOs, even though the multistarting approach is sometimes useful to avoid stagnation. Parallel independent runs ACO implementations frequently obtain similar results quality than sequential ACOs, and they are often outperformed by parallel models that use communication.

Table 3
Summary of parallel independent runs ACO proposals.

Author	Year	Algorithm	Problem	Computational platform
Stützle [90]	1998	$\mathcal{MM}$AS	TSP	UltraSparc II workstation
Rahoual et al. [83]	2002	AS	SCP	Cluster
Alba et al. [4]	2005	ACS	MTTP	Cluster
Alba et al. [5]	2007	ACS	MTTP	Cluster
Bai et al. [11]	2009	$\mathcal{MM}$AS	TSP	GeForce 8800GTX GPU

5.5. Multicolony model

Parallel ACO implementations following the multicolony model have been extensively used. This approach provides a cooperative search mechanism that often allows obtaining superior results than the sequential model as well as outperforming other parallel ACOs. Multicolony also admits a simple implementation in distributed memory platforms such as clusters of computers.

The main features to be considered when designing a multicolony ACO were summarized by Janson et al. [57]: the communication frequency and the neighborhood topology; the type of information that is exchanged between the colonies; how the information received from other colonies is used; and homogeneous versus heterogeneous approaches.

Michel and Middendorf [73,74] introduced a configuration that became the standard for multicolony ACO, since it was later used by many other authors: a synchronous algorithm with a structured neighborhood and a fixed communication frequency for sharing the best solutions, each one of whom, if better, substitutes the best-so-far solution in the destination colony.

Michel and Middendorf solved the Shortest Common Supersequence Problem (SCSP), a problem with application in bioinformatics (DNA sequencing). Their multicolony model obtained slightly better solutions than a single colony ACO for strings with length up to 160, but no efficiency analysis were performed. Later, four different exchange strategies were studied by Middendorf et al. [75] to solve the TSP and the QAP. The multicolony model outperformed a sequential ACO in a TSPLIB instance with 101 cities, while no definitive conclusions were drawn for a QAPLIB instance with 60 locations. The best results were obtained when sending the best local solution using an unidirectional ring connection topology and substituting the best-so-far solution accordingly. The authors also concluded that the exchange frequency should be set to avoid the degradation of either the solution quality or the ACO computational efficiency.

Piriyakumar and Levi [82] solved the TSP with a standard multicolony ACO. The experimental evaluation solved a TSPLIB instance with 52 cities on a Cray T3E multiprocessor, studying several quality and efficiency metrics: the cost of the best solution, the total computing time and the single processor time, and the rates between communication/idle times and total computation time. The analysis showed that good quality results can be obtained while maintaining bounded values for idle and communication times. A speedup study was not included.

A weapon-target assignment problem was faced by Lee et al. [62] using a multicolony ACO that applied a local search method to the best solution from all colonies, and the improved solution was considered for an additional pheromone update process. The proposal outperformed a GA and other sequential and parallel ACOs regarding both the solution quality and the execution time when solving problem instances with 120 weapons and 100 targets.

PACS, the multicolony ACS to solve the TSP by Chu et al. [27], followed the standard approach but it included an additional update every time that a colony receives a solution. The experimental evaluation only studied the solution quality for three TSPLIB instances with up to 225 cities, omitting an efficiency analysis. Several connection topologies were evaluated, and all the proposed PACS variants obtained better solutions than sequential AS and ACS algorithms.

Chu and Zomaya [26] evaluated two multicolony models – using a circular exchange of solutions and a shared pheromone matrix, respectively – for solving a prediction of protein structure problem. The experimental evaluation was unconventional, since it used as the performance metric the number of CPU ticks needed to find the best solution in a cluster with 5 processors. Both multicolony methods outperformed a coarse-grain master-slave approach, and the multicolony with circular exchange of solutions achieved the best performance values.

Yang et al. [104] applied a traditional multicolony ACO using an unidirectional ring topology to maximize the density of the demand covered by direct travels on a bus network. The experimental evaluation studied the solution quality and the execution time in a cluster of 8 PCs. Both metrics improved in the parallel ACO when compared with a single colony model.

The standard approach was also applied in the multicolony $\mathcal{MM}$AS by Xiong et al. [100] to solve the TSP. The experimental analysis solved classical TSPLIB instances with up to 15915 cities in a Dawn 4000L massively parallel processing (MPP) computer with 64 nodes. The parallel $\mathcal{MM}$AS got better solutions than the sequential one as the problem size increased, but the efficiency significantly decreased when using more than 8 nodes.

Hongwei and Yanhua [54] solved the DNA sequence determination problem applying a multicolony $\mathcal{MM}$AS that used the strategy of sorting and exchanging information for pheromone updating. When executed in a Dawn TC4000 MPP supercomputer, the parallel $\mathcal{MM}$AS achieved better solutions than a sequential ACO, a TS method, and a GA. The authors claimed to have a sublinear scalability behavior due to the communications, but no speedup analysis was presented.

Jovanovic et al. [58,59] solved the Minimum Weight Vertex Cover Problem (MWVCP) with a standard multicolony ACO implemented with threads in a Intel Core2 PC. Several interconnection topologies and policies on how to use the exchanged information were studied for instances with up to 150 nodes. The multicolony model computed better solutions than both a parallel independent runs and a sequential ACO. The authors found out that increasing the number of colonies is not always a good strategy in order to improve the results. No efficiency analysis was carried out.

Taškova et al. [94] solved a finite element mesh decomposition problem using a method that combines a parallel multicolony ACO with a refinement algorithm. A standard multicolony approach is used, which exchanges the best solutions found on each refinement level. The experimental analysis was performed in a cluster with 8 nodes, each one with two AMD Opteron 1.6 GHz processors. The distributed implementation obtained the same quality of solutions than the sequential one, but low speedup values were achieved (up to 2.98 when using 8 processes), mainly due to the inherent sequential features of the method.

Recently, the multi-depot VRP was solved by Yu et al. [105] applying a multicolony ACO where outstanding ants are exchanged at certain intervals using a ring interconnection topology. Instances with up to 360 customers were solved in a cluster with 8 PCs. The multicolony ACO achieved competitive solutions when compared with other methods (less than 3% far of the best known solutions), but no efficiency analysis was performed.

Researchers have also introduced other variants of multicolony that differ from the standard implementation. Among several different approaches, asynchronous multicolony algorithms have often been proposed, and another frequent idea is to include dynamic or adaptive methods for the communication frequency and/or topology.

The multicolony ACO by Chen and Zhang [23] used adaptive methods for exchanging the best solutions and adjusting the frequency of information exchanges. TSPLIB instances with up to 318 cities were solved in a Dawn 2000 MPP computer, comparing the solutions quality and the execution time against a sequential ACO and a multicolony with circular exchange of the best local solution. The adaptive multicolony performed the best, and the adaptive frequency allowed reaching better results than using fixed intervals, though it required larger execution times. A sub-linear speedup behavior was detected up to 35 processors, and the values improved when facing TSP instances of increasing size. Efficiency

values up to 0.8 were obtained when using 25 processors to solve the largest instance.

