ADAPTIVE ANT-COLONY ALGORITHM FOR SEMANTIC QUERY ROUTING

Claudia Gómez Santillán, Laura Cruz Reyes, Elisa Schaeffer, Eustorgio Meza, Gilberto Rivera Zarate

Abstract:

The most prevalent P2P application today is file sharing, both among scientific users and the general public. A fundamental process in file sharing systems is the search mechanism. The unstructured nature of real-world large-scale complex systems poses a challenge to the search methods, because global routing and directory services are impractical to implement. This paper presents a new ant-colony algorithm, Adaptive Neighboring-Ant Search (AdaNAS), for the semantic query routing problem (SQRP) in a P2P network. The proposed algorithm incorporates an adaptive control parameter tuning technique for runtime estimation of the time-to-live (TTL) of the ants. AdaNAS uses three strategies that take advantage of the local environment: learning, characterization, and exploration. Two classical learning rules are used to gain experience on past performance using three new learning functions based on the distance traveled and the resources found by the ants. The experimental results show that the AdaNAS algorithm outperforms the NAS algorithm where the TTL value is not tuned at runtime.

Keywords: *parameter tuning, search algorithm, peer-to-peer, adaptive algorithm, local environment, ant-colony algorithms.*

1. Introduction

Although popular for other uses, the World Wide Web is impractical for user-to-user file sharing as it requires centralized infrastructure such as an HTTP server. In the past decade, a new class of networks called *peer-to-peer* (P2P) systems began to spread as a solution to the increasing demand of file sharing among Internet users. In P2P networks, the users interconnect to offer their files to one another [1]. The participants, called *peers*, may connect and disconnect freely, and do so constantly, which triggers frequent changes in the network structure [2].

One of the main advantages is that peers are equal in terms of functionality and tasks which are developed. This produces high fault tolerance and auto-organization: peers form unstructured networks with an acceptable connectivity and performance. The *Semantic Query Routing Problem* (SQRP) consists in deciding, based on a set of keywords, to which neighbor to forward the query to search files related with the keywords [2], [3].

The lack of global structure caused that flooding-based search mechanisms have been mainly employed. Flooding-based mechanisms are simple, but unfortunately generate vast amounts of traffic in the network and may produce congestion on Internet. Existing approaches for SQRP in P2P networks range from simple broadcasting techniques to sophisticated methods [1], [4], [5]. The latter includes proposals based on *ant-colony systems* [6] that are specifically suited for handling routing tables in telecommunications. There exist few algorithms used for SQRP, including SemAnt [3] and Neighboring-Ant Search (NAS) [7], the latter based on the former. In this work we propose an algorithm as an extension to NAS, called the Adaptive Neighboring-Ant Search (AdaNAS). AdaNAS is hybridized with three local strategies: learning, structural characterization and exploration. These strategies are aimed to produce a greater amount of results in a lesser amount of time. The time-to-live (TTL) parameter is tuned at runtime based on the information acquired by these three local strategies.

2. Background

In order to place the research in context, this section is divided in four parts. The first part models a P2P network with graph theory, and in the second part we continue with structural characterization. The third part describes the basic ant-colony algorithms for SQRP algorithms and the last part explains parameter tuning and adaptation.

2.1. Graph Theory

A P2P network is a distributed system that can be modeled mathematically as a *graph*, $G = (V,E)$, where V is a set of *nodes* and $E \subseteq V \times V$ is a set of (symmetrical) *connections*. For more information on graph theory, we recommend the textbook by Diestel [8]. Each peer in the network is represented by a node (also called a *vertex*) of the graph. The direct communications among the peers are represented by the connections (also called the *edges*) of the graph. We denote by n the number of nodes in the system and identify the nodes by integers, $V = 1, 2, 3, ..., n$. Two nodes that are connected are called *neighbors*; the set of all neighbors of a node i is denoted by $\Gamma(i)$. The number of neighbors of a node i is called degree and is denoted by k_i. Two nodes i and j are said to be connected if there exists at least one sequence of connections that begins at i, traverses from node to node through the connections of the graph, and ends at j. Such sequences of connections are called routes or paths and the number of connections traversed is the length of the route.

2.2. Structural Characterization using Degree Distribution

For the purpose of analyzing the structure and behavior of complex systems modeled as graphs, numerous characterization functions have been proposed [9]. There are two main types of these functions: those based on global information that require information on the entire graph simultaneously and those based on local information that only access the information of a certain node *i* and its neighborhood Γ(*i*) at a time.

The degree k_i of a node *i* is a local measure of network. *P*(*k*) denotes the number of nodes that have degree *k*, normalized by *n*. The measure of *P*(*k*) can be interpreted as the probability that a randomly chosen node *i* has degree *k*. The values of *P*(*k*) for $k \in [0, n-1]$ (supposing that there can only be at most one connection between each pair of distinct nodes) form the *degree distribution* of the graph. Whereas the degrees themselves are local properties, obtaining the degree distribution is a global computation.

The degree distribution is widely used to classify networks according to the *generation models* that produce such distributions. Among the first and most famous generation models are the *uniform random graphs* of Erdös and Rényi [10], and Gilbert [11] that yield a binomial distribution that at the limit, approaches the Poisson distribution and most of the nodes in the graph have similar degrees [12].

In the past decade, another type of generation models became popular as various studies revealed that the degree distribution of some important real-world networks (including the WWW, the Internet, biological and social systems) was not Poisson distribution at all, but rather a power-law distribution [13], [14], [15], $P(k) \sim k^{-\gamma}$ with values of γ typically ranging between two and three. The models that produce such distributions are called *scale-free* network models. The notable structural property in networks with power law distribution is the presence of a small set of extremely well-connected nodes that are called *hubs*, whereas a great majority of the nodes has a very low degree [16], [17]. This property translates into high fault tolerance under random flaws, but high vulnerability under deliberate attack [14].

2.3. Parameter Tuning and Adaptation

Metaheuristics offer solutions that are often close to the optimum, but with a reasonable amount of resources used when compared to an exact algorithm. Unfortunately, the metaheuristics are usually rich in parameters. The choice of the values for the parameters is nontrivial and in many cases the parameters should vary during the runtime of the algorithm [18], [19].

The process of selecting the parameter values is known as tuning. The goal of offline tuning is to provide a static initial parameter configuration to be used throughout the execution of the algorithm, whereas online tuning, also known as *parameter control* or *adaptation*, is the process of adjusting the parameter values at runtime. We design a discrete model for adaptation based on the proposed by Holland in 1992 [20]. We assume that the system takes actions at discrete steps *t* = 1, 2, 3,..., as this assumption applies to practically all computational System. The proposed model is described in section four..

