Universidad Carlos III de Madrid
e-Archivo

Institutional Repository

This document is published in:

Applied Intelligence 36 (2011) 4, pp. 899-917

DOI: 10.1007/s10489-011-0304-1

© 2011. Springer Science+Business Media, LLC.

Using the ACO algorithm for path searches in social networks

Jessica Rivero · Dolores Cuadra · Javier Calle · Pedro Isasi

Abstract One of the most important types of applications currently being used to share knowledge across the Internet are social networks. In addition to their use in social, professional and organizational spheres, social networks are also frequently utilized by researchers in the social sciences, particularly in anthropology and social psychology. In order to obtain information related to a particular social network, analytical techniques are employed to represent the network as a graph, where each node is a distinct member of the network and each edge is a particular type of relationship between members including, for example, kinship or friendship. This article presents a proposal for the efficient solution to one of the most frequently requested services on social networks; namely, taking different types of relationships into account in order to locate a particular member of the network. The solution is based on a biologically-inspired modification of the ant colony optimization algorithm.

Keywords Large graphs · Social networks · ACO · Dijkstra · Path search

J. Rivero (✉) · D. Cuadra · J. Calle · P. Isasi
Computer Science Department, Carlos III University of Madrid,
Avd. Universidad, 30, 28911 Leganés, Madrid, Spain
e-mail: jrivero@inf.uc3m.es

D. Cuadra
e-mail: dcuadra@inf.uc3m.es

J. Calle
e-mail: fcalle@inf.uc3m.es

P. Isasi
e-mail: isasi@ia.uc3m.es

1 Framework and motivation

Today, the great majority of people around the globe are citizens of the information society. Among these digital citizens, only a very few remain who are not users of some type of social network.

With the appearance and boom of social networks, new problems have arisen and users have new needs that pose unique challenges for experts, particularly within the field of information management.

One of the more frequent types of requests made by social network users is to locate other members of the network in order to verify whether any particular relationship exists that could be beneficial in establishing business or research partnerships. Used in this way, social networks can create mutually beneficial ties between people who otherwise would have difficulty connecting.

One of the first proposals made to directly address the challenges posed by these types of requests appeared in 1997 and the Referral Web system [19] was developed to carry out member searches on small-scale social networks. It is the aim of the present article to propose a near optimal solution for member search requests on very large social networks that efficiently produces high-quality results with rapid response times.

In order to respond both effectively and efficiently to the challenge of finding links between members of a social network, two concrete steps must be taken. (1) All data being handled must be transformed into a graph, that is, a domain model must be created, and (2) an algorithm capable of efficiently searching for paths between nodes on the graph (i.e., members of a particular social network) that were specified by a user has to be identified.

With regard to this first step, it is necessary both to identify the concepts one wants to represent and to transform

these concepts into nodes on the graph. Additionally, certain weights must also be assigned to each of the edges on the graph representing relationships between nodes. This, it must be noted, is not always an easy task. In the case of social networks, for example, the act of assigning weights to the edges connecting different nodes on the graph (i.e., network members) can be extraordinarily complex. In one study [1] where nodes on the graph represent students in Club Nexus, one of Stanford University's social networks, careful attention had to be given to different factors such as gender or class year of a user who another user wanted to friend.

With respect to the second step, there are a large number of studies with the aim of finding algorithms capable of dealing with the different difficulties that graphs may present. Nevertheless, in most cases, the existing literature focuses on algorithms capable of obtaining paths between nodes in small graphs. This fact is problematic insofar as the graphs obtained when modeling social networks are large. For instance, in one particular path search proposal [26] which functions efficiently on small graphs (i.e., with thousands of nodes), the reported success rate diminishes significantly with an increase in the number of nodes on the graph. The same problem is found in [16] which efficiently conducts path searches on graphs on the order of 2^{13}.

Given the current dearth of studies available on large graphs representing social networks, it is the aim of the present article to attempt to fill this gap in the literature. In order to aptly address the nature of such graphs (graphs with a high degree of clustering and with a small number of steps between any two nodes on the graph), the ant colony optimization (ACO [12–14]) algorithm has been selected. However, the ACO algorithm's use is often restricted to small graphs (hundreds or thousands of nodes) because on bigger graphs the solutions will consists of too many steps, causing most of the ants to get lost and resulting in poor performance and often unsolved services. The article proposes an extended version of the algorithm, thereby making it suitable for use in large graphs with hundreds of thousands or millions of nodes.

The extension of the ACO algorithm proposed here was inspired by the behavior of animals capable of locating paths to food sources by using their sense of smell. Thus, while the classic ACO algorithm engineers ants which, as is true of real ants in nature, locate food sources through pheromone tracking, the modified ACO algorithm proposed engineers ants that, in addition to their traditional pheromone tracking capabilities, are also endowed with a sense of smell. This gives them an additional way to locate food sources, namely, by the odor which those food sources give off. The ants' new behavioral capability in the modified ACO algorithm is particularly significant for two reasons:

- If a food source is located on a relevant node of the graph, the time needed to obtain a path to this node from another node on the graph will be reduced.
- The algorithm can learn over time on the basis of queries about paths, and it is possible to modify the number of food sources and the number of nodes affected by the diffusion of this odor during the execution of searches. During this process, the ants will search for new paths on the graph using the stochasticity typical of the ACO algorithm if there are no other food sources because odor is only an added help.

In order to clearly explain and test the proposal, the present article is organized as follows. Section 2 discusses relevant literature in the area of path searches, while Sect. 3 formalizes the scenario to be dealt with by the modified algorithm and then completely describe the modified algorithm to be used. Following the full explanation of the scenario and algorithm, Sect. 4 presents and analyzes the results obtained from several experiments carried out with the objective of demonstrating both the viability and scalability of the proposal. Finally, Sect. 5 offers conclusions and a brief discussion of interesting areas for future research.

2 Background

While the problems raised by path searches are in no way new to graph theory research, different solutions nevertheless continue to be proposed to this day. This is due to the fact that the ever-growing volume of information currently being handled must be represented as graphs if it is to be managed both effectively and efficiently.

Within the domain of social networks, two previously-mentioned works [16, 26] offering solutions based on the k-nearest neighbor algorithm may be highlighted. In the two studies, a social network is represented as a graph and local information is utilized such that, from a given start node to a specified end node, the next node along the path from the start en route to the end may be located. Both proposals work with scenarios of a reduced size and, as clearly demonstrated [26], the probability of obtaining a solution decreases with an increase in the size of the scenario. This particular result demonstrates the inadequacy of these proposals for dealing with the new necessities and challenges raised by the large social networks currently in use.

With regard to studies searching for paths on graphs from different domains (i.e., not representing social networks) with a large number of nodes and edges, from a survey of the literature it appears that all utilize some type of preprocessing technique to structure the graph such that, despite the large volume of data being handled, only certain fragments of the graph are taken into account at different phases of the search. In this way, the studies are able to considerably

reduce response times for a given query. Once the graphs have been processed, the studies use classic path search algorithms (e.g., Dijkstra's algorithm or the A^* search algorithm) which they slightly modify in order to efficiently work with various fragments of the graphs. The principal ways in which these studies differ, then, is not with regard to the simple presence of pre-processing techniques, but rather to the methods used for graph storage as well as the specific types of pre-processing techniques used (e.g., hierarchies, clusters, etc.):

Storage in main memory. Some examples of this type of storage can be found in certain studies [4, 9] which organize the respective graphs using a hierarchy, or in another [10] which divides the main graph into sub-graphs and uses a pre-processing method to determine which of the existing edges in the graph ought to be taken into account in each of the sub-graphs. For proposals of this type, it is extremely important that the algorithm used in the pre-processing phase obtains a highly compact or reduced graph. Due to this, the amount of time required to obtain these types of graphs and the resulting costs in the pre-processing phase are necessarily large.

Secondary memory. In some particular proposals [5, 6], file systems are used to save both the sub-graphs as well as the paths obtained between start and end nodes (i.e., the nodes found along the borders between different sub-graphs). In others [27, 28], graphs are organized into tree structures, taking into account the distances between nodes, and saved in databases. With respect to this last method of data storage, a vast number of studies are gathered together in [31]. In each of them, the graphs used are pre-processed and later, all relevant information is stored in a database. While the pre-processing algorithms used in these cases are less heavy-handed than those discussed in the earlier point, the time required is very high.

One common characteristic to all proposals discussed here is the large amount of time needed to complete the pre-processing. As a result it is very difficult to include changes to the pre-processing factors based on studies of the solved path queries because any change in the pre-processing requires a lot of time. This means that these proposals are not adaptable to modifications to the pre-processing factors.

An important aspect not taken into account in previous proposals is the consideration of the kind of topology that graphs which represent social networks have. This kind of graph, as well as its large number of nodes, possesses a very peculiar structure (small-world topology [23, 30]) in which the number of edges to be covered in travelling from one particular node to another is extremely low and the clustered index is extremely high. Considering these different characteristics, then, it may be understood intuitively that the ACO algorithm [12–14] can offer particularly good solutions in this domain. ACO has been used for a multitude of applications: a new version of ACO (BACO) to solve the problem of job scheduling in grid computing [7], ACO as an algorithm to create personalized guides within museums using mobile devices [18], evolved ACO algorithm (HACO) to determine medical diagnoses [24], algorithm EACO (modified ACO algorithm) to efficiently solve the vehicle routing problem capacitive [20], a modified fuzzy ant clustering to solve problems in image retrieval application [29], a version of ant colony search algorithm to solve the multi-objective problems of strategical planning in the design of electrical distribution systems [17] or ACO algorithm shown in [33] that solves the problem of scheduling resource control satellite. However, from a review of the available literature, it appears that no studies exist that apply the algorithm to large graphs with millions of nodes with relevant information stored in databases. Although some studies [2] may approach such an aim, they are nevertheless restricted by the limitations imposed by the storage of graphs in main memory.

