We are IntechOpen, the world’s leading publisher of Open Access books
Built by scientists, for scientists

3,500
Open access books available

108,000
International authors and editors

1.7 M
Downloads

151
Countries delivered to

TOP 1%
Our authors are among the most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™
Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com
1. Introduction

In the last years, there has been a large growth in the research of computational techniques inspired in nature. This area, named Bioinspired Computing, has provided biologically motivated solutions for several real world problems. Among Bioinspired Computing techniques, one can mention Artificial Neural Networks (ANN), Evolutionary Algorithms (EA), Artificial Immune Systems (AIS) and Ant Colony Optimization (ACO).

ACO is a meta-heuristic based on the structure and behavior of ant colonies. It has been successfully applied to several optimization problems. Several real world optimization problems may change the configuration of its search space with time. These problems are known as dynamic optimization problems. This chapter presents the main concepts of ACO and shows how it can be applied to solve real optimization problems on dynamic environments. As a case study, it will be illustrated how ACO can be applied to process scheduling problems.

The chapter starts describing how real ants forage for food, environmental modifications related to this activity (for example, the blockage of a path) and the effects of these modifications in the colony. After, a brief review of previous works on Ant Colony Optimization is presented.

For computer simulations, the activities of an ant colony may be modeled as a graph. After modeling the problem of foraging for food (as a graph), the goal (finding the shortest path at a given moment) is defined and the mechanism to solve the problem is presented.

The need to model an on-line problem as a graph, the goal of finding the shortest path and the mechanisms adopted for solving the problem (an adapted version of the AntSystem) are detailed.

Before ACO is applied to the process scheduling problem, the problem is analyzed and modeled using a graph representation. Next, simulation results obtained in a set of experiments are presented, which are validated by results obtained in a real implementation. Important issues related with the use of ACO for process scheduling, like parameter adjustments, are discussed.

In the conclusion of this chapter, we point out a few future directions for ACO researches.

The use of computational techniques inspired in nature has become very frequent in the last years. This area, named Bioinspired Computing, provides efficient biologically motivated solutions for several real world problems.
solutions for several real world problems. Among the most popular bioinspired techniques, we may cite Artificial Neural Networks (ANN), Evolutionary Algorithms (EA), Artificial Immune Systems (AIS) and Ant Colony Optimization (ACO).

ACO [DMC96], a meta-heuristic based on the structure and behavior of ant colonies, has been successfully applied to several optimization problems [FMS05, PB05, BN06, SF06, PLF02, WGDK06, CF06, HND05]. For such, it has attracted a large deal of attention lately. Next, we will briefly describe key recent works using ACO to solve real world problems.

Foong et al. [FMS05] proposed a power plant maintenance scheduling optimization considering ant colony optimization algorithms. In this work, the performance of two ACO algorithms were compared: Best Ant System (BAS) and Max-Min Ant System (MMAS). Experimental results suggested that the performance of the studied algorithms can be significantly better than those obtained by other meta-heuristics, such as genetic algorithms and simulated annealing, in this particular application. The work considered as case study a 21-unit power plant maintenance problem investigated in previous researches. In this study, parameters like the number of ants, initial pheromone level, reward factor and others were varied to investigate the ACO search sensitivity.

Pinto and Baran [PB05] presented two multiobjective algorithms for the Multicast Traffic Engineering problem considering new versions of the Multi-Objective Ant Colony System (MOACS) and the Max-Min Ant System (MMAS). These ACO algorithms simultaneously optimize the maximum link usage, the cost of a multicast routing tree, the average delay and maximum end-to-end delay. Results showed a promising performance of the proposed algorithms for a multicast traffic engineering optimization.

Bui and Nguyen [BN06] proposed an algorithm to solve the graph coloring problem. The algorithm employed a set of agents, called ants, to color the graph. The ants were distributed on the vertices of the input graph based on the conflicts. Each ant colored a portion of the graph. Although based on traditional ACO algorithms, each ant solved part of the problem, making it suitable for distributed problems.

Smaldon and Freitas [SF06] investigated a new version of the Ant-Miner algorithm [PLF02], named Unordered Rule Set Ant-Miner, which produces an unordered set of classification rules. The proposed version was compared to the original Ant-Miner algorithm in six public-domain datasets, presenting similar accuracy. However, it discovered more modular rules, which could be interpreted independently from others, supporting its application for interpreting discovered knowledge in data mining systems.

Wang et al. [WGDK06] proposed a design exploration method to exploit the duality between the time and resource constrained scheduling problems. The proposed approach used the Max-Min Ant Colony Optimization to solve both the time and the resource constrained scheduling problems. Compared to using force directed scheduling exhaustively at every time step, the proposed algorithm provided relevant solution quality savings with similar execution time.