3. SQRP Search Strategies

In this section we present the problem focused in this work. First, we describe the semantic query routing problem (SQRP) as a search process. Then, strategies for solve SQRP are shown including our proposed algorithm which uses an adaptive strategy for adjusting an important parameter for the search process: TTL.

3.1. SQRP Description

SQRP is the problem of locating information in a network based on a query formed by keywords. The goal in SQRP is to determine shorter routes from a node that issues a query to those nodes of the network that can appropriately answer the query by providing the requested information. Each query traverses the network, moving from the initiating node to a neighboring node and then to a neighbor of a neighbor and so forth, until it locates the requested resource or gives up in its absence. Due to the complexity of the problem [2], [3], [5], [21], [22], [23], solutions proposed to SQRP typically limit to special cases.

The general strategies of SQRP algorithms are the following. Each node maintains a local database of documents r_i called the *repository*. The search mechanism is based on nodes sending messages to the neighboring nodes to query the contents of their repositories. The *queries* q_i are messages that contain keywords that describe searched resource for possible matches. If this examination produces results to the query, the node responds by creating another message informing the node that launched the query of the resources available in the responding node. If there are no results or there are too few results, the node that received the query forwards it to one or more of its neighbors. This process is repeated until some predefined stopping criteria is reached. An important observation is that in a P2P network the connection pattern varies among the net (*heterogeneous topology*), moreover the connections may change in time, and this may alter the routes available for messages to take.

3.2. SQRP Algorithms

The most popular technique for searching in P2P systems is flooding, where each message is assigned a positive integer parameter known as the *time-to-live* (TTL) of the message. As the message propagates from one node to another, the value of TTL is decreased by one by each forwarding node. When TTL reaches zero, the message will be discarded and no longer propagated in the system. The main disadvantage of flooding is the rapid congestion of the communication channels [24]. Another widely used search strategy is the *random walk* [21]. A random walk in a graph is a route where the node following the initiating node is chosen uniformly at random among its neighbors.

3.2.1. AntSearch

Wu *et al.* [23] propose an algorithm called *AntSearch*. The main idea in the AntSearch algorithm is using pheromone values to identify the free-riders, prevent sending messages to those peers in order to reduce the amount of redundant messages. The estimation of a proper TTL value for a query flooding is based on the popularity of the resources. Wu *et al.* use three metrics to measure the performance of the AntSearch. One is the *number of searched files* for a query with a required number of results, *R*: a good search algorithm should retrieve the number of results over but close to *R*. The second one is the *cost per result* that defines the total amount of query messages divided by the number of searched results; this metric measure how many average query messages are generated to gain a result. Finally, search latency is defined as the total time taken by the algorithm.

3.2.2. SemAnt

Algorithms that incorporate information on past search performance include the SemAnt algorithm [3], [25] that uses an ant-colony system to solve SQRP in a P2P environment. SemAnt seeks to optimize the response to a certain query according to the popularity of the keywords used in the query. The algorithm takes into account network parameters such as bandwidth and latency. In SemAnt, the queries are the ants that operate in parallel and place pheromone on successful search routes. This pheromone evaporates over time to gradually eliminate old or obsolete information. Also Michlmayr [3] considers parameter tuning for the SemAnt algorithm, including manual adjustment of the TTL parameter from a set of possible values 15, 20, 25, 30, 35 and concludes that 25 is the best value for the parameter. The adjustment of the TTL is made without simultaneous tuning of the other parameters.

3.2.3. Neighboring-Ant Search

NAS [7] is also an ant-colony system, but incorporates a local structural measure to guide the ants towards nodes that have better connectivity. The algorithm has three main phases: an evaluation phase that examines the local repository and incorporates the classical lookahead technique [4], a transition phase in which the query propagates in the network until its TTL is reached, and a retrieval phase in which the pheromone tables are updated.

Most relevant aspects of former works have been incorporated into the proposed NAS algorithm. The framework of AntNet algorithm is modified to correspond to the problem conditions: in AntNet the final addresses are known, while NAS algorithm does not has a priori knowledge of where the resources are located. On the other hand, differently to AntSearch, the SemAnt algorithm and NAS are focused on the same problem conditions, and both use algorithms based on AntNet algorithm.

However, the difference between the SemAnt and NAS is that SemAnt only learns from past experience, whereas NAS takes advantage of the local environment. This means that the search in NAS takes place in terms of the classic local exploration method of Lookahead [4], the local structural metric DDC[26] which measures the differences between the degree of a node and the degree of its neighbors, and three local functions of the past algorithm performance. This algorithm outperforms methods proposed in the literature, such as Random-Walk and SemAnt [7].

3.2.4. Adaptative Neighboring-Ant Search

The proposed algorithm in this work, *Adaptive Neighboring Ant Search* (AdaNAS) is largely based on the NAS algorithm, but includes the adaptation of the TTL parameter at runtime, in addition to other changes. The mechanism that may extend the TTL for an ant is called the *survival rule*. It incorporates information on past queries relying on the learning strategies included in AdaNAS, basic characteristics of SQRP and a set of parameters that are adjusted according to the results found when using the *survival rule* itself. The rule evaluates the length of the shortest known route that begins with the connection (i, j) from the current node i to a node that contains good results for the query t. The form in which the algorithm operates is explained in detail later in Sections 4 and 5.

4. AdaNAS Model

In this section, we present a multi-agent model in order to describe the adaptive behavior of AdaNAS.

4.1. The General Model

The environment is the P2P network, in which two stimuli or inputs are observed:

- I_1: the occurrences of the documents being searched,
- I_2: the degree k_i of the node i.

The environment has the following order to send stimuli: observing I_1 has a higher priority than observing I_2. AdaNAS is an ant-colony system, where each ant is modeled as an agent. AdaNAS has four agent types:

- The *query ant* is accountable for attending the users' queries and creating the *Forward Ant*; moreover it updates the pheromone table by means of evaporation. There is a *query ant* for each node in the net and it stays there while the algorithm is running.
- The *Forward Ant* uses the learning strategies for steering the query and when it finds resources creates the backward ant. It finishes the routing process when its TTL is zero or the amount of found resources is enough that is denoted by R then, it creates an *update ant*.
- The *backward ant* is responsible for informing to *query ant* the amount of resources in a node found by the *Forward Ant*. In addition, it updates the values of some learning structures that are the bases of the *survival rule* which will be explaining later (Section 4.2.3).
- The *update ant* drops pheromone on the nodes of the path generated by the *Forward Ant*. The amount of pheromone deposited depends on quantity of found resources (*hits*) and number of edges traveled (*hops*) by the *Forward Ant*.