Chen et al. [22] also proposed a novel strategy for information exchange, where each colony dynamically determines a destination colony to send its best solution using an adaptive method. The frequency of information exchange depended on the diversity of solutions, and the pheromone matrix was updated considering the best solution of the colony and the received solution. A Shenteng 1800 MPP supercomputer was used to solve TSPLIB instances with up to 318 cities. The results improved over a sequential ACO, and the adaptive information exchange strategy allowed keeping a balance between the convergence and the diversity of the solutions. The speedup did not linearly increase when adding processors, but good efficiency values were obtained for large sized problems when using up to 25 processors.

In the same line of work, Ellabib et al. [46] proposed MACS, a synchronous multicolony ACS with an adaptive mechanism for the global pheromone update. In MACS, each colony worked independently, sharing the best solutions through an exchange module that encapsulated all the communication issues. Star, hypercube, and unidirectional ring interconnection topologies were compared in the experimental evaluation on a cluster of 8 PCs for solving standard medium-sized instances of the VRP with time windows (VRPTW) and the TSP. The analysis studied the effect of the different topologies on the solutions quality, showing that the best results were obtained with the star topology. In addition, star-MACS also outperformed a previous ACS for the VRPTW, and a PACS for the TSP. No efficiency analysis was reported.

Manfrin et al. [71] made an in-depth analysis of parallel ACOs to solve the TSP. Multicolony was identified as a promising parallel model applied to $\mathcal{MM}$AS to solve TSPLIB instances with up to 2392 cities in a cluster of 8 PCs. Better results were achieved when reducing the frequency of the communication between colonies. So, the authors suggested using sophisticated communication patterns, dependant on the size of the instances and the execution time, and other complex mechanisms such as reinitializing the pheromone trails and dividing the search space to avoid premature convergence. Recent experiments on multicolony configuration by Twomey et al. [98] showed that the best communication strategies depend on whether a local search method is used or not, thus a specific analysis should be performed to avoid a subpar search behavior. The analysis concluded that preventing high communication rates makes more likely that colonies focus on different regions of the search space, emphasizing the exploration and possibly improving the results. The two previous works focused on the solution quality, and they did not present efficiency analysis.

Lucka and Piecka [69] solved the VRP with a multicolony savings-based ACO in a multicore architecture, using a different thread for each colony. The colonies asynchronously exchange the best solutions using the shared memory within the same node and using shared files across different nodes. The experimental analysis solved VRP instances with up to 420 customers in a cluster with 72 Sun X4100, each one with two dual core processors. When increasing the number of colonies up to 32, the quality of the solutions improved and the execution times reduced. Moreover, the communication time remained bounded even for the largest VRP instances solved.

The works by Xu et al. [102] and Xiong et al. [101] introduced a multicolony ACS with dynamic transition probability. The information exchange occurs with a fixed frequency, but the global best solution and the full pheromone matrix are communicated in different iterations. A partially asynchronous implementation was studied to solve the TSP in a Dawn 4000L, achieving similar result quality than a sequential ACS. The parallel ACS scaled poorly, since the best efficiency values were obtained when using only two nodes, and it quickly drops when increasing the number of nodes up to 16. Later, Xiong et al. studied a polymorphic version of the previous method (i.e. using different kind of ants and pheromone), and similar results were obtained.

Sameh et al. [87] also studied a multicolony ACS that exchanges the full pheromone matrix, placing each colony in a different processor. Unlike the previous approach, the best-so-far solution and the full pheromone matrix were exchanged between the same pairs of colonies, and the received information was used in the pheromone update process. The experimental results for a TSP instance with 318 cities demonstrated that the time required to find an optimal solution decreases when using more colonies, and that the information exchange frequency affects the time required to find an optimal solution. No efficiency analysis was carried out.

A different approach has been applied to solve optimization problems with multicolony ACOs using a problem-decomposition strategy. The multicolony ACO by Chen et al. [24] solved the classification rule discovery problem using several colonies to independently search for the antecedent part of an input set of rules and broadcasting the training set each time that a colony updated it. Four different data sets were solved in a Dawn 2000 MPP computer. The multicolony found simpler and more accurate rules than both a sequential ACO and a decision-tree-based algorithm, but no efficiency study was reported.

An original multicolony ACO to solve large-dimension decomposable problems was presented by Lin et al. [65]. The model used two coupled colonies to optimize different parts of the objective function applying local procedures for the construction phase and the pheromone update process. A stagnation-based asynchronous migration exchanged information between colonies. The proposed method got better solutions and convergence speed than a sequential ACO when solving eight continuous problems. Lin et al. claimed that this parallel model should be useful for optimization in high dimensional spaces, but no efficiency analysis was performed.

5.5.1. Summary: multicolony parallel ACO

Multicolony models have been widely used in parallel ACOs as they provide an accurate search exploration pattern based on the cooperative behavior of many ant colonies. Indeed, Table 4 summarizes the proposals of multicolony parallel ACO algorithms. Many authors have followed the traditional configuration by Michel and Middendorf [73,74], but some other valuable variants have been proposed, including asynchronous implementations, dynamic and adaptive models for the communication frequency and/or the neighborhood topology, and multicolonies following a problem-decomposition approach. Most of the multicolony implementations were developed using the distributed memory paradigm in cluster platforms.

Multicolony ACOs showed promising results since the first implementation in the last years of the 1990's, but the model has been continuously improved, paying special attention to the mechanisms used for exchanging information among colonies. Recent studies confirmed that a trade-off between the isolated exploration within each colony and the cooperation using information exchange is desirable in order to achieve accurate results and performance. Increasing the frequency of communication puts emphasis in exploiting good solutions, so this strategy is useful when solving instances of increasing size. However, it also could have a negative impact in the parallel ACO efficiency due to the cost of the communications.

5.6. Hybrid models

There have been many proposals for designing hybrid parallel ACOs that combine the characteristics of more than one parallel model. However, in practice, only a few hybrid parallel ACO methods have been implemented.

Table 4
Summary of multicolony parallel ACO proposals.