In the present article, ideas presented in a previous study [25] are expanded upon in order to propose an adaptation of the ACO algorithm equipped to deal with the different concerns discussed above (i.e., graphs with hundreds of thousands or millions of nodes and with a high degree of clustering). As mentioned earlier, in addition to the pheromone tracking capabilities of the ants of the classic ACO algorithm, the ants in the modified ACO algorithm are further endowed with a sense of smell to locate sources of food. As is the case for many animals in nature, the capacity to detect the odor given off by a particular food source from, at times, large distances effectively reduces the zone in which an animal must search and allows for efficiently reaching that food source even when it lies outside of the animal's line of sight. In this way, the modification proposed here reduces the search space of the graph, thereby making it viable to use the modified ACO algorithm in large graphs without, in the process, altering the basic functioning capabilities of the classic ACO algorithm. As a result of this and despite the other modifications made, the proposal can nevertheless maintain the characteristics of the classic ACO algorithm and make feasible the adaptation of the algorithm to the changes based on the studies of the solved paths in a short period of time.

3 Proposal

Having discussed the questions relevant to graphs of social networks not previously addressed in the literature, the article may now describe the modified ACO algorithm proposed in greater detail.

The modified ACO algorithm proposed here allows ants to conduct path searches in graphs with hundreds of thou-

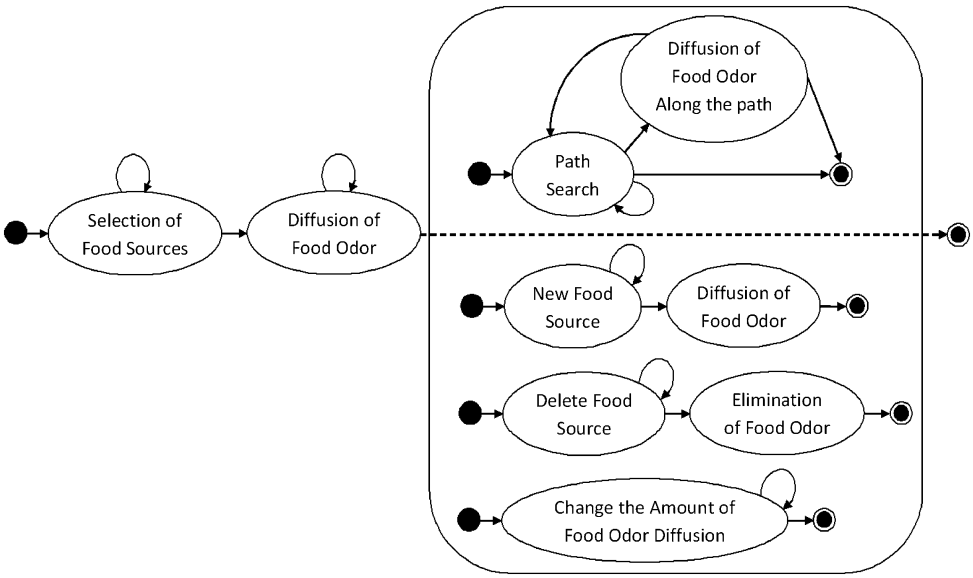


Fig. 1 State diagram of the algorithm

sands or millions of nodes and with a high degree of clustering by further endowing them with a behavioral capacity characteristic of certain animals with a sense of smell. As is the case with these animals, the ants used here are designed not only with pheromone tracking capabilities, but also with the capability to locate paths from their nests to a given food source using the odor which that food source gives off. Insofar as it is no longer necessary for the ants to discover the node on the graph where a particular food source is located, but rather simply a node on which some amount of odor given off by that food source (i.e., from another node on the graph) is present, these modified ants are able to reduce the number of nodes they inspect in their search and, therefore, may well find the path to food sources more quickly than their counterparts from the classic ACO algorithm [14].

In order to carry out this modification, the search process of the algorithm is divided into three principal phases:

Selection of the nodes on the graph where food sources are to be located. It is good if this selection can be done before the phase of path searches (first state in Fig. 1), but it is not necessary. This selection may be done at any moment, and concurrently with the task being executed at that moment (state "New Food Source" in Fig. 1).

Diffusion of food odor. Similar to the previous phase, this second phase is carried out prior to the execution of path search services (second state in Fig. 1). However, it is important to note that even after service execution has begun, or during service execution, these prior phases may nevertheless be revisited by the algorithm any time that studies of the services executed determine that new food sources should appear ("New Food Source" state in Fig. 1), that old food sources should disappear ("Delete Food Source" state in Fig. 1) or when these studies show that the amount of diffused odor is incorrect and ought to be increased/decreased ("Change the Amount of Food Odor Diffusion" state in Fig. 1).

Path searches with each requested service. This phase also permits the revisiting of prior phases insofar as when an ant retrieves part of a food source to bring back to its nest, as in nature, a new trail of food odor is thereby created along the path travelled by the ant from the food source to the nest ("Diffusion of Food Odor along the Path" state in Fig. 1). In this way, the zone on the graph with food odor from the food source increases.

The first sub-section of this point shows a formalization of the scenario in which path searches are to be conducted, the following sub-sections explain each of these different phases in greater detail and the last sub-section shows an example to make the algorithm clearly understood. However, before continuing, it is important to state the obvious fact that, prior to executing any of these phases, the information corresponding to a given social network must first be transformed into a graph (if such a preliminary step has not already been taken).

3.1 Problem formalization

Given a particular work scenario, this scenario may be represented by the connected graph $G(t) = \{N, L\}$ at a particular moment of time t, such that:

- $N(t) = \{(x, y) \in \mathcal{R} \times \mathcal{R}\}$ represents the set of nodes in G at moment in time t.
- $L(t) = \{l_{ij} = (n_i, n_j) \in N \times N, n_i \neq n_j\}$ is the set of edges in G at moment in time t, where $l_{ij} = l_{ji}$.

Each of the edges $l_{ij} \in L$ is assigned a particular weight which can be defined by the function $W : L \to \mathcal{R}^+$, where $w_{ij} = W(l_{ij})$.

The size of G is determined by the number of nodes it possesses, that is, the number of elements in $N(|N|)$. In the particular scenario to be considered in later sections of the present article, G possesses many hundreds of thousands of nodes.

A series of services is carried out on G requesting a path search between any two nodes $n_i, n_j \in N$, such that the quality of the path obtained $p_{ij} \in P(P : N \times N \rightarrow L' \subseteq L)$ be localizable between certain limits and that the time t_{answer} taken in obtaining the path be less than a time $t_{threshold}$ fixed by the user requesting the service.

3.2 Selection of food source nodes

In all social networks there are certain members (e.g., celebrities) who have a greater number of friends than other members and who have a greater probability that other members will want to friend them or use them in order to connect with others in the network. That is, the nodes representing these particular members on a graph are present (i.e., as start, end or internal nodes) in a much larger number of path searches executed than other nodes on the graph. In other words, the nodes representing these more popular individuals have a greater centrality than the others.

Due to the frequency with which these particular nodes are used in path searches, it is particularly interesting and useful to identify them within the graph and differentiate them in some way from the other components of N.

To achieve this in the present proposal, all nodes with a centrality above a specified threshold c are included in a new set *Food* (F), where $F \subseteq N$ and each element of F is labeled f_i. The number of elements within F, therefore, is greater than or equal to zero, according to the number of nodes in the graph with a centrality greater than c.

As previously indicated, it is possible that new nodes of interest could appear or that nodes already included in F could lose their importance relative to other nodes in the graph when studies are done of path searches that have been carried out. This possibility, however, is not problematic since such a change would merely require the inclusion of a new node in F or the removal of a pre-existing node from F.

3.3 Diffusion of food odor

Once all nodes comprising F have been identified, the following phase of the proposed algorithm must be explained, namely, how these relevant nodes in G are used by the algorithm.

Just as it is true for graphs of social networks discussed in the previous sub-section, in the natural world there are zones of greater relevance for a given animal. Among these zones, the animal will always place importance on those in which food—be it fresh grass, its prey or a piece of bread—can be found. Applying this real-world analogy to the formalization described in the previous sub-section, these zones in G where food is present are those represented by the set F.

Attracted to a particular food source by the odor which that source gives off, an animal can successfully reach the food source by following an odor trail that increases in intensity as the distance from it decreases.

The algorithm proposed in the present article attempts to simulate this particular aspect of animal behavior. That is, around every food source f_i with the maximum food odor are a set of nodes where that odor is nevertheless present, but with an intensity that is inversely proportional to the distance those nodes are from the corresponding f_i.

The concept described above may be formally expressed in the following way:

- $O(n_i)$ indicates the food odor associated with the node n_i.
- m is the maximum food odor intensity that a particular node on the graph may possess.
- $O(f_i) = m, \forall f_i \in F$, insofar as f_i, as a food source, represents a point from which food odor is diffused to the rest of the nodes on the graph.
- $\forall f_i \in F$ there exists a subset s_i of nodes to which the food odor has been diffused, and where each $s_i \subseteq N$ and contains at least one element, f_i.
- All zones of the graph where food odor is present are found within S; that is, $\forall s_i \in S$, where $|S| = |F|$.

Before discussing the algorithm used to diffuse the food odor from f_i to corresponding nodes, it is important to explain certain terms that directly influence on the creation of each zone where food odor is present:

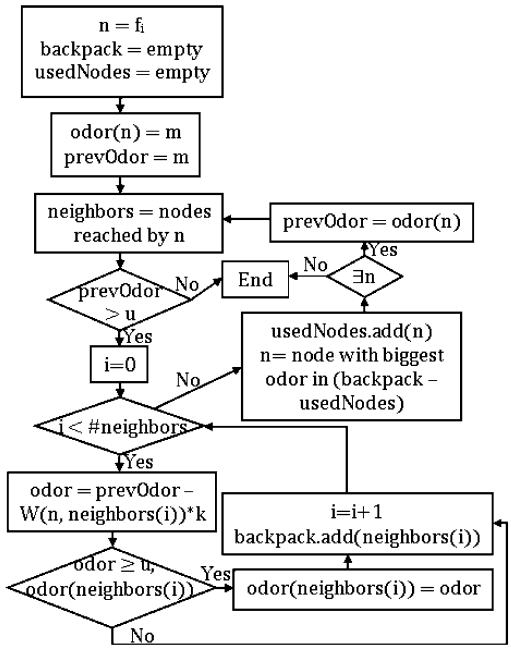
- Insofar as the intensity of a food odor must decrease as a node's distance from the food source increases, it is necessary to identify in the algorithm a specific rate according to which this decrease in food odor is quantitatively expressed. In the algorithm proposed here, the odor given off by a food source decreases according to both the costs associated with graph edges, as well as a factor k which determines the weight of each edge's cost in the decrease. This can be formally represented as $O(n_i) = O(n_j) - k \cdot w_{ij}$, where n_i is the node to which the food odor is diffused from node n_j.
- Regarding the creation of the distinct subsets s_i, no element shall form part of that subset if its value falls below that specified by a food odor threshold u in each distinct application.

