An Evolutionary Algorithm to the Density Control, Coverage and Routing Multi-Period Problem in Wireless Sensor Networks

Iuri Bueno Drumond de Andrade, Tiago de Oliveira Januario, Gisele L. Pappa and Geraldo Robson Mateus

***Abstract*—Wireless Sensor Networks (WSNs) are composed of autonomous and resource-constrained (power, sensing, radios, and processors) nodes. These networks are conceived to have a large number of nodes working on monitoring phenomena. A major challenge for theses networks is to provide solutions that maximize quality of service (QoS) requirements, such as coverage and data routing, and minimize the energy consumption. This paper presents an evolutionary algorithm (EA) to solve the Density Control, Coverage and Routing Multi-Period Problem (DCCRMP) in WSN. The results are compared to the optimal solutions obtained by an Integer Linear Program model and a GRASP heuristic from the literature. The EA obtains significant improvements in both quality of solutions and computational time.**

I. Introduction

The advances in the development of more economical and sophisticated techniques for the manufacture of electronic components, such as embedded processors, radios, and microchips, made Wireless Sensor Networks (WSNs) a reality. WSNs are a special type of *ad hoc* network composed of hundreds or even thousands of small autonomous computational devices called sensor nodes. Each sensor node works by detecting events (e.g., monitoring phenomena such as temperature or humidity), performing quick local data processing, and transmitting data through a wireless communication protocol to a special node, named sink. The sink node then sends collected data to a monitoring station for further analysis, using a more powerful communication infrastructure (e.g., satellite links). However, the activity of each sensor node is constrained by resource availability, such as processor, memory, radio, devices for sensing, and battery. In order to overcome these problems, sensor nodes in a WSN work in a cooperative way.

WSNs have become extremely popular due to their wide applicability in environmental, medical, industrial, or military scenarios, as they can be used for monitoring phenomena (such as tracking people, for example) in areas of hard-access or even inhospitable [9]. Nevertheless, even though WSNs are classified as *ad hoc* networks, they have specific features that differ among them from traditional *ad-hoc* networks, such as: (i) strong constraints of energy, processing, and communication, (ii) high density of nodes, and (iii) high dependence of the application. The aforementioned characteristics make it difficult for existing solutions for coverage and routing in traditional *ad hoc* networks to be applied to WSNs. Hence, new algorithms need to be developed to ensure the network routing and coverage, especially regarding constraints on energy, since recharging the sensors batteries is an infeasible task in hard-access areas (e.g., dense forests).

The problem of energy saving can be addressed by taking advantage of the high density of nodes in the WSNs using schemes such as nodes scheduling. Generally, in node scheduling, some nodes are kept active while others are scheduled to become inactive or remain in a sleep state that consumes less energy than the active state. As a consequence, this type of algorithm, named density control, also simplifies data communication, as a smaller number of nodes it also reduces collisions and radio interference [11].

Following the idea of density control (or scheduling schemes), Cardei and Wu [3] and Gao et al. [4] found out that the usage of coverage scheduling schemes in static networks allows significant power saving, especially when the redundant coverage area of the nodes is used as a deactivation criterion. In this same line, Siqueira et al. [10] developed an integrated scheduling scheme to solve the coverage and routing problems simultaneously in static networks. The approach ensured a correct and efficient operation for WSNs, assuring area coverage and data dissemination.

Both works for network routing cited above consider the network topology is static. However, network topology can change over time. Hence, a more efficient resource planning can be done if the estimated lifetime period of the network is divided into fixed time intervals, and the planning of time period $t + 1$ takes into account the final state of the network in time period t. Methods following this approach are called multi-period.

This paper addresses the *Density Control*, *Coverage* and *Routing Multi-Period* problem (DCCRMP) in WSNs. Its goal is to provide a multi-period density control algorithm that maximizes the Quality of Service (QoS) requirements in WSNs - especially coverage and data routing - while minimizing the network energy consumption. This problem was originally proposed by Nakamura et al. [8] and modeled using Integer Linear Programming (ILP).

The DCCRMP problem was also addressed by Andrade et al. [2] using a GRASP heuristic. They use a local search strategy which aims, period by period, to find a solution to the problem by analyzing features such as the redundant coverage area of the sensors. The results show that the proposed approach finds solutions very close to the optimal and with reduced computational time if compared to the ILP model originally proposed in [8].

Iuri Bueno Drumond de Andrade, Tiago de Oliveira Januario, Gisele L. Pappa and Geraldo Robson Mateus are with the Departamento de Ciência da Computação, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Minas Gerais, Brasil (email: {bueno, januario, glpappa, mateus}@dcc.ufmg.br).