Author	Year	Algorithm	Problem	Computational platform
Michel and Middendorf [73]	1998	EAS	SCSP	Multiprocessor
Michel and Middendorf [74]	1999	EAS	SCSP	Multiprocessor
Middendorf et al. [75]	2002	EAS	QAP	PC
Piriyakumar and Levi [82]	2002	ACO	TSP	Cray T3E
Lee et al. [62]	2002	ACO	Weapon-target assignment	PCs
Chu et al. [27]	2004	ACS	TSP	N/D
Chen and Zhang [23]	2005	ACO	TSP	Dawn 2000
Chu and Zomaya [26]	2006	ACO	Protein structure prediction	Cluster
Chen et al. [24]	2006	ACO	Classification rule discovery	Dawn 2000
Manfrin et al. [71]	2006	$\mathcal{MM}$AS	TSP	Cluster
Yang et al. [104]	2007	ACO	bus network design	Cluster
Ellabib et al. [46]	2007	ACS	VRPTW,TSP	Cluster
Chen et al. [22]	2008	ACO	TSP	Shenteng 1800
Xiong et al. [100]	2008	$\mathcal{MM}$AS	TSP	Dawn 4000L
Lin et al. [65]	2008	ACO	Decomposable problems	PC
Hongwei and Yanhua [54]	2009	$\mathcal{MM}$AS	DNA sequence determination	Dawn TC4000
Lucka and Piecka [69]	2009	ACO	VRP	Cluster
Xu et al. [102]	2009	ACS	TSP	Dawn 4000L
Jovanovic et al. [58]	2009	ACO	MWVCP	Intel Core2
Jovanovic et al. [59]	2010	ACO	MWVCP	Intel Core2
Taškova et al. [94]	2010	ACO	Finite element mesh decomposition	Cluster
Twomey et al. [98]	2010	$\mathcal{MM}$AS	TSP	Cluster
Xiong et al. [101]	2010	ACS	TSP	Dawn 4000L
Yu et al. [105]	2010	ACO	Multi-depot VRP	Cluster
Sameh et al. [87]	2010	ACS	TSP	Cluster

The *parallel combination of ants and evaluation of solution elements* was included as one of the categories of the taxonomy by Randall and Lewis [84]. This model used two levels of master-slave parallelism; one between the colony and the ants, and another one between each ant and the evaluation of the solution components. Later, Delisle et al. [33] suggested an hierarchical hybrid in two levels. The upper level was a multicolony model that employed message passing for the intercolony communications, and the lower level was a fine-grain master-slave in which the master process was replaced by a global shared memory. No specific implementations have been presented for the first theoretical proposal, while the second one has been recently adopted in the work by Liu et al. [66], where it was used to solve a routing problem in mobile networks.

Iimura et al. [56] studied a parallel Queen Ant Strategy (AS_{queen}), a method using a group of agents to build solutions and a *queen ant* giving directives to the agents about when to diversify the search or to exploit good solutions. This parallel ACO followed an hybrid approach that combined a master-slave model with a multicolony: the queen ant was the master, which controlled the search, and each slave held a colony that worked independently on the same problem, periodically sending its best solution to the master. A TSPLIB instance with 76 cities was solved in the experimental evaluation in an heterogeneous cluster with 9 PCs. The parallel model slightly improved the solutions quality when considering an increasing number of agents, while also reducing the execution time.

An original parallel ACO implementation was proposed by Liu et al. [67] for solving multi-stage decision problems following a decomposition strategy using a construction graph. Each process performed the ACO search in a subproblem and the whole solution search was accomplished cooperatively by the set of processes. Several processors were arranged in a pipeline-like structure that allowed having many ants concurrently solving subproblems at the same time, and the pheromone update was performed after each ant built a whole solution. The experimental evaluation solved an ad-hoc example problem using a cluster of PCs, and the efficiency analysis showed that the speedup values increased when solving large problems or when using many ants.

Roozmand and Zamanifar [86] improved the previous proposal from Chen et al. [24] to solve the classification rule discovery problem with an hybrid parallel ACS. In the new method several colonies worked independently on different categories with all the input set, and a coarse-grain master slave model – implemented with multithreading – was used for constructing and evaluating the solutions. An additional pheromone update is applied using the values of the best colony each time that a rule is found. Five public-domain desease data sets were used in the experimental evaluation. The method was able to discover simple rules than the parallel ACO by Chen et al. [24], with higher predictive accuracy than those obtained by other previous ACO-based methods. The authors claimed that "by using efficient communication methods" the multicolony model was able to reduce the speed of convergence to a local optimum (thus achieving more accurate results), but no further details were provided about the computational efficiency.

Following a different line of work, the parallel ACO by Liu et al. [66] implemented a model similar to the two-level hierarchical hybrid proposal by Delisle et al. [33]. The upper level used a standard multicolony approach employing the Message Passing Interface for communication; and the lower level used a fine-grain master-slave model to manage the shared pheromone matrix using multithreading (the master process is replaced by a global shared memory). This hybrid model was applied to solve a multi-path routing problem in Mobile Ad-hoc Networks (MANETs), and the parallel ACO achieved better solutions than both the AODV and DSR protocols regarding the packet delivery ratio and average end-to-end packet delay. No efficiency analysis was reported.

A summary of the hybrid proposals for parallel ACO is presented in Table 5.

6. Comparative analysis

This section presents a comparative analysis of the parallel ACO models in the new taxonomy, regarding the two main goals when using a parallel implementation: the computational efficiency and the quality of results. The study is aimed at offering an insight on the main results from the related works, providing specific contributions to the research community in order to evaluate the benefits of each parallel model, and possibly helping them to select one of them for solving a given optimization problem.

Table 5
Summary of hybrid parallel ACO proposals.

Author	Year	Algorithm	Problem	Computational platform
Randall and Lewis [84]	2002	ACO	N/D	N/D (theoretical)
Delisle et al. [33]	2005	ACO	N/D	N/D (theoretical)
Iimura et al. [56]	2005	AS_{queen}	TSP	Cluster
Liu et al. [67]	2006	ACO	Multi-stage problems	Cluster
Roozmand and Zamanifar [86]	2008	ACS	Classification rule discovery	Cluster
Liu et al. [66]	2008	ACO	Multi-path routing in MANETs	Cluster

6.1. Computational efficiency

The overview of the related works allows to conclude that coarse-grain master-slave and multicolony are the most promising models for achieving high computational efficiency when implementing parallel ACOs.

The multicolony model has a certain degree of flexibility that allows developing implementations in several parallel computing platforms without downgrading the efficiency values it usually obtains in distributed memory computers. Multicolony provides the best scalability behavior, and it is the most suited model to be executed in large clusters, allowing to tackle hard-to-solve optimization problems that involves complex computations and/or manages large volumes of data [26,54,104]. However, a comprehensive experimental study on the capabilities of the multicolony model implemented in other hardware platforms, regarding the computational efficiency and scalability metrics, remains to be done.

The performance of the coarse-grain and the fine-grain master-slave models is strongly related to the frequency of communications and the workload of each slave. For this reason, usually coarse-grain implementations are able to achieve better speedup values than fine-grain ACOs. However, when the number of slaves highly increases, the speedup of the coarse-grain master-slave model may deteriorate due to bottleneck in the communications to the master. To tackle this problem, some researchers have grouped several ants in a single slave [25,37,51,70], while other proposals have incorporated submasters that control several slaves in order to minimize the communications to the master [29,30]. A third alternative is to use an asynchronous model of communications, which modifies the behavior of the sequential ACO since the slaves work with the information sent by the master in a previous communication [16,20,29,51]. Regarding the fine-grain master-slave model, the empirical analysis indicate that the large volume of required communications conspires against achieving large speedup values. To deal with this issue, some authors have proposed to space the communications and/or synchronizations to improve the speedup values [33,84], modifying the algorithmic behavior of the sequential ACO. A recently proposed new alternative consists of implementing the fine-grain master-slave model on GPU [47], using the shared memory to store the pheromone matrix and to perform the communications. In this new proposal, a key issue is to reduce the bottleneck in the communication between CPU and GPU.