The system is subdivided into four parts: the structures A to adapt to the environment (called *agents*), the adaptation plan P, the memory M, and the operators O. Typically, A has various alternative states A_1, A_2, A_3,... among which one is to be chosen for the system, according to the observations made on the environment. On the other hand, P is typically a set of rules, one or more which can be applied. These rules apply the operations in the set O. An operator is either a deterministic function, denoted as $(A_i, P_j) \rightarrow A_k$, or a stochastic function to a probability distribution over a set of states for selecting A_k. The memory M permits the system to collect information on the condition of the environment and the system itself, to use it as a base for the decision making. The observations of the environment are taken as stimuli that trigger the operators.

The routing process implemented in the *Forward Ant* is required to be adaptive, thus A is defined in function of this agent. The possible states for A are five:

A_1: No route has been assigned and the *Forward Ant* is at the initial node. The ant can be only activated when the *query ant* send it a query and can only receive once time each stimulus.
A_2: A route has been assigned and TTL has not reached zero.
A_3: TTL is zero.
A_4: *Forward Ant* used the survival rule to extend TTL.
$A_5 = X$: Terminal state is reached by the *Forward Ant*.

The Figure 1 shows the AdaNAS adaptive model. According to the stimuli -the number of documents found (dotted line, I_1) and degree of the node (solid line, I_2) - an operator is selected. The line style for state transitions follows that of the stimuli: dotted line for transitions proceeding from I_1 and solid for I_2.

The memory M is basically composed of four structures that store information about previous queries. The first of these structures is the three dimensional pheromone table τ. The element $\tau_{i,j,t}$ is the preference for moving from node i to a neighboring node j when searching by a keyword t. In this work, we assume that each query contains one keyword and the total number of keywords (or *concepts*) known to the system is denoted by C.

The pheromone table $M_1 = \tau$ is split into n bi-dimensional tables, τj, t, one for each node. These tables only contain the entries $\tau_{j,t}$ for a fixed node i and hence have at most dimensions $C \times |\Gamma(i)|$. The other three structures are also three-dimensional tables $M_2 = D$, $M_3 = N$ and $M_4 = H$, each splits into n local bi-dimensional tables in the same manner. The information in these structures is of the following kind: currently being at node i and searching for t, there is a route of distance $D_{i,j,t}$ starting at the neighbor j that leads to a node identified in $N_{i,j,t}$ that contains $H_{i,j,t}$ hits or matching documents.

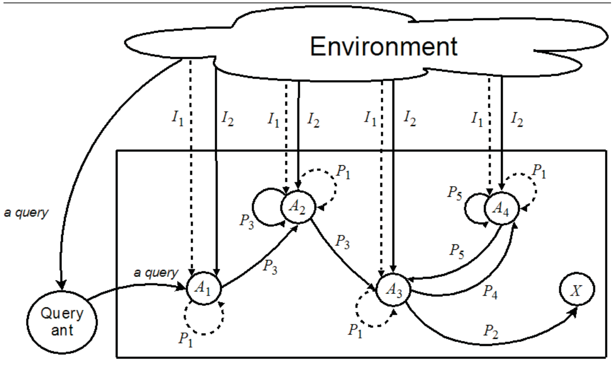

Fig. 1. AdaNAS Adaptive Model General.

The adaptive plans P are the following:

P_1: **The backward ant.** Created when a resource is found; modifies M_2, M_3 and M_4.
P_2: **The update ant.** Modifies the pheromone table $M_1 = \tau$ when the *Forward Ant* reached 0 and *survival rule* can not to proceed.
P_3: **The transition rule.** Selects the next node applying the inherent learning stored in pheromone trails and in the memory structure $M_2 = D$.
P_4: **The survival rule.** Proceeds when the learning stored in M_2, M_3 and M_4 permits to extend TTL and determines how much TTL must be extended.
P_5: **The modified transition rule.** A variation of transition rule that eliminates the pheromone and degree effects.

The operators O of AdaNAS are the following:

O_1: $(A_1, I_1) - P_1 \rightarrow A_1$:
The ant has been just created and documents were found from the initial node (I_1), so *backward ant* updates the short-time memory (P_1) and no change in the agent state of the system is produced.
O_2: $(A_1, I_2) - P_3 \rightarrow A_2$:
The ant has been just created and must select a neighbor node (I_2) according with the transition rule (P_3), this will produce a tracked route by the ant (A_2).
O_3: $(A_2, I_1) - P_1 \rightarrow A_2$:
The ant has assigned a route (A_2) and documents were found from initial node (I_1), so *backward ant* updates the short-time memory (P_1) and no change in the agent state of the system is produced.
O_4: $(A_2, I_2) - P_3 \rightarrow A_n | A_n \in \{A_2, A_3\}$:
When the ant has a partial route (A_2), it must select the next node from the neighborhood (I_2), so it applies the transition rule (P_3). The application of the transition rule can cause than TTL is over (A_3) or not (A_2).
O_5: $(A_3, I_1) - P_1 \rightarrow A_3$
Idem O_3, but now the ant has TTL = 0 (A_3).
O_6: $(A_3, I_2) - P_4 \rightarrow A_4$
The ant is over TTL (A_3) and with neighborhood information (I_2), so it applies the survival rule (P_4). When P_4 is applied the ant performs its activity in an extended time to live (A_4).
O_7: $(A_3, I_2) - P_2 \rightarrow \mathrm{X}$
The ant is over TTL (A_3) and with neighborhood information (I_2), so it decides that the path must end. In order to reach its final state (X), the *forward ant* must create an *updating ant* which performs the pheromone update (P_2).
O_8: $(A_4, I_1) - P_1 \rightarrow A_4$
Idem O_3, but now the ant performs it activity in an extended time to live (A_4).
O_9: $(A_4, I_2) - P_5 \rightarrow A_1 | A_1 \in \{A_3, A_4\}$
In an extended TTL by the modified transition rule (P_5), the ant must choose a vertex from the neighborhood (I_2) like the next node in the route. The application of transition rule can cause than TTL is over (A_3) or not (A_2) again.

The general model is illustrated in Figure 1 where can be observed the transitions among states of the *Forward Ant*.