One of the main drawbacks of the previously proposed approaches to solve the DCCRMP is that they search for the best network configuration for each period independently. Hence, the energy consumption of different configurations over time is ignored. In this direction, this paper presents an evolutionary algorithm (EA) [5] to solve the DCCRMP. The main advantage of the EA over the other methods is its global search, which was also combined with a powerful local search operator that explores specific regions of the search space, improving the EA convergence and escaping local minima. Results show that the method proposed is able to successfully meet the QoS requirements considered (i.e., coverage and energy saving), and provides solutions close or equal to the optimum with less computational time compared than the ILP model and the algorithm proposed in [2].

This paper is organized as follows: Section II defines the DCCRMP. Section III briefly explains the GRASP metaheuristic proposed in [2], as well as a modified version directly comparable with the proposed evolutionary algorithm. Section IV explains the proposed Evolutionary Algorithm. The results are presented in Section V, and conclusions are drawn in Section VI.

II. Definitions

We make some assumptions to formally define the DCCRMP. First, in order to quantify the monitoring area and the nodes coverage, we discretized the monitoring area into a set of points that demand sensing, called *demand points*. These points are located in the center of the squares in which the area is divided. The number of demand points depends on the application requirements, and the more demand points the closer to a continuous area. Hence, the coverage area of a single node corresponds to a circle of radius R_s [6]. The neighborhood of each node is composed by nodes within its communication range, called communication radius, R_c. We also assume that areas do not have obstacles, which means that a demand point is covered simply if it is within the coverage radius of an active node. Thus, the coverage problem is restricted to check if every demand point is covered by, at least, one sensor node.

The problem addressed in this work can be formally defined as follows.

Definition 1: Let us consider a monitoring area A, a set of demand points D, a set of sensor nodes S, a set of sink nodes M, and a number of time periods T (expected network lifetime). The Density Control, Coverage and Routing Multi-Period Problem (DCCRMP) consists in assuring, if possible, that every demand point $d \in D$ in A is covered by at least one sensor node $s \in S$, and there is a route between every active sensor node $s \in S$ and a sink node $m \in M$ at each period $t \in [1, T]$, respecting the energy limits of the sensor nodes.

III. GRASP Heuristic

This section describes the work presented in [2], from now on referred as GRASP-H, where a GRASP heuristic was used to solve the DCCRMP. In contrast with the method proposed in this paper, GRASP-H optimizes the network QoS requirements period by period, instead of considering them simultaneously. This main disadvantage of this approach is that solutions can be trapped in local minima.

Algorithm 1 shows GRASP-H. For a maximum number $GMAX$ of iterations, where $GMAX$ is the number of iterations since the best solution k^* and its respective evaluation function f^* were updated, the algorithm first builds a solution using a greedy procedure (constructive phase, Line 4), and then improves it following a local search method (Line 5). During the constructive phase, α controls the randomness of the algorithm, and β is a specific parameter of GRASP-H that controls the maximum redundant coverage area of a sensor. During the local search, $BMAX$ is a stopping criterion, and represents the maximum number of iterations since the best solution found by the local search was updated. GRASP-H is executed as many times as the number of periods considered in the instance for the problem being analyzed. The next subsections detail the evaluation function f and the local search procedure.

1: $f^* \leftarrow \infty$
2: $k^* \leftarrow \varnothing$
3: **for** iteration = 1,...,GMAX **do**
4: $k \leftarrow$ Constructive_Phase(α, β)
5: $k^{'} \leftarrow$ Local_Search(k, BMAX)
6: **if** $f(k^{'}) < f^*$ **then**
7: $f^* \leftarrow f(k^{'})$
8: $k^* \leftarrow k^{'}$
9: **end if**
10: **end for**
11: **return** k^*

Fig. 1. GRASP Heuristic

A. Local Search

The Local Search process is based on three operators: node insertion, swap and removal. These three operators are applied, in this order, to the current best solution. This order is necessary because, when inserting (activating) a new node to the current solution, it will appear in the neighborhood of the active nodes that are enclosed into the circular region with center on the new node and radius R_p. This region is called *perturbed region* and the radius R_p, or *radius of perturbation*, is equivalent to the communication radius. This perturbation in the neighborhood of the active nodes can change the configuration of the network in the perturbed region, since it allows a node overloaded with many children to donate a child to another node. After these swaps, some nodes can lose their routing function, or became completely redundant [4], i.e., all their coverage areas are also covered by other active nodes. Hence, they should be removed from the solution.