Chan and Freitas [CF06] proposed a new ant colony algorithm for multi-label classification, named MuLAM (Multi-Label Ant-Miner). This algorithm is a major extension of Ant-Miner, the first ant colony algorithm for discovering classification rules. According to experimental results, MuLAM presented a better predictive accuracy than other classification techniques investigated.

Hijazi et al. [HND05] used an ant colony algorithm in the wireless communication domain. The problem investigated was how to detect users in a multi-user environment in
synchronous MC-CDMA (Multi-Carrier Code Division Multiple Access) systems, minimizing the interference noise. The optimization solutions found by the ACO reduced the execution time requirements by as much as 98% when compared to an exhaustive search method.

All previous works found biological inspired motivations to solve real world problems. We believe that the approach followed by ant colonies to find the shortest path between their nest and a food source can provide an efficient solution to the process scheduling problem. For such, this chapter presents concepts on Ant Colony Optimization and how it can be applied to optimize process scheduling. This chapter is organized as follows: the section 2 presents ant colony optimization concepts; the problem of foraging for food and how ants solve it is presented in section 3; examples of Ant Colony Optimization algorithms are shown in section 4; the section 5 presents how Ant Colony Optimization can be used for a real class problem, in this case the process scheduling; the section 6 shows the conclusions and future directions, finally the references.

2. How real ants work

Apparently simple organisms, ants can deal with complex tasks by acting collectively. This collective behavior is supported by the release of a chemical substance, named pheromone. During their movement, ants deposit pheromone in their followed paths. The presence of pheromone in a path attracts other ants. In this way, pheromone plays a key role in the information exchange between ants, allowing the accomplishment of several important tasks. A classical example is the selection of the shortest path between their nest and a food source.

For instance, consider four ants and two possible paths, $P_1$ and $P_2$ (Figure 1), which link a nest $N_E$ to a food source $F_S$, such that $P_1 > P_2$. Initially, all the ants ($A_1$, $A_2$, $A_3$ and $A_4$) are in $N_E$ and must choose between the paths $P_1$ and $P_2$ to arrive to $F_S$.

![Figure 1. Ants foraging for food](image)

1. In $N_E$, the ants ($A_1$, $A_2$, $A_3$ and $A_4$) do not know the localization of the food source ($F_S$). Thus, they randomly choose between $P_1$ and $P_2$ with the same probability. Assume that ants $A_1$ and $A_2$ choose $P_1$, and ants $A_3$ and $A_4$ choose $P_2$.

2. While the ants pass by $P_1$ and $P_2$, they leave a certain amount of pheromone on the paths, $\tau_1$ and $\tau_2$, respectively.

3. As $P_1 < P_2$, $A_3$ and $A_4$ arrive to $F_S$ before $A_1$ and $A_2$. In this moment, $\tau_2 = 2$. Since $A_1$ and $A_2$ have not arrived to $F_S$, $\tau_1 = 0$. In order to come back to $N_E$, $A_3$ and $A_4$ must choose
again between \( P_1 \) and \( P_2 \). As in \( F_S \), \( \tau_{C2} > \tau_{C1} \), the probability of these ants choosing \( P_2 \) is higher. Assume that \( A_3 \) and \( A_4 \) choose \( P_2 \).

4. When \( A_3 \) and \( A_4 \) arrive to \( N_E \) again, \( \tau_{C2} \) arrives to 4. This increase in \( \tau_{C2} \) and consolidates the rank of \( P_2 \) as the shortest path. When \( A_1 \) and \( A_2 \) arrive to \( F_S \), \( \tau_{C2} = 4 \) and \( \tau_{C1} = 2 \). Thus, the probability of \( A_1 \) and \( A_2 \) coming back to \( N_E \) through \( P_2 \) becomes higher.

In the previous example, at the beginning, when there is no pheromone, an ant looking for food randomly chooses between \( P_1 \) and \( P_2 \) with a probability of 0.5 (50% of possibility for each path). When there is pheromone on at least one of the paths, the probability of selecting a given path is proportional to the amount of pheromone on it. Thus, paths with a higher concentration of pheromone have a higher chance of being selected.

However, this simple approach leads to the problem of stagnation. Suppose, for example, that ants get addicted to a particular path. Sometimes in near future, that path may become congested, becoming non-optimal. Another problem arises when a favorite path is obstructed and can no longer be used by the ants. In the case of real ants, the environment solves this problem by evaporation\(^1\), i.e., reducing the pheromone values to prevent high concentration in optimal paths (which avoid the exploration of possible - new or better - alternatives).

### 3. Solving problems using ACO

In order to understand how ant colonies may be used to solve problems, we have to understand the problem of foraging for food and how ants solve it. Suppose that each location (nest, food source, etc.) is represent by a node and each path by an edge in a graph, as shown in Figure 2.