4.2. Behavior Rule

An ant-colony algorithm has rules that determine its behavior. These rules define why the ants construct and evaluate the solution and why the pheromone is updated and used. Although the pheromone is the main learning structure, AdaNAS has three more: D, N and H, for know the distances toward the nodes that contain in its repository matching documents. AdaNAS own several behavior rules: the *transition rule*, the *update rules*, the *survival* *rule* and the modified *transition rule*.

4.2.1. Transition Rule

The transition rule P_3 considers two structures to determine the next state: τ and *D*. This transition rule is used by an ant *x* that is searching the keyword *t* and is located in the node *r*. The rule is formulates in the following Equation 1:

$$l(x,r,t) = \begin{cases} \text{argmax}_{i\in(\Gamma(r)/\Lambda x)} \{\psi(r,i,t)\}, \text{si} \quad p < q \\ L(x,r,t) \quad \text{otherwise;} \end{cases} \tag{1}$$

where *p* is a pseudo-random number, *q* is a algorithm parameter that defines the probability of using of the exploitation technique, Γ(*r*) is the set of neighbors nodes of *r*, Λ*x* is the set of nodes previously visited by *x*, and Equation 2, defined by:

$$\psi(r,i,t) = (w_d \cdot \kappa(r,i) + w_i \cdot (D_{r,i,t})^{-1})^{\beta_1} \cdot (\tau_{r,i,t})^{\beta_2}, \tag{2}$$

where w_d is the parameter that defines the degree importance, w_i defines the distance importance toward the nearest node with matching documents ($D_{r,i,t}$), β_1 intensifies the local metrics contribution (degree and distance), β_2 intensifies pheromone contribution ($\tau_{r,i,t}$), κ(*r; i*) is a normalized degree measure expressed in Equation 3:

$$\kappa(r,i) = \frac{k_i}{\max_{j\in\Gamma(r)}\{k_j\}}, \tag{3}$$

and *L* is the exploration technique expressed, in Equation 4:

$$L(x,r,t) = f(\{p_{x,r,i,t} \mid i \in \Gamma(r)\}), \tag{4}$$

where $f(\{p_{x,r,i,t} | i \in \Gamma(r)\})$ is a roulette-wheel random selection function that chooses a node i depending on its probability $p_{x,r,i,t}$ which indicates the probability of the ant *x* for moving from *r* to *i* searching by keyword *t* and it is defined in Equation 5:

$$P_{x,r,i,t} = \frac{\psi(r,i,t)}{\sum_{i\in(\Gamma(r)/\Lambda x)} \psi(r,i,t)} \tag{5}$$

The tables *D* and τ were described in the previous section. The exploration strategy *L* is activated when $p \geq q$ and stimulates the ants to search for new paths. In case that $p < q$, the exploitation strategy is selected: it prefers nodes that provide a greater amount of pheromone and better connectivity with smaller numbers of hops toward a resource. As is shown in the transition rule, β_2 is the intensifier of the pheromone trail, and β_1 is the intensifier of the local metrics, this means that the algorithm will be only steered by the local metrics when $\beta_2 = 0$, or by the pheromone when $\beta_1 = 0$. In this work the initial values are $\beta_1 = 2$ and $\beta_2 = 1$.

4.2.2. Update Rules

There are two basic update rules in an ant colony algorithm: the evaporation and increment of pheromone. The evaporation method of AdaNAS is based on the technique used in SemAnt [3], while the increment strategy is based on the proposed in NAS [7]. Both update rules are described below.

Pheromone Evaporation Rule, the pheromone evaporation is a strategy whose finality is avoid that the edges can take very big values of pheromone trail causing a greedy behavior on the algorithm. Each unit time the query ant makes smaller the pheromone trail of the node where the query ant is, by multiplying the trail by the evaporation rate ρ, which is a number between zero and one. To avoid very low values in the pheromone the rule incorporates a second term consisting of the product $\rho\tau_0$, where τ_0 is the initial pheromone value. The Equation 6 expresses mathematically the evaporation pheromone rule.

$$\tau_{r,s,t} \leftarrow (1-\rho)\bullet\tau_{r,s,t} + \rho\bullet\tau_0 \tag{6}$$

Pheromone Increment Rule, when a *Forward Ant* finishes, it must express its performance in terms of pheromone by means of an *update ant* whose function is to increase the quantity of pheromone depending on amount of documents found and edges traversed by *Forward Ant*. This is done each time that an *update ant* passes on one node. The Equations 7 and 8 describe the *pheromone increment rule*.

$$\tau_{r,s,t} \leftarrow \tau_{r,s,t} + \Delta\tau_{r,s,t}(x) \tag{7}$$

where $\tau_{r,s,t}$ is the preference of going to *s* when the Forward Ant is in *r* and is searching by keyword *t*, $\Delta\tau_{r,s,t}(x)$ is the amount of pheromone dropped on $\tau_{r,s,t}$ by a *backward ant* generated by the *Forward Ant x* and can be expressed like:

$$\Delta\tau_{r,s,t}(x) \leftarrow \left[w_h \frac{hits(x,s)}{\text{R}} + (1-w_h)\frac{1}{hops(x,r)} \right] \tag{8}$$

where *hits*(*x, s*) is the amount of documents found by the *Forward Ant x* from *s* to end of its path, and *hops*(*x, r*) is the length of the trajectory traversed by the *Forward Ant x* from *r* to the final node in its route passing by *s*.

4.2.3. Survival Rules

P_1 (the backward ant) updates the memory structures $M_2 = D$, $M_3 = N$, and $M_4 = H$. These structures are used in the survival rule (P_4) to increase time to live. This survival rule can be only applied when TTL is zero. The survival rule can be expressed mathematically in terms of the structures *H*, *D* and *N* as see in Equation 9:

$$\Delta TTL(x,i,t) = \begin{cases} D_{i,\omega(x,i,t),t}, \; si\ \Omega(x,i,t) > Z_x \\ 0, \quad en\ otro\ caso \end{cases} \tag{9}$$

where ÄTTL(*x, i, t*) is the increment assigned to the TTL of ant *x* (that is, number of additional steps that the ant will be allowed to take) when searching for resources that match to *t*, currently being at node *i*. The number of additional steps $D_{i,\omega(x,i,t),t}$ for arriving in the node ω(*x, i, t*) is determined from the shortest paths generated by previous ants, and is taken when its associated efficiency Ω(*x, i, t*) is better than *Zx* which is a measure of current performance of the ant x. The auxiliary functions are shown in Equations10 and 11:

$$\Omega(x,i,t) = \max_{j \in (\Gamma(i)/\Lambda_x)} \left\{ \frac{H_{i,j,t}}{D_{i,j,t}} \middle| N_{i,j,t} \notin \Lambda x, \right. \quad (10)$$

$$\omega(x,i,t) = \arg \Omega(x,i,t), \quad (11)$$

where $\Gamma(i)$ is the set of neighbors of node i and Λx is the set of nodes previously visited by the ant x. The tables of hits H, of distances D, and of nodes N were explained in the previous section. The function $\omega(x, i, t)$ determines which node that is neighbor of the current node i and that has not yet been visited has previously produced the best efficiency in serving a query on t, where the efficiency is measured by $\Omega(x, i, t)$.