The medium-grain master-slave model is a novel approach not present in other metaheuristics that do not involve a construction process (e.g., EAs). As this process consists in sequentially incorporating components to a empty solution, it can be split between multiple slave process that compute solutions to subproblems. This approach is helpful to cope with very large instances that otherwise would be computationally very expensive or even impossible to tackle due to memory limitations. The medium-grain master-slave model has obtained large speedup values when implemented in grid environments [76] and acceptable speedup values in clusters [36].

The parallel independent runs is conceptually the simplest one among the models in the new taxonomy. It trivially achieves almost linear speedup values, since it does not involve communications. In general, this is the only parallel model able to outperform the efficiency of the multicolony ACO [4,5].

The cellular model has the capability of achieving high speedup values, specially when implemented in modern parallel computing platforms. In this model of parallelization there is a trade-off between the size of the neighborhood and the computational efficiency, so an asynchronous model of communication is suggested as the best strategy for not downgrading the performance [3].

The hybrid models have generally been implemented using a multicolony in which each colony works following a master-slave model. The computational efficiency of these models is quite complex to analyze, because it involves two different levels of communication. When an accurate tunning is performed, hybrids models are able to inherit the good levels of efficiency of the multicolony ACO [56,67].

6.2. Quality of results

The algorithmic behavior of the coarse-grain and the fine-grain master-slave models is similar to a sequential implementation, except for minor effects produced by concurrence (for example, when a master-slave implementation uses a local pheromone deposit or an asynchronous model of communication). As a consequence, no major differences in the quality of the solutions are observed when comparing with a sequential ACO for most optimization problems.

The parallel independent runs model executes identical copies of a sequential ACO, thus it also obtains similar quality of results. However, by using several independent colonies the model is sometimes able to achieve slightly better results than a sequential ACO, specially when the traditional method suffers a stagnation situation.

The quality of the solutions computed by the multicolony model is clearly superior than the sequential one for most optimization problems, mainly due to the multiple search mechanism, the improved diversity, and the ability of handling large problem instances. The impact of the frequency of the communications in the solution quality was extensively studied by Twomey et al. [98], showing that preventing high communication rates between colonies makes more likely that the parallel ACO focus on different regions of the search space, emphasizing the exploration and improving the results. When implemented as a multicolony combined with another parallel ACO model, hybrids models shares this capability, and they usually obtain more accurate results than both sequential and non-hybrids parallel models.

Up to now, the only one implementation of the cellular model for parallel ACO showed a slight deterioration in the quality of the solutions when solving a network design problem. Several issues about this method need to be further studied, specially the balance between the neighborhood size (and the communications) and the quality of solutions.

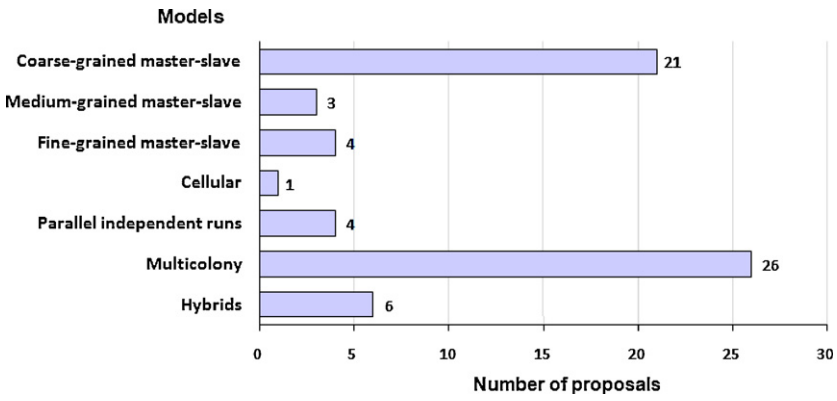

Fig. 4. Number of parallel ACO proposals grouped by model.

7. Trends and perspectives

As it has been made clear along the previous sections, the reviewed literature includes many articles introducing various alternatives for parallel ACO algorithms, following different approaches. Some general perspectives can be extracted from the study of the existing proposals.

7.1. Overview

The parallel independent runs model usually obtains results of similar quality as the single colony implementations, but the multistart approach is sometimes useful to avoid stagnation and the model can trivially achieve linear efficiency gains. Concerning the search pattern, most master-slave models essentially work as a single-colony algorithm, thus achieving the same quality of results. The medium grain master-slave model, which applies a decomposition approach, performs a different search, specially adapted to large dimension problem instances. All master-slave parallel ACOs are able to obtain efficiency gains which depend on the load balancing; usually the coarse grain master-slave model fare much better than the fine-grain model, due to their reduced communication loads. Multicolony models usually are able to get better quality of solutions than other parallel models, and they provide acceptable efficiency values in most platforms. Tuning the frequency of communication is the key element for reaching a good trade-off between efficiency and quality of solutions in a multicolony ACO; this is not always easy to achieve, and is typically done empirically.

The main trend in the field is to choose multicolony models and to tune the communication frequency, as the best alternative for ACO parallelization. Coarse-grain master-slave models are a robust second choice, which has also been employed in many articles. The newly proposed cellular ACO is not consolidated yet, and further work is needed in order to determine its usefulness. Fig. 4 presents the number of publications grouped by parallel ACO model (as previously mentioned, the pioneering works are not classified).

7.2. Software issues and tools

In a general view, the metaheuristics research community has proposed and implemented frameworks including parallel versions of many well-known techniques. Generic frameworks help developing new parallel metaheuristics variants, experimenting with existing ones, tackling new applications, and quickly performing fair comparisons in a well-known and stable environment. Such a framework for parallel ACO algorithms has not been found out in the exhaustive review of the literature performed to prepare this work. Most research works have developed ad-hoc implementations, without taking into account the design of general software libraries for parallel ACO models. The solely exception is the parallel skeleton for ACO included in the MALLBA library [6], but it has only been used by its creators [4,5]. One of the reasons which may explain the absence of a generic parallel ACO framework is that the implementations seem to be more closely tied to the particular problems solved than in other metaheuristic techniques. As a consequence, each work essentially starts the implementation from scratch, making it difficult to reuse the existing work, and to compare alternative methods. There is, then, an open challenge to develop such a framework for parallel ACO.

C and C++ have been the most used languages to develop parallel ACO implementations. Some other languages and tools have been sporadically used, such as Java and Matlab. Ad-hoc implementations of parallel ACO algorithms have been developed using libraries for parallel computing, such as implementations of the MPI standard [50] for distributed memory platforms, OpenMP [21] and Intel Threading Building Block [85] for shared memory computers.