![Figure 2: The forage problem modeled as a graph](image)

Thus, to solve a problem using ant colony optimization we have to represent its domain as a graph and its goal as finding a good path. Assume the problem of the traveling salesman [FS91] where, given a set of \( n \) towns, it is necessary to find a minimal length closed tour that visits each town once. The towns are easily represented as the graph nodes and the paths, as the graph edges.

---

\(^1\) When applying ant colonies to solve real problems, this approach may be used together with an heuristic one, combining the pheromone concentration with another information to take decisions.
Another example is the problem of graph coloring: consider a graph \( G = (V, E) \) where \( V \) is a set of vertices and \( E \) is a set of edges. The goal is to assign colors to the vertices in such a way that connected vertices do not have the same color. This must be done using as few different colors as possible.

4. Algorithms for Ant Colony Optimization

Based on the representation presented in the previous section, Dorigo et al. [DMC96] proposed an algorithm called ant-cycle which mimics the ants behavior. Dorigo showed how this algorithm can be used to solve the traveling salesman problem.

In Dorigo's algorithm, \( n \) ants are distributed in the \( n \) towns, one ant at each town. Each ant chooses among the linked towns the next visited town. The probability of each town being selected is a function of the next town distance and the amount of pheromone on the link between the current and the next towns. Ants can visit each town only once (this is achieved using a tabu list [BR03]). After completing a path, each ant leaves an amount of pheromone on the visited links. The shortest path found by an ant or set of ants is usually followed by the remaining ants, defining an optimal or suboptimal path. A pheromone evaporation process is frequently used to avoid the selection of the first tours and stagnation. A generic version of the ACO algorithm may be seen in Algorithm 1.

Set parameters, initialize pheromone trails
loop
  while termination conditions not met do
    ConstructAntSolutions
    ApplyLocalSearch (optional)
    UpdatePheromones
  end while
end loop
Choose best solution

Algorithm 1 Dorigo's algorithm (ACO heuristic)

The same approach (Ant Colony Optimization) was used by Negara [G.N06] to solve the coloring graph problem (Algorithm 2). The author considered an adjacent matrix \( A \) to represent the graph \( G = (V,E) \) where: \( a_{uv} = 1 \), if the edge \((u, v) \in E\) \( a_{uv} \leq 0 \), otherwise (considering \( u, v \in V \)). Such work considers the parallel solution of the problem where agents cope among themselves sharing experiences. It also considers elitism where the good agents can modify the mutual knowledge.

5. Applying on the real world: process scheduling

The increasing availability of low cost microprocessors and the evolution of computing networks have made economically feasible the construction of sophisticated distributed systems. In such systems, processes execute on computers and communicate to each other to perform a collaborative computing task. A load balancing algorithm is frequently adopted to distribute processes among available computers.
read matrix A
for iteration do
  for ant $a$ in Ants do
    $a$ colors the graph based on previous experience using one of the specific coloring algorithms
    if $a$ is elitist then
      $a$ modifies the matrix A
    end if
  end for
end for

Use results

Algorithm 2. Negara's algorithm

A load balancing algorithm is responsible to equally distribute the processes load among the computers of a distributed environment [SKS92]. Krueger and Livny [KL87] demonstrate that such algorithms reduce the mean and standard deviation of process response times. Lower response times result in higher performance.

The load balancing algorithms involve four policies: transference, selection, location and information [SKS92]. The transference policy determines whether a computer is in a suitable state to participate in a task transfer, either as a server or as a process receiver. The selection policy defines the process to be transferred from the busiest computer to the idlest one. The location policy is responsible to find a suitable transfer partner (sender or receiver) for a computer, once the transfer policy has decided about its state. A serving computer distributes processes, when it is overloaded; a receiving computer requests processes, when it is idle. The information policy defines when and how the information regarding the computer availability is updated in the system. Several works related to load balancing can be found in the literature [ZF87, TL89, SKS92, MT04, SdMS+05].

Zhou and Ferrari [ZF87] evaluated five server-initiated load balancing algorithms, i.e. initiated by the most overloaded computer: Disted, Global, Central, Random and Lowest. In Disted, when a computer suffers any modification in its load, it emits messages to the other computers to inform its current load. In Global, one computer centralizes all the computer load information and sends broadcast messages in order to keep the other computers updated. In Central, as in Global, a central computer receives all the load information related to the system; however, it does not update the other computers with this information. This central computer decides the resources allocation in the environment. In Random, no information about the environment load is handled. Now, a computer is selected by random in order to receive a process to be initiated. In Lowest, the load information is sent when demanded. When a computer starts a process, it requests information and analyzes the loads of a small set of computers and submit the processes to the idlest one, the computer with the shortest process queue.