4.2.4. Modified Transition Rule

The *modified transition rule* is a special case of *transition rule* (see Equations 4 and 5) where $\beta_2 = 0$, $W_d = 0$ and $q = 1$. This rule is greedy and provokes the replication of paths generated by previous ants. This rule takes place when TTL has been extended canceling the normal t*ransition rule*. Mathematically can be express in Equations 12 and 13, like:

$$l_m(x,r,t) = \left\{ \arg\max_{i \in (\Gamma(r)/\Lambda x)} \{\psi(r,i,t)\} \right. \quad (12)$$

where l_m is the *modified transition rule*, r is the current node in the path, t is the searched keyword, Λx is the set of nodes visited by the *Forward Ant x* and

$$\psi(r,i,t) = (w_i \cdot (D_{r,i,t})^{-1})^{\beta_1} \quad (13)$$

where w_i is a parameter that defines the influence of $D_{r,i,t}$ that is the needed distance for arriving in the known nearest node with documents with keyword t, from r passing by i and β_1 is the distance intensifier.

5. AdaNAS Algorithm

AdaNAS is a metaheuristic algorithm, where a set of independent agents called ants cooperate indirectly and sporadically to achieve a common goal. The algorithm has two objectives: it seeks to maximize the number of resources found by the ants and to minimize the number of steps taken by the ants. AdaNAS guides the queries toward nodes that have better connectivity using the local structural metric degree [26]; in addition, it uses the well known *lookahead* technique [25], which, by means of data structures, allows knowing the repository of the neighboring nodes of a specific node.

The AdaNAS algorithm performs in parallel all the queries using query ants. The process done by *query ant* is represented in Algorithm 1. Each node has only a query ant, which generates a *Forward Ant x* for attending only one user query, assigning the searched keyword t to the *Forward Ant*. Moreover, the *query ants* realize periodically the local pheromone evaporation of the node where it is.

In the Algorithm 2 is shown the process realized by the *Forward Ant*, as can be observed all Forward Ants act in parallel. In an initial phase (lines 4-8), the ant checks the local repository, and if it founds matching documents then creates a *backward ant y*. Afterwards, it realizes the search process (lines 9-25) while it has live and has not found R documents.

The search process has three sections: Evaluation of results, evaluation and application of the extension of TTL and selection of next node (lines 24-28).

Algorithm 1: Query ant algorithm

```
1  in parallel for each query ant w located in the node r
2  While the system is running do
3      if the user queries to find R documents with
       keyword t then
4          create Forward Ant x(r,t,R)
5          activate x
6      End
7      apply pheromone evaporation
8  End
9  end of in parallel
```

The first section, the evaluation of results (lines 10-15) implements the classical Lookahead technique. That is, the ant x located in a node r, checks the lookahead structure, that indicates how many matching documents are in each neighbor node of r. This function needs three parameters: the current node (r), the keyword (t) and the set of known nodes (*known*) by the ant. The set *known* indicates what nodes the lookahead function should ignore, because their matching documents have already taken into account. If some resource is found, the *Forward Ant* creates a *backward ant* and updates the quantity of found matching documents.

The second section (lines 16-23) is evaluation and application of the extension of TTL. In this section the ant verifies if TTL reaches zero, if it is true, the ant intends to extend its life, if it can do it, it changes the normal *transition rule* modifying some parameters (line 21) in order to create the *modified transition rule*.

The third section (lines 24-30) of the search process phase is the selection of the next node. Here, the *transition rule* (normal or modified) is applied for selecting the next node r and some structures are updated. The final phase occurs when the search process finishes; then, the *Forward Ant* creates *an update ant z* for doing the pheromone update.

The Algorithm 3 presents the parallel behavior for each *backward ant* which inversely traverses the path given by the *Forward Ant*. In each node that it visits, it tries to update the structures *D, H* and *N*, which will be used for future queries (lines 7-11). The update is realized if the new values point to a nearer node (line 7). After that, it informs to *ant query* of the initial node of the path how many documents the *Forward Ant* found and which path used (line 13).

The Algorithm 4 presents the concurrent behavior for each *update ant* which inversely traverses the path given by the *Forward Ant*. In each node that it visits, it updates the pheromone trail using the Equation 6 (line 5).

6. Experiments

In this section, we describe the experiments we carried during the comparisons of the AdaNAS and NAS algorithms.

Algorithm 2: Forward ant algorithm

```
1  in parallel for each Forward Ant $x(r,t,R)$
2  initialization: $TTL = TTLmax$, $hops = 0$
3  initialization: $path=r$, $\Lambda=r$, $known=r$
4  $Results = get\_local\_documents(r)$
5  if $results > 0$ then
6     create backward ant y($path, results, t$)
7     activate y
8  End
9  while $TTL < 0$ and $results < R$ do
10       $La\_results$= look ahead($r,t,known$)
11       if $la\ results > 0$ then
12            create backward ant y($path, la\ results, t$)
13            activate y
14            $results \ \ results + la\ results$
15       End
16       if $TTL > 0$ then
17            $TTL = TTL - 1$
18       Else
19            if ($results < R$) and ($\Delta TTL(x, results, hops) > 0$) then
20                 $TTL = TTL + \Delta TTL(x, results, hops)$
21                 change parameters: $q$=1, $W_d$=0, $\beta_2$=0
22            End
23       End
24 $Hops = hops + 1$
25 $Known = known \cup [\,(r \cup \Gamma(r))$
25 $\Lambda = \Lambda \in \ r$
27 $r = l(x,r,t)$
28 add to path($r$)
29 End
30 create update ant $z(x, path, t)$
31 activate z
32 kill x
33 end of in parallel
```

6.1. Generation of the test data

A SQRP instance is formed by three separate files: topology, repositories, and queries. We generated the experimental instances following largely those of NAS reported by Cruz *et al.* [7] in order to achieve comparable results. The structure of the environment in which is carried out the process described is called *topology*, and refers to the pattern of connections that form the nodes on the network. The generation of the topology (*T*) was based on the method of Barabási *et al.* [27] to create a scale-free network. We created topologies with 1024 nodes; the number of nodes was selected based on recommendations in the literature [3], [28].