7.3. Parallel computing platforms

When studying any class of parallel algorithms, it is important to take into account which computing platforms are used, as their architecture notably impacts in the time required to perform the communications, the synchronization, and the data sharing. Fig. 5 shows the number of publications grouped by the type of parallel architecture employed. Cluster platforms have been the most popular choice for implementing parallel ACO algorithms, followed by parallel ACO proposals implemented on multiprocessors and massively parallel computers. In the last years, an important trend in the field of parallel computing has focused on new platforms, in particular grid computing, multi-core servers and GPU-based computing. Nevertheless, in the parallel ACO community, the experimentation with these new architectures has just started. Some examples include: two implementation conceived to execute on a grid environment [76,99], four based on GPU cards [11,19,47,106], and nine works which employed multi-core architectures [48,55,58,59,63,68,95–97]. The near future will certainly witness a growing number of implementations on these new platforms, leading to novel proposals specifically adapted to take advantage of the infrastructure characteristics.

7.4. Application domains

Table 6 summarizes the number of parallel ACO proposals regarding the application domains. Many of the published works have focused on providing experimental results based on solving well-known combinatorial optimization problems, like TSP, QAP, MAXSAT, and other classical problems. These works have been mainly devoted to demonstrate the effectiveness of the parallel

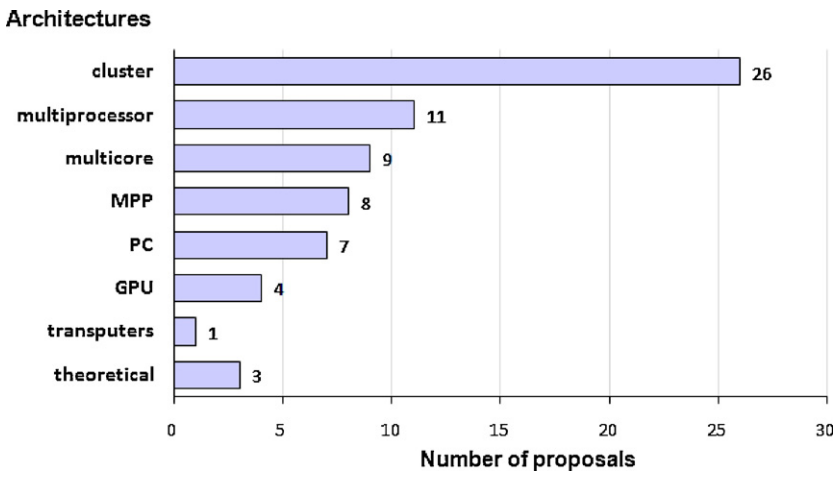

Fig. 5. Number of publications grouped by parallel computational platform.

models of ACO to outperform the traditional sequential implementations, regarding both the computational efficiency metrics and/or the quality of results. There have been a predilection for working with TSP, which was the problem solved by the first papers on ACO algorithms, and since then, it has been the benchmark of preference.

Nevertheless, in recent years there has been a growing number of papers applying parallel ACOs to solve a larger variety of problems, including some real-life based ones, instead of idealized benchmark situations.

Concerning those real-world problems, parallel models of ACO have been used in practice in many scientific application domains. Among the most important ones solved in the last years, we can mention:

- *Real-world routing and planning*, where optimization problems are used to model complex real-life situations such as vehicle routing, military planning, etc. The master-slave model have been used to solve the Orienteering Problem in its coarse-grain [19] and medium-grain [76] flavors, and the coarse-grain was also applied to target assignment [48]. The multicolony model was applied to multi-depot vehicle routing [105], emergency fleet dispatching [55], bus network design [104], and weapon-target assignment [62]. Both master-slave and multicolony have shown an acceptable level of success to cope with the inherent difficulties of those realistic situations.
- *Industrial and engineering design*, fields where complex functions are used to evaluate the solutions. The coarse-grain master-slave model have allowed the researchers to deal with these kind of difficult problems where sequential ACO tend to perform poorly or are difficult to apply. The problems solved with master-slave ACOs include: packing problem [80], image restoration [79], codebook design [64], pump scheduling [68], and radio frequency antenna design [99].
- *Bioinformatics*, where parallel models of ACO are helpful tools to deal with computation-ally-intensive optimization problems in molecular biology that often also needs to manage very large amount of data. Multicolony ACO have been used with success in protein structure prediction [26] and DNA sequencing [54], while the master-slave ACO has also been applied to protein structure prediction [51].
- *Telecommunications*, a field that have grown at a fast pace in recent years, posing difficult challenges to the research community due to the large size of the infrastructures, the need for obtaining real-time results, etc. Multicolony, master-slave, and the new cellular model have shown a great impact on facing these challenges, providing accurate and efficient solutions to the related optimization problems in network routing [66] and network design [58,59,78].

Table 6
Parallel ACO proposals regarding the application domains.

Application area	Publications			
	Journals	Books/thesis	Conferences	Total
Combinatorial optimization	11	3	24	38
Real-life routing and planning	3	0	4	7
Industrial design	1	0	4	5
Bioinformatics	0	2	3	5
Telecommunications	2	0	2	4
Other scientific problems	1	0	6	7
Theoretical works	1	1	1	3
Total	19	6	44	69

Besides the application domains previously highlighted, parallel ACOs have been recently applied to solve other real-life based problems (as it was shown in Section 5). This fact demonstrates the growing maturity of the research in parallel ACO models, and probably in the near future there will be many more real-life situations tackled with parallel ACO algorithms.

8. Conclusions

This article presents a general overview of parallel ant colony optimization and an exhaustive survey of the proposed implementations. It includes a conceptual discussion of these methods, looking at different classification criteria and previous efforts to develop categories for parallel ACO algorithms. The survey has been the basis to develop a proposal for a new taxonomy, which is a helpful conceptual tool to both understand and organize the existing work, and to identify possible areas for future research.

The work also includes an exhaustive review of the literature in the area, starting from the pioneering works in parallel ACO, up to the most recent proposals (up to December 31, 2010). The reviewed papers are organized according to the new taxonomy, and the main characteristics of the methods employed, as well as the application problems and results obtained, are presented. The discussion of each class concludes with a summary presenting its main features and a list of general conclusions about the efficacy of the corresponding methods. A comparative analysis regarding the computational efficiency and quality of results is also presented.

The final section of the paper discusses some trends and perspectives about parallel ACO, including recommendations about the most effective parallel models and implementations. It also

provides observations about the software issues and libraries, the employed parallel platforms and the application domains, which can be a source of inspiration for future research in the field.

Acknowledgements

The work of M. Pedemonte, S. Nesmachnow, and H. Cancela was partly supported by Programa de Desarrollo de las Ciencias Básicas, Universidad de la República, and Agencia Nacional de Investigación e Innovación, Uruguay.