Theimer and Lantz [TL89] implemented algorithms similar to Central, Disted and Lowest. They analyzed these algorithms for systems composed of a larger number of computers (about 70). For the Disted and Lowest algorithms, a few process receiver and sender groups were created. The communication within these groups was handled by using a multicast protocol, in order to minimize the message exchange among the computers. Computers
with load lower than a inferior limit participate of the process receiver group, whilst those with load higher than a superior limit participate of the process sender group. Theimer and Lantz recommend decentralized algorithms, such as Lowest and Disted, as they do not generate single points of failure, as Central does. Central presents the highest performance for small and medium size networks, but its performance decreases in larger environments. They concluded that algorithms like Lowest work with the probability of a computer being idle [TL89]. They assume system homogeneity, as they use the size of the CPU waiting queue as the load index. The process behavior is not analyzed, therefore, the actual load of each computer is not measured.

Shivaratri, Krueger and Singhal [SKS92] analyzed the server-initiated, receiver-initiated, symmetrically initiated, adaptive symmetrically initiated and stable symmetrically initiated algorithms. In their studies, the length of the process waiting queue at the CPU was considered as the load index. This measure was chosen because it's simple and, therefore, can be obtained with fewer resources. They concluded that the receiver-initiated algorithms present a higher performance than the server-initiated ones. In their conclusions, the algorithm with the highest final performance was the stable symmetrically initiated. This algorithm preserves the history of the load information exchanged in the system and takes actions to transfer the processes by using this information.

Mello et al. [MTPY04] proposed a load balancing algorithm for distributing processes on heterogeneous capacity computers. This algorithm, named TLBA (Tree Load Balancing Algorithm), organizes computers in a virtual tree topology and starts delivering processes from the root to the leaves. In their experiments, this algorithm presented a very good performance, with low mean response time.

Senger et al. [SdMS05] proposed GAS, a genetic scheduling algorithm which uses information regarding the capacity of the processing elements, applications' communication and processing load, in order to allocate resources on heterogeneous and distributed environments. GAS uses Genetic Algorithms to find out the most appropriate computing resource subset to support applications.

Motivated by all the previous ACO presented works (section 1) and the scheduling problem, Nery et al. [NMdCYOB] proposed an algorithm inspired in ant colonies to schedule processes on heterogeneous capacity computer clusters, which can be considered as an alternative approach for load balancing systems. Such algorithm is named Ant Scheduler and is based in ant colony optimization techniques. The next sections describe the algorithm and compare its results with others found in literature.

5.1 The Ant Scheduler
The problem of process allocation in heterogeneous multicompuring environments can be modeled by using graphs, as illustrated in Figure 3 [AAB00]. In this case, each process is a request for execution that has the nodes $S$ and $T$ as origin and destination, respectively. $S$ and $T$ are connected by $N$ different paths, each corresponding to a computer in a cluster. This graph is employed to improve the general performance of the system by minimizing the mean congestion of the paths.

The good results obtained by ACO in graph-based problems favor the use of ACO for the optimization of process allocation on heterogeneous cluster computing environments. For such, each initiated process can be seen as an ant looking for the best path starting in the
nest in order to arrive as fast as possible to the food source. In this search, each computer can be seen as a path and the conclusion of the program execution as the food source.

![Figure 3. A cluster modeled as a graph of resources](image)

The Ant Scheduler algorithm is based on the ant-cycle proposed by Dorigo et al. [DMC96]. When the computer responsible for the distribution of processes (master) in the cluster is started, each edge in the graph has its pheromone intensity initiated with a value $\tau_i = c$. When a process is launched, it is seen as an ant able to move. Thus, this process must select one of the paths (the computers of the cluster) to its destination (complete execution). The probability of an ant choosing a path $i$ is given by Equation 1, where $\tau_i$ is the pheromone level on path $i$, $\eta_j$ is a value associated to the computer $i$ by a heuristic function, and the parameters $\alpha$ and $\beta$ control the relevance of $\tau_i$ and $\eta_j$.

$$p_i = \frac{\tau_i^\alpha \cdot \eta_j^\beta}{\sum_{j=1}^{N} \tau_j^\alpha \cdot \eta_j^\beta} \quad (1)$$

This heuristic function is proportional to the load of the $i$th computer. The denominator is the sum of the pheromone levels weighted by the heuristic function and controlled by the parameters $\alpha$ and $\beta$. When an ant arrives to its destination (when a process finishes), it deposits a $\Delta_i$ amount of pheromone in the covered path (equation 2: where $Q$ is a constant and $T$ is the time spent by the ant to arrive at its destination (the process running time)).