An example of a node insertion can be seen in Figure 2a. The insertion of node V causes a perturbation in the nodes around it {sink, A, B, C, D, E }. This perturbation allows

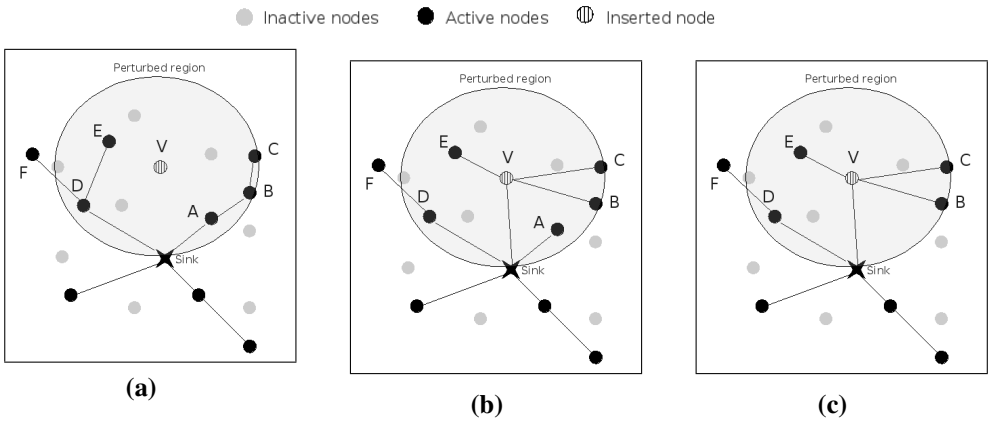

Fig. 2. Example of a network restructuring created by the insertion of a new node

nodes B, C and E to be adopted by the inserted node (Figure 2b), leaving node A with no children and node D with only one child. Thus, node A loses its contribution to the network routing and, as A has a completely redundant area, it can be removed from the solution (Figure 2c).

1) Insertion: As mentioned before, the insertion operation is used to activate nodes that may improve the current solution. This improvement occurs if the activation of a node perturbs the ones within the circular area around it, leading to network reconfiguration and minimization of the consumed energy.

The insertion operator randomly chooses a node from a set of eligible (deactivated) nodes and activates it. The chosen node then seeks, in its neighborhood, a parent that can adopt it, i.e., an active node that can route packages through it. The parent chosen must be the one where the connection between itself and the inserted node causes the least increase in the network energy consumption. If the new node cannot find a parent with these properties, it is discarded, and another node is picked up instead. Note that nodes that have already been analyzed, but did not improved the solution, will not be tested again until the solution changes. Figure 2 shows node V being inserted and adopted by the sink node.

2) Swapping: The swapping operation is used to find the best connections between the active nodes in the perturbed region that minimizes the network energy consumption. Thus, considering C the set of active nodes belonging to the perturbed region, for each $c \in C$, all possible connections between c and $c^{'} \in C, c \neq c^{'}$ are tested, looking for the ones that better restructure the routing tree, leading to energy saving.

To ensure routing, swap operations exclude possibilities that may cause cycles or connectivity problems (e.g., nodes that cannot afford energy discount after restructuring of routing tree) in the network. Hence, this operation maintains at least one route, without cycles, from all active nodes to the sink. In Figure 2b the nodes A, B, C, D, E belong to the neighborhood of the inserted node V. After several swap tests, nodes B, C and, E choose V as their new parent.

3) Removal: The removal operation is used to turn off nodes that do not contribute to improve coverage or are not needed to ensure the network connectivity. In this work, these nodes are called *unproductive nodes*. Node removal is necessary because, after the constructive phase, we might end up with nodes that do not contribute for the coverage, and are only used to guarantee data routing. However, after the swapping operation is applied, these nodes might come unproductive, as their children may be adopted by other nodes. Hence, their removal will contribute to energy saving.

The removal operator acts only on the perturbed region, searching for nodes that have completely redundant coverage area. However, nodes that were previously inserted to the solution by the Insertion operator cannot be discarded here. Figure 2b shows nodes B and C being adopted by the inserted node V, making node A unnecessary to the routing. As this node does not contribute to cover the area, it is removed (2c). Notice that, even though node D has a completely redundant coverage area, it cannot be deactivated because it is important for data routing from F.

B. Evaluation Function

The evaluation function is used to assess the quality of the solutions created by GRASP-H and the Local Search. This function minimizes the number of active nodes and the coverage gaps, i.e., maximizes coverage. Besides considering the network QoS requirements, such as coverage and routing, the function also minimizes the energy consumption per period, while respecting the battery capacity of the sensors. Formally the function is defined as:

$$min \sum_{i \epsilon S} \frac{E_i^t}{A_i^t} + (1 - \sum_{i \epsilon S} \frac{A_i^t}{|D|}) + \sum_{i \epsilon S} y_i^t \qquad (1)$$

where

- S is the set of sensor nodes,
- D is the set of demand points,
- A_i^t is the coverage area of the sensor node i in period t,
- E_i^t is the energy consumed by the sensor i in period t,
- y_i^t is a decision variable that has value 1 if the sensor i cannot guarantee connectivity in period t, and 0 otherwise, and
- t is the current period.

