Robot Routing in Sparse Wireless Sensor Networks with Continuous Ant Colony Optimization

Giovanni Comarela, Kênia Gonçalves, Gisele L. Pappa, Jussara Almeida and Virgílio Almeida
Computer Science Department, Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
{giovannicomarela, keniacarolina, glpappa, jussara, virgilio}@dcc.ufmg.br

ABSTRACT

Sparse wireless sensor networks are characterized by the distances the sensors are from each other. In this type of network, gathering data from all sensors in a point of interest might be a difficult task, and in many cases a mobile robot is used to travel along the sensors and collect data from them. In this case, we need to provide the robot with a route that minimizes the traveled distance and allows data collection from all sensors. This problem can be modeled as the classic Traveling Salesman Problem (TSP). However, when the sensors have an influence area bounded by a circle, for example, it is not necessary that the robot touches each sensor, but only a point inside the covered area. In this case, the problem can be modeled as a special case of the TSP with Neighborhoods (TSPN). This work presents a new approach based on continuous Ant Colony Optimization (ACO) and simple combinatorial technique for TSP in order to solve that special case of TSPN. The experiments performed indicate that significant improvements are obtained with the proposed heuristic when compared with other methods found in literature.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication, Sensor networks

General Terms

Algorithms, Performance, Experimentation

Keywords

Global Optimization, Ant Colony Optimization, Wireless Sensor Networks

1. INTRODUCTION

Aggregating information collected by sensors in wireless sensor networks is essential. However, this task might become difficult for a special type of sensor network which is characterized by nodes being geographically distant from each other. In this case, making sensors communicate among themselves and deliver information to a point of interest can be very costly or even prohibitive, specially due to energy restrictions (in case of battery-powered sensors).

A possible solution to this problem is to use a mobile robot to travel to all sensors, download the collected data and then return to its base station. However, note that it is not necessary for the robot to make physical contact with the sensors to communicate, once each sensor has a influence area, specified by a circumference or disk. Hence, the robot only needs to reach the border (one point) of this area to start the data transfer. In this case, the problem of routing robots is find a path from where the robot can track and collect information from all sensors with the lowest possible cost.

Figure 1 shows a sample instance of the routing problem with five sensors, and a valid path connecting all regions from starting point s. The influence areas of the sensors may be different due to different battery charges and different sensor types. Although there might be overlapping in these influence areas, this work assumes that no influence areas overlap.

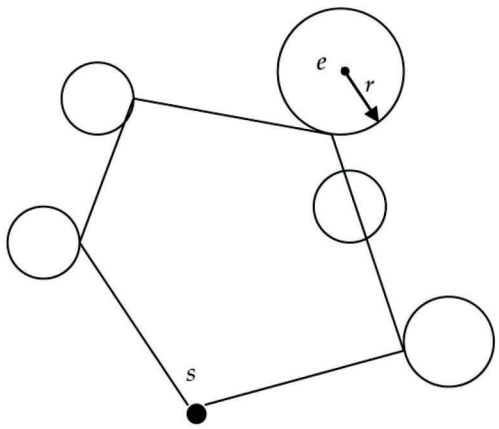

Figure 1: Example of robot routing in a network with five sensors [9].

The robot routing problem in a wireless sensor network can be treated as a special case of the Traveling Salesman Problem with Neighborhood (TSPN) [7], where the neighborhoods are disjoint circles in a two-dimensional space. The TSPN is a generalization of the Traveling Salesman Problem

GECCO'11, July 12–16, 2011, Dublin, Ireland.

(TSP), which is known to be NP-Hard [1]. For simplicity of notation, throughout this text the TSPN acronym will be used to refer to the problem of optimal routing in sparse sensor networks with mobile robots.

Heuristics to solve the TSPN usually divide the problem solution in two phases [9, 2]: a continuous and a combinatorial phase, and each of these phases is solved separately. The approach proposed here differs from the previous as it considers both the continuous and combinatorial phases simultaneously. The idea is to combine the strategies used by Yuan et. al. [9] and Safra et. al. [7] using a continuous version of Ant Colony Optimization (ACO) [4].

Experiments show that solving both phases simultaneously leads to better results than those obtained by other approaches found in the literature (and that address each phase independently) with high statistical confidence.

The remainder of this paper is organized as follows: Section 2 presents a formal definition of the TSPN problem and Section 3 presents related work. The new approach proposed and the methods previously published that are its base are in Section 4. Section 5 presents the findings of this study, and compare it with the methods already proposed int he literature. Finally, Section 6 presents conclusions and interesting points for future studies.

2. PROBLEM DEFINITION

The objective of this section is provide the basic definitions of objects in a sparse wireless sensor network. The notation was borrowed from [9].

In the networks considered, each wireless sensor has a neighborhood (influence area), represented by a circular region in the plane, and represented by a center coordinate $e = (x, y)$ and radius r. An instance of the problem is specified by a initial point s and a set of n disks $\{s, (e_1, r_1), \ldots, (e_n, r_n)\}$. The radius of each disk can be different from each other depending on the type of sensor and how much battery it has left. It is assumed that the sensor network is sparse and there is no intersection between the circles. In practice, note that if the robot initially have access to data of one or more sensors, these can be removed from the routing problem.

The robot can start communicating with the sensor as soon as it reaches their influence area (in other words, once it reaches its border). After downloading data from a sensor, the robot moves to the next point. Consequently, the path of the robot is built sequentially, connecting s with the other n border points (access points) of each sensor.

Let p be the set of n points to visit, and π a permutation on p. Then a valid path for the robot will be specified by the tuple $< s, p, \pi >$, where the objective of the problem is to find p and π that minimize the following function:

$$d(s, p_{\pi_1}) + d(s, p_{\pi_n}) + \sum_{i=1}^{n-1} d(p_{\pi_i}, p_{\pi_{i+1}}), \qquad (1)$$

where $d(P_1, P_2)$ is a function that computes the Euclidian distance between points P_1 and P_2.

In Eq. 1 there are two groups of parameters to be optimized: the visiting points (p) and the order they are visited in the path (π). Once that each region is represented by a circle, each point can be represented by an angle α, as shown in Figure 2. This representation was used to have only one variable to represent each point $p_i = (x_i, y_i)$ instead of two. In addition, note that the points will always be in the border of the influence area of each sensor.

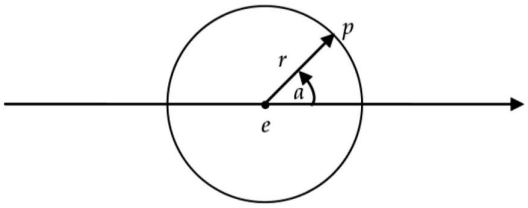

Figure 2: Visiting point p represented by α [9].

3. RELATED WORK

What makes TSPN an interesting problem to be studied is the fact that in any given wireless network with n sensors to be visited, $2n$ variables must be optimized. From these $2n$ variables, half are related to the vector π (combinatorial component of the problem) and the other half regard the visiting point (continuous component, represented by the angles α_i, $i = 1, \ldots, n$). Simultaneously optimizing these $2n$ parameters (ideal situation) is a very complex task, as it involves exploring two search spaces of very different nature.

Due to this degree of complexity, the main approaches found in the literature divide this problem into two subproblems: one to solve the continuous part and another to solve the combinatorial part of the problem. In order to illustrate that, Sections 3.1 and 3.2 present two common approaches used to deal with this problem, while Section 4 presents the heuristic proposed in this paper, which combines combinatorial and continuous searches for TSPN.

3.1 Greedy Solution

Elbassioni et al. [2] proposes a greedy approach for solving the Euclidean TSPN with disjoint regions of varying sizes. It is described in Algorithm 1:

Algorithm 1: Greedy Solution by [2]

Data: n circles in the plane specified by $(e_1, r_1), \ldots (e_n, r_n)$
1 Sort the n regions in increasing radius order
2 Select p_1 randomly in the smallest region
3 **for** $i \leftarrow 2$ **to** n **do**
4 | Select p_i that minimizes $d(p_i, p_j)$, $j = 1, \ldots, i-1$
5 **end**
6 Solve an Euclidian TSP for points $p_1, \ldots, p_n$

In order to find a solution for TSPN, Algorithm 1 first determines the n points to be visited in each region (part of the continuous solution) and then establishes the visiting order of these points by solving a TSP (solution for the combinatorial part). In the context of the problem addressed here, line 2 will select the actual s, which is treated as a circumference of radius 0.

This algorithm has low computational complexity and is deterministic (assuming an algorithm for solving the TSP also has these characteristics). However, due to its greedy nature, in certain TSPN instances it can obtain poor results. In fact, it can produce results up to 36 times worst than the optimal [2]. Note that this is without considering the error obtained by the TSP approach, which is assumed to

be optimal. In this work, the heuristic used to obtain a solution for TSP is the Nearest Neighbor with 2-OPT [5].

3.2 Continuous Heuristics Solution

The work presented in [9] proposes a methodology for solving the TSPN that is also based on the principle of decomposing the problem into continuous and combinatorial steps. The main difference of this approach to Algorithm 1 is that it solves TSP using the coordinates of the center of circumferences and then determines the points of visit in each region using an optimization algorithm. Algorithm 2 illustrates this idea.

Algorithm 2: TSPN Algorithm [9]

Data: n circles in the plane given by $(e_1, r_1), \ldots (e_n, r_n)$
1 Solve an Euclidean TSP for the points $e_1, \ldots, e_n$
2 Search for the best n hitting points based on permutation given in step 1

Let π^c be a permutation of sensors (center of the circles) obtained from an optimal TSP solution and π^r a permutation of the regions obtained by an optimal solution of TSPN. The key to Algorithm 2 is to use π^c instead of unknown π^r in the search for optimal points of visit.

The advantage of this approach is that, given a permutation of the circles, it is possible to accurately determine the access points. This can be done using advanced optimization techniques rather than the greedy approach used in Algorithm 1. The validity of this heuristic is given by the strong relationship between the TSPN and the associated TSP. For example, when all circles have radius equal to 0, the TSPN solution is identical to the TSP. The idea is to rely on the fact that as the sensors are distant from each other and the sizes of the disks become smaller compared with the distances between their centers, this relation tends to remain true. However, there is no guarantee that π^c and π^r are always equal.

For any disk with radius greater than zero, the visiting point is given by the angle α, as shown in Figure 2. In general, $\alpha \in [0, 2\pi)$. An important point of the algorithm presented in [9] is the fact that, knowing the order to visit the regions, it is possible to reduce the search space significantly without impacting the accuracy of the optimization algorithm. In order to understand how this can be done, consider two regions A and B, where A must be visited immediately after B. All possible visiting points in A can be represented as a result of the intersection between the circular boundary of A and the lines passing through the points visited in B. Figure 3 shows the typical situation described in this scenario. Assuming that the path is directed from B to A, the angles values θ and γ (in absolute) are given by:

$$\theta = \arcsin\left|\frac{r_A - r_B}{d}\right| \text{ and} \quad (2)$$

$$\gamma = \arcsin\left|\frac{y_A - y_B}{d}\right|, \quad (3)$$

where r_A and y_A are the y of A coordinate and radius, r_B and y_B are the y of B coordinate and radius, and d is the distance between the centers of A and B.

Observe that in, Figure 3, regardless of the access point the robot uses to visit point B, $\alpha \in [-\frac{\pi}{2} - \gamma + \theta, \frac{\pi}{2} - \gamma - \theta]$. In the general case, the signs of the angles θ and γ can vary according to the relation between r_B and r_A, and also according to the direction of the path (B for A or otherwise). A discussion of these details can be found in [9].

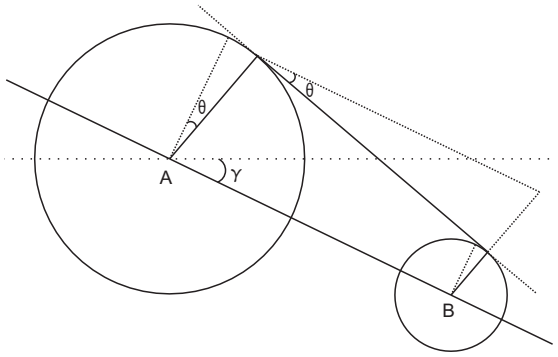

Figure 3: Example of distribution of the visited points in disk A

Step 2 of Algorithm 2 involves a process of continuous optimization in a n-dimensional space, where n is the number of sensors in the network. In [9], three evolutionary algorithms were used for this task. Experiments conducted showed that the three strategies had similar results. Thus, for purposes of comparison with [9], this work uses only one of these approaches regarding the continuous optimization problem. This approach is known as (1+1)-*Evolution Strategy* [8].

It is not in the scope of this paper to explain how these algorithms work. However, its pseudo-code is presented in Algorithm 3. In this algorithm, X is a n-dimensional vector and $f(X)$ is the objective function to be minimized. $G_j(X) \geq 0$, $j = 1, \ldots, m$ is a set of restrictions that must be respected. $X_E^{(g)}$ represents the best solution found until the iteration g and $X_N^{(g)}$ the candidate solution in iteration g.

Algorithm 3: $(1+1)-$ Evolution Strategy

Data: function $f : R^n \mapsto R$ and constraints $G_j : R^n \mapsto R$, $j = 1, \ldots, m$
1 Define $X_E^{(0)} = \{X_{E,i}^{(0)},\ i = 1, \ldots, n\}$ such that $G_j(X_E^{(0)}) \geq 0$, $j = 1, \ldots, m$ (initial solution)
2 $g \leftarrow 0$
3 **repeat**
4 $\quad X_N^{(g)} = X_E^{(g)} + z^{(g)}$ with components $X_{N,i}^{(g)} = X_{E,i}^{(g)} + z_i^{(g)}$, where $z_i^{(g)}$ is a random variable normally distributed with mean μ and variance σ^2
5 $\quad$ **if** $f(X_N^{(g)}) < f(X_E^{(g)})$ **and** $G_j(X_N^{(g)}) \geq 0\ \forall j$ **then**
6 $\quad\quad X_E^{(g+1)} \leftarrow X_N^{(g)}$
7 $\quad$ **else**
8 $\quad\quad X_E^{(g+1)} \leftarrow X_E^{(g)}$
9 $\quad$ **end**
10 $\quad g \leftarrow g + 1$
11 **until** *stop criterion*;

Preliminary experiments set $\mu = 0$, $\sigma = 0.01$ and the stop criterion to a maximum of 200 000 iterations in the main loop (parameters similar to those used in [9]). In the context

of TSPN, the function $f(X)$ is the path length that will be traveled by the mobile robot to visit all sensors, and the constraints are given by the limits obtained for the angles α.

4. HEURISTIC PROPOSED

In contrast with the previous methods proposed in the literature, this work suggests to combine the combinatorial and continuous phases of TSPN to optimize the paths found by the mobile robots. This is done by using a Ant Colony Optimization (ACO) algorithm, which uses as its initial solution a path built by Algorithm 1, and optimizes this initial path using a continuous ACO. After this solution converges, the ACO looks for a better sensor path again. This procedure is detailed in Algorithm 4, which describes the two main steps of the heuristic, which are:

1. *Solution Construction:* In this step, a set of artificial ants construct solutions for the instance of the optimization problem. This process is stochastic and considers a pheromone trail and specific information about the problem;

2. *Pheromone Update:* Pheromone trails must be modified according to past experiences of the ants. This process consists of two phases:*evaporation* and *intensification*.

In the context of TSPN, each artificial ant constructs a solution by generating a pseudo-random number from a random variable with normal distribution. This random variable represents the angle α showed in Figure 2. In other words, for each sensor, a random variable with normal distribution with mean μ_i and variance σ_i^2, $(i = 1, \ldots, n)$ is generated, and solutions are constructed sampling from these variables.

The values generated from the normal distribution (here denoted by x_i, $i = 1, \ldots, n$) may not be in the valid interval $[0, 2\pi)$. Hence, for each of them the following correction must be made:

$$x_i \leftarrow \begin{cases} 0, \text{ if } & x_i < 0 \text{ or } x_i \geq 2\pi \\ x_i, \text{ if } & 0 \leq x_i < 2\pi \end{cases} \quad (4)$$

During the solution construction phase, two types of pheromone are incorporated to the method: the first is represented by the vector $\mu = [\mu_1, \ldots, \mu_n]$ and the second by $\sigma = [\sigma_1, \ldots, \sigma_n]$.

In the second step, the pheromone trails are updated in order to make the algorithm converge. In [4] the authors suggest that this update must be performed in two phases. In the phase of *evaporation* μ and σ are modified as follow:

$$\mu \leftarrow (1 - \rho)\mu \quad (5)$$

$$\sigma \leftarrow (1 - \rho)\sigma, \quad (6)$$

where ρ is the evaporation constant $(0 < \rho < 1)$.

The phase of *intensification* aims to make solutions generated in the next iteration closer to the best ones generated by the algorithm in the current iteration. The values of μ and σ are modified as follow:

$$\mu \leftarrow \mu + \rho x \quad (7)$$

$$\sigma \leftarrow \sigma + \rho|x - \mu|, \quad (8)$$

where x is the vector containing the best solution found by the algorithm.

One important process of the algorithm is how to initialize μ and σ. Each component of μ is randomly initialized in such a way that $0 \leq \mu_i < 2\pi$. For the components of σ, the values assigned correspond to the arithmetic mean of the lower and upper bounds of each variable, that is, $\sigma_i = (2\pi - 0)/2 = \pi$.

The steps of *solution construction* and *pheromone update* must be repeated until the algorithm converges. In order to detect this state of convergence, a factor, denoted by c_f, must be computed. In this work c_f is given by:

$$\frac{1}{n} \sum_{i=1}^{n} \frac{\sigma_i}{\pi}. \quad (9)$$

In the initialization step, $c_f = 1$, and the idea is to execute the algorithm until c_f becomes close to 0. When this happens, each σ_i is also close to 0 (in Equation 9 is possible to see that $c_f \to 0$ if, and only if, for all i, $\sigma_i \to 0$) . So, once the value of c_f become smaller than a threshold it is possible to say that the algorithm has converged.

Algorithm 4 presents the pseudo-code of the proposed method.

Algorithm 4: *Proposed Heuristic*

```
   Data: n circles in the plane given by
         (e_1, r_1), ... (e_n, r_n)
 1 Use Algorithm 1 to solve the problem
 2 bestCost = cost of solution found by Algorithm 1
 3 bestSolution = solution found by Algorithm 1
 4 π = circles permutation found by Algorithm 1
 5 Initialize μ and σ
 6 Initialize a premature convergence threshold p
 7 repeat
 8     Generate Candidate Solution (Equation 4)
 9     cost = Cost of Candidate Solution (Equation 1)
10     if cost < bestCost then
11         bestCost = cost
12         bestSolution = Candidate Solution
13     end
14     Evaporate μ and σ
15     Intensify μ and σ
16     c_f = Convergence Factor (equation 9)
17     if c_f < p then
18         Use 2-OPT to improve π, resulting in π_new
19         cost = Cost of Solution (use π_new in Eq. 1)
20         if cost < bestCost then
21             π = π_new
22             bestSolution = Solution generating π_new
23         end
24         Update p
25         Re-initialize μ and σ
26     end
27 until stop criterion;
```

In lines 1 to 4, the greedy algorithm (Algorithm 1) presented in Section 3.1 is used to solve the original problem. The permutation of circles and the points obtained from this algorithm are used as initial solution for the heuristic proposed.

From this point, the continuous version of ACO is used

to search the best angles (α_i, $i = 1, \ldots, n$), representing the visiting points given the actual permutation of circles π.

Line 5 initialize the vectors μ and σ. The parameter p in line 6 is a threshold used to decide if the algorithm has converged or not. The stop criterion for the main loop (lines 6 to 26) is to check if $c_f < p$.

Line 7 generates candidates solutions for the angles from the pheromone vectors μ and σ. This procedure is done by generating, for each sensor, pseudo-random number from a random variable with normal distribution with mean μ_i and variance σ_i^2, and correcting the solution using Equation 4. After that, Line 8 verifies if this new solution is better that the best solution founds so far. In a positive case, Lines 9 to 12 are responsible for saving the new solution as the best one.

In the next step the pheromone trails are updated through the evaporation and intensification of the vectors μ and σ in the lines 13 and 14.

Line 15 computes the convergence factor according to Equation 9. If c_f is smaller than p, it indicates the algorithm is close to converge. At this point, the algorithm tries to improve the order of visit of the sensors. The idea consists of using the algorithm 2-OPT [5] to search a better order to visit the sensors. This algorithm consists of iteratively removing two edges of the tour and replace these with two different edges that reconnect the fragments created by the edge removal, resulting in a new, and possibly shorter, tour. Such approach is a simple and classical way to improve TSP tours. More sophisticated strategies can be also considered in this point. If a better solution is found in line 15, lines 18 to 22 perform the necessary operations to save this solution as the best one for the next iterations.

Line 23 updates the threshold p. In this work, the rule $p \leftarrow 0.1p$ is used. After that the pheromone trails are reinitialized. The vector μ receives the value associated with the best solution found and σ its initial value. The idea of decreasing p is to try to improve the combinatorial component of the problem several times, but only when the continuous part is stable.

5. EXPERIMENTAL RESULTS

This section presents the experimental results obtained and compares the proposed heuristic with the solution for this TSPN proposed in [9]. To represent a wireless sensor network, points were generated randomly in the Euclidian Plan (limited on the region $[10^5] \times [10^5]$) and each point was assigned a random radius in such a way that there were no overlapping circumferences. The number of sensors (n) varied in the set of values $\{10, 50, 100, 300\}$, which are almost the same values used in [9] [1]. For each n, 10 random networks were generated, making a total of 40 test instances. For each instance, Algorithms 4 and 2 were run 15 times, in order to perform a stronger statistical analysis.

It is important to highlight that the proposed method is not being compared with Algorithm 1, since Algorithm 4 uses its solution as an initial path for the ACO to optimize (hence, the results of Algorithm 4 are always at least as good as those obtained by Algorithm 1), and also a comparison between Algorithm 1 and Algorithm 2 is provided in [9], where the latter outperforms the former.

The results presented here use parameter configurations obtained in initial experiments for Algorithm 4. p (line 6) receives as its initial value 0.01, and this value decreases over time until 10^{-7}. The evaporation parameter is $\rho = 0.00001$.

Experiments were performed using an Intel *duo core* processor with 2.1GHz and 4GB of RAM. The programming language used was Java (version 1.6). All statistical analysis was performed with **R** [6].

5.1 Analysis

This section presents the results obtained for the 40 test instances simulated. Table 5.1 shows the results for instances with 10 and 50 sensors while Table 5.1 for instances with 100 and 300 sensors. Each cell of the table contains two values, the first is the mean of the 15 executions and the second (between parenthesis) is the standard deviation. The columns present, respectively, Algorithm 4 and 2.

Table 1: Comparison between Algorithm 4 and Algorithm 2 for $n = 10$ and $n = 50$

Alg.	4	2	4	2
	n			
Instance	10		50	
1	**279314.12** (0.10)	**279314.11** (0.02)	461642.25 (1394.38)	496530.11 (1.31)
2	**256195.40** (247.77)	**256101.35** (0.02)	531722.49 (971.7)	546163.48 (1.12)
3	244152.56 (1751.18)	244604.74 (0.01)	550652.81 (200.21)	580817.77 (1.99)
4	**220174.03** (22.44)	**220163.25** (0.01)	513158.63 (1013.61)	533571.87 (3.51)
5	258053.98 (120.64)	265651.95 (0.03)	543353.19 (1325.31)	555439.65 (2.12)
6	230221.85 (31.22)	264413.66 (0.02)	479034.41 (2174.24)	554386.14 (1.35)
7	218135.54 (162.35)	257118.70 (0.04)	491604.99 (1245.77)	528963.79 (1.19)
8	**279756.11** (5.21)	**279754.18** (0.01)	560036.78 (472.99)	601427.53 (2.82)
9	215990.25 (78.78)	216437.85 (0.02)	537140.42 (878.20)	554850.16 (1.98)
10	**235199.00** (203.64)	**235145.14** (0.04)	504115.72 (2330.49)	594933.80 (4.79)

It can be noticed that, for 10 sensors, each algorithm had a better performance (shorter path length) in 5 test cases. For 50 and 100 sensors the proposed algorithm had a lower mean for all simulated cases. In the last case (300 sensors) Algorithm 4 was better than Algorithm 2 in 8 out of 10 cases. Even though these results indicate that Algorithm 4 is better than 2 when the average path length is analyzed, it is important to note that the variance of Algorithm 4 is much greater than the one in Algorithm 2. Hence, from this descriptive analysis it is not possible to say which algorithm is the best. Hence, a *Wilcoxon* statistical test [3] was performed, and the following hypotheses established:

$$H_0 : \mu_4 - \mu_2 = 0 \text{ versus } H_1 : \mu_4 - \mu_2 < 0,$$

where μ_4 represents the average path length found by Algorithm 4 and μ_2 represents the average path length found by Algorithm 2.

Table 5.1 presents the p-values for the results of the hypotheses tests performed to compare the two algorithms (remarking that a low p-value indicates that H_0 has high probability to be false). This table gives statistical confidence

[1] In [9] the authors present their results for $n \in \{19, 50, 100, 200, 300\}$. But only one instance for each n.

Table 2: Comparison between Algorithm 4 and 2 for $n = 100$ and $n = 300$

Alg.	4	2	4	2
	n			
Instance	100		300	
1	687595.79 (779.51)	727089.87 (4.46)	1181593.45 (940.46)	1261379.67 (28.10)
2	638779.09 (426.55)	714987.02 (2.10)	**1282028.74** (1334.31)	**1230194.1** (19.68)
3	701310.79 (817.85)	763780.80 (1.67)	1310030.34 (4376.59)	1355969.36 (17.68)
4	698096.98 (1600.14)	780147.42 (1.82)	**1256780.78** (788.77)	**1255346.5** (14.57)
5	702397.56 (240.64)	737594.03 (2.19)	1229095.85 (2043.82)	1258425.07 (11.02)
6	737609.59 (1242.04)	753524.60 (3.80)	1283278.27 (2133.70)	1332464.88 (12.32)
7	749465.35 (677.13)	792600.49 (3.41)	1257135.44 (1213.70)	1335256.02 (17.17)
8	674670.63 (204.92)	693008.23 (7.21)	1145354.16 (1898.42)	1284000.23 (12.70)
9	720883.43 (6211.17)	792159.34 (5.34)	1226059.68 (1547.62)	1340092.48 (17.41)
10	661476.58 (1092.54)	705310.28 (5.19)	1207814.39 (869.96)	1293996.4 (26.16)

to the conjectures discussed in the last paragraph. With a small significance level, it is possible to say that in tests with 50 and 100 sensors Algorithm 4 had a better performance than Algorithm 2. Moreover, int 8 of ten cases considering 300 sensors, as discussed before, the results were obtained by Algorithm 4 are also better than those obtained by Algorithm 2. In the case of 10 sensors it is not possible to conclude that one algorithm is better than the other.

Table 3: p-values of the tests comparing Algorithms 4 and 2

Instance	n 10	50	100	300
1	**0.8053**	3.05e-05	3.05e-05	3.05e-05
2	**0.1651**	3.05e-05	3.05e-05	**1**
3	0.003357	3.05e-05	3.05e-05	3.05e-05
4	**0.1651**	3.05e-05	3.05e-05	**1**
5	3.05e-05	3.05e-05	3.05e-05	3.05e-05
6	3.05e-05	3.05e-05	3.05e-05	3.05e-05
7	3.05e-05	3.05e-05	3.05e-05	3.05e-05
8	0.04163	3.05e-05	3.05e-05	3.05e-05
9	3.05e-05	3.05e-05	3.05e-05	3.05e-05
10	**0.9156**	3.05e-05	3.05e-05	3.05e-05

5.2 Examples of instance

The objective of this section is to present two examples of a wireless sensors networks and the routes generated by the algorithms. The networks have only 25 and 40 sensors in order to provide a better visualization. Figures 4 and 5 present the networks and routes for each algorithm.

In Figure 4, for Algorithm 2 the path length is 409,936.07, while for Algorithm 4 it is 326,781.98 representing an improvement of over 8%. In the second case, the path length is 445126.58 and 427560.80 for Algorithms 2 and 4, respectively (an improvement of 4%). Notice that, in these cases, the main difference between the two solutions is the order

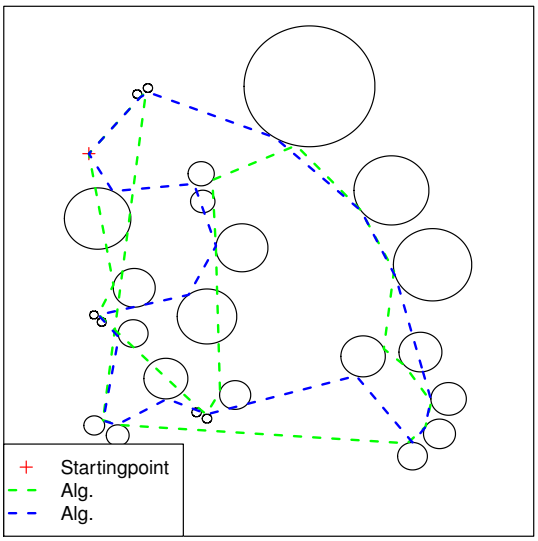

Figure 4: Example of an instance for a network with 25 sensors.

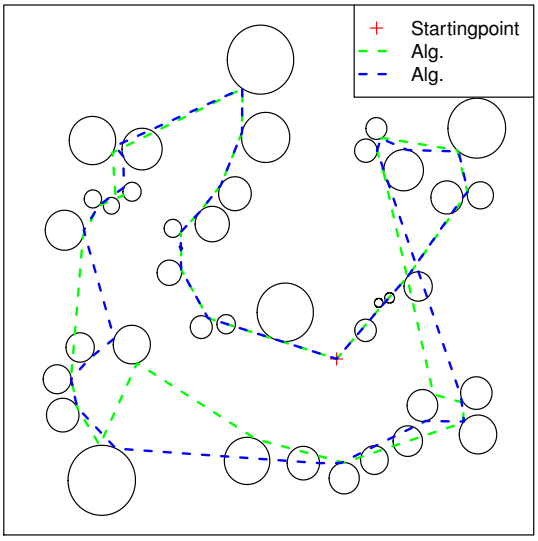

Figure 5: Example of an instance for a network with 40 sensors.

in which the sensors are visited. That's why Algorithm 4 found better solutions than Algorithm 2.

6. CONCLUSIONS

This work presented a new heuristic to solve the problem of robot routing in sparse sensor networks when the sensor neighborhoods are circumferences in the Euclidian plane. We proposed a heuristic that evaluates simultaneously the continuous and combinatorial components of the problem using the Ant Colony Optimization technique in its continuous version.

In order to evaluate the performance of the proposed heuristic, it was compared with the most recent heuristic found

in the literature in 40 test instances. Statistical hypotheses tests show that the proposed heuristic outperforms the other strategies for networks with many sensors. However, when the size of the network is small ($n = 10$) no significant improvement is achieved. Note that in practical terms large networks are the ones we have most interest in.

The proposed heuristic can be easily adapted to solve the same problem with other kinds of neighborhoods. For example, instead of circumferences in the plane, 3D scenarios, using spherical coordinates can be considered. One important application would be to work with aquatic wireless sensors networks, where sensors can have different locations in the Euclidian plane and different depths.

7. REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, New York, 2001.

[2] K. Elbassioni, A. V. Fishkin, N. H. Mustafa, and R. Sitters. Approximation algorithms for euclidean group tsp. In *In Automata, languages and programming : 32nd International Colloquim, ICALP 2005*, pages 1115–1126. Springer, 2005.

[3] R. V. Hogg, A. Craig, and J. W. Mckean. *Introduction to Mathematical Statistics*. Prentice Hall, 6th edition, June 2004.

[4] M. Kong and P. Tian. Application of aco in continuous domain. In L. Jiao, L. Wang, X. Gao, J. Liu, and F. Wu, editors, *Advances in Natural Computation*, volume 4222 of *Lecture Notes in Computer Science*, pages 126–135. Springer Berlin / Heidelberg, 2006.

[5] C. Nilsson. Heuristics for the traveling salesman problem. Technical report, Linköping University, 2003.

[6] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[7] S. Safra and O. Schwartz. On the complexity of approximating tsp with neighborhoods and related problems. *Computational Complexity*, 14(4):281–307, 2006.

[8] H.-P. P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[9] B. Yuan, M. Orlowska, and S. Sadiq. On the optimal robot routing problemin wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1252–1261, September 2007.