$$\Delta_i = \Delta_i + \frac{Q}{T} \quad (2)$$

In order to prevent an addiction to a particular computer, the paths face continuous pheromone evaporation. Thus, in regular time intervals, the amount of pheromones changes according to the rule of equation 3, where $\rho$ is a coefficient such that $(1 - \rho)$ represents the pheromone evaporation between $t$ and $t + 1$. Additionally, $\Delta$ is reseted ($\Delta = 0$) in regular time intervals.

$$\tau_i(t + 1) = \rho \cdot \tau_i(t) + \Delta_i \quad (3)$$

One problem with this approach is the possibility of a poor performance due to the different range of values for $\tau_i$ and $\eta_j$. In order to overcome this problem, these values are normalized using a logarithmic scale, modifying the equation 1 and originating the equation 4.
Artificial Ants in the Real World: Solving On-line Problems Using Ant Colony Optimization

Another problem found was the frequent allocation of a low value, between 0 and 1, to $i$, making $\log i < 0$, leading to unrealistic values for the probability function. This problem was solved by using $\log (\epsilon + i)$ instead of $\log i$, where $\epsilon = 1$. This resulted in the equation 5.

$$p_i = \frac{(\log \epsilon + \tau_i)\alpha \cdot (\log \epsilon + \eta_i)\beta}{\sum_{j=1}^{N}(\log \epsilon + \tau_j)\alpha \cdot (\log \epsilon + \eta_j)\beta}$$

The Ant Scheduler is composed of the Algorithms 3, 4 and 5. The first algorithm is executed when a new process, with possibility of migration, is initiated. When a process completes its execution, the second algorithm starts its execution. The third algorithm is periodically executed, in order to perform the pheromone evaporation.

Choose a computer with probability $p_i$ calculated using equation 5

Algorithm 3. Ant Scheduler: process started

Update the amount of pheromone $\Delta$ using equation 2

Algorithm 4. Ant Scheduler: process finished

loop
  for all $i$ such that $i$ is a cluster node do
    Update the amount of pheromone $\tau_i$ using equation 3
    Reset the amount of pheromone $\Delta$ ($\Delta = 0$)
  end for
end loop

Algorithm 5. Ant Scheduler: pheromone evaporation

5.2 Simulation

Several experiments have been carried out on environments with 32 computers for the evaluation of the Ant Scheduler algorithm behavior. The Ant Scheduler parameters used were $\alpha = 1$, $\beta = 1$, $\rho = 0.8$ and $Q = 0.1$. Parallel applications of up to 8, 64 and 128 tasks have been evaluated. This configuration allows the evaluation of the algorithm in situations where there are many tasks synchronized with others, that is, tasks that communicate among themselves to solve the same computing problem.

The workload imposed by such applications follows the workload model by Feitelson [FJ97]. This model is based on the analysis of six execution traces of the following production environments: 128-node iPSC/860 at NASA Ames; 128-node IBM SP1 at

http://www.cs.huji.ac.il/labs/parallel/workload/models.html
According to this model, the arrival of processes is derived from an exponential probability distribution function (pdf) with mean equal to 1,500 seconds. This model was adopted to simulate and allow a comparative evaluation of Ant Scheduler and other algorithms found in the literature.

In order to carry out the experiments and evaluate the scheduling algorithm proposed in this study, the authors used the model for creation of heterogeneous distributed environments and evaluation of the parallel applications response time - UniMPP (Unified Modeling for Predicting Performance) [dMS06]. The adopted model is able to generate the mean execution time of the processes submitted to the system. The mean response time is generated after reaching the confidence interval of 95%.

In this model, every processing element (PE), $PE_i$, is composed of the sextuple ${pc_i, mm_i, vm_i, dr_i, dw_i, lo_i}$, where $pc_i$ is the total computing capacity of each computer measured in instructions per time unit, $mm_i$ is the main memory total capacity, $vm_i$ is the virtual memory total capacity, $dr_i$ is the hard disk reading throughput, $dw_i$ is the hard disk writing throughput, and $lo_i$ is the time between sending and receiving a message.

In this model, every process is represented by the sextuple ${mp_j, sm_j, pdf_{dm_j}, pdf_{dr_j}, pdf_{dw_j}, pdf_{net_j}}$, where $mp_j$ represents the processing consumption, $sm_j$ is the amount of static memory allocated by the process, $pdf_{dm_j}$ is the probability distribution for the memory dynamic occupation, $pdf_{dr_j}$ is the probability distribution for file reading, $pdf_{dw_j}$ is the probability distribution for file writing, and $pdf_{net_j}$ is the probability distribution for messages sending and receiving.