The first sum in Eq. 1 prioritizes nodes that have low energy consumption and large coverage area, and the second sum maximizes the coverage area. The third sum is responsible for removing solutions that present at least one node that does not have enough energy to guarantee the network routing. Because these criteria have different magnitudes, the function was normalized.

C. Revised GRASP Heuristic

This paper proposes a new evolutionary algorithm that searches for the best network configurations at all periods simultaneously. GRASP-H, in contrast, searches for the best configurations one period at a time. Hence, in order to make a fair comparison of a GRASP-H and the proposed algorithm, a revised version of GRASP-H, named GRASP-R, was proposed, and is described in Algorithm 3.

1: $f_g^* \leftarrow \infty$
2: $v^* \leftarrow \varnothing$
3: **for** iteration = 1,...,MAX **do**
4: **for** i = 1,...,NPERIOD **do**
5: $v[i] \leftarrow$ Constructive_Phase(α, β)
6: $v^{'}[i] \leftarrow$ Local_Search(k[i])
7: **end for**
8: **if** $f_g(v^{'}) < f_g^*$ **then**
9: $f_g^* \leftarrow f_g(v^{'})$
10: $v^* \leftarrow v^{'}$
11: **end if**
12: **end for**
13: **return** v^*

Fig. 3. Revised GRASP Heuristic

This new version stores the solutions found for each period in $v^{'}$ and evaluated by $f_(g)$ (described in Section IV-D). Furthermore, the new version has a new stopping criterion for both the GRASP-R and local search procedures. GRASP-R now runs for a maximum number of iterations, MAX, and the local search goes on until all inactive sensors in the network are evaluated. This last modification reduces the number of parameters of the algorithm.

The GRASP heuristic evaluates the objective function for each period of the solution. This strategy does not consider the fact that the changes performed by local search in the current period affect subsequent periods and thus the entire solution. The heuristic GRASP-R evaluates the objective function after applying the local search in all periods. Besides to permit a global analysis of all periods of the solution, the result is in the worst case, equal to the result of the GRASP.

Applying GRASP-R can be considered the same as applying GRASP-H as many times as the number of periods being analyzed, represented by NPERIOD in Algorithm 3).

IV. Evolutionary Algorithm

This section presents the proposed evolutionary algorithm for the DCCRMP. As explained before, one of the motivations to explore EAs in this context is to take advantage of its global search to find simultaneously the best network configurations for different time periods. Even though GRASP-R also evaluates the solutions for all periods simultaneously, it done this inefficiently, once it has no memory structure or mechanisms capable to reduce the search space and escape of minima locals.

Alg. 4 presents the pseudo-code of the EA. Initially, a population of $popSize$ individuals is created, where each individual represents the network configuration for each of the t periods considered in the problem being tackled, as detailed in Section IV-A. Then, until a maximum number of generations $NGenerations$ is reached, the individuals are evaluated using a small variation of the function defined in Eq. 2. Following this step, a tournament selection of size $tSize$ is performed. A tournament selection randomly chooses $tSize$ individuals from the population, and declares the one with the best value of fitness the winner of the tournament. After that, the selected individuals undergo union and mutation operations, or a local search procedure based on the original GRASP-H, according to used defined probabilities Pu, Pm and Pl. Note that the local search operator used here is the same one described in Section III-A. Previous experiments showed that the combination of both a global and a local strategy lead to better results than using them independently.

1: Pop = Generate initial population(*popSize*)
2: **for** i = 1 until NGenerations **do**
3: Selection by tournament (Pop, *tSize*)
4: Union (Pop, *Pu*)
5: Mutation (Pop, *Pm*)
6: Local Search (Pop, *Pl*)
7: **end for**
8: **return** the best solution of Pop

Fig. 4. Evolutionary Algorithm

A. Individual Representation and Population Generation

Each individual in the proposed EA is represented by a $n \times t$ matrix, where n is the number of sensor nodes and t is the number of periods being considered. The matrix is initially empty, and a node i is considered active at time period j if it has a parent to transmit collected data to. In this case, the cell S_{ji} receives the number of the parent node of i in time period j in the solution S. Hence, each line in the matrix is an abstraction of a path between the sink and the nodes of WSN.

Figure IV-A shows the representation of a solution for a network with five sensor nodes and four periods of time. For instance, in the second period, node two is connected to sink (represented by node 0) and is the parent of node one, which in turn is the parent of node three. Notice that all networks form trees, since every node must route the information to a single adjacent sensor node, and cycles are not allowed.

When generating the individuals, we always guarantee that they provide network connectivity. Hence, the initial popula-

	1	2	3	4	5
1	0		1		
2	2	0	1		
3			5	5	0
4	4		1	0	

Fig. 5. Representation of an Individual

Ind_1

	1	2	3	4	5
1	0	0	1		
2	2	0	1		
3			5	5	0
4	4		1	0	

Ind_2

	1	2	3	4	5
1	0		1	1	4
2	4			0	0
3			5	5	0
4			4	5	0

Ind_3

	1	2	3	4	5
1	0	0	1	1	4
2	4		4	5	0
3	3		5	5	0
4	2	0	1	0	0

Fig. 6. An example of Union operator

tion is built using a simple procedure for generating paths, as follows.

