Contents lists available at ScienceDirect

Expert Systems With Applications

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

Extending ACO for fast path search in huge graphs and social networks

Javier Calle, Jesica Rivero, Dolores Cuadra*, Pedro Isasi

Computer Science Department, Universidad Carlos III, Avda. Universidad 30, Leganés 28911, Spain

ARTICLE INFO

Article history:
Received 6 July 2016
Revised 27 April 2017
Accepted 27 May 2017
Available online 31 May 2017

Keywords:
Huge graphs
ACO
Bio inspired algorithms
Path search
Social networks

ABSTRACT

This paper presents the bio-inspired algorithm SoSACO-v2 that is explained as an extension of the Ant Colony Optimization in which the ants are empowered with the sense of smell, directing them straightly to privileged nodes when they are near enough of them. This algorithm is an evolution of a former version which main feature is efficiency through path search task in huge graphs of high connectivity. New requirements regarding this task in most applications include processing vast graphs, immediate comeback, and dealing with dynamicity. The here proposed algorithm gives response to new needs the former approaches cannot fulfill: fast finding of paths between two nodes through vast dynamic graphs. SoSACO-v2 does not provide the optimum path, but it is the quicker algorithm in providing a response. It stands for domains where optimality is not required, and often the path search takes more time than covering the path. The approach is evaluated, both in a generic huge graph and in a small-world type graph from a real social network, showing satisfactory results.

1. Framework and motivation

One of nowadays topics in Computer Science involves representing and managing efficiently and effectively large volumes of information. A largely user formalization is the mathematic notion of *graph*. Its representation capabilities range from plain geographical maps to any conceptual map, where the information concepts are nodes and the relationships between them are links.

With this information modeling, relating a pair of concepts is solved as the common problem of searching a path between the nodes representing those concepts. If the graph supports a navigation system, the path is the route to be followed by the user to reach a place from another. In social networks, the path would represent the chain of links relating two users, and can also be applied to find a person that satisfies certain characteristics (which are searches for specific or generic paths in the graph, respectively). Because of its versatility and wide spread use, this formalization has been the focus of many research works, making available a considerable amount of tools for its processing and management. As requirements evolve, the technology is refined and the tools enhanced aimed to attain certain goals.

A recent goal is to observe huge graphs, comprising million of nodes. Processing those graphs raises the essential requirement of efficiency, given that in many cases the time required to apply a solution (path) is much lower than the time required to find the best solution. In other words, it is preferable to find a fast solution rather than to find the optimum path. In another vein, some systems require solutions in a limited time, so they can take advantage of algorithms able of providing a quick response and then refine it as much as possible until time runs out. The quick increase of hardware capabilities is not enough to give response to the even quicker increase of requirements in terms of scalability (bigger problems) and efficiency (faster response time). Besides, new requirements, as dealing with dynamicity, force to explore new algorithms.

Classical path search algorithms are mostly focused in searching for the best solution, which entails serious limitations. On one hand, they don't scale well, showing poorer response times as the size of the graph grows. Moreover, they can become useless when dealing with very high dimension graphs and limited response time, because they take more time than provided in finding any solution (Dorigo & Blum, 2005). Even evolved versions of these algorithms (supported either by fragmentation of graph, pruning, or preprocessing) show these weaknesses. On the other hand, in dynamic environments their limitations are even greater (Kar, 2016): when they reach a solution, its validity has often expired (some of the nodes/arcs included in the solution may have disappeared). Some other approximate methods are more capable of dealing with dynamicity, but they do not behave very well when the dimension is really high.

* Corresponding author.
E-mail addresses: fcalle@inf.uc3m.es (J. Calle), jess.rivero@gmail.com (J. Rivero), dcuadra@inf.uc3m.es (D. Cuadra), isasi@ia.uc3m.es (P. Isasi).

http://dx.doi.org/10.1016/j.eswa.2017.05.066
0957-4174/© 2017 Elsevier Ltd. All rights reserved.

This paper describes an extension of the 'Sense of Smell - Ant Colony Optimization' (hereinafter SoSACO) algorithm (Rivero, Cuadra, Calle, & Isasi, 2011), which in turn extends the well-known ACO algorithms (Dorigo & Stützle, 2004). The former SoSACO algorithm is aimed to provide fast paths linking two nodes within huge graphs. Apart from ACO main features, it is supported by a preprocessing that attaches to some nodes clues about the location of other privileged nodes (the *food nodes*). Such auxiliary information does not affect the network structure, and can be updated at runtime to not impair the dynamic characteristics of the ACO algorithms. The idea also comes from biological inspiration, specifically on how some animal species direct their foraging guided by the smell coming from the food. However, earlier versions of this algorithm were restricted to services involving privileged nodes (starting from or arriving at a food node). This work presents and evaluates a general algorithm able of fast pathfinding between any two nodes within a huge graph, and also able of refining the solution if extra-time is available. Therefore, the applications are those in which it is necessary to find a solution as quickly as possible, and also those in which there is a certain time limit and it is advantageous to be improving the solution before the deadline expires.

On the other hand, interest in social networks has boosted developments adapted to (or suitable for) the small-world topology (Newman, 2003; Tang & Liu, 2010), which characterizes these networks. A small-world network is a graph in which most nodes are not neighbors to each other, but almost all nodes can be accessed from any other node through a relatively short number of steps. Again, one of most required services for these networks is the path search (for finding individuals with an affinity (Kautz, Selman, & Shah, 1997)), for which there are some specific algorithms (Feng, Li, Gu, Lu, & Chen, 2006; Sandberg, 2006) limited to small/medium size graphs. However, these networks must frequently be represented by means of huge graphs in which every two users are related through short paths.

The concept of small-world network was introduced by Watts and Strogatz (1998), though it had been suggested long before by a social psychologist Milgram (1967) who discovered the called *'six degrees of separation'* principle, which states that every node can be reached with six steps (on average) from any other node. Gladwell (2000) pointed out that this phenomenon is due to the existence of extraordinary people (connectors) *hubbing* the network by holding thousands of links. Three important features define real-world complex networks: short average path lengths, high clustering coefficient, and free-scale degree distribution (Wang & Chen, 2003). The average path length is defined as the average number of steps along the shortest paths for all possible pair of nodes in the network; clustering coefficient is the extent to which the nodes of a graph tend to cluster together; and finally, the degree distribution is the probability distribution of node's degrees (when defined by a power law of the form $P(k) \sim k - \gamma$, it is called free-scale network). In the field of *complex networks*, the small-world effect and the free-scale degree distribution cover many real-world complex networks (Albert & Barabási, 2009). The here proposed algorithm can take advantage of these features to find paths in huge networks with little time consumption (and improve the solution if more time is provided).

In fact, these concepts are similar to those supporting the first version of the SoSACO algorithm, which can be briefed as setting food nodes easily reachable from anywhere (applied to the quickest path search when one of these nodes is either the start node or the goal). Because of such adequacy, and of the increasing importance of social networks, it has been found this field as an interesting evaluation case, along with the general topology case, for showing both the efficiency and the versatility of the SoSACO's family algorithms.

Several systems and applications solving pathfinding in huge graphs find most of their service requests featured by *relevant nodes*, that is, either the start node or the destination belong to a reduced subset of nodes. However, it is also true that there are many applications in which the definition of the services is completely random, providing paths between any two nodes in the graph. For this reason, it was found as a natural extension of that algorithm SoSACO overcoming that restriction without serious impairment of its proved efficiency. It should be remarked that the here explored focus is to minimize the time consumed in finding the path, because for many applications executing the path is less costly than waiting for a better one (or the optimum). Besides, the secondary goal is to find a good quality solution, as close to optimal as possible while respecting the minimum time cost.

Through the following sections, and after a quick review of the theoretical background and related work, the algorithm design and formalization will be exposed. All related mechanisms are explained in Section 4, highlighting differences to other similar approaches. Then, its performance will be evaluated for finally discussing its benefits, its application perspective and the current space for improvement it shows.

2. Background

Given the long tradition in the use of graphs as a tool to formalize problems of various kinds, over the years there have been proposed many algorithms for processing them. With the more recent development of computer technology, this trend has not been slowed but rather boosted, due to the versatility of this tool and its ease of use and implementation. Specifically, there exist a great variety of algorithms for path finding in graphs. As needs and the definition of the problems to be solved have evolved, so have the requirements to impose on these algorithms.

Algorithms appearing in a first period were aimed to provide the optimal solution or to demonstrate that there is no solution. With the growth of the problems dimension, and because of the importance of response time, more efficient algorithms were sought to avoid exploring the entire graph. These algorithms were usually based on extensions of the former supported by some preprocessing, and often aimed to fragment the graph in sub-graphs. In the meanwhile, algorithms based on heuristics for pruning the search tree were proposed. These algorithms are able of producing quasi-optimal solutions (optimality could be lost in some cases) that were obtained in less time (although eventually they could lead to slow full scans). Typically, the instances of the problem to be solved that produce poor results when applying heuristics are theoretical in nature, or at least they are infrequent, and therefore the application of these techniques is advantageous and quite widespread.

However, for certain problems it is either hard to find or to implement effective heuristics, or perhaps the problem changes structurally over time affecting the effectiveness of the heuristic. For this reason, recently there has been proposed a variety of metaheuristic algorithms of general nature, which are able to adapt their working to diverse problems and even to different states of the same problem. These algorithms are not most adequate in many cases for which a raw heuristic is available, but they provide an alternative way of achieving a solution, which could eventually be the most efficient (or even the only one). In the following, some typical examples of each kind are briefly revisited, focusing on the most adequate for large sized graphs (over 10^5 nodes and 5×10^5 links), and comparing their advantages and shortcomings.

For such huge graphs, most solutions apply some preprocessing technique to re-structure the graph into minor subgraphs in a way that, despite the large volumes of data being handled, only small fragments of the graph are taken into account at any given phase of the search, thus reducing their complexity and the response time. For processing the smaller subgraphs, classic path search algorithms are usually applied (e.g., Dijkstra's or A* algorithms).