Having explained each of the factors involved in food odor diffusion in the graph, Algorithm 1 presents the specific instructions for its execution and a flowchart that represents this execution graphically.

The algorithm presented above functions by selecting the node with the most intense food odor from a set of nodes

Algorithm 1 Creation of s_i

```
s_i, backpack, usedNodes=empty;
n=f_i;
odor(n), prevOdor=m;
neighbors=reachedNodes(n);
backpack.add(n);
while (prevOdor>u) do
    for (h=1...neighbors.size) do
        nAux=neighbors(h);
        odorAux=odor(nAux);
        cost=W(n,nAux);
        odor=prevOdor-cost*k;
        if ((odor≥odorAux) and (odor≥u))then
            odor(nAux)=odor;
            backpack.add(nAux);
        end if;
    end for;
    usedNodes.add(n);
    n=nodeBiggestOdorNotInBackPack();
    if (∃n) then
        prevOdor=odor(n);
        neighbors=reachedNodes(n)-usedNodes;
    else
        prevOdor=u;
    end if;
end while;
s_i.add(backpack);
```



saved in a backpack, initially containing only f_i. The algorithm then finds other nodes that can be reached from the previously selected node with food odor. This node is saved within the variable n and the nodes reachable from it in *neighbors*. For each of the latter nodes (i.e., those saved in *neighbors*), respectively, a food odor will be assigned that is equal to that of the more intense, previously-selected node, lowered by w, the cost of the edge connecting each with the previously-selected node, and adjusted by k. This assignment of food odor to *neighbor* nodes, therefore, may be formally represented as $O(n) - k \cdot w$ and is always carried out by the algorithm so long as the value obtained is greater than u, as well as any other values which may have been previously assigned to the neighbor nodes. In these cases where a food odor has been assigned to a node, this node is also placed in the backpack.

Once this process has been carried out for all nodes reachable from n, it is marked (i.e., placed in *usedNodes* in order to avoid additional analyses by the algorithm) and the process is repeated for other nodes in the backpack.

After the complete process has been finished, that is, when $backpack - usedNodes = \emptyset$, each of the elements in the backpack are placed in s_i.

Once the diffusion has been carried out, studies of its use in the next search paths may show that f_i is not used very often, that is, f_i is not an important node. Therefore, f_i should be deleted from F and all the elements forming its s_i would be erased (s_i should be deleted from S).

It is necessary to highlight here the importance of the value assigned to u, inasmuch as said value determines the size of each s_i and, returning to the real life analogy used throughout the article, the degree to which a food odor is diffused. If the value assigned to u is too high, for instance, no real aid is given to the animals in locating the food source insofar as the animals would not have access to any odor trail to follow. As a result, the animals would not know that a food source existed until they literally and randomly stumbled upon it.

If, on the other hand, the value of u is low, the food source is easily discovered by the animals which would simply have to follow the food odor trail. However, the time required to carry out the diffusion would be quite long (in the case of the previous paragraph, the time required is relatively short).

With the importance of this aspect clearly in mind, the evaluation section of the present article carries out a study of different u values so as to observe their influence on the proper functioning of the algorithm.

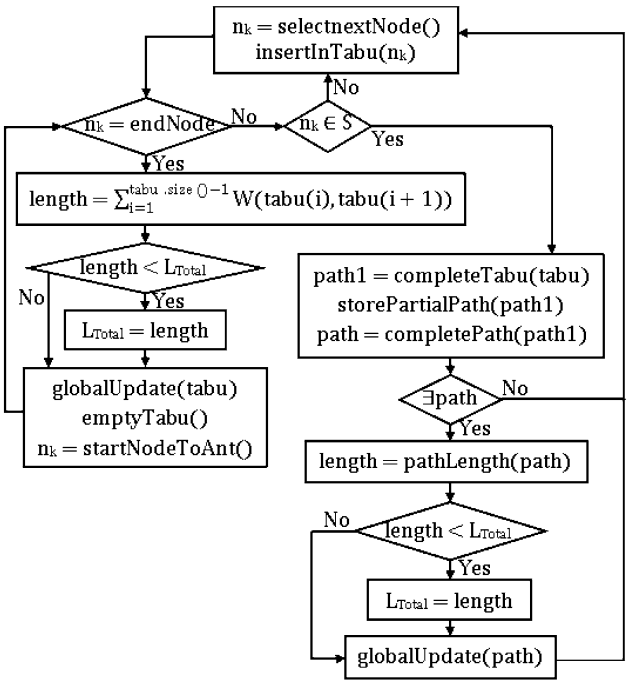
3.4 Path searches: the ACO algorithm modified

The number of edges that an ant should cover before reaching its destination ought to be relatively low, since the contrary could result in the ant getting lost along the way or in the production of an inferior quality solution. This is why the representation of nodes with food odor is particularly helpful for the ACO algorithm, insofar as an ant needs only to find an s_i and follow the corresponding odor trail of growing intensity to obtain a solution.

In this way, the number of edges covered by an ant is reduced considerably without requiring a restructuring of the

Algorithm 2 Path search algorithm

```
if (nk=end_node) then
    length=0;
    for (h=1...(tabu.size-1)) do
        aux=W(tabu(h),tabu(h+1));
        length=length+aux;
    end for;
    if (length<Ltotal) then
        Ltotal=length;
    end if;
    globalUpdate(tabu);
    emptyTabu();
    nk=startNodeToAnt();
elseif (nk ∈ S) then
    path1 = completeTabu(tabu);
    storePartialPath(path1);
    path = completePath(path1);
    if (∃path) then
        length=pathLength(path);
        if (length<Ltotal) then
            Ltotal=length;
        end if;
        globalUpdate(path);
    end if;
    nk=selectNextNode();
    insertInTabu(nk);
else
    nk=selectNextNode();
    insertInTabu(nk);
end if;
```



graph, be it through fragmentation, reorganization as a tree or any other related method. Thus, if a deletion, insertion or modification of an s_i occurs simultaneously with an ant's path search, the ant may nevertheless carry on normally with its search. This is due to the fact that the zones in the graph with food odor act merely as an aid to the tracking capabilities of ants specified under the classic ACO algorithm and preserved in the proposed algorithm.

Despite this fact, the inclusion of zones with food odor within the graph nevertheless marks a significant departure from other published works discussed in Sect. 2 of the present study. This is particularly true given the fact that the most proposals in other studies are incapable of executing a search without having previously carried out a preprocessing of the graph.

Below is a detailed account of the specific extensions proposed for the classic ACO algorithm in order to respond to the current necessities and challenges discussed in previous sections of this paper.

Initialization phase (three separate processes)

- Each of the edges in G is reset to a uniform and fixed amount of pheromone. It is important to remember here that, while food odor is a characteristic specific to the nodes of G, pheromone, on the other hand, is specific to the edges of G.
- At any particular moment, an ant can move to any node accessible from the point at which the ant is located, save those nodes which the ant has visited previously. Only in the case where all of the nodes reachable from the point where the ant is located have already been visited, can the ant return to a previously-visited node. In order to keep track of all the nodes reached, the ants in the algorithm make use of a tabu list into which each visited node is entered. This tabu list is consulted prior to any movement made by an ant and is emptied in the initialization phase of the algorithm.
- Finally, the ants are divided in half, placed in their corresponding nests at opposing ends of a particular path and, to the extent that neither of these two end nodes pertains to F, are given the objective of reaching the end node at the opposite end of the path. If one of these end nodes does, in fact, pertain to F, all ants are situated on the opposing non-F node to search for the corresponding f_i.

Once both the ants and certain characteristics of the graph have been initialized, the search for the requested path may be carried out.

Path search phase In the algorithm to be followed and executed in this phase (executed, that is, so long as the execution time does not exceed $t_{threshold}$) that is represented below (see pseudocode and its flowchart in Algorithm 2), the following aspects possess a particular relevance:

- An ant's selection of a single node out of all those nodes which can possibly and permissibly be visited from a

particular point in G (following the application of the tabu list rule discussed above regarding previously visited nodes) is based on the application of the probabilistic formula expressed below (see formula (1)), where n_i is the node on which the ant is located at a particular moment in time t, *NodesR* are the totality of indexes of visitable nodes following the application of the tabu list rule, n_j are all those nodes with an index pertaining to *NodesR*, and $\tau_{ij}(t)$ is the amount of pheromone existing for edge l_{ij} at moment t

$$p(n_i, n_j) = \frac{\tau_{ij}(t)}{\sum_{k \in NodesR} \tau_{ik}(t)} \tag{1}$$

The selection is executed in the algorithm by the method *selectNextnode()*.

- Only after an ant has found its corresponding end node can it return to its nest and begin the path search again.
- After having reached an s_i, an ant will attempt to follow its current path to its corresponding end node. To do so, the ant's tabu list is completed using the method *completeTabu(tabu)* for the section of the path spanning from a node found with food odor to its corresponding $f_i \in s_i$. As discussed earlier, the discovery of the path from a node with food odor to its corresponding $f_i \in s_i$ is relatively quick and easy, insofar as the ant needs only to follow the increasingly strong odor trail leading to the food source. After this section of the path spanning from one end node to an f_i has been discovered, an attempt is made using the method *completePath(path1)* to complete the full path by continuing from the same f_i to its opposing end node. As discussed below, this latter task of path completion is achievable only when the latter section of the full path (i.e., from f_i to the opposing end node of the path) has previously been found by other ants.
- The partial paths found by ants (i.e., paths found leading to/from a start/end node to an f_i) are stored using the method *storePartialPath(path1)* so that they may be used by other ants in the future to fully complete the different paths of G. A partial path is saved if it is the first time such a path has been discovered between a particular start/end node and an f_i, or if the partial path discovered is superior in quality to another, previously-discovered path between the same two nodes.
- Lastly, it is important to note that the only type of pheromone update carried out in the algorithm is one of a global nature (using the method *globalUpdate(path)*). In other words, the pheromone is only updated by the algorithm when a complete path has been found, from a start node to an end node. The formula governing this pheromone update appears below (see formula (2)), where $\tau_{ij}(t)$ is the pheromone for the edge l_{ij} at moment in time t, ρ is the dissipation rate of the pheromone and *length* is the length of the path encountered

$$\tau_{ij}(t) = \begin{cases} \tau_{ij}(t-1) \cdot (1-\rho) + \frac{constant}{length}, & n_j \in path \\ \tau_{ij}(t-1) \cdot (1-\rho), & other\ case \end{cases} \tag{2}$$

The decision to execute only global pheromone updates is due to the fact that the graph is extremely large and the number of ants required for the ACO algorithm is, therefore, necessarily high. Were local pheromone updates included as well, a large number of transactions would be needed and response times, as a result, would be negatively affected.