1) Choose, randomly, a node n and insert it to path c.
2) If the distance between n and the sink is not larger than R_c, go to step 5.
3) Choose, randomly, a node w that does not belong to c, and whose distance from n is not larger than R_c.
4) Insert w in c, making $n \leftarrow w$, and go to step 2.
5) Assign path c to a empty period. Make $c \leftarrow \varnothing$.
6) If there are empty periods, return to step 1.

After a path is generated for a period, the local search operator is applied to it, guaranteeing that each individual is close to a local minimum. Next, we verify if each active sensor supports the energy spent to route the information of their descendants. In a negative case, the node and all its sub-trees are deactivated. In this way, all solutions generated will guarantee connectivity.

We chose to apply the local search operator here because preliminary experiments show it improves considerably the effectiveness of the union and mutation operators.

B. Union Operator

Given two individuals, the union operator proposed in this paper randomly chooses a period to be unified until all the periods in the third individuals are filled. Let p be the period randomly chosen from Ind_1 and Ind_2. We chose the individual with the greater number of sensor nodes to guide the construction of the new individual. Suppose a in Ind_1 is the winner. In this case, we copy the nodes of period a in Ind_1 to the first period in Ind_3, and then insert the nodes of a of Ind_2 in Ind_3 also in the first period of Ind_3.

Figure IV-B illustrates this process. Here, the first period p_1 in Ind_3 is the combination of p_1 in Ind_1 and p_1 in Ind_2. In this case, p_1 in Ind_2 guided the search. Hence, it was copied to Ind_3, and then united with p_1 in Ind_1. We can see that p_1 in Ind_3 has more sensors than their parents in the same period, which allows it to escape local minima. When we choose p_3 in both Ind_1 and Ind_2, the union of them generates a solution equal to their parents, which indicates that solution can be good to improve the fitness of Ind_3. Again, after generating each new individual, we need to check if each active sensor in each period supports the discount of energy required to route information from their descendants. If a node cannot support it, the node and its sub-tree are deactivated.

C. Mutation Operator

Although the local search described in Section III-A can lead to solutions close to optimal, its operators can constraint the search space in such a way that the solutions get trapped in local minima. This is because, both during the operations of insertion (Section III-A1) and swapping (Section III-A2), the energy consumption of the network is analyzed only for the current period. The same happens with removal (Section III-A3) since the choice of sensors to be removed only takes into account the coverage area and connectivity.

Thus, the local search applied to each period tends to make the solution for a period to be repeated in subsequent ways until a node can no longer support the discount of energy needed to ensure the routing. This causes the appearance of *critical nodes*, i.e., nodes that are active at a given period but in the following have been disabled due to energy problems. This can be seen in Figure 7. Fig. 7a shows the configuration of a network presenting a critical node in a period P_1, and what happens with the network in P_2. Fig. 7b shows a configuration for the network in order to preserve the critical node in the period P_2. In both settings, the requirements of service quality are being reached. Although the mutation operator will act mainly on critical nodes, it can also consider non-critical nodes if critical nodes are absent. If it occurs, a random node is selected for mutation.

Node A is critical in P_1, since in P_2 this node is disabled due to energy problems. Therefore, nodes B and C are activated in P_2 to guarantee QoS requirements. If we activated node B in P_1, the number of descendants of node A would decrease from four to one, as this node could take some of the descendants of A. Thus, node A is still connected in P_2 and the network would save energy, since only the node B had to be active.

Mutation first identifies the critical nodes, and then randomly selects one to apply the mutation operator. The EA uses two types of mutation: Feasible Insertion and Feasible Swapping. Regardless of the type of mutation, it is always applied to the period where the critical node stands. The EA chooses one out of these two with equal probabilities.

The reduction of energy consumption of a critical node is commonly achieved by (i) activating a new node, (ii) passing one of the children of the critical node to another active node. These two situations are taking into account by two different mutation operators: *feasible insertion* and *feasible swapping*.