A major difference is found in the way they handle the massive data. Some approaches as Bast's (Bast, Funke, Matijevic, Sanders, & Schultes, 2007) or Delling's (Delling, Sanders, Schultes, & Wagner, 2006) organize the graph using a hierarchy stored in main memory. An evolution of the latter (Delling, Holzer, Müller, Schulz, & Wagner, 2009) divides the graph into subgraphs and runs a preprocessing (which result is also stored in main memory) to determine which edges should be observed for each subgraph. Most recently, these authors summarize algorithms applied in route planning for transportation networks in Bast et al. (2014) where the preprocessing effort, space requirements, and query time are taking into account. Basically, the algorithms solve point-to-point shortest path and the start point is that data fits in RAM. They assume that while preprocessing may take a long time (e.g., hours), queries need to be fast.

On the other hand, there are also several proposals supported by secondary storage, thus avoiding the problems caused by hardware resources constraints. Chan and Lim (2007) and Chan and Zhang (2007) preprocess the graph and divide it into subgraphs to be stored in raw files. Some works apply database technology to manage the secondary storage (Yu & Cheng, 2010), which will serve for keeping the graph, the results of the storage, and a cache of solutions to frequently requested services. For example, Sankaranarayanan and Samet (2009) and Sankaranarayanan, Samet, and Alborzi (2009) use databases for organizing graphs into tree structures, as a result of their preprocessing, which will enable processing huge graphs in less time.

As a conclusion, it should be pointed out that, virtually, all the works of this kind are supported by a costly preprocessing (up to several days time), performed before putting the system into operation. Besides, most of them are focused to be run on the whole graph, requiring to be completely restarted if any structural change is performed in the graph (dynamic graph conditions). As a good prospect for the future, it would be advantageous to aim the development of new algorithms to local preprocessing, thus avoid preprocessing the entire graph when any structural change occurs. The proposal presented in Binh and Duong (2015) applies two algorithms for reducing the original graph. This proposal solves the specific problem in a communication network, where the response time is not the most important parameter to be optimized. The idea is seeking a sub-graph with the smallest weight in which all customers are connected to infrastructure nodes.

A typical dynamic graph example is that of the social networks. Regarding path search algorithms in this domain, it should be mentioned one of the first relevant works presented by Kleinberg (1999) who performs a decentralized search observing only local data. Some further works extending those techniques are found in Liben-Nowell (2010) and Mogren, Sandberg, Verendel, and Dubhashi (2009), improving performance and observing more information such as the population distribution. Honiden, Houle, Sommer, and Wolff (2010) propose an algorithm for vast graphs supported by Voronoi duals (graphs enunciated by Mehlhorn (1988) and Erwig (2000) inspired in classic Delaunay triangulation for Voronoi diagrams, which size is smaller than the original graph). The search is performed over the Voronoi duals applying the algorithm of Dijkstra, and the result (a path in the dual graph, not in the original one) is used for directing a search on the original graph. Main drawback is that Voronoi duals are created through a costly preprocessing. Recently, some proposals are focused on applying heuristic algorithms to prove the small-world structure in networks. The main idea shown in Aparicio, Villazón-Terrazas and Álvarez (2015) is the calculation of the shortest path between any two nodes of the network. In this approach, the network is huge but the execution time is not relevant. This work deals with dynamics changes simulating these changes in the Twitter network. In Chakraborty and Manoj (2015), the string network typology is adapted to a small-world structure. The algorithm has a good performance but the experimentation is carried out in networks with moderate size (20–100 nodes).

Regarding metaheuristics for path searching, it is outstanding the bio-inspired Ant Colony Optimization algorithm (Dorigo & Stützle, 2004). For over twenty years, its versatility has been proven in many and diverse domains such as: vehicle routing problem (Bullnheimer, Kotsis, & Strauss, 1999; Lee, Lee, Lin, & Ying, 2010), calculating trajectories for unmanned vehicles (Ma, Duan, & Liu, 2007), project management (Abdallah, Emara, Dorrah, & Bahgat, 2009), improvement of software quality prediction models (Azar & Vybihal, 2011), medical risk profile recognition (Ramos, Hatakeyama, Dong, & Hirota, 2009), or cybersecurity in the web (Manaskasemsak & Rungsawang, 2015; Wang, Lin, & Wang, 2016), community detection in social networks (Ben Romdhane, Chaabani, & Zardi, 2013), among many others. Recently, in the vehicle routing problem domain, the ACO has been adapted to exploit parallel environments (Niu, Wang, He, & Xiao, 2014; Jindal & Bedi, 2016), and to reduce the problem complexity applying time windows (local search) (Brito, Martínez, Moreno, & Verdegay, 2015; Senarclens de Grancy & Reimann, 2016). In Myszkowski, Skowroński, Olech, and Oślizło (2015) an ACO inspired algorithm is proposed for solving the Project scheduling problem. A dataset, called iMOPSE, is defined as benchmark to compare results in this domain (the dataset size is around 100 and 200 nodes). However, most works based in ACO limit their focus to small graphs (one thousand nodes at most), which can always be kept in main memory. Besides, although ACO is especially suited to dynamic scenarios (Kar, 2016), it has hardly been applied to social networks (barely on a static network comprising just one hundred nodes, Ahmad and Srivastava (2008)), perhaps due to difficulties in handling with such graph extent. Besides, in the transportation networks domain it has been shown that ACO is very suitable to dynamic changes, though response time is yet unacceptable in real world situations (Eaton, Yang, & Mavrovouniotis, 2015). Facing dynamicity in huge graphs is still a distant challenge to be approached step by step. The chart below (Table 1) provides comparison between the different types of algorithm when dealing with the *fast searching of paths* problem on vast graphs.

In sum, adapting ACO algorithm to huge graphs by incorporating a preprocessing is an appealing innovative challenge, with interesting application prospects. Scalability is a key issue to be observed, and a clear weakness of both classical approaches like Dijkstra and other implementations of ACO. For extending the proposal to dynamic problems, it would be preferable to focus the preprocessing as local. Finally, in order to avoid main memory resources constraints, it is appropriate to support the storage by a database (or in general, secondary storage technologies). An algorithm meeting this description can be found in Rivero et al. (2011), seeking the fastest response time when facing huge graphs ($2*10^5$ nodes). However, that proposal was restricted to paths involving a subset of frequent nodes (either as the start or the destination nodes) and there is no work of this kind observing general service definition yet.

3. Framework

This work aims to extend and improve a previous algorithm aimed to fast pathfinding in huge graphs. Through this section, the problem basics will be briefly reviewed, and then the base algorithm SoSACO will be explained to lay the foundations of the proposal, which will be explained in a later section.

Table 1
Comparison of capabilities through different types of path-search algorithm.

Features/Algorithms	Classic	Fragmented	Pre-processed	Metaheuristic	Target
Provide **optimal** solution	√	–	√	–	–
Scalable	–	√	√	–	√
Supports **dynamicity**	–	–	–	√	√
Provide **partial** solutions	–	√	–	√	√
Close to optimal solution	N.A	√	N.A	√	√

3.1. Problem formalization

Let be G a connected graph which at a particular moment of time t is defined as $G(t) = \{N(t),L(t)\}$, meeting:

- $N(t) \equiv \{ 1,2,\ldots, n \}$ represents the set of nodes in G (which cardinality is $|N| = n$).
- $L(t) \equiv \{ l_{ij} = (i,j) \in N(t) \times N(t),\ i \neq j \}$ is the set of edges in G at moment in time t, where $l_{ij} = l_{ji}$.

Each of the edges $l_{ij} \in L(t)$ is assigned a weight notated w_{ij} and defined by the function $W\colon L \rightarrow R^{+}$, where $w_{ij} = W(l_{ij})$. In any given moment in time t, the operation *path search* between any two nodes $x, y \in N(t)$ provides a sequence of links $P_{xy} \equiv \{l(i) = l_{ab} \in L(t)\}$ of length k, where $i = 1\ldots k$ and meeting that:

- $l(1)=(x,a)$, where $a \in N(t)$
- $l(k)=(b,y)$, where $b \in N(t)$
- *if* $l(i)=(a,b)$ and $l(i+1)=(c,d)$, then is met that $b=c\ \forall\ i < k$, where $a,b,c,d \in N(t)$

The quality of the path is inversely proportional to its cost, defined as the summation of the weights associated to the links included in the path, and formalized as $C(P_{xy}) = \Sigma_i\ w_{ab}$ where $l(i) = l_{ab}$.

The path search operation has a limited response time rt_{max}, so that any solution attained after that time is not considered. The success rate is defined as the percentage of services for which at least one solution is provided (before rt_{max}). The main goal of this work is to minimize the response time in the path search on huge graphs ($|N| > 10^5$) keeping satisfactory values of both the quality (close to the optimal) and the success rate (close to 100%).

3.2. The SoSACO algorithm

The SoSACO algorithm (Rivero et al., 2011) is an extension of the well-known metaheuristic algorithm ACO (Dorigo, 1992; Dorigo & Blum, 2005; Dorigo & Stützle, 2004), keeping its main features and adding some new ones, such as a preprocessing. The primary reason behind the introduction of this preprocessing is to increase the success rate: since the algorithm is intended to huge graphs (hundreds of thousands of nodes), is probabilistically difficult (depending on various factors) that an ant with random movement reaches a particular node (destination node). Therefore, the target is to multiply that destination node by observing a set of nodes from which it is easy to reach that destination, so if any ant reaches one of those nodes it will have found a path. This extension is also biologically inspired, as the behavior sought is that shown by of some animal species which, being in a place near to the food location, are able to be guided by their smell sense to that food. Therefore, the algorithm mainly consist in endowing the ants with smell sense, and hence its name (Sense of Smell - Ant Colony Optimization).

Apart from the classic processes and features inherited from the ACO family of algorithms, this algorithm has some new features that should be briefly revisited before extending it. The following subsections go over those features to support better comprehension of further proposal. Finally, it should be pointed out that, yet the following exposition departs from basic ACO algorithm, the extensions included by SoSACO are compatible with most variations of ACO (elitism, maxmin, etc.). Those variations would possible improve the results, but they are set aside by now to prevent them hindering the understanding of the proposal.