In order to evaluate the Ant Scheduler algorithm, a class was included in the object-oriented simulator $^3$ [dMS06]. This class implements the functionalities of Ant Scheduler and has been aggregated to the UniMPP model simulator to generate the mean response times of an application execution. These results were used to evaluate the performance of Ant Scheduler and to allow comparisons with other algorithms.

### 5.2.1 Environment parameters

Experiments were conducted in environments composed of 32 computers. In these experiments, each $PE_i$ for the UniMPP model was probabilistically defined. The parameters $pc_i, mm_i, vm_i, dr_i, dw_i$ were set by using an uniform probability distribution function with the mean of 1,500 Mips (millions of instructions per second), 1,024 MBytes (main memory), 1,024 MBytes (virtual memory), 40 MBytes (file reading transference rate from hard disk) and 30 MBytes (file writing transference rate to hard disk). These measures were based on the actual values obtained using a group of machines from our research laboratory (Distributed Systems and Concurrent Programming Laboratory). These measures followed the benchmark proposed by Mello and Senger [dMS06]. These parameter values and the use of probability distributions allow the creation of heterogeneous environments to evaluate the Ant Scheduler algorithm.

The Feitelson's workload model was used to define the occupation parameter (in Mips) of the processes (or tasks) that take part of the parallel application. The remaining parameters

---

4 available at http://www.icmc.usp.br/~mello/
required for the UniMPP to represent a process were defined as: \(sm_j\), the amount of static memory used by the process, based on an exponential distribution with a mean of 300 KBytes; \(pdf \, dm_j\), the amount of memory dynamically allocated, defined by an exponential distribution with a mean of 1,000 KBytes; \(pdf \, dr_j\), the file reading probability, defined by an exponential distribution with a mean of one read at each 1,000 clock ticks, same value used to parameterize the writing in files (\(pdf \, dw_j\)); \(pdf \, net_j\), the receiving and sending of network messages, parameterized by an exponential distribution with a mean of one message at each 1,000 clock ticks.

During the experiments, all computers were located at the same network, as an ordinary cluster. Within the network, the computers present a delay (RTT - Round-Trip Time according to the model by Hockey [Hoc96]) of 0.0001 seconds (mean value extracted by the net benchmark by Mello and Senger [dMS06] for a Gigabit Ethernet network).

### 5.2.2 Algorithms simulated

The performance of Ant Scheduler is compared with 5 other scheduling and load balancing algorithms proposed in literature: DPWP [ASSS99], Random, Central, Lowest [ZF87], TLBA [MTPY04] and GAS [SdMS+05].

The DPWP (Dynamic Policy Without Preemption) algorithm performs the parallel applications scheduling taking into account a distributed and heterogeneous execution scenario [ASSS99]. This algorithm allows the scheduling of the applications developed on PVM, MPI and CORBA. The details involved in the task attributions are supervised by the scheduling software, AMIGO [SouOO]. The load index used in this algorithm is the queue size of each PE (processing element). Through this index, the load of a PE is based on the ratio between its number of tasks being executed and its processing capacity. The processing capacity is measured by specific benchmarks [SouOO, SSSSO]. The DPWP scheduling algorithm uses load indexes to create an ordered list of PEs. The parallel application tasks are attributed to the PEs of this list, in a circular structure.

The Lowest, Central and Random algorithms were investigated for load balancing in [ZF87]. These algorithms are defined by two main components: LIM (Load information manager) and LBM (Load balance manager). The first component is responsible for the information policy and for monitoring the computers' load in order to calculate the load indexes. The latter defines how to use the collected information to find out the most appropriate computer to schedule processes. The approach followed by these components to perform their tasks allows the definition of distinct algorithms. These algorithms differ from the scheduling algorithms by being designed to perform the load balance, thus there is no global scheduling software to which the applications are submitted. In fact, each PE should locally manage the application tasks that reach it, initiating them locally or defining how another PE will be selected to execute tasks.

\footnote{We have compared our results to this work, because it was also developed in our laboratory.}
The **Lowest** algorithm aims to achieve the load balance by minimizing the number of messages exchanged among its components. When a task is submitted to the environment, the LIM receiving the request defines a limited set of remote LIMs. The loads of the PEs of this set are received and the idllest PE is selected to receive the task. The **Central** algorithm employs a master LBM and a master LIM. Both of them centralize the decision making related to the load balance. The master LIM receives the load indexes sent by the slave LIMs. The master LBM receives the requests to allocate processes to the system and uses the information provided by the master LIM to make these allocations. The **Random** algorithm does not use information regarding the system load to make decisions. When a task is submitted to the execution environment, the algorithm randomly selects a PE. The load index used by the **Lowest** and **Central** algorithms is calculated based on the number of processes in the execution queue. Zhou and Ferrari [ZF87] have observed that the **Lowest** and **Central** algorithms present similar performance and that the **Random** algorithms present the worst results of all. They also suggested the **Lowest** algorithm for distributed scenarios, because it's not centralized.