Feasible Insertion works in two phases: (i) chooses an inactive node to be inserted into the period K of the critical node, (ii) guarantees that this new node also appears active in the following time periods. In (i), the insertion operator of the local search (see Section III-A1) is used to perform the insertion, inserting the parent that causes the lowest

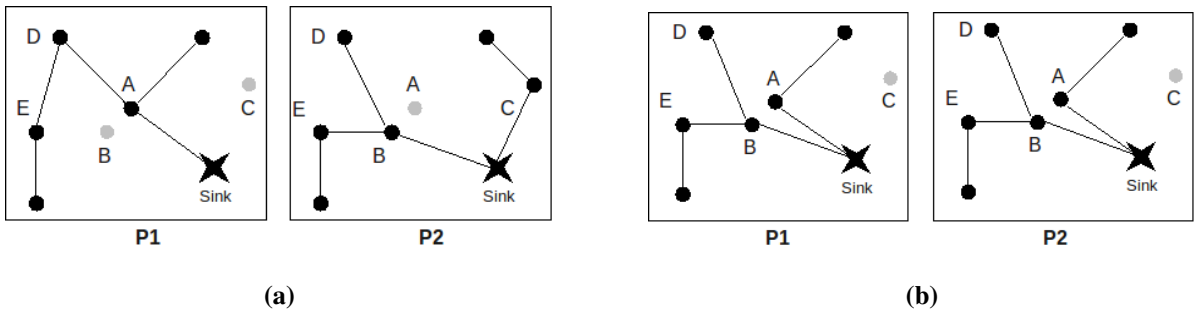

Fig. 7. Examples of critical nodes

total energy consumption to the network in the period being considered.

The selected inactive node N should be close to the center of mass of the critical node descendants d. In this way, the chances of N adopting any node in d is bigger. That done, the swapping (III-A2) and removal (III-A3) operators will be applied to connect active nodes to the newly inserted sensor (same case as Figure 7) if necessary.

After that, the local search is applied to the subsequent periods of K, with one exception: the node N is inserted in the period immediately after K, and cannot be removed during the search in that period. This allows modifications made in one period to propagated to the next.

Feasible swapping also works in two phases. In the first one, it randomly chooses a critical node descendant W to be analyzed. If the node has no descendants, the feasible insertion is called, since the insertion of a new node can reduce the energy consumption of the critical node. After that, an active node within the range of W is chosen to be its new parent. In the second phase, the local search is applied to the subsequent periods of K. Note that the connection between W and its new father cannot be changed in the subsequent immediate period of K.

D. Fitness Function

The proposed fitness function is an extension of the function presented in Section III-B. However, instead of evaluating the energy criteria, coverage and connectivity only for one period, we evaluate them for all periods simultaneously. Besides, as we verify the network connectivity after all operations, the criterion responsible for this (variable y in Eq.1) not need more necessary. The modified function can be formally defined as follows:

$$min \sum_{j=1}^{T} (\sum_{i \in S} \frac{E_i^t}{A_i^t} + (1 - \sum_{i \in S} \frac{A_i^t}{|D|})) \quad (2)$$

V. Computational Results

The proposed EA and GRASP-R were tested using 10 problem instances with a variable number of nodes randomly placed with a uniform distribution in a $100m^2$ monitored area. The instances have a variable number of periods, all with one hour of duration. The communication and sensing radiuses are $2m$ and $3m$, and the battery capacity is 400 mAh. Another necessary parameters to calculate the energy discount for each sensor (see [8] for more details) were: *activation energy* (energy spent to activate a node) equals to $10mAh$, *maintenance energy* (energy spent to keep a node active during a period) equals to $30mhA$, *reception energy* (energy spent by a sensor to receive packets from the others) equals to $10mhA$, and *transmission energy* (energy spent by a sensor to transmit packets from the others) based on the energy consumption of the sensor node MICAz [1]. We consider that all sensors spent 50% of the time transmitting packets.

The sink node is fixed in the center of the monitored area to allow more homogeneous energy consumption around the sensor field. As nodes placed closer to the sink tend to consume more energy, because they route a larger number of packets, they are usually the first nodes to run out of energy. Hence, placing the sink node in the center of the monitored area increases the number of nodes around it, better distributing the traffic in this region.

All the algorithms were programmed in C using gcc 4.3.3 compiler, system Ubuntu 9.10 64 bits, and the tests were performed on a Pentium Core 2 Quad 2.5 GHz and 4 GB of RAM. The proposed EA was compared to the solutions provided by the mathematical model of Integer Linear Programming (ILP) proposed by Nakamura et al. [8] and by the extend GRASP heuristic (GRASP-R) proposed in Section III-C. The ILP model was solved with the commercial optimization package CPLEX 10.1 [7].

The first phase of our experiments was to set the values of the parameters of the EA and GRASP-R. Preliminary experiments using a test instance with 60 sensors and 6 periods were performed. This instance was chosen because it presented the largest computational effort among the ones analyzed (Figure V), and keeping the execution time of an instance low was also a priority here. We vary the number of generations, size of the population and the local union, mutation and local search probabilities. All experiments were run 30 times, and the average and best solutions regarding the distance to the optimal (in percentage) are reported in Tables I, II and III, together with their computational time.