References

[1] E. Alba, Parallel evolutionary algorithms can achieve super-linear performance, Information Processing Letters 82 (1) (2002) 7–13.
[2] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, Wiley-Interscience, 2005.
[3] E. Alba, B. Dorronsoro, Cellular Genetic Algorithms, Springer, 2008.
[4] E. Alba, G. Leguizamón, G. Ordoñez, Analyzing the behavior of parallel ant colony systems for large instances of the task scheduling problem, in: Proceedings of the 19th International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2005, p. 14.
[5] E. Alba, G. Leguizamón, G. Ordoñez, Two models of parallel ACO algorithms for the minimum tardy task problem, International Journal of High Perfomance Systems Architecture 1 (1) (2007) 50–59.
[6] E. Alba, G. Luque, J. García-Nieto, G. Ordoñez, G. Leguizamón, MALLBA: a software library to design efficient optimization algorithms, International Journal of Innovative Computing and Applications 1 (1) (2007) 74–85.
[7] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (5) (2002) 443–462.
[8] P. Albuquerque, A. Dupuis, A parallel cellular ant colony algorithm for clustering and sorting, in: Proceedings of the 5th International Conference on Cellular Automata for Research and Industry, Lecture Notes in Computer Science 2493 (2002) 220–230.
[9] S. Alonso, O. Cordón, I. Fernández, F. Herrera, Integrating evolutionary computation components in ant colony optimization, in: L. de Castro, F. Von Zuben (Eds.), Recent Developments In Biologically Inspired Computing, IGI Publishing, 2004, pp. 148–180 (Chapter VII).
[10] G. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of the Spring Joint Computer Conference, ACM, New York, NY, USA, 1967, pp. 483–485.
[11] H. Bai, D. OuYang, X. Li, L. He, H. Yu, Max-min ant system on gpu with cuda, in: Proceedings of the 2009 Fourth International Conference on Innovative Computing, Information and Control, IEEE Computer Society, 2009, pp. 801–804.
[12] M. Blesa, C. Blum, Finding edge-disjoint paths in networks: an ant colony optimization algorithm, Journal of Mathematical Modelling and Algorithms 6 (3) (2007) 361–391.
[13] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Computing Surveys 35 (3) (2003) 268–308.
[14] M. Bolondi, M. Bondaza, Parallelizzazione di un algoritmo per la risoluzione del problema del commesso viaggiatore, Master's thesis, Politecnico di Milano, Italy, 1993.
[15] T. Bui, T. Nguyen, J. Rizzo, Parallel shared memory strategies for ant-based optimization algorithms, in: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, ACM, 2009, pp. 1–8.
[16] B. Bullnheimer, G. Kotsis, C. Strauss, Parallelization strategies for the ant system, in: R. de Leone, A. Murli, P. Pardalos, G. Toraldo (Eds.), in: High Performance Algorithms and Software in Nonlinear Optimization, vol. 24 of Applied Optimization, Kluwer Academic Publishers, 1998, pp. 87–100.
[17] E. Cantú-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer Academic Publishers, 2000.
[18] G. Caro, M. Dorigo, AntNet: distributed stigmergetic control for communications networks, Journal of Artificial Intelligence Research 9 (1998) 317–365.
[19] A. Catalá, J. Jaen, J. Mocholí, Strategies for accelerating ant colony optimization algorithms on graphical processing units, in: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE Press, 2007, pp. 492–500.
[20] M. Catalano, F. Malucelli, Parallel randomized heuristics for the set covering problem, in: M. Paprzycki, L. Tarricone, L. Yang (Eds.), Practical Parallel Computing, Nova Science Publishers, 2001, pp. 113–132.
[21] B. Chapman, G. Jost, R. Pas, Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation), The MIT Press, 2007.
[22] L. Chen, H. Sun, S. Wang, Parallel implementation of ant colony optimization on MPP, in: Proceedings of the International Conference on Machine Learning and Cybernetics, vol. 2, IEEE Press, 2008, pp. 981–986.
[23] L. Chen, C. Zhang, Adaptive parallel ant colony algorithm, in: Proceedings of the 1st International Conference in Advances in Natural Computation, Lecture Notes in Computer Science 3611 (2005) 1239–1249.
[24] Y. Chen, L. Chen, L. Tu, Parallel ant colony algorithm for mining classification rules, in: Proceedings of the IEEE International Conference on Granular Computing, IEEE Press, 2006, pp. 85–90.
[25] J. Chintalapati, M. Arvind, S. Priyanka, N. Mangala, J. Valadi, Parallel ant-miner (pam) on high performance clusters, in: Proceedings of the International Conference On Swarm, Evolutionary and Memetic Computing, Lecture Notes in Computer Science vol. 6466 (2010) 270–277.
[26] D. Chu, A.Y. Zomaya, Parallel ant colony optimization for 3D protein structure prediction using the HP lattice model, in: N. Nedjah, L. de Macedo, E. Alba (Eds.), Parallel Evolutionary Computations, vol. 22 of Studies in Computational Intelligence, Springer, 2006, pp. 177–198 (Chapter 9).
[27] S. Chu, J. Roddick, J. Pan, Ant colony system with communication strategies, Information Sciences 167 (1–4) (2004) 63–76.
[28] T. Crainic, H. Nourredine, Parallel metaheuristics applications, in: E. Alba (Ed.), Parallel Metaheuristics, Wiley, 2005, pp. 447–494 (Chapter 19).
[29] M. Craus, L. Rudeanu Multi-level parallel framework, International Journal of Computing 3 (3), electronic publication.
[30] M. Craus, L. Rudeanu, Parallel framework for ant-like algorithms, in: Proceedings of the 3rd International Symposium on Parallel and Distributed Computing, IEEE Computer Society, 2004, pp. 36–41.
[31] V. Cung, S. Martins, C. Ribeiro, C. Roucairol, Strategies for the parallel implementation of metaheuristics, in: P. Hansen, C. Ribeiro (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, 2002, pp. 263–308 (Chapter 13).
[32] J. De Jong, M. Wiering, Multiple ant colony system for the bus-stop allocation problem, in: Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence, 2001, pp. 141–148.
[33] P. Delisle, M. Gravel, M. Krajecki, C. Gagné, W. Price, Comparing parallelization of an ACO: message passing vs. shared memory, in: Proceedings of the 2nd International Workshop on Hybrid Metaheuristics, Lecture Notes in Computer Science vol. 3636 (2005) 1–11.
[34] P. Delisle, M. Gravel, M. Krajecki, C. Gagné, W. Price, A shared memory parallel implementation of ant colony optimization, in: Proceedings of the 6th Metaheuristics International Conference, 2005, pp. 257–264.
[35] P. Delisle, M. Krajecki, M. Gravel, C. Gagné, Parallel implementation of an ant colony optimization metaheuristic with OpenMP, in: International Conference of Parallel Architectures and Complication Techniques, Proceedings of the third European workshop on OpenMP, 2001, pp. 8–12.
[36] K. Doerner, R. Hartl, S. Benkner, M. Lucká, Parallel cooperative savings based ant colony optimization – multiple search and decomposition approaches, Parallel Processing Letters 16 (3) (2006) 351–370.
[37] K. Doerner, R. Hartl, G. Kiechle, M. Lucká, M. Reimann, Parallel ant systems for the capacitated vehicle routing problem, in: Proceedings of the 4th European Conference Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science 3004 (2004) 72–83.
[38] K. Doerner, R. Hartl, M. Lucká, A parallel version of the D-ant algorithm for the vehicle routing problem, in: Proceedings of the International Workshop on Parallel Numerics, 2005, pp. 109–118.
[39] M. Dorigo Optimization, learning and natural algorithms, Ph.D. thesis, Politecnico di Milano, Italy (1992).
[40] M. Dorigo, Parallel ant system: an experimental study, unpublished manuscript, Cited by [41], 1993.
[41] M. Dorigo, G. Di Caro, The ant colony optimization meta-heuristic, in: D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, K. Price (Eds.), New Ideas in Optimization, McGraw-Hill, 1999, pp. 11–32.
[42] M. Dorigo, G. Di Caro, L. Gambardella, Ant algorithms for discrete optimization, Artificial Life 5 (2) (1999) 137–172.
[43] M. Dorigo, L. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 53–66.
[44] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man and Cybernetics Part B 26 (1) (1996) 29–41.
[45] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, 2004.
[46] I. Ellabib, P. Calamai, O. Basir, Exchange strategies for multiple ant colony system, Information Sciences 177 (5) (2007) 1248–1264.
[47] J. Fu, L. Lei, G. Zhou, A parallel ant colony optimization algorithm with gpu-acceleration based on all-in-roulette selection, in: Proceedings of the 3rd International Workshop on Advanced Computational Intelligence, 2010, pp. 260–264.
[48] D. Gao, G. Gong, L. Han, N. Li, Application of multi-core parallel ant colony optimization in target assignment problem, in: Proceedings of the International Conference on Computer Application and System Modeling, vol. 3, 2010, pp. 514–518.
[49] F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, 57, Springer, 2003.
[50] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, Cambridge, MA, 1994.
[51] H. Guo, Q. Lu, J. Wu, X. Huang, P. Qian, Solving 2D HP protein folding problem by parallel ant colonies, in: Proceedings of the 2nd International Conference on BioMedical Engineering and Informatics, IEEE Press, 2009, pp. 1–5.
[52] J. Gustafson, Reevaluating Amdahl's law, Communications of the ACM 31 (5) (1988) 532–533.
[53] A. Hara, T. Ichimura, N. Fujita, T. Takahama, Effective diversification of ant-based search using colony fission and extinction, in: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE Press, 2006, pp. 1028–1035.
[54] X. Hongwei, L. Yanhua, Parallel ACO for DNA sequencing by hybridization, in: Proceedings of the World Congress on Computer Science and Information Engineering, IEEE Computer Society, 2009, pp. 602–606.