The **TLBA** (Tree Load Balancing Algorithm) algorithm aims at balancing loads in scalable heterogeneous distributed systems [MTPY04]. This algorithm creates a logical interconnection topology with all PEs, in a tree format, and performs the migration of tasks in order to improve the system load balance. The **GAS** (Genetic Algorithm for Scheduling) algorithm uses Genetic Algorithms to propose optimized scheduling solutions [SdMS+05]. The algorithm considers knowledge about the execution time and applications’ behavior to define the most adequate set of computing resources to support a parallel application on a distributed environment composed of heterogeneous capacity computers. **GAS** uses the crossover and mutation operators to optimize the probabilistic search for the best solution for a problem. Based on Genetics and Evolution, Genetic Algorithms are very suitable for global search and can be efficiently implemented in parallel machines.

### 5.2.3 Experimental results

For the validation of the Ant Scheduler algorithm, its performance was compared with results obtained by the five algorithms previously described. For such, the authors carried out simulations where all these algorithms were evaluated running parallel applications composed of different numbers of tasks. Figures 4 and 5 show the process mean response times for parallel applications with up to 64 and 128 tasks, respectively.

Random had the worst results, while Ant Scheduler presented the best performance. The poor performance obtained by GAS can be explained by the fact that its execution time increases according to the number of computers. This occurs due to the use of larger chromosomes (this approach is based on Genetic Algorithms), which have to be evaluated by the fitness function. This evaluation requires a long time, which is added to the scheduling cost, jeopardizing the process execution time. It is hard to observe the curve for Ant Scheduler in Figure 4 due to the small mean response times in comparison with the other algorithms. These results show that, in all the scenarios investigated, the Ant Scheduler presented the best performance.
5.3 Implementation

In order to allow real experiments, Ant Scheduler was implemented using the Linux kernel 2.4.24. This implementation uses the process migration service of the openMosix\(^\text{6}\) patch. openMosix is a software designed to balance the load of clusters by distributing processes.

---

\(^{6}\) openMosix is a Linux kernel patch developed by Moshe Bar which allows automatic process migration in a cluster environment — Available at http://openmosix.sourceforge.net/
The implementation was performed by adding a set of traps inside the Linux kernel. The first trap was implemented in the system call do_fork. Whenever a new process is started, do_fork is called. This system call executes the first trap of Ant Scheduler, which chooses the node where the new process will run. This phase is based on the pheromone level and the computing capacity of each node. Similarly, when a process finishes, the system call do_exit is made. This system call executes the second trap of Ant Scheduler, which updates the amount of pheromone in the computer (ant’s path) where the process was running. These traps were implemented in a kernel module by using function pointers, allowing simple changes to use another process scheduling algorithm. When the module is loaded, it registers its functions (traps). This module also starts a thread that periodically updates the pheromone level of each computer applying the equation 3. Experiments were carried out to evaluate the process execution time for an environment using Ant Scheduler and openMosix on a set of five Dual Xeon 2.4 Ghz computers. Table 1 presents the results in process mean execution time (in seconds) for a load of 10 low-load, 10 mean-load and 10 high-load applications executing simultaneously. According to these results, the use of Ant Scheduler reduced the mean response time.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>without Ant Scheduler</th>
<th>with Ant Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>351.00</td>
<td>327.00</td>
</tr>
<tr>
<td>2</td>
<td>351.00</td>
<td>336.00</td>
</tr>
<tr>
<td>3</td>
<td>351.00</td>
<td>318.00</td>
</tr>
<tr>
<td>4</td>
<td>354.00</td>
<td>321.00</td>
</tr>
<tr>
<td>5</td>
<td>351.00</td>
<td>318.00</td>
</tr>
<tr>
<td>6</td>
<td>351.00</td>
<td>333.00</td>
</tr>
<tr>
<td>7</td>
<td>351.00</td>
<td>321.00</td>
</tr>
<tr>
<td>8</td>
<td>351.00</td>
<td>336.00</td>
</tr>
<tr>
<td>9</td>
<td>348.00</td>
<td>309.00</td>
</tr>
<tr>
<td>10</td>
<td>348.00</td>
<td>315.00</td>
</tr>
<tr>
<td>Mean</td>
<td>350.70</td>
<td>323.40</td>
</tr>
<tr>
<td>Std Dev</td>
<td>1.615</td>
<td>8.777</td>
</tr>
</tbody>
</table>