In Tables I and II, we observe that the quality of the average

TABLE I
RESULTS OBTAINED WHEN VARYING THE NUMBER OF GENERATIONS

NGeneration	Average (%)	Best (%)	Execution time(s)
30	0,92	0,75	5,92
50	0,86	0,32	8,64
80	0,81	0,29	13,79
100	0,78	0,24	16,96
150	0,75	0,24	25,06
200	0,7	0,24	33,14

TABLE II
RESULTS OBTAINED WHEN VARYING THE SIZE OF THE POPULATION

PopSize	Average (%)	Best (%)	Execution time(s)
10	0,92	0,56	5,23
20	0,79	0,32	10,28
30	0,78	0,24	16,91
40	0,76	0,29	22,73
50	0,79	0,29	28,76
60	0,72	0,27	34,19
70	0,64	0,29	39,87
80	0,57	0,29	46,9
90	0,49	0,29	52,62
100	0,55	0,29	58,11

and best solutions improves as the number of generations and the population size increase. In both cases, we can also note that the best solution found among the 10 runs converges as the number of solutions evaluated increase. In addition, more individual evaluations also imply more computational time. Hence, in order to find the best compromise between these two, we chose to use 100 generations and 40 individuals, considering as a parameter the convergence of the algorithm to the best solution.

Finally, Table III shows the variation on the probability of the Local Search Operator. We can see that as we increase the probability, we improve the average and best distances from the optimal solution, but also increase the required computational time. Notice that this operator works as an adjustment factor, performing a fine grained search in the promising areas of the search space. After many tests, we opt to use a probability of 90%, as in general, for other instance, it works better than a lower probability. In sum, the final parameter configuration was: population size equals to 40, maximum number of generations equals to 100, a tournament size of 2, and probabilities of union, mutation and local search of 0.4, 0.3 and 0.2, respectively.

For GRASP-R, the values to α and β were varied using the values 0.1, 0.5 and 0.9. These results were also analyzed using the distance from the optimal solution. We obtained the best results with α = 0.5 and β = 0.9. The maximum number of iterations was set to 3000, which is equivalent to the number of generations $\times$ the number of individuals used by the EA.

Table IV shows the distances between the solutions found

TABLE III
RESULTS OBTAINED WHEN VARYING THE LOCAL SEARCH PROBABILITY

Pl (%)	Average (%)	Best (%)	Execution time(s)
0	1,18	0,67	3,93
5	0,95	0,32	5,06
6	0,96	0,68	5,25
7	0,95	0,32	5,49
8	0,93	0,32	5,6
9	0,96	0,29	5,62
10	0,92	0,32	5,81
20	0,87	0,32	7,22
30	0,85	0,32	8,88
40	0,79	0,32	9,75
50	0,8	0,32	10,62
60	0,83	0,32	11,91
70	0,84	0,32	12,62
80	0,83	0,32	13,44
90	0,78	0,24	14,34
100	0,78	0,32	14,98

by the EA and GRASP-R (between brackets) when compared with the optimal solution. Each line corresponds to a different number of sensors, and results for 4, 5, 6 and 7 periods are presented. Note that instances considering seven periods and 100 sensors could not be solved accurately using the ILP model due to their time complexity. Table V

Observe that the EA found the optimal solution in all instances except the ones with 60 sensors and 5 and 6 periods. Besides, the EA reduced both the average value of the solutions found and the value of the worst solution found, when compared with the GRASP-R. This implies that both the strategy of reducing the search space and the escape of local minima proved to be very effective. Another thing to notice is that even those instances were the optimum is unknown, the EA was able to find solutions that were better or equal to the ones obtained by GRASP-R.

Although GRASP-R was also able to find solutions very close to the optimal, the computational time required when compared to the EA execution time was at least 6 times greater (see Table V). One reason for this discrepancy is that the local search is applied at each GRASP-R iteration, while in the EA its application depends on a user defined probability (in this case, 90%). Hence, we can conclude that, despite the local search is applied less times in the EA, it uses this search more efficiently, allowing that this approach can be more scalable than the GRASP-R.

VI. CONCLUSIONS AND FUTURE WORKS

This work presented an evolutionary algorithm (EA) to solve the Density Control, Coverage and Multi-Period Routing problem (DCCRMP) in Wireless Sensor Network. The great advantage of this work over others proposed in the literature is to optimize routing considering simultaneously different time periods, as well as to use a combined global and local search