[55] S. Ibri, H. Drias, M. Nourelfath, A parallel hybrid ant-tabu algorithm for integrated emergency vehicle dispatching and covering problem, International Journal of Innovative Computing and Applications 2 (4) (2010) 226–236.
[56] I. Iimura, K. Hamaguchi, T. Ito, S. Nakayama, A study of distributed parallel processing for queen ant strategy in ant colony optimization, in: Proceedings of the 6th International Conference on Parallel and Distributed Computing Applications and Technologies, IEEE Computer Society, 2005, pp. 553–557.
[57] S. Janson, D. Merkle, M. Middendorf, Parallel ant colony algorithms, in: E. Alba (Ed.), Parallel Metaheuristics, Wiley, 2005, pp. 171–201 (Chapter 8).
[58] R. Jovanovic, M. Tuba, D. Simian, Analysis of parallel implementations of the ant colony optimization applied to the minimum weight vertex cover problem, in: Proceedings of the 9th World Scientific and Engineering Academy and Society International Conference on Simulation, Modelling and Optimization, 2009, pp. 254–259.
[59] R. Jovanovic, M. Tuba, D. Simian, Comparison of different topologies for island-based multi-colony ant algorithms for the minimum weight vertex cover problem, WSEAS Transactions on Computers 9 (1) (2010) 83–92.
[60] H. Kawamura, M. Yamamoto, K. Suzuki, A. Ohuchi, Multiple ant colonies algorithm based on colony level interactions, Transactions on Fundamentals of Electronics, Communications and Computer Sciences E83-A (2) (2000) 371–379.
[61] H. Koshimizu, T. Saito, Parallel ant colony optimizers with local and global ants, in: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IEEE Computer Society, 2009, pp. 1655–1659.
[62] Z. Lee, C. Lee, S. Su, Parallel ant colonies with heuristics applied to weapon-target assignment problems, in: Proceedings of the 7th Conference on Artificial Intelligence and Applications, 2002, pp. 201–206.
[63] N. Li, D. Gao, G. Gong, Z. Chen, Realization of parallel ant colony algorithm based on tbb multi-core platform, Proceedings of the International Forum on Information Technology and Applications 1 (2010) 177–180.
[64] X. Li, X. Yu, X. Luo, Parallel implementation of ant colony optimization for vector quantization codebook design, in: Proceedings of the 3rd International Conference on Natural Computation, IEEE Computer Society, 2007, pp. 787–791.
[65] Y. Lin, J. Zhang, J. Xiao, A pseudo parallel ant algorithm with an adaptive migration controller, Applied Mathematics and Computation 205 (2008) 677–687.
[66] C. Liu, L. Li, Y. Xiang, Research of multi-path routing protocol based on parallel ant colony algorithm optimization in mobile ad hoc networks, in: Proceedings of the 5th International Conference on Information Technology: New Generations, IEEE Computer Society, 2008, pp. 1006–1010.
[67] H. Liu, P. Li, Y. Wen, Parallel ant colony optimization algorithm, in: Proceedings of the 6th World Congress on Intelligent Control and Automation, vol. 4, IEEE Computer Society, 2006, pp. 3222–3226.
[68] M. López-Ibáñez, T. Prasad, B. Paechter Parallel optimisation of pump schedules with a thread-safe variant of epanet toolkit, in: Geotechnical Special Publication, vol. 187, (2009), pp. 462–471.
[69] M. Lucka, S. Piecka, Parallel posix threads based ant colony optimization using asynchronous communication, in: Proceedings of the 8th International Conference on Applied Mathematics, vol. 2, 2009, pp. 229–236.
[70] Q. Lv, X. Xia, P. Qian, A parallel ACO approach based on one pheromone matrix, in: Proceedings of the 5th International Workshop on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science vol. 4150 (2006) 332–339.
[71] M. Manfrin, M. Birattari, T. Stützle, M. Dorigo, Parallel ant colony optimization for the traveling salesman problem, in: Proceedings of the 5th International Workshop on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science vol. 4150 (2006) 224–234.
[72] D. Merkle, M. Middendorf, Fast ant colony optimization on runtime reconfigurable processor arrays, Genetic Programming and Evolvable Machines 3 (4) (2002) 345–361.
[73] R. Michel, M. Middendorf, An island model based ant system with lookahead for the shortest supersequence problem, in: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science 1498 (1998) 692–701.
[74] R. Michel, M. Middendorf, An ACO algorithm for the shortest common supersequence problem, in: D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, K. Price (Eds.), New Ideas in Optimization, McGraw-Hill, 1999, pp. 51–62.
[75] M. Middendorf, F. Reischle, H. Schmeck, Multi colony ant algorithms, Journal of Heuristics 8 (2002) 305–320.
[76] J. Mocholí, J. Martínez, J. Canós, A grid ant colony algorithm for the orienteering problem, in: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE Press, 2005, pp. 942–949.
[77] A. Noé, K. Verbeeck, P. Vrancx, Multi-type ant colony: the edge disjoint paths problem, in: Proceedings of the 4th International Workshop Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science vol. 3172 (2004) 202–213.
[78] M. Pedemonte, H. Cancela, A cellular ant colony optimisation for the generalised Steiner problem, International Journal of Innovative Computing and Applications 2 (3) (2010) 188–201.
[79] W. Peng, R. Tong, G. Qian, J. Dong, A constrained ant colony algorithm for image registration, in: Proceedings of the International Conference on Intelligent Computing, Lecture Notes in Computer Science vol. 4115 (2006) 1–11.