The *local repository* (*R*) of each node was generated using "topics" obtained from ACM Computing Classification System taxonomy (ACMCCS). This database contains a total of 910 distinct topics. Also the content are scale-free: the nodes contain many documents in their repositories on the same topic (identified by keywords) and only few documents on other topics.

Algorithm 3: Backward ant algorithm

```
1  initialization: $hops = 0$
2  in parallel for each backward ant $y(path, results, t)$
3  for $I = |path| - 1$ to 1 do
4       $R = path_{(i-1)}$
5       $s = path_i$
6       $hops = hops + 1$
7       if Dr, s, t > hops then
8            $D_{r,s,t} =$ hops
9            $H_{r,s,t} =$ result
10           $N_{r,s,t} =$ path$_h$
11      End
12 End
13 Send ($results, path$) to the query ant located in $path1$
14 kill y
15 end of in parallel
```

Algorithm 4: Update ant algorithm

```
1  in parallel for each update ant $z(path, t, x)$
2  for $i = |path| - 1$ to 1 do
3       $R = path_{(i-1)}$
4       $s = path_i$
5       $\tau_{r,s,t} = \ \tau_{r,s,t} + \Delta\tau_{r,s,t}(x)$
6  End
7  kill z
8  end of in parallel
```

For the generation of the *queries* (*Q*), each node was assigned a list of possible topics to search. This list is limited by the total amount of topics of the ACMCCS. During each step of the experiment, each node has a probability of 0.1 to launch a query, selecting the topic uniformly at random within the list of possible topics of the node repository. The probability distribution of *Q* determines how often the query will be repeated in the network. When the distribution is uniform, each query is duplicated 100 times in average.

The topology and the repositories were created static, whereas the queries were launched randomly during the simulation. Each simulation was run for 15,000 queries during 500 time units, each unit has 100 ms. The average performance was studied by computing three performance measures of each 100 queries:

- **Average hops**, defined as the average amount of links traveled by a Forward Ant until its death, that is, reaching either the maximum amount of results required R or running out of TTL.
- **Average hits**, defined as the average number of resources found by each Forward Ant until its death.
- **Average efficiency**, defined as the average of resources found per traversed edge (hits/hops).

6.2. Parameters

The configuration of the algorithms used in the experimentation is shown in Tables 1 and 2. The first column is the parameter, the second column is the parameter value and the third column is a description of the parameter. These parameter values were based on recommendations of the literature [3], [6], [7], [29], [30].

6.3. Results

The goal of the experiments was to examine the effect of the strategies incorporated in the AdaNAS algorithm and determine whether there is a significant contribution to the average efficiency. The main objective of SQRP is to find a set of paths among the nodes launching the que-

Table 1. Parameter configuration of the NAS algorithm.

PARAMETER	VALUE	DEFINITION
α	0.07	Global pheromone evaporation factor
ρ	0.07	Local pheromone evaporation factor
β	2	Intensifier of pheromone trail
τ_0	0.009	Pheromone table initialization
q_0	0.9	Relative importance between exploration and exploitation
R	10	Maximum number of results to retrieve
TTL_{max}	10	Initial TTL of the Forward Ants
W	0.5	Relative importance of the resources found and TTL

Table 2, Parameter configuration of the AdaNAS algorithm.

PARAMETER	VALUE	DEFINITION
ρ	00.07	Local pheromone evaporation factor
β_1	2	Intensification of local measurements (degree and distance) in transition rule.
β_2	1	Intensification of pheromone trail in the in the transition rule.
T_0	0.009	Pheromone table initialization
q_0	0.9	Relative importance between exploration and Exploitation in the transition rule.
R	10	Maximum number of results to retrieve
TTL_{max}	10	Initial TTL of the Forward Ants
w_h	0.5	Relative importance of the hits and hops in the increment rule
w_d	1	Degree's influence in the transition rule
w_i	1	Distance's influence in the transition rule

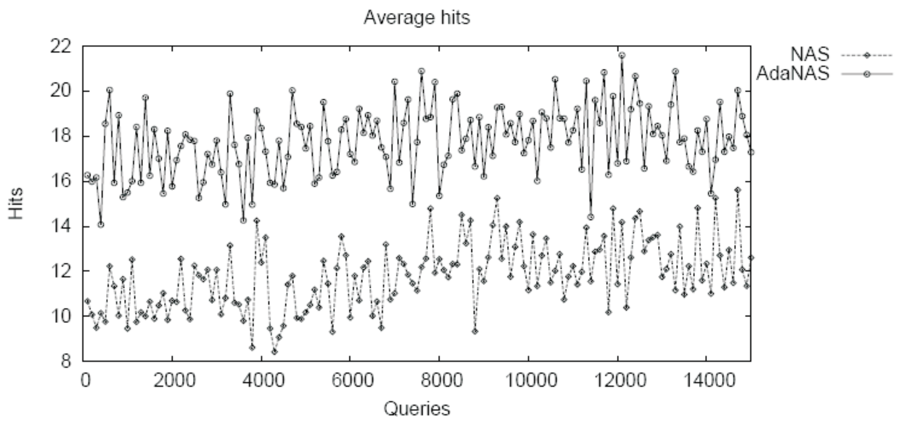

Fig. 2. Learning evolution in terms of the number of resources found for AdaNAS and NAS algorithms.

ries and the nodes containing the resources, such that the efficiency is greater, this is, the quantity of found resources is maximized and the quantity of steps given to find the resources is minimized.

Figure 2 shows the *average hits* performed during 15,000 queries with AdaNAS and NAS algorithms on an example instance. NAS starts off approximately at 13.4 hits per query; at the end, the average hit increases to 14.7 hits per query. For AdaNAS the average hit starts at 16 and after 15,000 queries the average hit ends at 18.3. On the other hand, Figure 3 shows the *average hops* performed during a set of queries with NAS and AdaNAS. NAS starts approximately at 17.4 hops per query; at the end, the average hops decrease to 15.7 hops per query. For AdaNAS the average hops starts at 13.7 and after 15, 000 queries the average hops ends at 9.1. Finally, Figure 4 shows the *average efficiency* performed during a set of queries. NAS starts approximately at 0.76 hits per hop; at the end, it increases to 0.93 hits per hop. For AdaNAS the average efficiency starts at 1.17 hits per hop and after 15, 000 queries the average efficiency ends at 2.