As can be observed in the search process presented, the fact that the value of $|S|$ or each $|s_i|$ may change during the search process does not imply that the search process must, therefore, come to a halt since, if a food odor cannot be detected, the ants continue to search according to the procedure spelled out by the classic ACO algorithm (in Fig. 1 this concurrency is observed clearly).

It is important to again briefly note the possibility that, following the search phase, the algorithm can, in fact, return to the odor diffusion phase. This return to an earlier phase in the process is due to the fact that if the ants used an s_i to reach their destination on the path obtained, a certain amount of food odor is diffused along the nodes of the path ("Diffusion of the Food Odor along the Path" state in Fig. 1), thereby increasing $|s_i|$ and making future searches quicker and easier.

Food odor is retroactively assigned to the nodes along the path obtained by following the path in the opposite direction, that is, beginning from the node with a food odor and arriving at the node from which the ant originally departed. From one node to the next along this inverse path, the food odor assigned decreases according to the same formula described earlier in Sect. 3.3, that is, $O(n_i) = O(n_j) - k \cdot w_{ij}$, where n_j is the node assigned a food odor directly preceding n_i, the node currently being assigned a food odor.

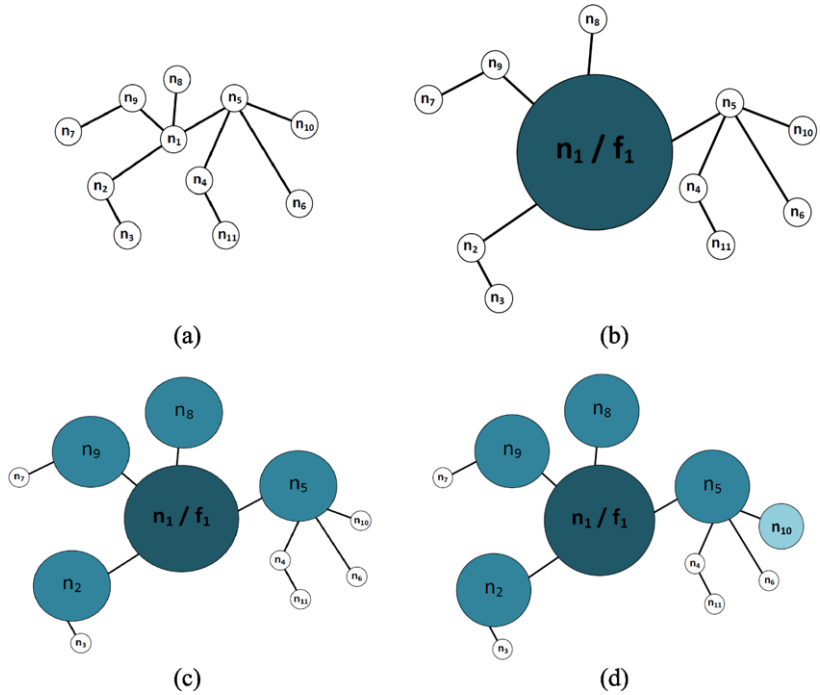
The principal difference between this particular extension and that generated in the second phase of the algorithm is that, in this particular case, the food odor assigned to nodes here need not be greater than or equal to u. Rather, it is necessary that its value simply be greater than zero.

3.5 Example of application

After describing the modified ACO algorithm in detail, this present sub-section aims to ensure the clear understanding of the algorithm and its component phases by means of the following example.

As demonstrated in Fig. 2(a) below, the example begins with an initial graph of 11 nodes, $N = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_{11}\}$, and 10 edges, $L = \{l_{1,2}, l_{1,5}, l_{1,8}, l_{1,9},$

Fig. 2 Diffusion of food odor



$l_{2,3}, l_{4,11}, l_{5,4}, l_{5,6}, l_{5,10}, l_{9,7}\}$, with the following weights: $W = \{w_{1,2}, w_{1,5}, w_{1,8}, w_{1,9}, w_{2,3}, w_{4,11}, w_{5,4}, w_{5,6}, w_{5,10}, w_{9,7}\} = \{2, 2, 2, 2, 4, 3, 4, 5, 1, 4\}$.

Having defined the graph used in the present example, the different phases of the algorithm are executed on the graph in the following way:

Selection of graph nodes with food sources. Observing Fig. 2(a), were the graph the representation of a social network, it could easily be deduced that node n_1 represented an important person, resulting in its selection as a node at which a food source should be located. In this way, the node enters the set F as f_1, such that $F = \{f_1\}$. Visually representing the node's greater importance and the maximal presence of food odor at this node (as is expected at nodes where a food source is located), Fig. 2(b) identifies n_1 with dark color and a larger size than that of the remaining nodes on the graph. Specifically, the amount of food odor present at the node is equal to $m : O(n_1) = O(f_1) = m = 10$.

Diffusion of food odor. Once the node with the food source has been selected, the following phase of the algorithm can be commenced whereby the food odor is diffused to other nodes on the graph. For this particular example where u and k values are set at $u = 6$ and $k = 1$, the steps to be followed are listed below:

Step 1
- *Backpack* $= \{n_1\}$.
- *Used nodes* $= \{\}$.

Step 2
- The node with the greatest amount of food odor in the backpack, n_1, is selected, such that $n = n_1$.
- $O(n_2) = O(n_1) - k \cdot w_{1,2} = 10 - 1 \cdot 2 = 8$.
- $O(n_5) = O(n_1) - k \cdot w_{1,5} = 10 - 1 \cdot 2 = 8$.
- $O(n_8) = O(n_1) - k \cdot w_{1,8} = 10 - 1 \cdot 2 = 8$.
- $O(n_9) = O(n_1) - k \cdot w_{1,9} = 10 - 1 \cdot 2 = 8$.
- *Backpack* $= \{n_1, n_2, n_5, n_8, n_9\}$.
- *Used nodes* $= \{n_1\}$.

In Fig. 2(c) below, in order to differentiate the nodes to which food odor has been diffused in this particular step from the other nodes of the graph (i.e., the nodes to which food odor arrives directly from f_1), the nodes with diffused food odor have been colored with a lighter color and are represented by circles that, while larger than those of the remaining nodes, are nevertheless smaller than that of the node where the food source is found. This is due to the fact that the node with the food source has more food odor than any of the four newly incorporated nodes in the backpack. Additionally, no differences can be observed in either color or size among the newly incorporated nodes due to the fact that, as demonstrated in the calculations above, each of them possesses the same amount of food odor.

Step 3
- The node with the greatest amount of food odor in the backpack not selected previously is chosen. In this case, as there remain various nodes with the same amount of food odor, one is selected at random such that $n = n_8$.
- No node exists that is reachable from n_8 that has not already been reached in earlier steps of the process. Thus, no (additional) food odor may be dispersed from this particular node to other nodes on the graph.

– *Backpack* $= \{n_1, n_2, n_5, n_8, n_9\}$.
– *Used nodes* $= \{n_1, n_8\}$.

Step 4

– The node with the greatest amount of food odor in the backpack not selected previously is chosen. Similar to Step 3, as various nodes with the same amount of food odor remain, one is selected at random such that $n = n_2$.
– $O(n_3) = O(n_2) - k \cdot w_{2,3} = 8 - 1 \cdot 4 = 4 < u = 6 \rightarrow O(n_3) = 0$.

This node is not added to the backpack since its odor is less than u. Therefore, neither this node nor any other nodes which hang from it exclusively may be included in s_1.

– *Backpack* $= \{n_1, n_2, n_5, n_8, n_9\}$.
– *Used nodes* $= \{n_1, n_8, n_2\}$.

Step 5

– The node with the greatest amount of food odor in the backpack not selected previously is chosen. Similar to the previous steps, as various nodes with the same amount of food odor remain, one is selected at random such that $n = n_5$.
– $O(n_4) = O(n_5) - k \cdot w_{5,4} = 8 - 1 \cdot 4 = 4 < u \rightarrow O(n_4) = 0$.
– $O(n_6) = O(n_5) - k \cdot w_{5,6} = 8 - 1 \cdot 5 = 3 < u \rightarrow O(n_6) = 0$.
– $O(n_{10}) = O(n_5) - k \cdot w_{5,10} = 8 - 1 \cdot 1 = 7$.

Of all the nodes reachable from n_5, only n_{10} is added to s_1. Insofar as the other two nodes sharing an edge with n_5 possess food odors determined to be less than u, they cannot be included in s_1. As shown in Fig. 2(d), the new addition to s_1 is colored lighter than the other nodes of s_1, yet distinct from the colorless nodes not in s_1, and is represented by a circle that is smaller than those of the other nodes in s_1, yet larger than the nodes not in s_1. In this way, n_{10}'s lesser odor than that of the other nodes in s_1 and greater odor than that of the nodes not in s_1 is visually represented.

– *Backpack* $= \{n_1, n_2, n_5, n_8, n_9, n_{10}\}$.
– *Used nodes* $= \{n_1, n_8, n_2, n_5\}$.

Step 6

– The node with the greatest amount of food odor in the backpack not selected previously is chosen such that $n = n_9$.
– $O(n_7) = O(n_9) - k \cdot w_{9,7} = 8 - 1 \cdot 4 = 4 < u \rightarrow O(n_7) = 0$.

Insofar as the food odor corresponding to n_7 is less than u, the node is not included in s_1.

– *Backpack* $= \{n_1, n_2, n_5, n_8, n_9, n_{10}\}$.
– *Used nodes* $= \{n_1, n_8, n_2, n_5, n_9\}$.

Step 7

– The only node not having been selected previously is chosen such that $n = n_{10}$.
– No node exists that is reachable from n_{10} that has not already been reached in earlier steps of the process. Thus, no new nodes can be included in the backpack.
– *Backpack* $= \{n_1, n_2, n_5, n_8, n_9, n_{10}\}$.
– *Used nodes* $= \{n_1, n_8, n_2, n_5, n_9, n_{10}\}$.