3.2.1. ACO basics

In their search for food, the ants deposit a hormone (called pheromone) that will guide other individuals of the same colony if they are in the same place (in a certain period of time, as the pheromone tends to evaporate). ACO consists in creating a colony (of a certain number of individual) positioned in the start node, and let each individual wander through the network, following a probabilistic formula (see Eq. (1)): the probability of going from node i to node j (where $i, j \in N$ and j is adjacent to i) is proportional to the amount of pheromone at the link τ_{ij} multiplied by a heuristic value η_{ij}, and inversely proportional to the summation of that product ($\tau_{ix} \cdot \eta_{ix}$) for the rest of the candidate nodes.

$$p(i,j) = \frac{[\tau_{ij}]^{e_1} \cdot [\eta_{ij}]^{e_2}}{\sum_{x \in Adj(i)} [\tau_{ix}]^{e_1} \cdot [\eta_{ix}]^{e_2}}, \quad where\, i, j \in N\, and\, j \in Adj(i) \tag{1}$$

Eq. (1): *Next node selection probability*

The amount of pheromone found at each node varies in time, due to three types of events: (a) anytime an ant uses a link (local update); (b) anytime a complete path is attained (global update); and (c) periodically all pheromone trails are decremented. *Local update (2)* consists in increasing the pheromone deposited just after choosing a link, by adding a value inversely proportional to the cost assigned to the link (w_{ij}) and proportional to a constant value ($k_1/1000$) as shown in Eq. (2). Through *Global update (3)*, the pheromone is increased by a constant value (k_2) divided by the global cost of the complete path found, as formalized in the Eq. (3). Finally, pheromone trails should be eventually removed causing unsuccessful attempts to be forgotten. Specifically, just before any global update, the *evaporation (4)* will be applied to all nodes in the graph, as stated in Eq. (4). These processes introduce some constant values (k_1, k_2 and the evaporation rate ρ) as parameters of the method that should be suited to each specific scenario through preliminary experimentation.

$$\tau_{ij}(t) = \tau_{ij}(t-1) + \frac{k1}{1000 \cdot w_{ij}} \tag{2}$$

$$\tau_{ij}(t) = \tau_{ij}(t-1) + \frac{k2}{pathLength} \tag{3}$$

$$\tau_{ij}(t) = (1-\rho) \cdot \tau_{ij}(t-1) \tag{4}$$

3.2.2. The 'food node' of SoSACO

As the sense of smell works in nature to guide individuals to find food, the smell in SoSACO is a trail spread in preprocessing time to guide ants to the destination node. Thus, the food node is a frequently requested privileged node for which the path search will be faster (no matter the starting node). For not directed graphs, the food node can also be a frequent start node. Finally, it can be set several food nodes, with a different identifier for each smell trail.

Given a food node $i \in N$, its trail (odor) at another node $j \in N$ is notated as $O(i,j)$. The maximum odor value is limited by parameter m, and is restricted to the food nodes themselves, that is $O(i,i) = m$ $\forall i \in N$. Regarding the minimum odor value, it should be noted that is not usually adequate to spread the odor of a food node all over the graph. In first place, that trail has to be stored and later processed and observing such volume of data could require too many resources. On the other hand, with some perspective for the future, if some structural change is performed in the graph (dynamicity) the odor trail(s) found at the elements changed must be updated. Reducing the coverage area of the odor trail reduces the set of nodes with that benefit, but also enhances processing time and maintenance time (in case of dynamical graphs). Therefore, in each case it is necessary to find a balance between these factors, which is specified in the 'odor threshold' μ. Any trail below that value μ will not be recorded during the preprocessing, thus shaping a reasonable sized smell-area $S(i)$ around the food node i, and preventing the network to be saturated with the odor trails.

The preprocessing is supported by an auxiliary data structure, the bag B of nodes to be examined at any given iteration. The odor initialization algorithm starts with $B(0) \equiv \{i\}$, being i the focused food node. The preprocessing algorithm is described as follows:

```
step 0  $n := 0$; $B(n) \equiv \{i\}$; /* initialization; first bag only includes the food
        node */
step 1  Bag for further iteration $B(n+1)$ is empty (set to $\emptyset$)
step 2  For each node j $\in B(n)$, let be R the set of nodes $x \in N$ adjacent to j
            For each node $x \in R$,
                calculate odor i coming from node j as $O_j(i,x) = O(i,j) - k \cdot w_{jx}$
                If that new odor $O_j(i,x)$ is greater than the currently stored $O_j(i,x) > O(i,x)$ and greater or equal than the odor threshold $O_j(i,x) \geq \mu$
                then update odor $O(i,x) := O_j(i,x)$ and include current node $x$ into $B(n+1)$
            end loop;
        end loop;
step 3  If the bag for further iteration is not empty $B(n+1) \neq \emptyset$, then start
        a new iteration n := n+1, and go back to step 1;
        in other case (bag $B(n+1)$ is empty), END.
```

where k is a constant value between 0 and 1 (*odor loss*) and w_{jx} is the cost of the link between nodes j and x. Once this process ends (for each food node) it can be identified a non-empty subset of nodes $S_i \subseteq N$ namely the *odor coverage of node i*, which nodes $j \in S_i$ meet $O(i,j) \geq \mu$.

3.2.3. Pathfinding in SoSACO

The basic SoSACO process is very similar to the fore explained ACO algorithm, except for the success condition. When searching for node i, not only reaching that node involves finding a valid path, but also reaching any node within the odor coverage area S_i.

Indeed, any node j with an odor trail $O(i,j) > 0$ is connected with the food node i (located at the goal) by a growing trace of odor. Therefore, when such a node j is reached by any ant, it can complete the path by choosing the higher odor trail among the nodes adjacent to j, and so on till it reaches the food node i (which is characterized by the maximum odor trail, m). This behavior is depicted in Fig. 1.

Notice that such procedure provides a fast first response (yet not probably the optimum path), that can be later refined, by following several cycles as in the basic ACO. However, for many applications it is better to apply that first solution directly, even not being optimal, because waiting for refinement is more costly than applying a mid-quality solution.

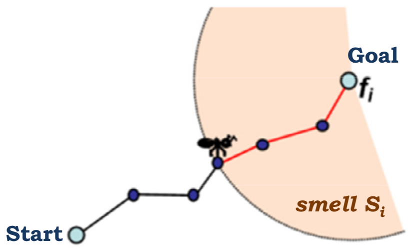

Fig. 1. Example of pathfinding with SoSACO.

3.2.4. Radial diffusion in SoSACO

There is a last feature of that SoSACO to be taken into account. It has been indicated that saturating the storage with food trails is not adequate, and because of that the trail was limited to be equal or greater than μ. However, including a reduced amount of that trails will not hinder processing severely and could mean interesting advantages if properly selected. And a good selection is that gained from experience, since the trace from the start node to the food node is the best found (typically, after many cycles).

Consequently, after a service is ended and the best path provided, the nodes comprising that path will be impregnated with their correspondent odor trail, even though those trails are below the μ threshold. Thus, experience gained in a service can be of use in future services. The coverage area is now complemented with linear trails centered in the food node, with an appearance (and function) similar to that of the radial main roads departing from big cities. Such analogy is depicted in Fig. 2.

However, the excessive accumulation of experience can jeopardize system's performance (again by network saturation with odor traces). To avoid this risk, several techniques can be applied:

- *limit the overall number of traces*: a new upper boundary will be set, which will be checked periodically; if exceeded, less relevant traces (lower values) will be selected and deleted until the boundary constraint is met again.
- *store only distant paths*: when a new path is obtained, instead of directly adding odor traces to its nodes, calculate the percentage of nodes adjacent to those nodes in the path that have already the same odor trace. Then, if that percentage is over a certain threshold, the path is close to already existent paths and thus it will be discarded.
- *reset periodically the coverage area S_i*: by erasing the odor trace for node i and setting it again. This is the most adequate solution for dynamic graphs, because they involve continuous updating of the odor areas, so there is no need to do extra-operations to avoid the risk of odor trails saturation.

Finally, regarding the maintenance time during which the odor trails are being updated, it should be reminded that this is auxiliary information. Its existence helps to find solutions faster but it is not essential to perform searches (without it, basic ACO is applied). Therefore, the system is always available (even during maintenance operations).

4. Proposal

The focus of the work is to extend SoSACO in order to solve general pathfinding services across huge graph taking as less time as possible. First version of that algorithm was able of finding paths involving a food node. The basis of this extension is that to find a path from node A to node C and another path from node B to node C means to have found a path from node A to B via node C. Since SoSACO enables fast finding of the food node from anywhere, it should be easy to find a valid solution quickly from the start node to the destination via the food node.

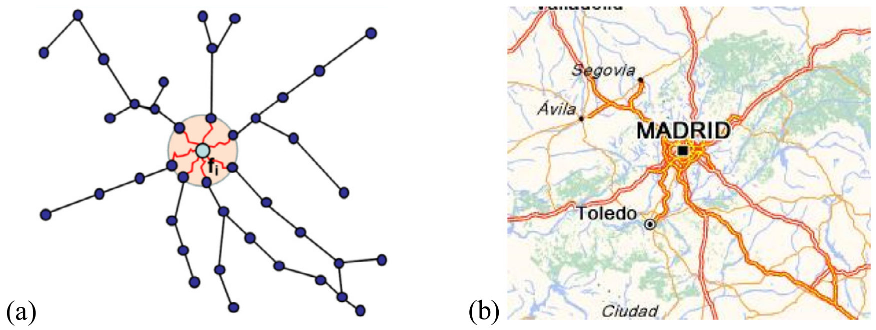

Fig. 2. Parallelism between the odor radial diffusion (a) and the radial main roads around a city (b).