TABLE IV
EVOLUTIONARY ALGORITHM AND GRASP-R RESULTS CONSIDERING THE DISTANCE FROM THE OPTIMAL SOLUTION

Number of Sensors	Number of time periods											
	4			5			6			7		
	Average (%)	Best (%)	Worst (%)	Average (%)	Best (%)	Worst (%)	Average (%)	Best (%)	Worst (%)	Average (%)	Best (%)	Worst (%)
40	0(0)	0(0)	0(0)	0(0,05)	0(0)	0(0,10)	0(0,07)	0(0)	0(0,08)	0(0)	0(0)	0(0)
45	0(0)	0(0)	0(0)	0(0,09)	0(0)	0(0,10)	0(0,08)	0(0)	0(0,08)	0(0)	0(0)	0(0)
50	0(0,39)	0(0,04)	0,04(0,49)	0,04(0,16)	0(0,07)	0,07(0,17)	0(0,08)	0(0,08)	0(0,08)	0(0)	0(0)	0(0)
55	0(0,10)	0(0)	0(0,34)	0,01(0,07)	0(0)	0,08(0,08)	0,02(0,12)	0(0,07)	0,07(0,18)	0(0,19)	0(0)	0(0,40)
60	0(0)	0(0)	0(0)	0,06(0,21)	0,06(0,06)	0,06(0,32)	0,78(0,96)	0,24(0,75)	1,23(0,99)	0,03(0,04)	0(0)	0,20(0,80)
100	3,63(1,80)	0(0)	4,70(3,66)	2,10(3,73)	0(1,13)	5,60(4,63)	2,29(3,33)	0(0,60)	6,19(4,99)	3,31(4,81)	0(4,26)	5,77(5,62)

TABLE V
EXECUTION TIMES OF THE CPLEX, GRASP AND EVOLUTIONARY ALGORITHM

Number of Sensors	Number of time periods											
	4			5			6			7		
	CPLEX (s)	GRASP (s)	EA (s)	CPLEX (s)	GRASP (s)	EA (s)	CPLEX (s)	GRASP (s)	EA (s)	CPLEX (s)	GRASP (s)	EA (s)
40	94,79	49,88	5,89	266,05	64,52	7,89	2500,02	77,12	11,06	4878,54	92,18	14,34
45	167,53	65,99	7,71	619,19	85,96	9,92	8863,49	99	13,85	35865,74	114,6	18,07
50	1151,77	82,35	9,7	3920,2	108,52	12,83	285918,46	126,3	17,56	-	146,28	22,46
55	2259,04	92,87	10,38	37991,68	123,7	12,9	53565,94	150,6	16,08	-	179,78	20,77
60	1213,12	106,23	8,02	3774	136,89	12,76	670633,64	166,3	16,47	-	197,53	20,11
100	-	343,69	18,77	-	438,39	22,3	-	512,03	24,77	-	627,5	34,37

strategy. We also modified the GRASP-based method originally proposed in Andrade et al. (Section III-C) to consider multi-period routing.

The results show that the proposed EA presents better results than those found by the revised version of GRASP. Although the computational cost of both methods is lower than the one obtained by CPLEX to solve the ILP model (Table V), the excessive use of the local search procedure in GRASP-R made its cost to be 20 times worse than the computational cost required by the EA.

In conclusion, the proposed evolutionary algorithm was able to meet the requirements of quality of service (QoS) network, while finding solutions close to or equal to the optimal with less computational time compared with the time spent by CPLEX to solve the ILP model and by the GRASP proposed by [2]. In future works we aim to perform more comparisons between the algorithms and create memory strategies that they can further reduce the search space without affecting the quality of the solutions.

REFERENCES

[1] Micaz - wireless measurement system. fonte: http://www.xbow.com/Products/productdetails.aspx?sid=164, June 2007.

[2] I. B. D. Andrade, F. G. Nakamura, and G. R. Mateus. A grasp heuristic to density control: Solving multi-period coverage and routing problems in wireless sensor networks. In *IEEE Symposium on Computers and Communications*, 2009.

[3] M. Cardei and J. Wu. Energy-efficient coverage problems in wireless ad-hoc sensor networks. In *Computer Commun.*, volume 29, pages 413–420, 2006.

[4] Y. Gao, K. Wu, and F. Li. Analysis on the redundancy of wireless sensor networks. In *Proceedings of ACM WSNA'2003*, volume 29, pages 108–114, Feb. 2003.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[6] C. Huang and Y. Tseng. The coverage problem in a wireless sensor network. In *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 115–121. ACM Press, 2003.

[7] ILOG. Cplex. source: http://www.ilog.com/products/cplex/, May 2006.

[8] F. G. Nakamura, F. P. Quinto, G. C. Menezes, and G. R. Mateus. An optimal node scheduling for flat wireless sensor networks. In *4th International Conference on Networking - Lecture Notes in Computer Science*, volume 3420, pages 104–483, 2005.

[9] P. Rentala, R. Musunuri, S. Gandham, and U. Saxena. Survey on sensor network. In *Mobile Computing (CS6392) Course*, 2001.

[10] I. G. Siqueira, C. M. S. Figueiredo, A. A. F. L. J. M. S. Nogueira, and L. B. Ruiz. An integrated approach for density control and routing in wireless sensor networks. In *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS'06)*, 2006.

[11] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman. Infrastructure tradeoffs for sensor networks. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, pages 49–58. ACM Press, 2002.