Since no new nodes can be included in the backpack, the food odor diffusion is concluded with the following results: a node with food odor $F = \{f_1 = n_1\}$ and $S = \{s_1\} = \{\{n_1, n_2, n_5, n_8, n_9, n_{10}\}\}$.

Path searches with each requested service. In this phase of the example, all requested path searches are executed. As discussed in earlier sections of this article, insofar as the performance of the proposed algorithm for path searches is not negatively affected by previous phases of the process, these searches may be requested during the execution of the food odor diffusion, during the selection of the node with the food source or following the diffusion. Nevertheless, and in order to demonstrate how the inclusion of food odor in the proposed modification enhances the path search process, the path search phase of the present example begins following the completion of the food odor diffusion, that is, using the graph in the state illustrated in Fig. 2(d).

Having clarified the graph state over which service requests are to be executed, the particular path to be requested by the service in the present example is path $p_{1,11}$ linking node n_{11} with node n_1.

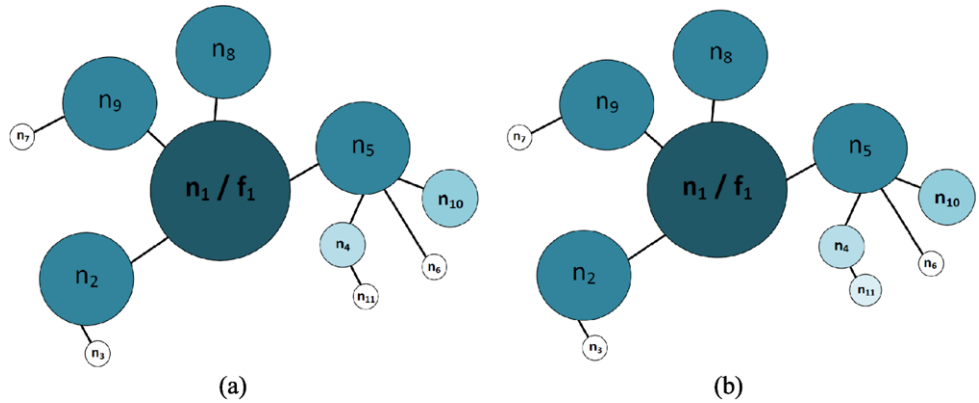
Following the phases described in Sect. 3.4 of the article, the path search in the present example is explained below:

Phase 1

– *Tabu* $= \{\}$.
  The empty set above refers to the initial tabu lists for each of the ants in the algorithm.
– The pheromone levels for each of the edges are set at *initFerom*.
– Given that one of the start/end nodes in the path search pertains to F, all of the ants are therefore placed on n_{11}.

Phase 2 Using the probabilistic equation presented earlier in formula (1), the ants begin to search for nodes on the requested path. This search is completed when an ant encounters a node with food odor since it is then easy for the ant to reach the node with the food source from a node with food odor. In the example, the fragment of the path $\{n_{11}, n_4, n_5\}$ is encountered. Having arrived at

Fig. 3 Inclusion of new elements in s_i



the node with food odor n_5, the ant terminates its use of the probabilistic formula and continues its search for the food source by following the path of increasingly intense food odor. In this way, the ant completes the path, obtaining $p_{11,1} = \{n_{11}, n_4, n_5, n_1\}$.

Since the zone with food odor has been used by the ants along the path obtained, following the completion of the service, the food odor is further diffused along the fragment of the path running from the start node to the first node where food odor has previously been detected. For this additional diffusion, the steps below are followed:

Step 1

First node encountered with food odor: n_5.

$s_1 = \{n_1, n_2, n_5, n_8, n_9, n_{10}\}$.

Step 2

Following node: n_4.

$O(n_4) = O(n_5) - k \cdot w_{5,4} = 8 - 1 \cdot 4 = 4$.

Since the food odor value obtained is greater than zero, the node is added to s_1.

$s_1 = \{ n_1, n_2, n_5, n_8, n_9, n_{10}, n_4\}$.

This new addition is illustrated in Fig. 3(a) in which the node n_4, colorless and small in previous representations of the graph, is now given a light color—lighter than the previously colored nodes yet darker than the nodes not included in s_1—and a larger size—smaller than the previously colored nodes yet larger than the nodes not included in s_1. This is due to the fact that the food odor present in n_4 is weaker than in nodes previously included in s_1 yet stronger than the other nodes not included in s_1.

Step 3

Following node: n_{11}.

$O(n_{11}) = O(n_4) - k \cdot w_{4,11} = 4 - 1 \cdot 3 = 1$.

Similar to Step 2 above, since the food odor value obtained is greater than zero, the node is added to s_1.

$s_1 = \{ n_1, n_2, n_5, n_8, n_9, n_{10}, n_4, n_{11}\}$.

In Fig. 3(b), the newly added node n_{11} appears with a light color and larger size than in previous depictions of the graph. Nevertheless, both the intensity of color and the size of n_{11} are less than those of the other nodes previously included in s_1. This is due to the weaker amount of food odor present in n_{11} with respect to all other previously included nodes.

Given that $|s_1|$ of the graph in question has increased to include the two new nodes, with food odor, n_{11} and n_4, future path searches are, therefore, made easier insofar as the ants need only to encounter n_{11} rather than travel all the way to n_5.

4 Evaluation of algorithm

After fully explaining the proposed algorithm, the present section of the article aims to demonstrate its proper functioning. To this end, a series of experimental path searches are executed with the proposed algorithm as well as other algorithms frequently used for path searches in graphs, such that the costs of the paths obtained by each algorithm and the times required for their obtainment may be compared. This section also studies the influence of different u values (i.e. threshold for food odor intensity), number of ants, and graph size on the proper functioning of the proposed algorithm as well as the proposal's scalability for graphs with a greater number of nodes and also, therefore, edges.

The proposal presented here is very specific and its goals are beyond the focus of most of the state of the art metaheuristics, there being no benchmark for evaluation. On the other hand, variants of ACO are usually compared with the same algorithm. Therefore, it was decided to compare the proposal with well-known and disseminated algorithm. Specifically, the comparison is made with:

Classic ACO algorithm [14]. This algorithm has been chosen in order to clearly demonstrate the importance of the improvements included in the version proposed here.

Dijkstra's algorithm. This algorithm has been selected due to its standing as the classic path search algorithm (e.g., database management systems generally offer Dijkstra's algorithm for path searches) as well as to the optimal costs of the paths it obtains. Thus, direct comparison between

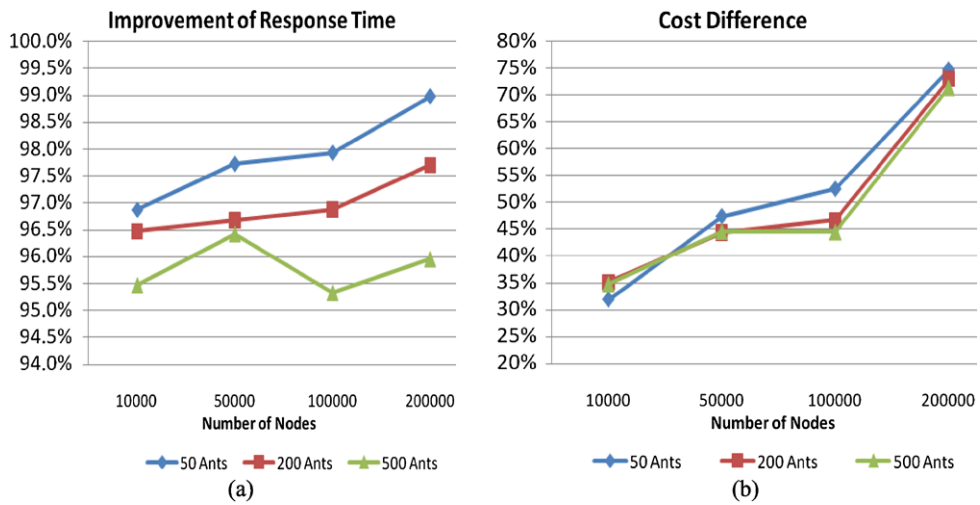


Fig. 4 Cost and response time of the first obtained path with the proposed algorithm with respect to Dijkstra

Dijkstra and the proposed algorithm will clearly show how close or far the results generated by the latter are from the optimum.

In the following sub-sections, these algorithms are executed over three distinct types of graphs: first, over a fictitious, generic graph of different sizes, and then over two different graphs corresponding to two distinct social networks, Slashdot [22] and Epinions [21].

4.1 Selection of number of ants

The first step before starting to test the algorithm is to set a value to some specific parameters of the way it works. Because the search algorithm used is based on ACO [12], it is important to set the number of ants to be used.

To set the value of this parameter, the algorithm is going to run on a generic graph of different sizes, and the best number of ants to be used will be the one which obtains the lowest path cost in the shortest period of time.

Regarding the test plan to be performed, 5 queries, each of 100 services (routes between two nodes of the graph), will run.

With respect to the generic graph, it is going to be generated so that there is an area with a higher concentration of nodes, and therefore links, in its center. With this design, the high degree of clustering in social networks will be simulated. To finish, the number of nodes that form the graph will take different values to see if there is any trend in the number of ants that is more appropriate in each case. The sizes of graphs considered are the followings: 10,000 nodes (with 30,000 links), 50,000 nodes (with 150,000 links), 100,000 nodes (with 300,000 links) and 200,000 nodes (with 600,000 links).

Figure 4 shows the results for 50, 200 and 500 ants on the graphs mentioned above: Fig. 4a shows the response time improvement of the proposed algorithm with respect to Dijkstra (algorithm used to choose the number of ants that makes being closer to the optimal cost possible), and Fig. 4b shows the difference in cost of the path obtained with respect to the optimal cost (obtained with Dijkstra).

That figure shows how the time used to give the first solution to the required path (Fig. 4a) increases when the number of ants used increases. This is to be expected because the processing that has to be carried out is greater. However, the response time is always better than the one required by Dijkstra (the value obtained is always over 95%).

With regard to the cost of the obtained path (Fig. 4b), its difference from Dijkstra decreases as the number of ants increases. This is because the probability of finding the best way from among all the possibilities increases when there are more ants looking for a solution.

Joining together the above comments in relation to Fig. 4, the following conclusion can be drawn: the response time is improved with respect to Dijkstra for any number of ants. However, the cost of the path is always higher than the cost obtained by Dijkstra, but it tends to be closer to the optimal value with the higher number of ants. For this reason, the number of ants in large graphs must be as great as possible.

Following this logic, in the next section 500 ants will be used because the graph used is that of 200,000 nodes (the selection of this graph is based on the fact that the algorithm requires testing in the worst conditions, that is, in the larger size graph).

Besides the number of ants that need to be used for the correct functioning of the algorithm, another conclusion can be drawn from the experimentation. This conclusion is that the algorithm is scalable with respect to the growth in the number of nodes/links. In fact the effect of growth in response time is practically inconsiderable, the cost grows linearly, and in all the experiments, all the services obtain a solution (the success rate is equal to 100%).

4.2 Experimentation with generic graph

The principal objective of this experiment is to demonstrate the quality of the path search solutions presented by the proposed algorithm for a generic graph, that is, a graph displaying no particular structure that could inherently favor the use of the proposed algorithm over any other. The quality of path search solutions obtained by the proposed algorithm, then, are determined experimentally through their comparison with the times and costs obtained by the application of Dijkstra's algorithm. While attempts were made to compare the proposed algorithm with the classic ACO algorithm as well, no results obtained from the application of the latter are presented in this sub-section. The reason for this decision is simple and clear: in more than 70% of the trials run with the classic ACO algorithm, the ants got lost due to the high volume of nodes to be visited and no results could be obtained, while in the remainder of the trials, the quality of the paths obtained was quite low. The results demonstrate what has already been noted in the literature; that is, the classic ACO algorithm is not suitable for large graphs.

For the generic scenario tested in the present experiment, a large, connected graph was created with 200,000 nodes and 600,000 bidirectional edges, such that the average number of edges per node in the graph is three.

With regard to the nodes of interest on the graph, only one was selected (i.e., only one element f_1 in F) and it was given a greater concentration of edges, that is, a particularly high connectivity.

With respect to the specific tests carried out for the present experiment, 10 queries with 1000 services each were executed for the graph presented above. The term *service* is used here to denote a request for a path search between two different nodes on the graph, where the end node is f_1 and the start node is selected randomly among all those appearing in the graph, thereby guaranteeing that no start node be repeated within a given query. This decision to designate f_1 as the end node of each of the path searches executed in the present experiment was made because, as in graphs representing social networks, nodes of interest are generally visited with greater frequency than others.

It is also important to highlight that the present experiment only takes the first path obtained by the ants into account. In other words, once a first solution has been obtained for a given service, the service is then immediately terminated and the following service is executed.

As for the aspects critical to the proper functioning of the proposed algorithm, it is first important to note that while the food odor in the graph is restarted for every new query, nonetheless, it is not restarted from one service to another. In this way, the evolution of the effect exerted by the expansion of $|s_1|$ on the quality of the paths and response times obtained can be carefully observed over the duration of the query. Secondly, and with regard to the pheromone trails in the graph, each is restarted at the beginning of every service. Additionally, the values assigned to the principal parameters of the proposed algorithm are shown in Table 1.

Table 1 Algorithm parameters

Parameter	Value
$t_{threshold}$	700 sec
#ants	500
ρ	0.6
m	1,000,000
k	100%

Table 2 Times in food odor diffusion

u	Nodes with food odor (%)	Time (sec)
999,700	1	11.5
998,600	17	178.3
998,000	30	326.6

It is significant that the value assigned to $t_{threshold}$ is approximately equal to the time taken by Dijkstra's algorithm to produce a solution. In this way, the ants of the extended ACO algorithm proposed here may never take longer than the time required to obtain a solution of optimal quality.

Additionally, no specific u value (i.e., the minimum threshold value for food odor diffusion) appears in Table 1. This is due to the fact that, as mentioned earlier, the functioning of the proposed algorithm with different u values is to be analyzed separately. The u values to be compared, then, can be seen below in Table 2.

With each distinct u value, different values of $|s_1|$ are obtained, such that the smaller the u value, the larger the number of nodes contained in $|s_1|$. As can be observed in Table 2, this increased number of nodes results in an increase in the time necessary to carry out the food odor diffusion. These times deserve careful consideration when drawing conclusions about the benefits or lack thereof of increased u values for the proposed algorithm with regard to the path quality and response times obtained.

Once the diffusion has been carried out, the different queries are executed and the following conclusions may be drawn:

Rapid evolution of path costs and response time throughout services (Fig. 5). Clearly observable from the graph is the ability of the proposed algorithm to rapidly diminish and stabilize costs and response times after the fiftieth service. The importance of the additional, retroactive food odor diffusion carried out at the end of each service is, therefore, confirmed.

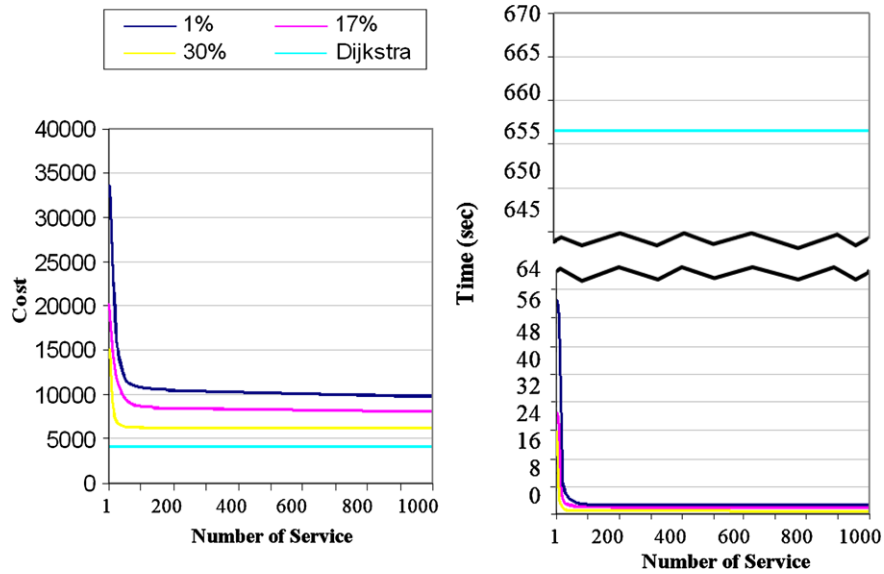


Fig. 5 Evolution in path quality and response time of first path obtained for each service executed

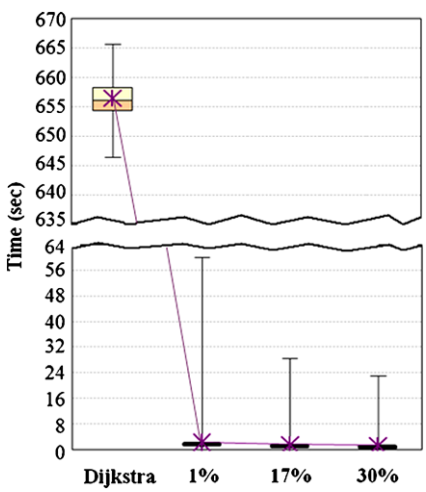


Fig. 6 Response times for first path obtained in each service

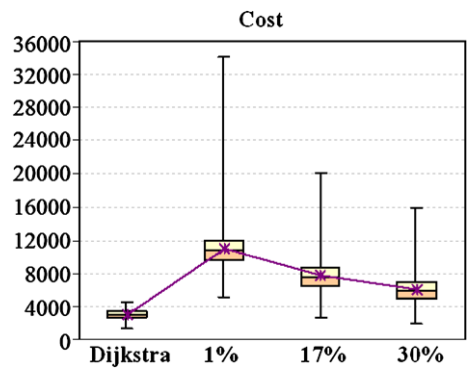


Fig. 7 Cost of first path obtained in each service

Furthermore, the influence of u values on this evolution is demonstrated by the graphs in Fig. 5 to be insignificant insofar as the algorithm results began to stabilize at approximately the same service number for each different u value. The only observable difference between different u values, then, is with respect to the cost at which the proposed algorithm results stabilize, being slightly lower for lower u values. These results are studied below in greater detail.

Observable reduction in response time (relative to Dijkstra) for all u values (Fig. 6), despite need for improvements in currently non-optimal obtained path costs (Fig. 7). One of the likely explanations for the non-optimal costs obtained by the proposed algorithm relative to Dijkstra is that, in the series of tests conducted for this study, the selected path is always the first to be obtained by the ants. It is for this reason that a future study is discussed in the last section of this paper in order to attempt to evaluate the cost evolution in consecutive solutions obtained by ants for the same service. Additionally, the number of nodes which the ants must visit in order to reach their destination is quite high due to the generic structure of the graph used in this particular experiment. As a result, the behavior of the algorithm with the generic graph is understandably less optimal than the expected behavior of the algorithm with graphs representing small-world networks [23], that is, the sort of graphs to which the proposed algorithm is designed to be applied and whose results are presented below in Sect. 4.3.

As a final comment, it is important to add that the influence of the different u values on cost and response times for paths attained by the algorithm is not comparable to its influence on the time necessary to execute the food odor diffusion. While costs and response times for the paths obtained with smaller and larger u values are approximately the same, such is not the case with regard to the time necessary to execute a food odor diffusion, which is significantly lower with the use of higher u values. Thus, in the case of considering the introduction of new food nodes, or the deletion of old ones,

or the change in the number of nodes affected by the odor diffusion, it is advisable to select larger rather than smaller u values as they make the algorithm more agile and adaptable to these changes within the graph.

4.3 Experimentation with a real-world graph

Having demonstrated the high performance of the proposed algorithm with a generic graph, the present sub-section observes and analyzes its behavior on a graph representing a real-world scenario. Given that the proposed algorithm was designed and selected specifically for its application to these sorts of graphs, the results obtained are, therefore, of particular significance.

For this real-world scenario, a graph representing a social network was selected in order to address the two principal characteristics repeated throughout the earlier sections of this article: that is, the graphs' large size and the relatively low number of edges between any two nodes.

The social network used in the present experiment is the Slashdot social network as it was encountered in February of 2009, as presented in [22]. The graph representing the network in this state is composed of 82,168 nodes and 948,464 edges, and it possesses a small-world structure.

For the present experiment, all three algorithms presented earlier—Dijkstra's algorithm, the classic ACO algorithm and the proposed algorithm with different u values—are used. With regard to the parameters affecting the functioning of the proposed algorithm, the values used are those previously recorded in Table 1. The table, it must be noted, is also important for the classic ACO algorithm insofar as it contains a number of parameters—such as $t_{threshold}$, number of ants and pheromone evaporation rate—which are pertinent to it as well.

The tests run in this experiment were identical to those carried out on the generic graph of the previous sub-section. The restarting of food odor and pheromone were also handled the same way as before.

The present experiment presents an F with only one element corresponding to the person with the greatest number of edges within the graph of the social network and serving as the end node for all services requested in the experiment.

Concerning the different u values for the proposed algorithm, as Table 3 indicates, two such values are used in the present experiment and correspond to a food odor diffusion covering 3% and 37% of the nodes on the graph respectively. As noted earlier in Sect. 4.2, despite the expected improvements in path search solutions obtained with the application of lower u values, differences in food odor diffusion times are nonetheless highly significant and ought to be seriously considered when evaluating the advisability of using lower rather than higher u values.

Detailed comparisons of the performance of these four algorithms—Dijkstra, classic ACO, proposed ACO with 3% initial diffusion and proposed ACO with 37% initial diffusion—are presented below in Tables 4, 5 and 6.

Table 3 Amount of food odor diffused throughout the graph

u	Nodes with food odor (%)	Time (sec)
999,999	3	17.4
999,998	37	234.1

Table 4 Costs of first path obtained for each service

	Dijkstra	ACO	Proposed ACO 3%	Proposed ACO 37%
Mean	3.18	18.97	3.62	3.18
SD (standard deviation)	0.13	2.82	0.20	0.14
% SD	4.20%	14.85%	5.41%	4.30%
Median	3.20	18.78	3.60	3.20
Q1	3.10	16.95	3.48	3.10
Q3	3.30	20.58	3.75	3.30
Minimum	3.00	12.03	3.18	3.00
Maximum	3.70	32.15	4.38	3.75

Table 5 Worsening of costs of first path obtained relative to Dijkstra

	ACO	Proposed ACO 3%	Proposed ACO 37%
Mean	100.45%	12.06%	0.16%
SD (standard deviation)	51.27%	3.46%	0.64%
% SD	51.04%	28.67%	387.11%
Median	91.00%	12.06%	0.00%
Q1	65.93%	9.77%	0.00%
Q3	118.67%	14.29%	0.00%
Minimum	23.48%	2.22%	0.00%
Maximum	494.44%	22.98%	6.38%

In Table 4, it can be observed that the proposed algorithm's cost is practically equal to the optimum obtained by Dijkstra. While the results obtained by the algorithm with 37% of the nodes initially assigned a food odor are closer to the optimum than those obtained by the algorithm with only 3% food odor coverage, the difference between the two in relation to the optimum is minimal. Due to the time required to diffuse food odor in the two cases (see Table 3), the proposed algorithm with only 3% food odor coverage is actually more advisable here. As the algorithm with only 3% coverage allows for faster food odor diffusion from f_1 than the algorithm with 37% coverage, it therefore possesses a greater adaptability to changes in the food nodes (includ-

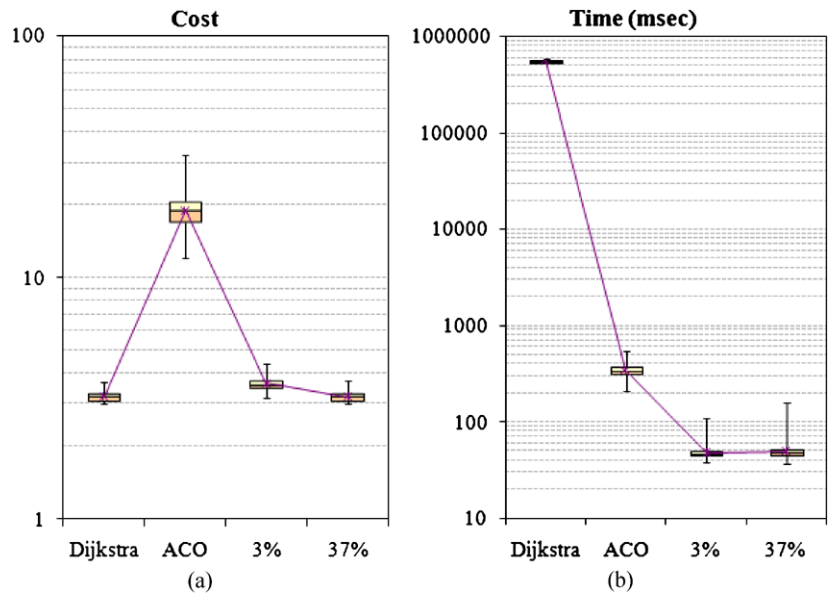


Fig. 8 The logarithmic scale for costs and response times for first path obtained for each service in each algorithm used

Table 6 Response times (msec) of first path obtained for each service

	Dijkstra	ACO	Proposed ACO 3%	Proposed ACO 37%
Mean	551284.80	345.78	47.55	48.55
SD (standard deviation)	6338.63	50.31	4.81	6.34
% SD	1.15%	14.55%	10.11%	13.05%
Median	550293.10	340.65	46.85	47.58
Q1	546963.00	311.24	44.47	44.84
Q3	554854.50	375.83	49.98	51.10
Minimum	537232.90	210.55	38.25	36.58
Maximum	577875.60	545.60	109.65	157.05

ing new food nodes, deleting old food nodes, changing the number of nodes affected by odor diffusion).

It is important to note the great improvements gained by the proposed ACO algorithm over the classic ACO. As can be observed in Table 4, the difference in costs obtained by the classic ACO and either of the two modified ACO algorithms is large despite the fact that the topology of the graph used here could be considered adequate for the classic ACO due to the low number of edges that must be traversed in order to get from one node to another (e.g., the worst case recorded in [22] is a path comprised of twelve different edges).

While Fig. 8(a) clearly shows the differences in the performance of the proposed and classic ACO algorithms, another positive characteristic of the proposed algorithm observable in Fig. 8(a) and present for both of the u values used is that the variability of results obtained is similar to that recorded in Dijkstra. This factor requires consideration since it may be concluded that there is a high probability that any service requested is the optimum or very close to the optimum.

In order to more clearly see how far the costs obtained both by the proposed algorithms and by the classic ACO algorithm fall from the optimum obtained with Dijkstra, Table 5 represents the cost differences as a percentage.

The table makes clear that while the classic ACO algorithm differs from the optimal value by a high percentage, the proposed algorithms actually come quite close to the optimum. More importantly, the distance from the optimum becomes nearly imperceptible in the case of the proposed algorithm with food odor coverage across 37% of the nodes.

It must be reiterated that the calculations presented here include the results of all services requested, including the first ones. Thus, it is understandable that the maximum values recorded are rather high since, as illustrated earlier in Fig. 5, the results yielded from the first services are always worse than those yielded from subsequent services.

Once costs have been fully discussed, it is important to analyze the times the different algorithms require to obtain a solution to a path search. With regard to this parameter, both Table 6 and Fig. 8(b) make it abundantly clear that the proposed algorithm is superior to the others. It must be noted, however, that the difference between the times required by the classic and proposed ACO algorithms is not particularly great. Nevertheless, the previously discussed difference in obtained path costs makes it clear that the proposed ACO algorithm with either of its u values is superior to its classic counterpart.

From the results presented and analyzed here, therefore, the advantages offered by the proposed algorithm relative to the other two algorithms studied, when applied to a real-world domain, become fully evident.

4.4 Performance on real-world graphs of different sizes

Having demonstrated the proper functioning and high performance of the proposed algorithm in terms of costs and path search times relative to both the classic ACO algorithm and Dijkstra's algorithm, which is used by most database management systems, it is now necessary to determine if the modified ACO algorithm is indeed capable of being used on real-world graphs of any size.

In order to do so, and just as in the experiments discussed above, 10 queries with 1000 services each were executed with start nodes selected at random from all possible nodes and the end node f_1 where $F = \{f_1\}$ (i.e., where f_1 was the only node with a food source on the graph).

For the present experiment, two graphs representing two distinct social networks—Slashdot [22], used in the previous experiment, and Epinions [21]—were used. While the first represents a classic social network, the second represents a trust network, a type of social network which always maintain a small-world topology! [32].

Insofar as the present experiment aims to ascertain how an increase in the number of nodes on a graph influences response times and the ability to obtain paths, only the proposed algorithm is tested here.

Using the parameters from Table 1 and u values of 999,999 and 999,998, the following experimental results shown in Table 7 were obtained.

As observable in Table 7, in each of the scenarios and regardless of the u value used, the proposed algorithm's path search success rate is the same, with paths obtained in 100% of requested services.

Considering the average times and standard deviations recorded in Table 7, it can be observed that the time required for the algorithm to obtain a path is nearly the same for the different u values used. The relatively insignificant time differences recorded, therefore, justify the use of either of the two u values.

Finally, comparing the response time data obtained for the two different graphs used, not only is the difference between recorded times quite small, but the time taken to obtain paths on the Epinions graph is actually lower than that taken for the Slashdot graph despite the fact that the former possesses a larger number of nodes than the latter ($|N|_{Epinions} = 131{,}828$ and $|N|_{Slashdot} = 82{,}168$). Consideration of the number of edges $|L|$ in the two graphs, however, where $|L|_{Epinions} = 841{,}372$ and $|L|_{Slashdot} = 948{,}464$ (i.e., $|L|_{Epinions} < |L|_{Slashdot}$), reveals that it is not the number of nodes, but rather the number of edges in a graph that affects response times, although its influence is minimal.

Table 7 Scalability tests

u	Slashdot		Epinions	
	999,999	999,998	999,999	999,998
Avg. time	47.55	48.55	37.84	41.04
SD (standard deviation)	4.81	6.34	5.87	6.86
Success rate (%)	100%	100%	100%	100%

With these results, it may be concluded that the proposed algorithm is indeed scalable. While the increase in the number of edges on a graph exerts a slightly negative effect on path search response times, the success rate nevertheless remains the same with paths obtained in 100% of the services executed. This is a great improvement over other proposals [26] whose success rates decrease with an increase in the number of nodes on a graph.

5 Conclusions and future research

This article has presented an adaptation of the classic ACO algorithm with the objective of creating an algorithm capable of working with large graphs with a high degree of clustering, that is, graphs with hundreds of thousands or millions of nodes with a high degree of centralization.