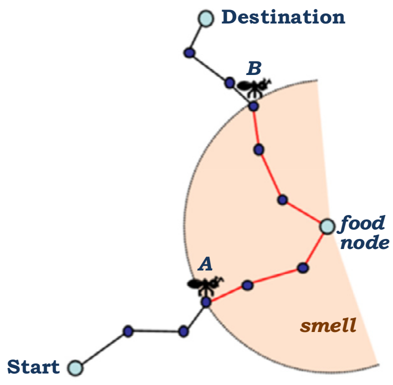

Fig. 3. Example of using a *food node* as a meeting point for quick path search.

The here presented extension consists in creating two colonies, each featured by a different pheromone-id. One colony will be settled in the start node and the one in the destination node, and both of them searching the food node as a meeting point. This idea is graphically represented in Fig. 3.

Notice that this mechanism is posed as for non-directed graphs but keeps its validity for directed graphs. It is only required to change the construction of the candidate set when the ant is moving from node *i*. The ants of breed A (departing from start node) will take as adjacent the node *j* if there is a link $i \rightarrow j$, while the ants of breed B (departing from destination node) will take as adjacent the node *j* if there is a link $j \rightarrow i$. Hereinafter, the exposure of the proposal will focus undirected graphs for simplicity, while maintaining its validity for all types of graphs.

On the other hand, this mechanism requires a more detailed explanation, in order to observe the eventuality of any ant of breed A reaching the destination (or ants of breed B reaching the start node) before finding the food odor. Furthermore, when any ant finds pheromone trails of the other breed it can take advantage of such information. This evolved mechanism will be explored through next Section 4.1.

The so obtained extended version of SoSACO (depicted in Section 4.2) can also take advantage of other enhancements widely applied to ACO algorithms. Specifically, this proposal includes a variant of the use of tabu lists (namely β-tabu) that will be explained later (in Section 4.3). Finally, by changing the role of the food node (no longer start/end node to be now a *meeting point*) also should change the selection criteria of foods nodes in design time. A brief discussion of this is found in Section 4.4.

4.1. Two colonies (and a branch)

By now, let us leave aside the existence of the food nodes to explore the possibilities of creating two colonies alone. Departing from the idea that there are two breeds of ants with different pheromone id, it should be stated that each ant will place pheromone of its breed when moving, while global pheromone update can involve both types of pheromone and the evaporation is always performed on both types.

However, the behavior of the ants changes slightly. When they start moving, they place lots of pheromone around the settlement of their colony. Thus, what an ant is really seeking is that pervaded by the pheromone of the other breed of ants. Consequently, when calculating probabilities regarding which node to select to move next, ants will first consider the pheromone of the other breed (by following the classic formula, stated in the Eq. (1)). If there is no trail of that pheromone, they will follow their own pheromone (also applying the Eq. (1)). Finally, if there is no pheromone trail, the ant will choose an adjacent node randomly.

Given that the pheromone trails of the other breed are stronger as its settlement is closer, with the explained behavior the ants will probably reach that node faster. Notice that until the ants begin to find trails of the other breed their behavior is somewhat erratic (directed only by their own pheromone, provided there is no heuristic available). However, that situation holds less time than before (with one breed). Since both breeds are continuously expanding, when any ant first finds trails of the other breed this will probably occur at halfway to its settlement node. At that moment, the coverage areas of both breeds (zone pervaded with each specific pheromone) start overlapping. But the sum of those areas at that moment is much lower than the area covered by an only breed finding the destination alone. Indeed, if the start and destination nodes are separated by a distance *d*, in the latter case of one breed the coverage area is $\pi \cdot d^2$. However, two colonies could cover only $\pi \cdot (d/2)^2$ each (in the best case, being the worst case that one of the breeds does not progress which is the same that having an only breed). In Fig. 4.a is shown the coverage area of an only breed in comparison to that of two breeds (Fig. 4.b).

In sum, till the meeting event both breeds could have jointly covered half the area ($2 \cdot \pi \cdot (d/2)^2$) than an only breed, and the time taken from that point to reach the other breed settlement by following its pheromone is quite reduced. This mechanism is in itself an improvement on the algorithm ACO that can save time and increase the success rate (less coverage area involves also fewer ants being lost).

Still, there is space for improvement in that proposal. When an ant (for example, of breed A) first reaches trails of the other breed's pheromone (B) in the node *c*, its breed will be devoted to two sub tasks: (i) refine the path from settlement A to node *c*; and (ii) find a path between nodes *c* and settlement B (and later refine it). This differentiation leads to specialize the ants of that breed A, by splitting it again into two breeds (A and C) respectively located at the original settlement (A) and at the node *c* (C). In order to keep a constant number of individuals, the sum of the populations of those two breeds will be equal to the population of the

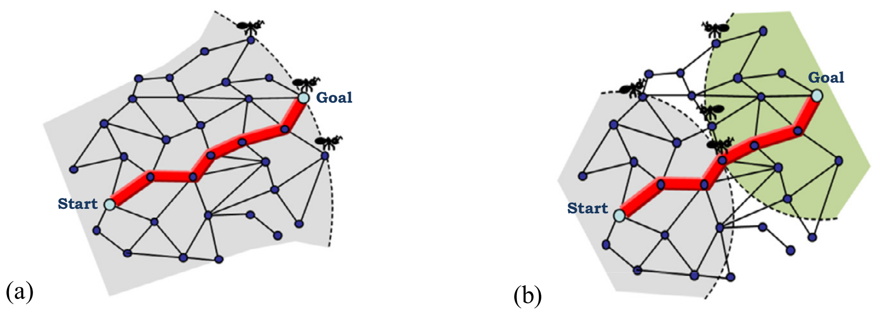

Fig. 4. Search area covered when departing from one breed (a) and two breeds (b).

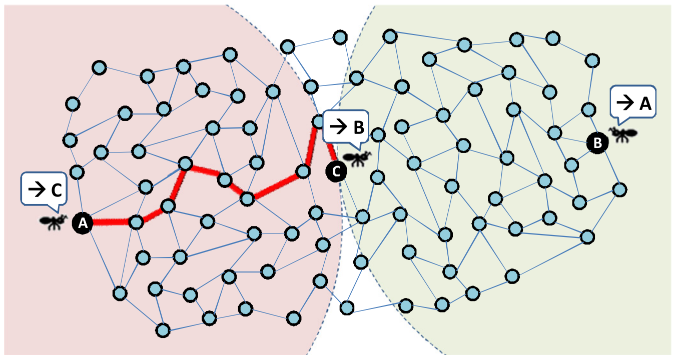

Fig. 5. Split of *breed A* into two new breeds (*A* and *C*), seeking nodes C and B (respectively).

former breed A. The percentage of individuals going to the new branch C is indicated by a new parameter θ (e.g., $\theta = 0.5$ involves each breed will take half the ants originally assigned to A). Therefore, the ants from breed A will seek a better path to node c, while the ants from breed C will keep on searching node B. Such split is graphically schematized in Fig. 5.

The new branch (C) could have its own pheromone id thus avoiding going towards settlement A, but in fact this is not essential since it is going to be directed by pheromone B. Notice that individuals from breed C will follow first pheromone B and then pheromone C, while ants from breed A will follow pheromone C and then pheromone A. Finally, individuals from breed B will keep on following pheromone A and then B.

On the other hand, it should be stated that the global update conditions change too. There can be found three kinds of path: from node A to node B, from node A to node *n*, and from node *n* to node B. In each of these cases, nodes in the path are reinforced with the pheromone trails settled in the road ends (A&B, A&C, and B&C, respectively). Each time a partial path is found, it is compared with the previously found (if any) and then stored (in case the new one is less costly than the currently stored).

Finally, yet the colony splitting could be applied several times, by now it is only proposed to be done once per service (that is, first ant to reach the pheromone of the other breed will cause its own colony to be split, but such operation will be observed no more in the rest of that service). Therefore, the experiments set for the evaluation of present proposal (see Section 5) include only one splitting process per service.

4.2. Fast finding of paths via food node

First version of SoSACO provided solutions in a short time, yet it was restricted to services which start node (or destination node) was set at the food node. Present extension focus general services for finding paths between any two nodes within the graph. Nevertheless, if when applying present proposal is found that the service departs from (or ends at) a food node, the applied method will be that of SoSACO version 1. That is, there will be only one colony settled at the node other than the food node. For those particular cases, that procedure performs better than this extension (creating two breeds with half the ants for each), as will be stated further in the evaluation of this proposal. However, that approach does not observe general services (paths between any two nodes, without restrictions), which is the aim of this extension.

The preprocessing will have created an odor coverage zone *S*, as described in Section 3.2.2. Anytime a general service is defined providing start and destination nodes, two colonies will be created and each will be settled at one of those two nodes, as afore stated in previous Section 4.1. The behavior of the ants is just the explained there, except for when any ant reaches a node with food odor. When this occurs, the complete path to the food node will be obtained by following the growing odor trace. Then, the breed of that ant will have a valid solution to reach the food node from that moment. If the breed had already a better way (less costly) to reach the food node, the new solution will be discarded. If not, this new solution will be kept associated to that breed. Once both breeds achieve a solution to reach the food node, both sub-paths will be combined and thus a complete path will be obtained to fulfill the service.

Such method is schematically depicted in Fig. 6. First snapshot represents the initial state, where both breeds are set in the starting and end nodes respectively, each seeking the other node. In the following image (snapshot 2), the ants had started drifting across the graph. When any ant firstly reaches the odor traces (snapshot 3) it will obtain the rest of the path to the food node F (following the increasing trails of odor). Notice that its location does not change (it stays at the node where it found the first odor trail). The breed of that ant (A) now counts on a way to reach the food node. Besides, the ant cannot possibly find a better path to the food node (until it is restarted), but it could find the pheromone of the other breed (B) and finally reach its goal (node B). Therefore, the ant will continue its drifting from its current location (where it firstly found the odor trail) seeking the other breed (B), as depicted in the fourth and fifth snapshots. Through the rest of its processing, that ant ignores all odor trails (as they are useful no more for that ant). When the ant meets the termination conditions (see Section 3.2.1) and thus it is restarted, it will observe the odor trails again for enabling the refinement of the path to the food node. Finally, as shown in the snapshot 4, when any ant (in this case, of breed B) discovers odor trails and obtains a path to the food node (F) and finds there is available the complementary solution (the other breed has found a way to the food node), then it will compose a complete path from starting node to destination (passing

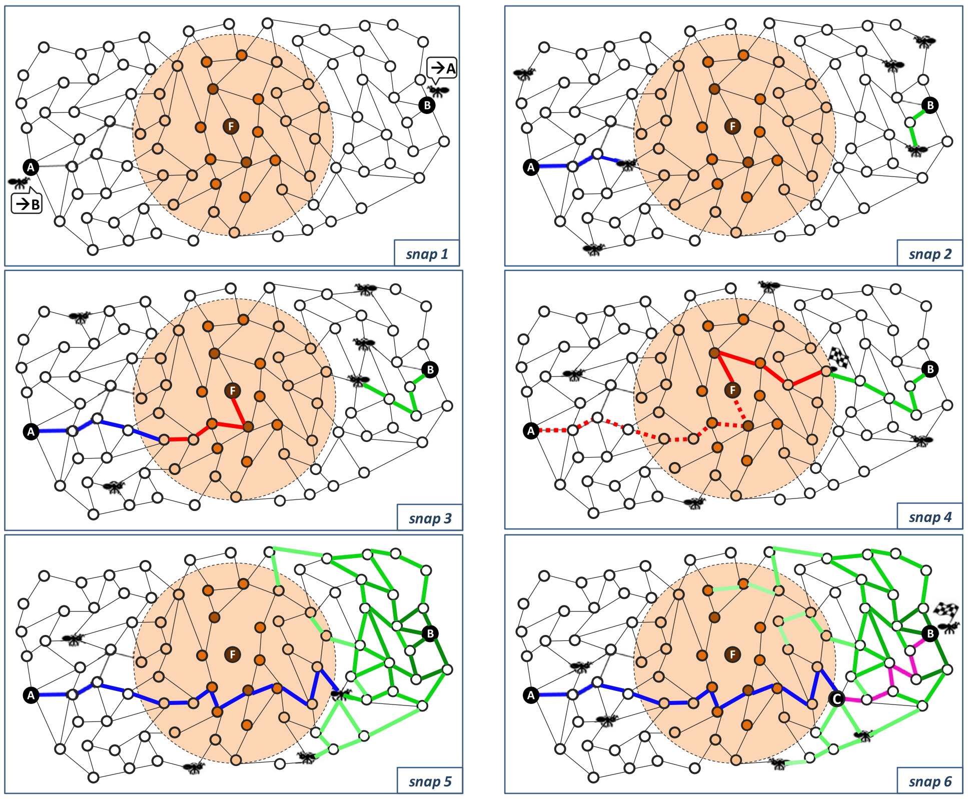

Fig. 6. Behavior of the ants when finding food odor trails.

through food node) by chaining both partial solutions, and immediately rise a solution.

The path found via the food node is a quick valid solution, but is quality may be far from optimal. In case there is extra time, the ants should be kept drifting to seek a refined solution. In the fifth snapshot it can be observed that the ant of breed A that found the odor continues drifting and ignoring the odor from that moment on. After a few steps, it finds a link with pheromone of the other breed. Then, it founds a new colony C (see Section 4.1) which ants can reach easily their goal (node B). By following the pheromone of the other breed, they finally arrive at their destination providing a refined solution, as depicted in the snapshot 6.

Notice that most times the ants finding odor and the pheromone of the other breed are different individuals, but they could be the same. Thus, the example illustrates also why is useful not to restart the ants finding odor, but left them drifting without paying attention to the odor trails (until restart).

Regarding the global pheromone update, it is usually set for further refinement of paths. Given that the path to the food node can be distant to the optimal path to the destination, reinforcing pheromone of paths leading to that food node can lead to improve the path to it, but can also hinder the finding of direct paths (paths that do not include the food node). Therefore, the finding of food odor should not cause global pheromone update (at least, not to the same extent). It could be advantageous to observe a minor global update for this event (considerably less than the observed for the events described in Section 4.1), yet exploring such alternative will be left further work (the evaluation of present approach will not observe global update when odor is found).

Summarizing, by combining both methods (two breeds of ants ant a food node as a meeting point), fast solutions can be attained. If extra time is available (after finding first solution), path refinement can improve the quality of the solution bringing it close to the optimal.

4.3. Tabu list enhancement

Using *tabu lists* is a widespread technique among ACO algorithms. It consists in recording the nodes visited by the ant (in a tabu list) and then inhibiting ants going to already visited nodes (by the same ant). Thereby, this technique prevents the appearance of loops in the ant path. The here presented new variant is a more relaxed constraint named *β-tabu*.

This β-tabu list technique is set not to prevent the ant going on loops (that can be removed from the ant's final path, in case it is successful) but to avoid the ant taking the same choice in the same situation. That is, the ant is allowed going to an already visited node, but it will be forced going to an unexplored node in the next step. Thus, the ant can cross its own path for exploring the other side of the graph.

Then there is the risk of the ant zigzagging around its own past steps. To prevent such eventuality, a refinement of this approach based in a new parameter β is introduced. It consists in allowing ants going anywhere (including already visited nodes) only if its β previous steps (at least) were performed to non-visited nodes. This is easy to implement with a counter for each ant indicating the number of steps taken by it. Already visited nodes can only be chosen if that number of taken steps is equal or greater than β. When any node is revisited (taken though it is already in the tabu list) the counter is reset to zero.

Notice that the firstly explained variant will be named 1-tabu, while 0-tabu and ∞-tabu are special cases of no use of tabu list and the classic tabu list, respectively.

4.4. Selection criteria for the ‘food node’

Given that the role of the food node has been changed from the previous version of the algorithm, its selection criteria (explained in Section 3.2.2) should be reviewed. Formerly, it was a frequently requested node (either as start point or destination). This criterion remains valid subject to more frequent use will be the aforementioned. But if the intended function of the food node is to serve as a meeting point for the two colonies of ants, it will be more appropriate to seek other selection criteria such as the described next:

- *Domain dependent:* it may be advantageous to choose those nodes that have a particular significance in the domain that is being represented (which Delling et al. (2006) called *landmarks*). To provide some examples, a famous person in a social network (as it will have more friends), or a device with Internet access in a network (since it will have more devices connected to it).
- *Graph structure:* generalizing the above criteria, for cases in which no specific knowledge of the domain is available, the food node would be characterized by a high degree of centrality in the graph (Borgatti, 2005), that is, nodes with high clustering coefficient. This criterion can be applied on either static or dynamic scenarios, while eventually changing the food node on the latter (Min-Joonge, Sunghee, & Chin-Wan, 2016). Besides, in those cases where the network topology is metrizable, the nearest node to the centroid of the graph could also be a good choice. Finally, a simple alternative is to choose the node with the highest connectivity degree.
- *Experience based:* nodes with high transit frequency can be good meeting points, thus excellent candidates to be chosen as food nodes. Consequently, it is posed a criterion based on the experience that may be used to adapt the preprocessing to the needs of recent services with the prospect of future services (short-term) will show similar needs.

Note that the here presented choice revolves around a single food node although the method could be extended to use several of them simultaneously. In dynamic scenarios, a background process could be set for discovering new good candidates for working as food nodes (and putting them into service), while another process is set for retracting less useful food nodes. However, such extensions require significant changes in the algorithm, which will be left for future work.

4.5. The SoSACO algorithm

Let be *G(t)* a connected non-directed graph on which a path search (P_{AB}) between starting node A and destination node B is going to be performed (A,B $\in$ *N(t)* and A$\neq$B) in rt_{max} response time, as defined in Section 3.1. Let be F a food node, for which the proper odor coverage area S_F is already set, as described in Section 3.2.2. Let be β the type of β-tabu to be applied.

Since there will be several breeds of ants, it necessary to differentiate their notation. Let be $\tau_{ij}^{X}(t)$ the trail of pheromone of colony X at link l_{ij} at time t, while ρ^X is the evaporation rate of colony X. Let be P_{XY} the best path found between nodes X and Y, and P_C the best path of the split colony to the intermediate node C. Let be $R_i(t)$ the sequence of links covered by each ant h_i till time t. Finally, let be α the total number of ants.

The proposal involves a centralized control, in charge of the initialization and the post processing, and the algorithm defining the behavior of any generic ant. On one hand, the initialization phase of SoSACO involves setting up the graph, arranging each ant, and preparing global storage of best path currently found P_{XY} for the relevant pairs of nodes (AB, AF, and BF), as formalized next.

```
initialization
step 0   $P_{AB} \equiv \varnothing$; $P_{AF} \equiv \varnothing$; $P_{BF} \equiv \varnothing$; $P_C \equiv \varnothing$; split:=FALSE;
step 1   $\tau_{ij}^{X}(0) = 0$,  $\forall$ colony X, $\forall$ link $l_{ij}$
step 2   if  B=F
             $\forall$ ant $h_i$ set starting node A, end node B, tabu($h_i$)={A};
          else if  A=F
             $\forall$ ant $h_i$ set starting node B, end node A, tabu($h_i$)={B};
          else  half := round($\alpha$/2);
             $\forall$ ant $h_i$ ,,  i $\in$ [1, half],  set starting node A, end node B, tabu($h_i$)={A};
             $\forall$ ant $h_j$ ,,  j $\in$ [(half+1),$\alpha$],  set starting node B, end node A, tabu($h_j$)={B};
```

On the other hand, after setting up all the ants, each of them will start drifting across the graph. At any iteration, any ant will first check if it is at the destination (thus it will have found a solution). If not, it will check if the pheromone of the other breed is firstly found (first ant finding trails of the other colony), so it will have to split its own colony. If not, it will finally move to an adjacent node, checking then if any odor is present. In case it is, and provided that individual has not found odor trails before, it will have found a partial path (to the food node) and eventually a full solution (if the other complementary partial path is already available). The described behavior is formalized in the following algorithm:

```
ant i behavior
while current_time < $rt_{max}$ do
    if current node = end node then
        /*prepare solution */
        prepare path (remove loops from $R_i$; if split colony, concatenate first half $P_C$);
        exploit solution ($R_i$);
        perform global update and evaporation (equations 3 and 4 in section 3.2.1);
        restart ant $h_i$ (empty tabu list and set current node := starting node);

    else if the other breed's pheromone is firstly found then split colony (not split and $\tau_{ij}^{Y}(0) > 0$)
    then
        split:=TRUE;
        prepare partial path (remove loops from $R_i$);
        store partial path $P_C := R_i$;
        $\forall$ ant $h_j$ of the same breed (starting node of $h_j$ = starting node of $h_i$) do
            if (j mod2)=0 then starting node of $h_j$ := current node;
                else end node of $h_j$ := current node;
            restart ant $h_j$;
    else
        /* make next move */
        choose next node n;
        include into $R_i$ the link from current node to node n ($l_{mn}$);
        perform local update of pheromone (equation 2 in section 3.2.1);
        current node := n;

        /* smell (check odor) */
        if odor firstly found ( O(F, current node) >0 and O(F,k)=0 $\forall$ k $\in$ tabu($h_i$))
        then
            follow odor, obtaining partial path R' leading to node F;
            /*prepare partial solution */
            prepare path (remove loops from $R_i$ and concatenate $R_i$');
            lock $P_{XF}$ (where X is the starting node);
            if first partial solution ($P_{XF} \equiv \varnothing$) OR is a better solution ($C(P_{XF}) > C(R_i|R_i')$)
                then
                    store it $P_{XF} := R_i|R_i'$;
                    if exists complementary solution ($P_{YF} \neq \varnothing$),
                        then exploit solution ($P_{XF}|P_{YF}$);
            unlock $P_{XF}$ ;
```

Notice that anytime an ant achieves a new solution, there is a simple way to exploit it, taking care of collisions between ants that could be accessing the same resources:

```
exploit solution P
lock $P_{AB}$ (prevent other ants from updating it);
if first solution ($P_{AB} \equiv \varnothing$), then provide path P and store it $P_{AB}$ := P;
    else if better solution ($C(P_{AB}) > C(P)$) then store it $P_{AB}$ := P;
unlock $P_{AB}$ ;
```

When choosing next nodes, ants prioritize the pheromone of the other breed. If not found nearby, they will seek odor traces (but only if they have not found odor before). Finally, if that is not the case, they will take any node with choice probability proportional to the trails of their own pheromone (if present; if not, a random choice is performed). Choices based on pheromone were defined in Eq. (1), Section 3.2.1.

```
choose next node n        for ant $h_i$ from node m, having start node X and end node Y

take nodes adjacent to node m as $J \equiv \{j,, \exists\ l_{mj} \in L(t)\}$;
if the other pheromone is present in accessible link ($\exists\ j \in J$ meeting $\tau_{mj}^{Y}(0) > 0$)
    then do stochastic choice of next node $n \in J$ proportional to pheromone Y (equation 1, section 3.2.1);
    else if not odor individually found yet ($O(F,k)=0\ \forall\ k \in tabu(h_i)$)
            and odor is present in adjacent node ($\exists\ j \in J$ meeting $O(F,j)>0$)
            then choose next node $n \in J$ maximizing odor (not $\exists\ j \in J$ meeting $O(F,j)> O(F,n)$);
    else do stochastic choice of next node $n \in J$ proportional to pheromone X (equation 1, section 3.2.1);
```

When an ant detects odor trails, it is able of finding its way to the food node by following the increasingly strong traces of odor. The method is formalized next:

```
follow odor        from node m, returning partial path R'

R' ≡ ∅;
stepnode := m;
while stepnode ≠ F do
    take nodes adjacent to node m as $J \equiv \{j,, \exists\ l_{mj} \in L(t)\}$;
    choose next step $n \in J$ maximizing odor (not $\exists\ j \in J$ meeting $O(F,j)> O(F,n)$);
    include link $l_{stepnode,n}$ into R';
```

Finally, after the service is completed, the best solution found is pervaded with radial food odor trails. Notice that, unlike the odor diffusion in preprocessing time, this process has no minimum odor boundary.

```
post processing        having best solution found $P_{AB}$

if solution passes through the food node F ($l_{iF} \subset P_{AB}$) then
obtain subpaths $P_{AF}$ and $P_{FB}$ arranged as sequences of links departing from node F;
for each subpath do
        current node n := F;
        while subpath is not empty do
                extract next link $l_{nx}$ from subpath;
                calculate odor for next node x coming from current node n as $O_n(F,x)= O(F,n)-k \cdot w_{nx}$
                if that new odor trail is greater than the currently stored $O_n(F,x)>O(F,x)$
                then update odor $O(F,x) := O_n(F,x)$;
```

4.6. Adaptation of SoSACO to dynamic conditions

One of the most appealing features of ACO-family algorithms is their adaptation to changes. In order to complete the proposal, the performance of SoSACO on dynamic environments should be analyzed.

Firstly, dynamicity has to be characterized. In this work, a dynamic graph is that subject to continuous changes with a certain time-lag. The observed changes are both dropping and adding graph elements (either nodes or links).

Though SoSACO counts on a preprocessing, it does not hinder its adaptation to dynamic environments. Anytime an odored node is removed, all decreasing odor trace departing from that point will be removed. Yet this eventuality can be hindering services processing to some extent, actually is not quite probable to be removing one of these nodes, and when doing so it is frequently inlayed in a radial odor trace (not a circle). Anyway, odor is soon restored as services progress. The weakness is found on removing the very 'food node'. All preprocessing for that node is removed, and that food node is no longer useful. Regarding this issue, food nodes choice should be restricted to constant (not volatile) nodes if possible.

Preliminary experimentation shows certain performance drop (more time spent in providing first solution). Eventually, some tests took less time than the same services in static conditions. This was due to the fortuity that some dropped elements were 'pruning' useless paths, so ants were saving their time in not following those paths. Conversely, when useful links were dropped, efficiency could be severely harmed, yet not always (when there are close detours, other ants coming later will be exploring and finding). Though their cost is not weighty on average, those cases could be unsuccessful if an upper bound is set on the response time (37.93% were taking more than double the time required for the service in static conditions).

The analysis of these cases throws two conclusions: moving ants which have passed through a dropped element are useless: even if they succeed, their path is no longer valid. It should be useful to eliminate them. Unfortunately, our attempts of doing so are computationally more costly than leaving them wandering until they are finished (as *zombie ants*), so it seems to be advisable to leave them alone and ignore their result later on. On the other hand, there are certain points (dropped links with high pheromone deposit) where additional help would be appreciated.

This work just proposes to add new individuals to the closed road. The number of added individuals will be proportional to the amount of pheromone in the dropped link and a new constant h, valued through preliminary experimentation. The total number of individuals is rebalanced later on, when other ants are finished. For preventing the ants from proliferating over the upper bound, the initial amount of individuals has to be set to a lower amount (for example, 80% of individuals set at the start, and up to 20% set as extra aid). The evaluation of this method can be found in the third subsection of the following part.

5. Evaluation of SoSACO v2

This section includes the results of the evaluation of the proposal. In order to show its versatility, the graph used in the evaluation should be of general type. However, taking into account that a wide extent of real complex networks show small-world topology and free-scale degree distribution (as afore stated), as for example the social networks, it has been found interesting to test the algorithm on two different scenarios:

1 Generic topology: a randomly generated connected graph comprising 2×10^5 nodes and 6×10^5 links.
2 Social network: specifically, it will be used the real social network *Slashdot* (Leskovec, 2010). The load is that of February 2009, with 82.168 nodes and 948.464 links.

Since performance can be affected by the hardware resources, the focus of the evaluation is to compare SoSACO performance with that other of a well-known algorithm (specifically, Dijkstra). Regarding the metrics, both response time and path quality (global cost) are observed. Since SoSACO is of stochastic nature, the success rate will be calculated too. Finally, as the method is characterized by several parameters, some preliminary experimentation should be performed to set the values to those parameters adapted to the specific implementation and available hardware resources. Through that experimentation, parameters are valued in a range and combined, seeking the combination producing minimum time response.

5.1. First scenario: generic topology

Ten experiments were performed, from which average values were taken. Before each experiment, any trace of odor in the network was removed. Then, the preprocessing was run to create the odor coverage S around the food node. Each experiment consists of one thousand services, which will be repeated four times to reduce the influence of stochastic algorithm. All traces of pheromone are eliminated after each service, and radial odor diffusion is performed after the four repetitions.

As before stated, the algorithm's parameterization is attained through preliminary experimentation. Firstly, total number of ants was set to 500 individuals and the evaporation rate ρ was 0,6. The *response time threshold* (maximum time after which no response is accepted) was set to 700 s.

The node with higher clustering coefficient was selected to be the food node. The maximum odor value was 10^6 (for assuring any node, even the farthest, could get an odor trail in case), and the odor threshold μ (maximum odor assignment during preprocessing) was chosen to limit coverage area *S* to less than a certain percentage of nodes. The valuation of this parameter can alter the results significantly, and because of this three different valuations were observed to compare performance (specifically, 1%, 17% and 30% of total nodes in the coverage).

Finally, regarding the β-tabu, preliminary experiments showed better path quality with higher β (paths obtained with 0-tabu and 1-tabu showed respectively 114% and 132% the cost of paths obtained with 2-tabu. However, the response time was also increased with β (1-tabu and 0-tabu took 30% and 42% of the time spent by tabu-2, respectively). Since main goal is to attain fast responses β was set to one (1-tabu is applied).

Table 2
Parameterization of SoSACO algorithm during these experiments.

Parameters	Values
# ants (α)	500
Evaporation rate (ρ)	0,6
Response time ($t_{threshold}$)	700 seg.
Tabu list type (β)	1-tabu
Maximum odor area (S)	1%, 17%, 30%
Maximum odor (m)	10^6

The evaluation results are shown in Table 2. As expected, the quality of the solutions provided by Dijkstra (optimal) is greater than the quality of those provided by SoSACO. However, it should be noted that the difference is acceptable in many cases. On the other hand, the difference in response time is quite significant (more than 547 times faster on average).

Regarding the different parameterizations of SoSACO, it should be stated that setting lower values for μ (higher coverage percentage) increases quality and reduces response time (provided there are enough hardware resources). For example, with $\mu = 0998 \times 10^6$ nearly 30% of the total nodes are pervaded, and the cost and response time are reduced down to 55% and 59%, respectively. However, the preprocessing time grows exponentially (from barely 10 s for covering 1% up to more than five minutes to cover 30%). With the prospect of future application to dynamic graphs, it will be better to choose smaller coverage areas, so they can be quickly refreshed when any change occurs.

5.2. Second scenario: social network

For the second scenario, the social network, the preprocessing was set to minimum diffusion. That is, only nodes with direct link to the food node will be pervaded, which amounts nearly 3% of the total nodes. In this case, the preprocessing (odor diffusion) took 10,2 s. The results of this evaluation are shown in Table 3. Of particular interest is the high standard deviation (50%) shown by the new SoSACO in the response times for services on this scenario. However, given the large difference in performance relative to other methods (46 times faster than Dijkstra) its use is advantageous in any case. It is even faster than the previous version of SoSACO (34% faster) which functionality was more restricted (to services directly involving food nodes), while it is also true that the quality of the solutions is worse (mean cost is 15,62% higher in this case).

To complete the evaluation, it was considered interesting to compare the behavior of the proposal with a metaheuristic algorithm in this scenario. Specifically, it was compared to ACO with similar parameterization. Due to the features of the scenario (huge graph of small world topology) it was found adequate to apply a tabu list in ACO, as ants otherwise got lost easily producing extremely low success rates. Supported by that tabu list, ACO reach a success rate higher than 90% yet at the expense of the quality of the solutions proposed. To understand such consequence, it is important to realize that most optimal solutions go through a *hub node*, that is, a node gathering tens of thousands of links (typically, nodes representing a famous person). Such nodes are easily reached by the ants, but departing from there they rarely direct their steps to the destination (due to the large amount of possibilities). Besides, the tabu list prevents them to go back to that node,

Table 3
Performance comparison of Dijkstra vs. SoSACO algorithms on generic scenario.

Topology		Generic (scenario i)			
Method		Dijkstra	SoSACO		
			$S \cong 1\%$	$S \cong 17\%$	$S \cong 30\%$
Cost	Mean	3046,72	10712,67	7552,39	5942,99
	%SD	17,08%	15,87%	21,39%	23,94%
	Median	3016,11	10667,96	7543,55	5862,50
Response time (ms)	Mean	656461,00	2036,96	1474,59	1199,28
	% SD	–	27,97%	33,11%	40,04%
	Median	–	1948,40	1392,70	1122,35

Table 4
Performance comparison of SoSACO algorithm vs. other methods on social networks.

Topology		*Small World* (scenario ii)					
Experiment		first path found				best path found	
Method		Dijkstra	ACO	SoSACO v1 $S \cong 3\%$	SoSACO v2 $S \cong 3\%$	SoSACO v1 $S \cong 3\%$	SoSACO v2 $S \cong 3\%$
Cost	Mean	4,51	387,49	8,16	9,43	5,89	5,12
	%SD	13,97%	63%	8,73%	9,12%	0,05%	0,16%
	Median	4,00	346,75	8,11	9,35	5,88	5,00
Response time (s)	Mean	563,39	253,32	2,02	1,33	< 25	
	%SD	–	62%	34,15%	36,84%		
	Median	–	238,81	1,98	1,22		
Success rate	%	100%	90,6%	100%	100%	100%	100%

resulting in long detours to reach the destination. Clearly, this disadvantage for ACO becomes an advantage when using SoSACO: the food node is going to be located at a hub node, being even easier to the ants to reach it. Moreover, the ants don't need to be rightly directed at this node, because there are two breeds (departing from start and destination nodes respectively) that always find this node and meet there, thus each providing half of the complete path.

Consequently, as shown in Table 3, the cost of the paths provided by ACO is very high (almost 86 times higher), though it should be stated that it is much faster than Dijkstra (requires less than half the response time). However, the new version of SoSACO is 20,85 times faster and produces quasi-optimal solutions, as before stated. Besides, SoSACO has proven to be robust in this scenario, throwing a success rate of 100% in these experiments. It must be stated that it is theoretically possible, due to the stochastic nature of the algorithm, that some services do not reach a valid solution before reaching the maximum time threshold for the first path (500 s in this case). Although this eventuality is improbable and did not occur through these experiments, taking into account the previous statement is more appropriate to speak of a *success rate close to 100%.*

When comparing new version of SoSACO with the former, it has proven to be faster (needs less time, 66% on average) at the cost solution quality (first path is 15.56% longer on average). Besides, given the nature of the SoSACO algorithm, it can improve the path provided whenever more time is available. Through preliminary experimentation, it was found that most path improvements were attained within 15 s after the first path achievement. Consequently, it was decided to extend the experiment not only to first path finding, but also recording best path within 25 s in order to check solution improvements (for both versions of SoSACO). The new version of SoSACO showed solution enhancement in 82.9% of the cases, providing better solutions than the former version (13% shorter) in the same time lapse. Average time consumption for best solution before 25 s is 12,15 s for SoSACO v1 and 16,35 s for v2, which shows that the new version takes better advantage of extra-time than its predecessor (Table 4).

5.3. Evaluation on dynamic graphs

In this section, the dynamic version of SOSACO (see Section 4.6) is evaluated. With this aim, the first scenario (generic topology) is taken and prepared with dynamic conditions for the algorithm to be tested. Specifically, the graph will be subject to continuous changes: adding nodes or links, and dropping nodes or links (notice than dropping a node involves dropping several links). For these experiments, the mentioned changes are directed by the Barabási-Albert model (BA model (Barabási & Albert, 1999)), so dynamism is more realistic and utterly leads to a free-scale network, which better reflects the characteristics of networks representing real-life problems.

The sets of experiments (services) will be the same as used for evaluation in static conditions, as will be the parameterization of the method. The adapted version of SOSACO involves to take apart some spare ants to reinforce special cases. In this evaluation, from the total set of 500 individuals in the colony, 80% will be set from the beginning and the rest (20%) will be left for reinforcing. Besides, parameter h is quite important in several ways. A high value could be forcing the incorporation of reinforcement ants on low pheromone links dropping. Furthermore, all spare ants could be set too soon, and stop reinforcing for some time. On the other hand, low h value leads to better distribution of spare ants, but only reinforce high pheromone links dropping.

When changes occur with a time lag of 20 ms., almost all services solved by Dijkstra are useless: it was found when testing that over 97% of the solutions provided contained dropped elements (either nodes or links). For this experiment, four time lag values (20 ms, 50 ms, 100 ms, 200 ms) will be taken.

Through preliminary experimentation on the generic topology, it has been demonstrated that h = 2 fits the dynamicity set, though lower dynamicity (or other type of scenario) could have required another value, as depicted in Fig. 7. It should be noticed that

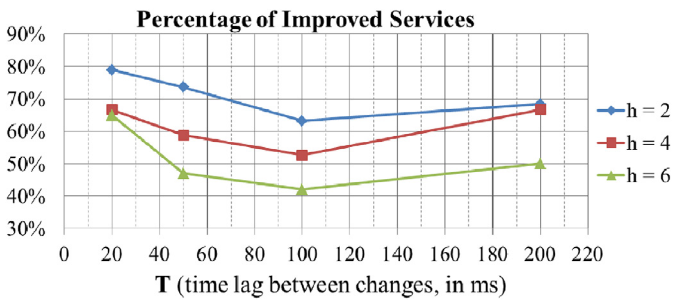

Fig. 7. Effect of the valuation of h on the generic scenario.

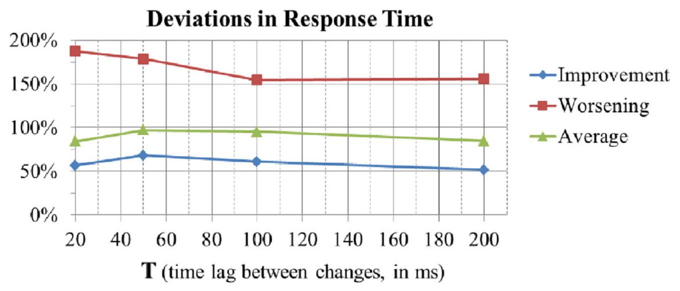

Fig. 8. Influence of Dynamicity on Response Time.

many services are being improved in response time (compared to the static graph case). This happens because of certain 'pruning' of misleading pheromone (small deposits) when removing links/nodes belonging to unsuccessful ant wandering. The amount of cases improving response time is increased with h = 2.

The improvement on SOSACO for adapting it to dynamic environments reduces the impact of losing useful links/nodes, but does not avoid it. Indeed, with high dynamicity, worsened services can show average response times of 188% the static case, as depicted in Fig. 8. When improved, the gain is lower, yet there are more cases. Globally, performance of this dynamic approach is even better than the static scenario. Though, there is still space for improvement in those few cases in which response time is nearly doubled.

6. Conclusions and future research

It has been described a new method for fast pathfinding in huge graphs. It is an extension of an already existent approach (Rivero et al., 2011). That previous version was based in preprocessing for creating *odor coverage areas* around a privileged node (the food node) that will later guide ants in their search, as if endowing them with the sense of smell. To the extent of our knowledge, that algorithm was faster than any other but unfortunately the search was restricted to services involving the food node (either as start node or destination).

This new version overcomes that restriction, providing paths between any two nodes. Main differences are found in (1) a variation of type *divide and conquer* for the base ant algorithm; (2) a better use of the advantages supported by the food node; and (3) the inclusion of a variant of the *tabu list* extension. The result is a new and more powerful method, which is able of providing solutions faster than the rest of the compared algorithms. The quality of the solutions is usually lower (the mean cost of the paths is higher), but it should be recalled that the main objective for this algorithm is precisely to minimize the response time, and that the focused applications are those systems that do not require the optimum solution but agility in responding. The loss of quality is due to the stochastic nature of the algorithm. Since the possible paths are millions, the possibility of attaining best route should be low, even with the progressive refinement provided by the algorithm. However, the quality of solutions found is very close to optimum. Most probable case is to be falling into a 'local maximum' that will be restraining its progress and preventing it from achieving optimum solution. Nevertheless, those local maximums are usually good enough solutions, and achieved in short time.

Besides, the here presented method is capable of refining the solution (that is, finding better solutions) when it is provided with more time. If more hardware resources are available, a parallel implementation can be considered. Due to its stochastic nature, parallelism will lead to generate more solutions (than a single server) in the same time, and therefore better solutions can be attained (with barely any change done on the method, just replicating it across a cluster of servers). Such performance enhancement is perhaps not relevant to be measured here, but should be taken into account when implementing the algorithm for real applications.

Regarding future challenges, authors aim enhance the possibilities of the method in dynamic graph scenarios. The adaptation in changing scenarios is an inherent characteristic to algorithms in the ACO family. Through the SoSACO design, these features have been observed. Though it counts on a preprocessing, it does not hinder the adaptation of SoSACO (if properly parameterized). On one hand, changes will rarely affect odored nodes, and in case they do they will be quickly restored. The efficiency in dynamic conditions is not only worse, but slightly better on average. The drawback is that those few cases in which efficiency is worse, performance has really been dropped. The challenge is to explore and improve those cases.

Acknowledgments

This study was partially funded by the Spanish Ministry of Industry, Tourism and Commerce through the CADOOH research project (TSI-020302201121), and the Spanish Ministry of Science and Innovation under the MOVES research project (TIN-201128336). Our sincere thanks to everyone involved in those projects, and to all who have supported us during this work.

References

Abdallah, H., Emara, H. M., Dorrah, H. T., & Bahgat, A. (2009). Using Ant Colony Optimization algorithm for solving project management problems. *Expert Systems with Applications, 36*(6), 10004–10015.
Ahmad, M., & Srivastava, J. (2008). An Ant Colony Optimization approach to expert identification in social networks. *First international workshop on social computing, behavioral modeling and prediction pheonix.*
Albert, R., & Barabási, A. L. (2009). Scale-free networks: A decade and beyond. *Science, 325*, 412–413.
Aparicio, S., Villazón-Terrazas, J., & Álvarez, G. (2015). A model for scale-free networks: Application to twitter. *Entropy, 17*(8), 5848–5867.
Azar, D., & Vybihal, J. (2011). An Ant Colony Optimization algorithm to improve software quality in prediction models: Case of class stability. *Information and Software Technology, 53*(4), 388–393.
Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science, 286*, 509–512. doi:10.1126/science.286.5439.509.
Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P. et al. (2014).Route planning in transportation networks. MSR Technical Report,
Bast, H., Funke, S., Matijevic, D., Sanders, P., & Schultes, D. (2007). In transit to constant shortest-path queries in road networks. In *Proceedings of WS on algorithm engineering and experiments*.
Binh, H. T. T., & Duong, N. T. (2015). Heuristic and genetic algorithms for solving survivability problem in the design of last mile communication networks. *Soft Computing, 19*(9), 2619–2632.
Ben Romdhane, L., Chaabani, Y., & Zardi, H. (2013). A robust ant colony optimization-based algorithm for community mining in large scale oriented social graphs. *Expert Systems with Applications, 40*(October 15 (14)), 5709–5718 ISSN 0957-4174.
Borgatti, S. P. (2005). Centrality and network flow. *Social Networks, 27*(1), 55–71.
Brito, J., Martínez, F. J., Moreno, J. A., & Verdegay, J. L. (2015). An ACO hybrid metaheuristic for close-open vehicle routing problems with time windows and fuzzy constraints. *Applied Soft Computing Journal, 32*, 154–163.
Bullnheimer, B., Kotsis, G., & Strauss, C. (1999). An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research, 89*, 319–328.
Chakraborty, A., & Manoj, B. S. (2015). An efficient heuristics to realize near-optimal small-world networks (2015). *21st National conference on communications NCC.*

Chan, E. P. F., & Lim, H. (2007). Optimization and evaluation of shortest path queries. *International Journal on Very Large Data Bases, 16*(3), 343–369.

Chan, E. P. F., & 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).

Delling, D., Holzer, M., Müller, K., Schulz, F., & Wagner, D. (2009). High performance multi-level routing. In C. Demetrescu, A. V. Goldberg, & D. S. Johnson (Eds.). In *The shortest path problem: 9th DIMACS implementation challenge, DIMACS book: Vol. 74* (pp. 73–92). American Mathematical Society.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2006). Highway hierarchies star. *9th DIMACS implementation challenge: Shortest paths, November*.

Dorigo, M. (1992). *Optimization, learning and natural algorithms* Doctoral Thesis. Italy: Dipartamento di Elettronica, Politecnico di Milano.

Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science, 344*, 243–278.

Dorigo, M., & Stützle (2004). *Ant colony optimization*. Cambridge: MIT Press.

Eaton, J., Yang, S., & Mavrovouniotis, M. (2016). Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays. *Soft Computing, 20*(8), 2951–2966.

Erwig, M. (2000). The graph voronoi diagram with applications. *Networks, 36*(3), 156–163.

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* (pp. 341–348).

Gladwell, M. (2000). *The tipping point: How little things can make a big difference*. © Little Brown.

Honiden, S., Houle, M. E., Sommer, C., & Wolff, M. (2010). *Approximate shortest path queries using Voronoi duals, transactions on computational science IX* (pp. 28–53). Springer.

Jindal, V., & Bedi, P. (2016). Reducing travel time in VANETs with parallel implementation of MACO (Modified ACO). *Advances in Intelligent Systems and Computing, 424*, 383–392.

Kar, A. K. (2016). Bio inspired computing – A review of algorithms and scope of applications. *Expert Systems with Applications, 59*(October 15), 20–32 ISSN 0957-4174.

Kautz, H., Selman, B., & Shah, M. (1997). Referral web: Combining social networks and collaborative filtering. *Communications of ACM, 40*(3), 63–65.

Kleinberg, J. (1999). The small-world phenomenon: An algorithmic perspective. *Proceedings of the 32nd ACM symposium on theory of computing, 2000, October* Also appears as Cornell Computer Science Technical Report 99-1776.

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. *Applied Intelligence, 32*(1), 88–95.

Leskovec, J. (2010). *SNAP: Network datasets: Slashdot social network*. Stanford University *http://snap.stanford.edu/data/soc-Slashdot0902.html* Accessed 09 November 2010.

Liben-Nowell, D. (2010). Wayfinding in social networks, algorithms for next generation networks. *Computer Communications and Networks Series*, 435–456.

Ma, G., Duan, H., & Liu, S. (2007). Improved Ant Colony Algorithm for global optimal trajectory planning of UAV under complex environment. *International Journal of Computer Science & Applications, 4*(3), 57–68.

Manaskasemsak, B., & Rungsawang, A. (2015). Web spam detection using trust and distrust-based ant colony optimization learning. *International Journal of Web Information Systems, 11*(2), 142–161.

Mehlhorn, K. (1988). A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters, 27*(3), 125–128.

Milgram, S. (1967). The small-world problem. *Psychology Today, 1*(May (1)), 61–67.

Min-Joonge, L., Sunghee, C., & Chin-Wan, C. (2016). Efficient algorithms for updating betweenness centrality in fully dynamic graphs. *Information Sciences, 326*(January 1), 278–296.

Mogren, O., Sandberg, O., Verendel, V., & Dubhashi, D. (2009). Adaptive dynamics of realistic small world networks. In *Proceedings of the European Conference on Complex Systems*.

Myszkowski, P. B., Skowroński, M. E., Olech, Ł. P., & Oślizło, K. (2015). Hybrid ant colony optimization in solving multi-skill resource-constrained project scheduling problem. *Soft Computing, 19*(12), 3599–3619.

Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM Review, 45*(2), 167–256.

Niu, Y., Wang, S., He, J., & Xiao, J. (2014). A novel membrane algorithm for capacitated vehicle routing problem. *Soft Computing, 19*(2), 471–482.

Ramos, G. N., Hatakeyama, Y., Dong, F., & Hirota, K. (2009). Hyperbox clustering with ant colony optimization (HACO) method and its application to medical risk profile recognition. *Applied Soft Computing, 9*(2), 632–640.

Rivero, J., Cuadra, D., Calle, J., & Isasi, P. (2011). *Using the ACO algorithm for path searches in social networks*. Applied Intelligence, Springer. doi:10.1007/s10489-011-0304-1.

Sandberg, O. (2006). Distributed routing in small-world networks. In *Proceedings of the 8th workshop on algorithm engineering and experiments* (pp. 144–155).

Sankaranarayanan, J., & Samet, H. (2009). Distance oracles for spatial networks. In *Proceedings of the 25th IEEE international conference on data engineering* (pp. 652–663).

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).

Senarclens de Grancy, G., & Reimann, M. (2016). Vehicle routing problems with time windows and multiple service workers: A systematic comparison between ACO and GRASP. *Central European Journal of Operations Research, 24*(1), 29–48.

Tang, L., & Liu, H. (2010). Graph mining applications to social network analysis. In C. Aggarwal, & H. Wang (Vol. Eds.), *Managing and mining graph data*. In *Advances in data base systems 40* (pp. 487–514). Springer.

Wang, P., Lin, H.-T., & Wang, T.-S. (2016). An improved ant colony system algorithm for solving the IP traceback problem. *Information Sciences, 326*, 172–187.

Wang, X. F., & Chen, G. (2003). Complex networks: Small-world, scale-free and beyond. *Circuits and Systems Magazine IEEE, 3*(1), 6–20.

Watts, D. J., & Strogatz, S. (1998). Collective dynamics of “small-world” networks. *Nature, 393*, 440–442.

Yu, J. X., & Cheng, J. (2010). Graph reachability queries: A survey. In C. Aggarwal, & H. Wang (Vol. Eds.), *Managing and mining graph data*. In *Advances in data base systems 40* (pp. 181–215).