[80] W. Peng, R. Tong, M. Tang, J. Dong, Ant colony search algorithms for optimal packing problem, in: Proceedings of the First International Conference on Advances in Natural Computation, Lecture Notes in Computer Science vol. 3611 (2005) 1229–1238.
[81] C. Pettey, Diffusion (cellular) models, in: T. Bäck, D. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, Oxford Univ. Press, 1997, pp. 1–6 (Chapter 8).
[82] D. Piriyakumar, P. Levi, A new approach to exploiting parallelism in ant colony optimization, in: Proceedings of the International Symposium on Micromechatronics and Human Science, IEEE Industrial Electronics Society, 2002, pp. 237–243.
[83] M. Rahoual, R. Hadji, V. Bachelet, Parallel ant system for the set covering problem, in: Proceedings of the 3rd International Workshop on Ant Algorithms, Lecture Notes in Computer Science, vol. 2463, Springer, 2002, pp. 262–267.
[84] M. Randall, A. Lewis, A parallel implementation of ant colony optimization, Journal of Parallel and Distributed Computing 62 (9) (2002) 1421–1432.
[85] J. Reinders, Intel Threading Building Blocks, 1st ed., O'Reilly & Associates, Inc, Sebastopol, CA, USA, 2007.
[86] O. Roozmand, K. Zamanifar, Parallel ant miner 2, in: Proceedings of the 9th International Conference on Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science vol. 5097 (2008) 681–692.
[87] A. Sameh, A. Ayman, N. Hasan, Parallel ant colony optimization, International Journal of Research and Reviews in Computer Science 1 (2) (2010) 77–82.
[88] B. Scheuermann, S. Janson, M. Middendorf, Hardware-oriented ant colony optimization, Journal of Systems Architecture 53 (7) (2007) 386–402.
[89] B. Scheuermann, K. So, M. Guntsch, M. Middendorf, O. Diessel, H. ElGindy, H. Schmeck, FPGA implementation of population-based ant colony optimization, Applied Soft Computing 4 (2004) 303–322.
[90] T. Stützle, Parallelization strategies for ant colony optimization, in: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science vol. 1498 (1998) 722–731.
[91] T. Stützle, H. Hoos, MAX-MIN ant system, Future Generation Computer Systems 16 (8) (2000) 889–914.
[92] E. Talbi, Metaheuristics: From Design to Implementation, Wiley Publishing, 2009.
[93] E. Talbi, O. Roux, C. Fonlupt, D. Robillard, Parallel ant colonies for the quadratic assignement problem, Future Generation Computer Systems 17 (4) (2001) 441–449.
[94] K. Tasˇkova, P. Korosěc, J. ěilc, A distributed multilevel ant-colony approach for finite element mesh decomposition, in: Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science vol. 6068 (2010) 398–407.
[95] S. Tsutsui, Cunning ant system for quadratic assignment problem with local search and parallelization, in: Proceedings of the 2nd International Conference on Pattern Recognition and Machine Intelligence, Lecture Notes in Computer Science vol. 4815 (2007) 269–278.
[96] S. Tsutsui, Parallel ant colony optimization for the quadratic assignment problems with symmetric multi processing, in: Proceedings of the 6th International Conference on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science vol. 5217 (2008) 363–370.
[97] S. Tsutsui, N. Fujimoto, Parallel ant colony optimization algorithm on a multi-core processor, in: Proceedings of the 7th International Conference on Swarm intelligence, Lecture Notes in Computer Science 6234 (2010) 488–495.
[98] C. Twomey, T. Stutzle, M. Dorigo, M. Manfrin, M. Birattari, An analysis of communication policies for homogeneous multi-colony aco algorithms, Information Sciences 180 (12) (2010) 2390–2404.
[99] G. Weis, A. Lewis, Using XMPP for ad-hoc grid computing – an application example using parallel ant colony optimisation, in: Proceedings of the International Symposium on Parallel and Distributed Processing, 2009, pp. 1–4.
[100] J. Xiong, C. Liu, Z. Chen, A new parallel ant colony optimization algorithm based on message passing interface, in: Proceedings of the IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, vol. 2, IEEE Computer Society, 2008, pp. 178–182.
[101] J. Xiong, X. Meng, C. Liu, An improved parallel ant colony optimization based on message passing interface, in: Proceedings of the First International Conference on Advances in Swarm Intelligence, Lecture Notes in Computer Science vol. 6145 (2010) 249–256.
[102] J. Xu, X. Han, C. Liu, Z. Chen, A novel parallel ant colony optimization algorithm with dynamic transition probability, in: Proceedings of the 2009 International Forum on Computer Science-Technology and Applications, IEEE Computer Society, Washington, DC, USA, 2009, pp. 191–194.
[103] X. Xu, L. Chen, P. He, A novel ant clustering algorithm based on cellular automata, Web Intelligence and Agent Systems 5 (1) (2007) 1–14.
[104] Z. Yang, B. Yu, C. Cheng, A parallel Ant Colony algorithm for bus network optimization, Computer-Aided Civil and Infrastructure Engineering 22 (1) (2007) 44–55.
[105] B. Yu, Z.-Z. Yang, J.-X. Xie, A parallel improved ant colony optimization for multi-depot vehicle routing problem, Journal of the Operational Research Society 62 (1) (2011) 183–188.
[106] W. Zhu, J. Curry, Parallel ant colony for nonlinear function optimization with graphics hardware acceleration, in: Proceedings of the 2009 IEEE international conference on Systems, Man and Cybernetics, IEEE Press, 2009, pp. 1803–1808.