Table 1. Experimental Results

In order to evaluate the statistical significance of the results obtained, the authors applied the Student’s t-test. In this analysis, the authors used the standard error $s_x$ for small data samples [W.C88], given by equation 6, where $s$ is the standard deviation and $n$ is the number of samples. Applying the equation, the standard errors of 0.51 and 2.775 were obtained without Ant Scheduler and with Ant Scheduler, respectively.

$$s_x = \frac{s}{\sqrt{n}}$$

(6)

In the test, the null hypothesis proposed is $H_0$: $\mu_{w/o} = \mu_{with}$, with the alternative hypothesis $H_A$: $\mu_{w/o} < \mu_{with}$ to evaluate whether the results are statistically equivalent. The hypothesis
$H_0$ considers the results of the Ant Scheduler and the standard openMosix to be similar. If the test is rejected, the alternative hypothesis $H_A$ is accepted. This hypothesis considers the process mean response time for the environment adopted. Such mean response time is lower when the processes are distributed using the Ant Scheduler, what confirms its superiority.

The significance level used for one-tailed test is $\alpha = 0.0005$. $\mu_{\text{without}}$ is the process mean response time with Ant Scheduler; $\mu_{\text{with}}$ is the process mean response time with the standard openMosix. For the adopted significance level $\alpha$ the data sets have to present a difference of at least 4.781 in the t-test to reject the hypothesis. This value is found in tables of critical values for the $t$-student distribution.

Applying the equation 7, the value 9.83 is found, confirming that the results present statistic differences with $p < 0.005$, rejecting the hypothesis $H_0$. In this way the hypothesis $H_A$ is valid and the system with Ant Scheduler presents better results than standard openMosix.

$$t = \frac{\mu_{\text{without}} - \mu_{\text{with}}}{s_e}$$ (7)

By applying statistic tools over the data sets, it's possible to find the most precise $\alpha = 0.0000018$ for a one-tailed test. This value shows how many times the alternative hypothesis is true. In this case, $H_1$ can be considered true in 9,999,982 out of 10,000,000 executions, showing that Ant Scheduler reduces the response time. Only in 18 of these executions the results of Ant Scheduler and openMosix would not present significant statistical differences.

6. Conclusions and future directions

The behavior of real ants motivated Dorigo et al. [DMC96] to propose the Ant Colony Optimization (ACO) technique, which can be used to solve problems in dynamic environments. This technique has been successfully applied to several optimization problems [FMS05, PB05, BN06, SF06, PLF02, WGDK06, CF06, HND05]. Such results have motivated this chapter which presents ACO concepts, case studies and also a complete example on process scheduling optimization.

Besides the successful adoption of ACO, it presents some relevant questions which have been motivating future directions such as: how to adjust parameters which depend on the optimization problem [SoCOSj; how to reduce the execution time [G.N06, MBSD06]; the optimization improvement by using incremental local search [BBSD06]; and the aggregation of different and new concepts to ACO [RL04]. Those works confirm ACO is an important optimization technique and also that it has been improved and present a promising future.

7. Acknowledgments

We would like thank Ines Tristao Silva for the ant figure composition. This paper is based upon work supported by CAPES - Brazil under grant no. 032506-6 and FAPESP - Brazil.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the CAPES or FAPESP.

---

7 The authors applied the Gnumeric tool in this analysis — a free software tool licensed under GNU/GPL terms.
8. References

Yair Amir, Baruch Awerbuch, R. Sean Borgstrom, Amnon Barak, and Arie Keren. An opportunity cost approach for job assignment and reassignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, 11:760-768, October 2000. [AAB+00]


Artificial Ants in the Real World: Solving On-line Problems Using Ant Colony Optimization


In the era globalization the emerging technologies are governing engineering industries to a multifaceted state. The escalating complexity has demanded researchers to find the possible ways of easing the solution of the problems. This has motivated the researchers to grasp ideas from the nature and implant it in the engineering sciences. This way of thinking led to emergence of many biologically inspired algorithms that have proven to be efficient in handling the computationally complex problems with competence such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), etc. Motivated by the capability of the biologically inspired algorithms the present book on “Swarm Intelligence: Focus on Ant and Particle Swarm Optimization” aims to present recent developments and applications concerning optimization with swarm intelligence techniques. The papers selected for this book comprise a cross-section of topics that reflect a variety of perspectives and disciplinary backgrounds. In addition to the introduction of new concepts of swarm intelligence, this book also presented some selected representative case studies covering power plant maintenance scheduling; geotechnical engineering; design and machining tolerances; layout problems; manufacturing process plan; job-shop scheduling; structural design; environmental dispatching problems; wireless communication; water distribution systems; multi-plant supply chain; fault diagnosis of airplane engines; and process scheduling. I believe these 27 chapters presented in this book adequately reflect these topics.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:
