Optimized Trajectory of a Robot Deploying Wireless Sensor Nodes

Ines Khoufi, Erwan Livolant, Pascale Minet, Mohamed Hadded, Anis Laouiti

Optimized Trajectory of a Robot Deploying Wireless Sensor Nodes

Ines Khoufi, Erwan Livolant, Pascale Minet,
Inria
Rocquencourt, 78153 Le Chesnay Cedex, France
Email: ines.khoufi@inria.fr, erwan.livolant@inria.fr,
pascale.minet@inria.fr

Mohamed Hadded, Anis Laouiti
TELECOM SudParis, CNRS Samovar UMR 5157
91011 Evry Cedex, France
Email: mohamed.hadded@telecom-sudparis.eu,
anis.laouiti@telecom-sudparis.eu

***Abstract*—Mobile robots can be used to deploy static wireless sensor nodes to achieve the coverage and connectivity requirements of the applications considered. Many solutions have been provided in the literature to compute the set of locations where the sensor nodes should be placed. In this paper, we show how this set of locations can be used by a mobile robot to optimize its tour to deploy the sensor nodes to their right locations. In order to reduce both the energy consumed by the robot, its exposure time to a hostile environment, as well as the time at which the wireless network becomes operational, the optimal tour of the robot is this minimizing the delay. This delay must take into account not only the time needed by the robot to travel the tour distance but also the time spent in the rotations performed by the robot each time it changes its direction. This problem is called the Robot Deploying Sensor nodes problem, in short RDS. We first show how this problem differs from the well-known traveling salesman problem. We then propose an integer linear program formulation of the RDS problem. We propose various algorithms relevant to iterative improvement by exchanging tour edges, genetic approach and hybridization. The solutions provided by these algorithms are compared and their closeness to the optimal is evaluated in various configurations.**

I. Context and motivations

More and more applications are supported by wireless sensor networks. They cover areas as diverse as structural health monitoring, smart metering, industrial process monitoring, precision farming, smart cities, control of traffic lights, smart home, etc. The main reason for this tremendous development lies in the ease of deployment of wireless sensor networks. However, to meet the application requirements in terms of coverage and connectivity while minimizing the number of wireless sensors deployed, some rules must be followed. In short, full coverage of an area means that any event occurring in this area will be detected by at least one sensor node. Connectivity means that the information related to any event detected by a sensor node can be delivered to a special wireless node, called the sink, in charge of processing the data gathered from the sensor nodes. Many papers in the literature deal with these two big issues that are coverage and connectivity, leading to various problems mainly depending on the item to cover (area, points of interest, barrier) and on the type of coverage requested (full/partial, permanent/intermittent). The interested reader can refer to [1] for a survey of these problems and their solutions.

Concerning the deployment achieved to meet these application requirements, they differ in their goal, their constraints and their implementation (e.g., centralized versus distributed). Most deployments aim at minimizing the number of sensor nodes deployed for cost reasons to achieve the application requirements. Another goal that is frequently encountered in crisis situation (e.g., after a disaster) where a wireless sensor network must be fastly deployed in order to on the one hand help rescuers to save victimes and on the other hand assist in damage assessment. In such cases, the goal is to minimize the time needed to deploy an operational wireless network. This goal is also targeted in hostile environment, where the exposure duration must be reduced. For cost reasons, static sensor nodes are more frequent than mobile ones. That is why in this paper, we focus on the computation of the minimum-delay tour of a mobile robot that has to deploy static wireless sensor nodes at positions that have been previously computed to meet the application requirements in terms of coverage and connectivity.

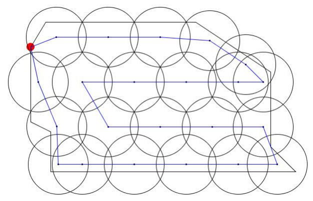
Fig. 1. A WSN deployment assisted by a robot.

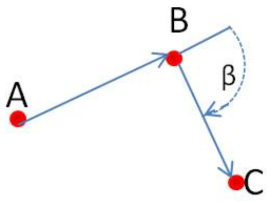

Fig. 2. Rotation angle between cities A, B and C.

Figure 1 illustrates an example of deployment in an area with irregular shape where a circle denotes the sensing range of the sensor node located in its center. The red point represents the initial position of the robot. This robot must visit each sensor node position to place each sensor node. The goal is to minimize the time needed by the robot to perform its mission. During the moves of the robot, the main energy consumption is due to robot moves. Minimizing the duration of these moves contributes to significantly reduce the energy consumed by the robot. Notice however, that the time and energy needed by the robot depend not only on the distance traveled by the robot but also on the rotations of the robot to change its directions to visit successively all sensor node positions (see Figure 2). Such rotations cannot be neglected because they consume additional time and energy. This problem is called the Robot Deploying Sensors problem, in short RDS. It can be seen as a generalized version of the traveling salesman problem, denoted TSP in the following.

This paper is organized as follows. In Section II, we give a brief state of the art related to wireless sensor network deployment assisted by a robot and TSP. For TSP, we distinguish between algorithms providing either the exact solution or approximated solutions. In Section III we formulate our RDS problem as an Integer Linear Program. In Section IV we present different approximated solutions (e.g., 2-Opt, Genetic and Hybrid) that are evaluated and compared to the exact solution provided by the CPLEX solver [2] for various configurations in Section V. We discuss further issues in Section VI and conclude in Section VII.

II. Related work

Robots are now commercially available. However, the robots vacuum and the robots lawn mowers that we can buy have random trajectories and consequently need a large amount of time to visit a whole area or only some points of interest (e.g., a dirty place or a grassy area). In the literature the problem of exploring a whole zone by a mobile robot has received a lot of attention. Different solutions have been proposed such as [3] and [4] that take into account the obstacles that may exist in the area considered, [5] that adapts itself to the area it discovers. The robot may move in three directions: left, straight and right that are ordered by decreasing priority order. The robot state depends on the number of directions it may take at this time: at least two directions for the normal state, one direction for narrow-lane and zero direction for dead-end. The algorithm used depends on this state. If the robot is initially located at the left-top corner of the rectangular area, it leads to a spiral trajectory. The spiral trajectory is usually preferred to the serpentine trajectory because it is shorter in the absence of obstacles. However, these solutions address the full exploration of a zone that may be combined with a deployment of sensor nodes.

Our problem has many analogies with the well-known Traveling Salesman Problem, called TSP, whose goal is to find the smallest tour visiting all sensor nodes positions (representing the cities) only once and going back to its initial position. This problem has been proved NP-hard. For problems of limited size, the optimal solution can be found with various branch and bound algorithms, integer linear programming solvers like for instance, the CPLEX solver [2]. For problems of large size, heuristic and approximation algorithms that give faster results close to optimal are used. These algorithms are either constructive or proceed by iterative improvement. For instance, the 2-Opt [6] algorithm improves the current solution by replacing two edges with two new ones to reduce the tour length. Other examples are given by Tabu [7], [8] and Genetic [9], [10] algorithms. In this paper, we will compare the solutions provided by the 2-Opt algorithm, a Genetic algorithm and an Hybridation of these last two algorithms, with the optimal one for various configurations.

However, our problem differs from the classical traveling salesman problem because the objective is not to minimize the distance traveled but the duration of the tour, taking into account the angular speed of the robot. Consequently, the cost associated with a tour is equal to the time needed by the robot to perform its tour. It is significantly more complex to evaluate that the simple distance between two cities B and C visited successively as illustrated in Figure 2. It should take into account the angle made by the direction the robot had when arriving from A to B and the direction given by BC. Let us consider an example of deployment assisted by a mobile robot. Figure 3 shows the optimal tour of the robot obtained when only the distance is taken into account: this is the optimal tour of TSP. We observe many direction changes in this tour. Figure 4 depicts the optimal tour when both distance and angle are accounted for: this is the optimal tour for RDS. This tour is smoother than the TSP optimal tour. This example clearly points out the difference between the TSP optimal tour and the RDS optimal one. The optimal tour has a duration of 271.55 seconds and a distance of 2035 meters in the RDS problem, whereas it has a longer duration of 385.66 seconds but a shorter distance of 1924 meters in the TSP problem.

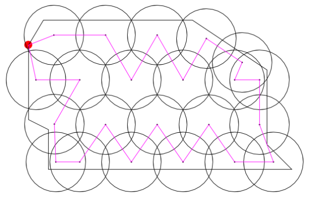
Fig. 3. Optimal tour with TSP.

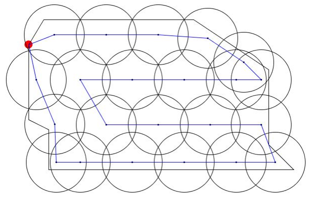
Fig. 4. Optimal tour with RDS.

Our contribution lies in the integer linear programming formalisation of the RDS problem, the proposal of algorithms providing exact or approximated solutions to the RDS problem and the identification of their respective application domains that can be deduced from our comparative evaluation results.

III. Formalization of the RDS problem

The Robot Deploying Sensor nodes problem can be formulated as an Integer Linear Program (ILP).

The robot tour is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices representing the node positions to be visited during the robot tour and $\mathcal{E}$ is the set of edges representing the path between node positions. The cardinal of $\mathcal{V}$ is n.

Let $d_{i,j}$ denote the distance between the node positions $i \in \mathcal{V}$ and $j \in \mathcal{V}$. Let $a_{i,j,k}$ denote the angle between the edges $(i,j) \in \mathcal{E}$ and $(j,k) \in \mathcal{E}$. Let ls and as be the robot linear speed and the robot angular speed, respectively.

We define $x_{i,j}$, where $i \in \mathcal{V}$ and $j \in \mathcal{V} \setminus \{i\}$, the utility of a path $p \in \mathcal{E}$, i.e. $x_{i,j} = 1$ if and only if p belongs to the robot tour and $x_{i,j} = 0$ otherwise. Furthermore, let $y_{i,j,k}$, where $i \in \mathcal{V}$, $j \in \mathcal{V} \setminus \{i\}$ and $k \in \mathcal{V} \setminus \{i,j\}$, be the robot rotation required at a node position. In other words $y_{i,j,k} = 1$ if and only if the rotation at node j position is effective and $y_{i,j,k} = 0$ otherwise. Finally we denote $z_{i,j}$, where $i \in \mathcal{V}$ and

$j \in \mathcal{V} \setminus \{i\}$, the robot stock of sensor nodes available on the edge (i, j).

The objective is to minimize the time used by the robot to visit all sensor node positions. This time takes into account the time due to both the distance and the rotation angle towards the next sensor node position:

$$min \left(\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V} \setminus \{i\}} d_{i,j}/ls * x_{i,j} + \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V} \setminus \{i\}} \sum_{k \in \mathcal{V} \setminus \{i,j\}} a_{i,j,k}/as * y_{i,j,k} \right)$$

This objective is subject to the following constraints:

$$\forall i \in \mathcal{V}, \sum_{j \in \mathcal{V} \setminus \{i\}} x_{i,j} = 1 \tag{1}$$

Contraint 1 allows only one departure for the robot from a node position.

$$\forall j \in \mathcal{V}, \sum_{i \in \mathcal{V} \setminus \{j\}} x_{i,j} = 1 \tag{2}$$

Contraint 2 authorizes only one arrival for the robot in a node position.

$$\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V} \setminus \{i\}} \sum_{k \in \mathcal{V} \setminus \{i,j\}} y_{i,j,k} = n - 1 \tag{3}$$

Contraint 3 means that the robot tour includes $n-1$ rotation costs to make a complete tour.

$$\begin{gathered} \forall i \in \mathcal{V}, \forall j \in \mathcal{V} \setminus \{i\}, \forall k \in \mathcal{V} \setminus \{i,j\}, \\ y_{i,j,k} \leq (x_{i,j} + x_{j,k})/2 \end{gathered} \tag{4}$$

Contraint 4 ensures that any rotation cost corresponds to a pair of consecutive edges followed by the robot.

$$\forall i \in \mathcal{V} \setminus \{1\}, \forall k \in \mathcal{V} \setminus \{i, 1\}, \quad y_{i,1,k} = 0 \tag{5}$$

Contraint 5 guarantees that the rotation cost of the robot in its start position is not accounted for. In fact, when the robot comes back to its start position, it does not need to rotate toward any next node position, since the tour is complete.

$$\forall i \in \mathcal{V}, \forall j \in \mathcal{V} \setminus \{i\}, \quad z_{i,j} \leq (n-1) * x_{i,j} \tag{6}$$

Contraint 6 denotes the maximum capacity of the robot in terms of sensor nodes.

$$\begin{gathered} \forall i \in \mathcal{V}, \\ \sum_{h \in \mathcal{V} \setminus \{i\}} z_{h,i} + \left\{ \begin{array}{ll} n & \text{if} \quad i = 1 \\ 0 & \text{otherwise} \end{array} \right\} = \sum_{j \in \mathcal{V} \setminus \{i\}} z_{i,j} + 1 \end{gathered} \tag{7}$$

Contraint 7 expresses that the robot carries a certain number of sensor nodes. This number decreases with the number of node positions already visited by the robot.

The ILP formulation of the RDS problem differs from this of the TSP problem on the following points:

- The second term in the objective function has been added to account for the time lost in rotations.
- Constraints 3, 4, 5 have been introduced to deal with the robot rotation constraints.

This model of RDS adopts the following assumptions:

A1: The robot knows its location and is able to move autonomously to any location specified in the area considered.

A2: The set of sensor nodes positions are given as well as the initial location of the robot.

A3: The connectivity graph of sensor nodes positions is assumed to be complete. In other words, it is always possible for the robot to go from any sensor node position to any other sensor node position. For each pair of sensor node positions, the distance is given. For each triple of sensor node positions, the rotation angle is given.

The next three assumptions are adopted for the sake of simplicity. They will be relaxed in Section VI.

A4: There is no obstacle in the paths between any two sensor node positions.

A5: The robot has enough capacity to carry all sensor nodes it has to deploy.

A6: The robot has enough energy to visit all sensor nodes locations in a single tour.

IV. Proposed algorithms

We now describe the algorithms solving the RDS problem that we will compare in the next section. We consider an algorithm providing the exact solution and three algorithms providing approximated solutions.

A. The exact solution

The exact solution of the RDS problem is provided by the solver CPLEX [2] using the problem formulation given in Section III. This exact solution will be used as a reference to evaluate the closeness to the optimal of approximated solutions. Various approximated solutions are used. The first one, called 2-Opt, is based on iterative improvement, the second one uses genetics and the third one is an hybridation between them.

B. 2-Opt algorithm

We adapt the well-known 2-Opt algorithm [6] to the RDS problem. 2-Opt starts with an initial solution and tries to iteratively improve it by replacing two edges with two new ones that reduce the tour duration. This algorithm provides a local optimum based on the initial solution.

C. Genetic algorithm

A genetic algorithm is inspired from the biological evolution process. To define a genetic algorithm, we have to define the selection of the initial population, the operators we use for the selection of parents, the crossover and the mutation and finally the constitution of the population used in the next iteration as well as the fitness function. In the traveling salesman problem, an individual is defined by an ordered sequence of the cities visited by the robot. The initial population is given by K individuals, K is a non-null natural integer, each individual being a random permutation of the C cities to visit. The first city is the initial location of the robot. Hence, it is given as an input.

Let $\mathcal{P}_i$ denote the population at the beginning of any iteration $i > 0$. The Genetic algorithm randomly selects $\lfloor K/2 \rfloor$ pairs of parents among the current population $\mathcal{P}_i$, apply the crossover operator on each pair to generate two children. Each gene (i.e. a city visited) of a child is subject to a mutation with a gene mutation probability of P_{mut}. A new population $\mathcal{P}_{new}$ is then generated. All individuals of $\mathcal{P}_i \cup \mathcal{P}_{new}$ are evaluated by the fitness function. The fitness of an individual is equal to one over the time needed by the robot to perform this tour. The $\lfloor K/2 \rfloor$ best individuals (i.e., maximizing the fitness function) among the $\mathcal{P}_i \cup \mathcal{P}_{new}$ are selected, they are completed by $K - \lfloor K/2 \rfloor$ individuals randomly selected among the unselected ones to constitute $\mathcal{P}_{i+1}$ the population considered in the next iteration. This principle enables the algorithm to always keep the $\lfloor K/2 \rfloor$ best individuals it has found during the $Imax$ iterations performed by the algorithm.

In [11], a genetic algorithm is built to solve TSP. The mutation operator exchanges two genes, selected at two random positions, of an individual. The three crossover operators considered, PMX (for Partially Matched Crossover), CX (for Cyclic Crossover) and OX (for Ordered Crossover), ensure that the crossover of any two individuals is still an individual (i.e. a permutation of the C cities to visit). Furthermore, it is shown that PMX outperforms the two other crossover operators CX and OX. Hence, we select PMX as our crossover operator, using two cross points randomly selected.

D. *Hybrid algorithm*

The Hybrid algorithm combines the 2-Opt algorithm with a genetic one. More precisely, instead of starting with an initial random population, the Hybrid algorithm applies the 2-Opt algorithm to optimize each individual of the initial population. In addition, at each iteration, the children obtained with the crossover operator are mutated with the gene mutation probability and then optimized by applying again the 2-Opt algorithm.

V. Comparative evaluation

We evaluate the different algorithms presented in Section IV on different configurations ranging from 10 sensor nodes to 154 sensor nodes. These configurations may meet various application requirements. For instance, small configurations with less than 30 nodes are representative of temporary industrial worksites, where coverage of some interest points and connectivity with a sink must be achieved. Medium to large configurations, from 50 to 150 nodes, can represent industrial warehouses where full coverage and connectivity with a sink must be met. Small and medium configurations with less than 70 nodes can also be encountered to improve data gathering about an industrial process to detect leakages for example.

For this performance evaluation, we take the following parameters values: $ls = 10$ meters per second, $as = 10$ degrees per second for the robot linear and angular speeds, respectively; $P_{mut} = 0.15$, $Imax = 1000$ iterations for Genetic and $Imax = 100$ for Hybrid, $K \geq 2 * C$ individuals, where C is the number of sensor nodes to deploy. Sensor nodes are deployed in the area depicted in Figure 1. The dimensions of the circumscribing rectangle are $530m$ x $300m$, the sensing range varies from $140m$ to $20m$ to match a number of sensor nodes from 10 to 154.

First, we compute the solutions to the TSP and RDS problems for a number of sensor node positions ranging from 10 to 154, using 2-Opt. Figure 5 clearly shows that if for very small configurations (i.e., configurations with at most 10 sensor nodes), the tours found by TSP and RDS may be the same. This is no longer the case when the number of sensor nodes increases, the difference between the TSP solution and the RDS one increases considerably.

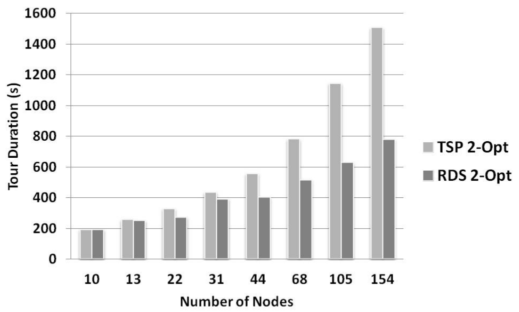

Fig. 5. Solutions found by 2-Opt for the RDS and TSP problems.

Secondly, we compare the accuracy (i.e., closeness to the optimal) of the solutions provided by each algorithm tested in small configurations (≥ 31 sensor nodes). Figure 6 depicts the solutions given by 2-Opt, Genetic and Hybrid versus the optimal one in small configurations. When the number of nodes is higher than 13 sensor nodes, Genetic fails to find the optimal tour in 1000 iterations. This is due to the fact that it generates many tours that are not interesting. On the contrary, 2-Opt provides a solution that is close to the optimal for the configurations tested. Hybrid provides the best results as an approximation algorithm.

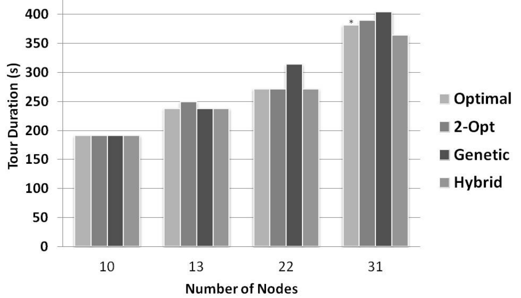

Fig. 6. Final solutions of Optimal, 2-Opt, Genetic and Hybrid for small configurations.

For large configurations, Hybrid improves the solution found by 2-Opt as shown in Figure 7. For instance, for 103 and 154 nodes, Hybrid decreases the tour duration from 629.1s to 628.15s and from 752.95s to 749.41s, respectively. This can be explained by the fact that 2-Opt can be blocked in a local optimal, whereas Hybrid uses mutations and crossovers to explore other tours. However, 2-Opt is better to exploit a given solution. We observe also that Genetic is very sensitive to the choice of the initial population: if it is far from the optimal, the final solution remains far from the optimal. In the configurations tested, 2-Opt improves the initial solution by at least 50%.

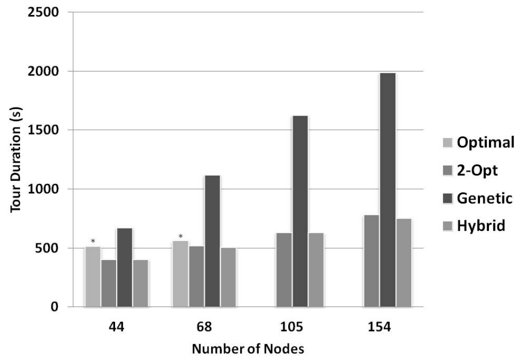

Fig. 7. Final solutions of Optimal, 2-Opt, Genetic and Hybrid for large configurations.

Another interesting result is given by the time needed by each algorithm to compute its final solution. CPLEX is run on a Quad-core Intel Xeon W3565 3.2GHz platform, whereas the other algorithms are run on a 8-core Intel i7-2760QM 2.4GHz platform. Tables I and II depict the computation time for each algorithm tested.

TABLE I
COMPUTATION TIME FOR OPTIMAL, 2-OPT, GENETIC AND HYBRID FOR SMALL CONFIGURATIONS.

Number of nodes	10	13	22	31
Optimal (s)	11.18	217.35	10866.24	87387.77*
2-Opt (s)	0.005	0.003	0.014	0.029
Genetic (s)	0.396	0.505	0.845	1.161
Hybrid (s)	2.808	6.788	44.319	276.9

TABLE II
COMPUTATION TIME FOR OPTIMAL, 2-OPT, GENETIC AND HYBRID FOR LARGE CONFIGURATIONS.

Number of nodes	44	68	105	154
Optimal (s)	174828.61*	103910.62*	-	-
2-Opt (s)	0.114	0.339	2.051	3.689
Genetic (s)	2.857	7.065	18.969	39.8
Hybrid (s)	870.45	3743.66	27111.4	41267.3

As expected, Optimal needs the largest computation time in all configurations tested except for 10 nodes. For example, it takes about 3 hours to solve the RDS problem with 22 sensor nodes. Hybrid needs the second largest time. This is due to the calls to the 2-Opt algorithm applied first on each individual of the initial population and then on each child generated by the crossover operator. For example, it takes 636 seconds (about 6,5mn) to generate the final solution of the RDS problem with 22 sensor nodes. The fastest algorithms are 2-Opt and Genetic to a lesser extent. In all configurations tested, 2-Opt is at least 10 times faster than Genetic, this ratio reaches 100 times in small configurations. Since Genetic may provide a solution far from the optimal one, 2-Opt is preferred. For larger configurations, (more than 31 nodes), we did not succeed to obtain the optimal solution with CPLEX after 24 hours computation. Since in configurations with more than 30 nodes, CPLEX needs a time higher than 24 hours, we take as final solution, the best solution found by CPLEX in 24 hours. This solution may be a non-optimal one, as depicted in Figure 7 by a star symbol. In all these configurations, Hybrid provides the best results. We recommend the Hybrid algorithm for large configurations because it provides the best trade-off between the optimal closeness and an acceptable computation time.

VI. DISCUSSION

In this section, we show how to relax the assumptions A4 (no obstacle), A5 (enough capacity) and A6 (enough energy).

A. *Obstacles*

Up to now we have considered a sensor deployment in an area without obstacles. However in the real life, obstacles may be present. In this section, we show how to relax the assumption A4 and extend the solutions given previously to cope with obstacles. In a previous paper [12], we proposed a deployment algorithm that copes with transparent and opaque obstacles and ensures full coverage; This algorithm computes the sensor node positions that are given as input to the RDS algorithm. It is hard to compute an optimized robot trajectory taking into account obstacles.

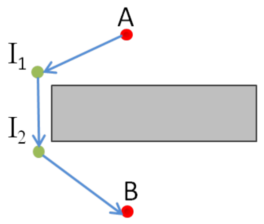

Fig. 8. Intermediate points between sensor node positions A and B

If there exists an obstacle between the sensor node positions A and B, the direct path from A to B is made impossible. Thus the connectivity graph of the cities is no longer a complete graph. We propose to define intermediate positions I_i that allow the robot to reach B avoiding the obstacles. We replace the impossible direct path between A and B by a possible one that can be seen as a juxtaposition of segments I_iI_{i+1} with $I_1 = A$ and $I_n = B$. The cost of this path is computed as the sum of the cost of any segment composing the path. Figure 8 shows the two intermediate points that are introduced in the path taken by the robot to reach B from A. The path AI_1I_2B replaces the direct path AB.

B. *Capacity constraint*

Up to now we have assumed that the robot has the capacity to carry all sensor nodes at the same time. If this is not the case, assumption A5 is no longer true. In such a case, the robot has to perform subtours starting at its initial position. How can we solve this new problem?

In order to handle the new carrying capacity constraint, we enhance the integer linear program of Section III as follows:

Let cap be the robot carrying capacity in terms of number of sensor nodes. The objective is the same as before and only three constraints are modified. Constraints 1 and 2 specifying that there is only one arrival and departure in each city are relaxed to enable multiple arrivals and departures in the initial robot position. In fact, the robot must come back to its initial position to refill its sensor node stock.

$$\forall i \in \mathcal{V} \setminus \{1\}, \sum_{j \in \mathcal{V} \setminus \{i\}} x_{i,j} = 1 \quad (8)$$

$$\forall j \in \mathcal{V} \setminus \{1\}, \sum_{i \in \mathcal{V} \setminus \{j\}} x_{i,j} = 1 \quad (9)$$

Furthermore, the capacity constraint 6 must be updated according to the capacity parameter cap.

$$\forall i \in \mathcal{V}, \forall j \in \mathcal{V} \setminus \{i\}, \quad z_{i,j} \leq cap * x_{i,j} \quad (10)$$

Figure 9 depicts an optimal RDS tour when the robot has to deploy 13 sensor nodes and its capacity is limited to 6 sensor nodes. This optimal tour comprises 3 subtours depicted in blue, each of them starting at the initial position of the robot.

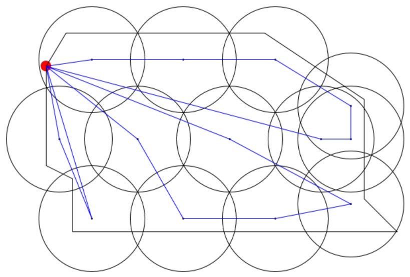
Fig. 9. Optimal RDS tour with a limited capacity of 6.

Table III gives the number of subtours done by the mobile robot when its capacity is 6, 5, or 4 sensor nodes respectively. When the robot capacity decreases, the number of subtour increases as expected, leading to a longer tour duration.

TABLE III
OPTIMAL TOUR WITH DIFFERENT ROBOT CAPACITIES.

Robot capacity	13	8	6	4
Subtours (number)	1	2	3	4
Length (m)	1712	2169	2562	3048
Time cost (s)	237.64	262.98	319.11	385.70

C. Energy constraint

What happens if the maximum energy level of the robot does not allow it to visit all sensor nodes locations in a single tour. Assumption A6 is no longer true. Here again, the robot proceeds by subtours, refilling its battery at its initial position that is also the starting point of each subtour. An idea could be to group sensor nodes into clusters, such that the robot deploys all the sensor nodes of the same cluster in a single subtour. Notice that the number of cluster members may differ from one cluster to another, since the robot consumes more energy to visit a far cluster than a close one.

VII. CONCLUSION

In this paper we have shown how to optimize the delay needed by a mobile robot to deploy sensor nodes, taking into account the rotations performed by the robot to change its direction. The delay-optimized tour of a mobile robot may result in a reduction of at least 50% in the time needed to deploy wireless sensor nodes. This smaller deployment time may be crucial not only in emergency applications but also in industrial process control because the latency before the first data gathering is reduced. This benefit is obtained by using the optimal solution that can be provided by an integer linear program solver like CPLEX for instance in small and medium configurations. For larger configurations, the time needed to obtain the optimal solution can become prohibitive. That is why we use the Hybrid algorithm that successfully combines the exploration of the Genetic algorithm with the exploitation of 2-Opt algorithm.

REFERENCES

[1] I. Khoufi, P. Minet, A. Laouiti, S. Mahfoudh, *Survey of deployment algorithms in wireless sensor networks: coverage and connectivity issues and challenges*, International Journal of Autonomous and Adaptive Communications Systems (IJAACS), to appear in 2014.

[2] IBM, *IBM ILOG CPLEX Optimization Studio - CPLEX Users Manual*, Version 12.5, 2011.

[3] M. Shyam, K. Anurag, *Obstacle constrained total area coverage in wireless sensor networks*, arxiv-web3.library.cornell.edu/pdf/1001.4753v1, January 2010.

[4] C.Y. Chang, C.T. Chang, Y.C. Chen, *Obstacle-resistant deployment algorithms for wireless sensor networks*, IEEE Trans. on Vehicular Technology, Vol.58, N6, July 2009.

[5] C.Y. Chang, C.T. Chang, C.Y. Hsieh, C.C. Chen, Y.C. Chen, *A dead-end free deployment algorithm for wireless sensor networks with obstacles*, IWCMC 2010, Caen, France, June 2010.

[6] G. A. Croes, *A method for solving traveling salesman problems*, Operations Research, Vol. 6, Issue 6, 1958, pp. 791-812.

[7] F. Semchedine, L. Bouallouche-Medjkoune, L. Bennacer, N. Aber and D. Assani, *Routing protocol based on Tabu search for wireless sensor networks*, Wireless Personal Communications , Volume 67, Issue 2 , pp 105-112, 2012.

[8] A. El Rhazi, S. Pierre, *A Tabu search algorithm for cluster building in wireless sensor networks*, Mobile Computing, IEEE Transactions on Mobile Computing, Vol.8 , Issue4, April 2009.

[9] A. Norouzi, A. H. Zaim, *Genetic algorithm application in optimization of wireless sensor networks*, The Scientific World Journal, 2014.

[10] N. Zhu, I. OConnor, *iMASKO: A Genetic algorithm based optimization framework for wireless sensor networks*, Journal of Sensor and Actuator Networks, 2013.

[11] N. Kumar, Karambir, R. Kumar, *A comparative analysis of PMX, CX and OX crossover operators for solving traveling salesman problem*, International journal of Latest Research in science and technology, V1, Issue2, pp98-101, July-August 2012.

[12] I. Khoufi, P. Minet, A. Laouiti, E. Livolant, *A Simple Method for the Deployment of Wireless Sensors to Ensure Full Coverage of an Irregular Area with Obstacles*, ACM MSWiM 2014, Montreal, Canada, September 2014.