In this proposed modification, two new concepts, *food* (i.e., the set F) and *food odor* (along with the related concept of food odor diffusion), have been added in order to reduce the number of steps an ant must take in order to reach its destination. As has been shown, the introduction of these new concepts allows the algorithm to be used on large graphs without the need to change the structure of the graph or carry out a complex pre-processing.

With these two new concepts inspired from the observation of animal behavior, the proposal has extended the classic ACO algorithm by equipping the ants with a sense of smell. Thus, to the ants' natural pheromone-tracking capabilities, the extension adds the further ability of other animals with a sense of smell to follow an ever-intensifying trail of food odor to the source from which that odor emanates. Thanks to these additions, the following two advantages have been obtained for the proposed algorithm which the classic ACO algorithm does not offer. First, the search space over which an ant must work is greatly reduced, thanks to the presence of zones with food odor in the graph. As a result, the algorithm can be successfully applied to large graphs. Second, and resulting from the aforementioned advantage, due to the fact that an ant must cover a significantly lower number of edges in the graph in order to reach its destination, the resulting path search times and path costs are also significantly reduced.

In order to demonstrate each of these advantages, a series of experiments was run and confirmed the superiority of the proposed algorithm over the other algorithms studied with respect to response times. Additionally, the experiments show that when the algorithm is applied to real-world scenarios, nearly optimal costs can be obtained. Moreover,

the algorithm has been proven to be scalable with paths obtained in 100% of all services executed.

Despite these achievements, and with the aim of further improving the results obtained in the experimental phase of this paper, it would be particularly interesting to see how the cost evolution for a path obtained is affected by allowing the ants to work for longer periods of time rather than immediately taking the first solution they obtain as was done in the present research. Additionally, more elements could be included in F, with the corresponding elements in S, in order to observe the influence of $|F|$ on the functioning of the algorithm.

Finally, an important conclusion extracted from the study of the proposed algorithm is that this algorithm may well have good results on dynamic graphs due to the particular characteristics of the food nodes, of odor diffusion and of the ACO algorithm. This would expand its scope of application, and would be even more appropriate for the case of social networks since they vary with time (users (nodes) and relationships (links) between old and/or new users that continuously appear/disappear). This conclusion derives from the fact that the search algorithm can continue the path search regardless of any changes affecting the odor or the food nodes because the graph on which the ants work is not restructured, only odor diffusion is necessary. Furthermore, the ACO algorithm has demonstrated its good performance in dynamic environments (reflected in different scenarios such as ad-hoc networks [11], the classical travelling salesman problem [3], electrical distribution systems management problem [15] or dynamic vehicle routing [8, 20]). For these reasons, the use of the proposed algorithm in huge dynamic graphs would be an interesting, and possibly satisfactory, future line of research.

Acknowledgements The authors would like to thank two anonymous reviewers and the editors for their helpful suggestions during the preparation of this article. This study was funded through a competitive grant awarded by the Spanish Ministry of Education and Science for the THUBAN Project (TIN2008-02711) and through MA2VICMR consortium (S2009/TIC-1542, http://www.mavir.net), a network of excellence funded by the Madrid Regional Government.

References

1. Adamic L, Adar E (2005) How to search a social network. Soc Netw 27(3):187–203
2. Alba E, Chicano F (2007) ACOhg: dealing with huge graph. In: Proceedings of the genetic and evolutionary computation conference of 2007, pp 10–17
3. Angus D, Hendtlass T (2005) Dynamic ant colony optimisation. Appl Intell 23(1):33–38
4. Bast H, Funke S, Matijevic D, Sanders P, Schultes D (2007) In transit to constant shortest-path queries in road networks. In: Proceedings of workshop on algorithm engineering and experiments of 2007
5. Chan EPF, Lim H (2007) Optimization and evaluation of shortest path queries. VLDB J 16(3):343–369
6. Chan EPF, Zhang J (2007) A fast unified optimal route query evaluation algorithm. In: Proceedings of the 16th ACM conference on conference on information and knowledge management, pp 371–380
7. Chang R-S, Chang J-S, Lin P-S (2009) An ant algorithm for balanced job scheduling in grids. Future Gener Comput Syst 25(1):20–27
8. De Oliveira SM (2009) A study of pheromone modification strategies for using ACO on the dynamic vehicle routing problem. In: Doctoral symposium on engineering stochastic local search algorithms of 2009, pp 6–10
9. Delling D, Sanders P, Schultes D, Wagner D (2006) Highway hierarchies star. In: The shortest path problem: 9th DIMACS implementation challenge. DIMACS book, vol 74, pp 141–174
10. Delling D, Holzer M, Müller K, Schulz F, Wagner D (2009) High-performance multi-level routing. Ser Discrete Math Theor Comput Sci 74:73–92
11. Di Caro G, Ducatelle F, Gambardella LM (2005) AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. Eur Trans Telecommun 16(5):443–455. Special Issue on Self Organ Mob Netw
12. Dorigo M (1992) Optimization, learning and natural algorithms. Doctoral Thesis, Dipartamento di Elettronica, Politecnico di Milano, Italy
13. Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comput Sci 344:243–278
14. Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge
15. Favuzza S, Graditi G, Sanseverino E (2006) Adaptive and dynamic ant colony search algorithm for optimal distribution systems reinforcement strategy. Appl Intell 24(1):31–42
16. Feng G, Li C, Gu Q, Lu S, Chen D (2006) SWS: small world based search in structured peer-to-peer systems. In: Proceedings on the international conference on grid and cooperative computing workshops of 2006, pp 341–348
17. Ippolito MG, Morana G, Riva Sanseverino E, Vuinovich F (2005) Ant colony search algorithm for optimal strategical planning of electrical distribution systems expansion. Appl Intell 23(3):139–152
18. Jaén J, Mocholí JA, Catalá A, Navarro E (2011) Digital ants as the best cicerones for museum visitors. Appl Soft Comput 11(1):111–119
19. Kautz H, Selman B, Shah M (1997) Referral Web: combining social networks and collaborative filtering. Commun ACM 40(3):63–65
20. Lee C-Y, Lee Z-J, Lin S-W, Ying K-C (2010) An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem. Appl Intell 32(1):88–95
21. Leskovec J (2010) SNAP: network datasets: epinions social network. Stanford University. http://snap.stanford.edu/data/soc-sign-epinions.html. Accessed 09 November 2010
22. Leskovec J (2010) SNAP: network datasets: slashdot social network. Stanford University. http://snap.stanford.edu/data/soc-Slashdot0902.html. Accessed 09 November 2010
23. Newman MEJ (2003) The structure and function of complex networks. SIAM Rev 45(2):167–256
24. Ramos GN, Hatakeyama Y, Dong F, Hirota K (2009) Hyperbox clustering with ant colony optimization (HACO) method and its application to medical risk profile recognition. Appl Soft Comput 9(2):632–640
25. Rivero J (2009) Fast search of paths through huge networks. In: Doctoral symposium on engineering stochastic local search algorithms of 2009, pp 46–50
26. Sandberg O (2006) Distributed routing in small-world networks. In: Proceedings of the 8th workshop on algorithm engineering and experiments, pp 144–155

27. Sankaranarayanan J, Samet H (2009) Distance oracles for spatial networks. In: Proceedings of the 25th IEEE international conference on data engineering, pp 652–663
28. Sankaranarayanan J, Samet H, Alborzi H (2009) Path oracles for spatial networks. In: Proceedings of the 35th international conference on very large data bases, pp 1210–1221
29. Supratid S, Kim H (2009) Modified fuzzy ants clustering approach. Appl Intell 31(2):122–134
30. Tang L, Liu H (2010) Graph mining applications to social network análisis. In: Aggarwal C, Wang H (eds) Managing and mining graph data. Advances in DataBase systems, vol 40, pp 487–514
31. Yu JX, Cheng J (2010) Graph reachability queries: a survey. In: Aggarwal C, Wang H (eds) Managing and mining graph data. Advances in DataBase systems, vol 40, pp 181–215
32. Yuan W, Guan D, Lee Y-K, Lee S (2010) The small-world trust network. Appl Intell 1–12. ISSN 0924-669X
33. Zhang N, Feng Z-R, Ke L-J (2010) Guidance-solution based ant colony optimization for satellite control resource scheduling problem. Appl Intell 1–9. ISSN 0924-669X

Jessica Rivero The Ph.D. Student Jessica Rivero got her degree in Telecommunication Engineering from Carlos III University of Madrid in 2005. In 2006, she joined the Advanced Databases Group, at the Computer Science Department of this university, where she currently works as assistant teacher. In 2007, she obtained the Master in Computer Science and Technology. Her research interests include advanced database technologies, spatio-temporal databases, metaheuristics algorithms and their applications to Situation management. During her stay in IRIDIA group (CoDE department of the Université Libre de Bruxelles (Belgium)) she validated her research in Spatio-Temporal databases and metaheuristics.

Dolores Cuadra received the M.Sc. in Mathematics from Universidad Complutense of Madrid in 1995. In 1997, she joined the Advanced Databases Group, at the Computer Science Department of Carlos III University of Madrid, where she currently works as lecturer. In 2003, she obtained the Ph.D. degree in Computer Science from Carlos III University of Madrid. Her research interests include advanced database technologies, spatio-temporal databases and their applications to Situation management. She has been working in Computer Science Department at Purdue University of West Lafayette (Indiana) for nearly a year, where she has applied her research in Spatio-Temporal database.

Javier Calle got his degree in Computer Science from the Technical University of Madrid in 1997. In 2000, he joined the Advanced Databases Group in the Computer Science Department at the Carlos III University of Madrid, where he is currently working as lecturer. In 2005, he obtained the Ph.D. degree in Computer Science from Technical University of Madrid, with a thesis in the Natural Interaction research. Apart from this, his research topics are focused on dialogue modeling, intentional processing and joint action models, and cognitive components of the interaction. He has also been working in several European and National research projects regarding Human-Computer Interaction.

Pedro Isasi is engineer in Computer Science for Polytechnic University of Madrid since 1991 and received his Ph.D. at this same university in 1994. Now, he is a Professor in Computer Science at Carlos III of Madrid University. He is funder and director of the Neural Network and Evolutionary Computation Laboratory. His principal researches are in the field of Machine Learning, Metaheuristics and evolutionary optimization methods, mainly applied to the field of finance and economics, forecasting and classification.