The adaptive strategies of AdaNAS show an increment of 24.5% of found documents, but the biggest contribution is a reduction of hops in 40%, giving efficiency approximately twice better on the final performance of NAS. This observation suggests that the use of degree instead of DDC was profitable. In addition, the incorporation of the survival rule permits to improve the efficiency, because it guides the Forward Ants to nodes that can satisfy the query. Moreover, in future works it will be important to study adaptive strategies for other parameters as well as the initial algorithm parameter configuration in search of further improvement in the efficiency.

Figure 5 shows the results of the different experiments applied to NAS and AdaNAS on thirty runnings for each ninety different instances generated with the characteristics described in Section 6.1. It can been seen from it that on all the instances the AdaNAS algorithm outperforms NAS. On average, AdaNAS had efficiency 81% better than NAS.

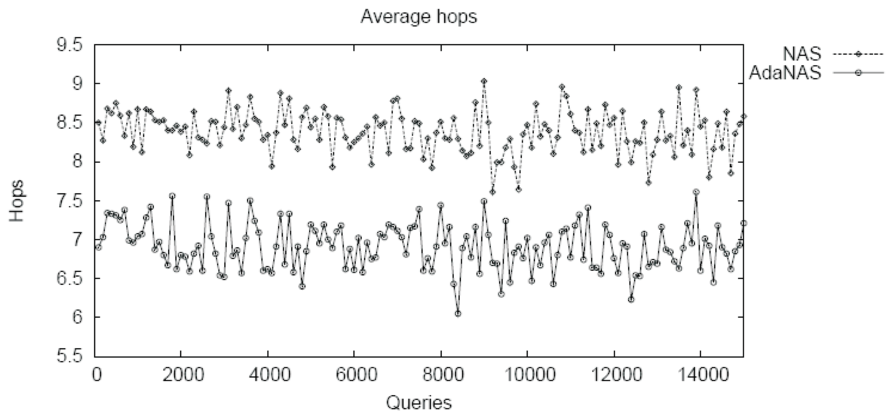

Fig. 3. Learning evolution in terms of the length of the route taken for AdaNAS and NAS algorithms.

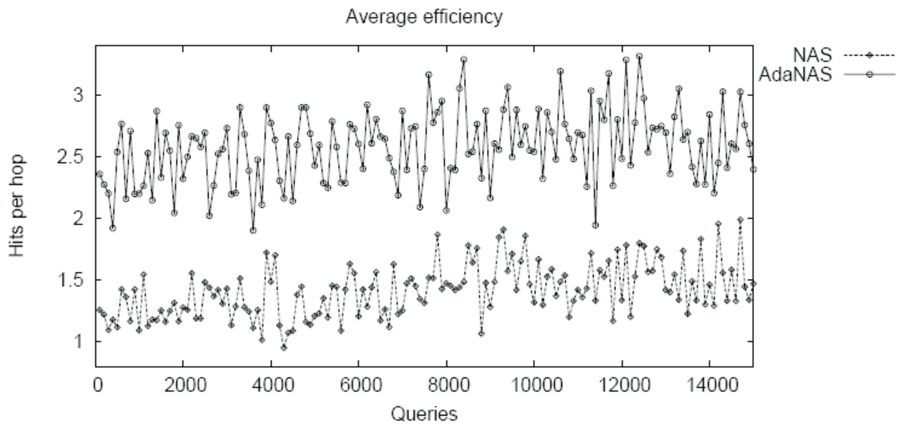

Fig. 4. Learning evolution in terms of the efficiency (hits/ hop) for AdaNAS and NAS algorithms.

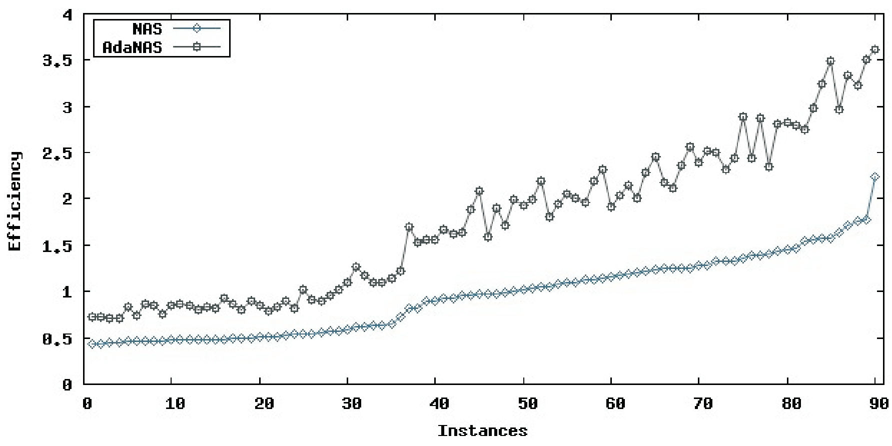

Fig. 5. Comparison between NAS and AdaNAS experi-menting with 90 instances.

7. Conclusions

For the solution of SQRP, we proposed a novel algorithm called AdaNAS that is based on existing ant-colony algorithms, which is a state of art algorithm. AdaNAS algorithm incorporates parameters adaptive control techniques to estimate a proper TTL value for dynamic text query routing.

In addition, it incorporates local strategies that take advantage of the environment on local level; three functions were used to learn from past performance. This combination resulted in a lower hop count and an improved hit count, outperforming the NAS algorithm. Our experiments confirmed that the proposed techniques are more effective at improving search efficiency. Specifically the AdaNAS algorithm in the efficiency showed an improvement of the 81% in the performance efficiency over the NAS algorithm.

As future work, we plan to study more profoundly the relation among SQRP characteristics, the configuration of the algorithm and the local environment strategies employed in the learning curve of ant-colony algorithms, as well as their effect on the performance of hop and hit count measures.

ACKNOWLEDGMENTS

This research was supported in part by CONACYT, DGEST and IPN.

AUTHORS

Claudia Gómez Santillán - Instituto Tecnológico de Ciudad Madero (ITCM). 1ro. de Mayo y Sor Juana I. de la Cruz s/n CP. 89440, Tamaulipas, México. Tel.: (52) 833 3574820 Ext. 3024 and Instituto Politécnico Nacional, Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada (IPN-CICATA). Carretera Tampico-Puerto Industrial Alt., Km.14.5. Altamira, Tamps., México. Tel.: 018332600124. E-mail: cggs71@hotmail.com.

Laura Cruz Reyes*, Gilberto Rivera Zarate - Instituto Tecnológico de Ciudad Madero (ITCM). 1ro. de Mayo y Sor Juana I. de la Cruz s/n CP. 89440, Tamaulipas, México. Tel.: (52) 833 3574820 Ext. 3024. E-mails: lcruzreyes@prodigy.net.mx and riveragil@gmail.com.

Eustorgio Meza, Elisa Schaeffer - CIIDIT & FIME, Universidad Autónoma de Nuevo León (UANL), Av. Universidad s/n, Cd. Universitaria, CP.66450, San Nicolás de los Garza, N.L., México. Phone: (52)8113404000 Ext.1509. E-mails: elisa@yalma.fime.uanl.mx, emezac@ipn.mx.

* Corresponding author

References

[1] Sakarayan G., *A Content-Oriented Approach to Topology Evolution and Search in Peer-to-Peer Systems*. PhD thesis, University of Rostock, 2004.

[2] Wu L.-S., Akavipat R., Menczer F., "Adaptive query routing in peer Web search". In: *Proc. 14th International World Wide Web Conference*, 2005, pp. 1074-1075.

[3] Michlmayr E., *Ant Algorithms for Self-Organization in Social Networks*. PhD thesis, Vienna University of Technology, 2007.

[4] Mihail M., Saberi A., Tetali P., "Random walks with look ahead in power law random graphs", *Internet Mathematics*, no. 3, 2004.

[5] Tempich C., Staab S., Wranik A., "REMINDIN': Semantic Query Routing in Peer-to-Peer Networks based on Social Metaphers" , in: *13th World Wide Web Conference (WWW)*, 2004.

[6] Dorigo M., Gambardella L.M. "Ant colony system: A cooperative learning approach to the traveling salesman problem", *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 53-66.

[7] Cruz L., Gómez C., Aguirre M., Schaeffer S., Turrubiates T., Ortega R., Fraire H., NAS algorithm for semantic query routing systems in complex networks. In: *DCAI, Advances in Soft Computing*, vol. 50, 2008, pp. 284-292.

[8] Diestel R., "*Graph Theory"*, *Graduate Texts in Mathematics*, vol. 173, Springer-Verlag: New York, USA, 2000.

[9] Costa L., Rodríguez F.A., Travieso G., Villas P.R., "Characterization of complex networks: A survey of measurements", *Advances in Physics*, no. 56, 2007, pp. 167-242.

[10] Erdos P., Rényi A., *On the evolution of random graphs*, vol. 2, Akademiai Kiad'o, Budapest, Hungary, 1976. First publication in MTA Mat. Kut. Int. Kozl. 1960, pp. 482-525.

[11] Gilbert E., "Random graphs", *Annals of Mathematical Statistics*, 30(4), Dec. 1959, pp. 1141-1144.

[12] Bollobás B., *Random Graphs*, *Cambridge Studies in Advanced Mathematics*., vol. 73, Cambridge University Press, Cambridge, UK, 2nd edition, 2001.

[13] Adamic L., Huberman B., "Power-law distribution of the World Wide Web". *Science*, no. 287(5461), 2000, p. 2115.

[14] Albert R., Jeong H., Barabási A., "Error and attack tolerance of complex networks". *Nature*, no. 506, 2000, pp. 378-382.

[15] Faloutsos M., Faloutsos P., Faloutsos C., "On power-law relationship on the internet topology". *ACM SIGCOMM Computer Communication Review*, 1999, pp. 251-262.

[16] Barabási A., *Emergence of scaling in complex networks*, *Wiley VHC*, 2003, pp. 69-82.

[17] Newman M. E. J., "Power laws, pareto distributions and zipf's law", *Contemporary Physics*, vol. 46(5), 2005, pp. 323-351.

[18] Birattari M., *The Problem of Tunning Mataheuristics as Seen From a Machine Learning Perspective*. PhD thesis, Bruxelles University, 2004.

[19] Glover F., Laguna M., *Tabú Search.* Kluwer Academic Publishers, 1986.

[20] Holland J.H., *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[21] Amaral L., Ottino J., "Complex systems and networks: Challenges and opportunities for chemical and biological engineers". *Chemical Engineering Scientist*, no. 59 2004, pp. 1653-1666.

[22] Liu L., Xiao Long J., Kwock C.C., "Autonomy oriented computing from problem solving to complex system modeling". In: *Springer Science + Business Media Inc*, 2005, pp. 27-54,

[23] Wu C.-J., Yang K.-H., Ho. J.M., "AntSearch: An ant search algorithm in unstructured peer-to-peer networks". In: *ISCC*, 2006, pp. 429-434.

[24] Goldberg P., Papadimitriou C., "Reducibility among equilibrium problems". In: *Proceedings of the 38th annual ACM symposium on Theory of Computing*, New York, NY, USA, 2005, pp. 61-70.

[25] Michlmayr E., Pany A., Kappel G., "Using Taxo-nomies for Content-based Routing with Ants". In: *Proceedings of the Workshop on Innovations in Web Infrastruc' ture, 15th International World Wide Web Conference (WWW2006)*, May 2006.

[26] Ortega R., *Estudio de las Propiedades Topológicas en Redes Complejas con Diferente Distribución del Grado y su Aplicación en la Búsqueda de Recursos Distribuidos*. PhD thesis, Instituto Politécnico Nacional, México, 2009. (in Spanish)

[27] Barabási A., Albert R., Jeong H., "Mean-field theory for scale-free random networks". *Physical Review Letters*, no. 272, 1999, pp. 173-189.

[28] Babaoglu O., Meling H., Montresor A., "Anthill: An framework for the development of agent-based peer to peer systems". In: *22nd InternationalConference On Distributed Computing Systems*. ACM, 2002.

[29] Ridge E., *Design of Expirements for the Tuning of Optimization Algorithms*. PhD thesis, University of York, 2007.

[30] Ridge E., Kudenko D., "*Tuning the Performance of the MMAS Heuristic in Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*". *Lecture Notes in Computer Science*, vol. 4638, T. Stutzle, M. Birattari, Eds. Berlin / Heidelberg: Springer, 2007, ISBN 978-3-540-74445-0, pp. 46-60.