Hindawi
Applied Computational Intelligence and Soft Computing
Volume 2022, Article ID 3745358, 21 pages
https://doi.org/10.1155/2022/3745358

Research Article

Wireless Sensor Network Coverage Optimization: Comparison of Local Search-Based Heuristics

Krzysztof Trojanowski and Artur Mikitiuk

Cardinal Stefan Wyszyński University in Warsaw, Wóycickiego 1/3, Warsaw 01-938, Poland

Correspondence should be addressed to Artur Mikitiuk; a.mikitiuk@uksw.edu.pl

Received 22 March 2022; Revised 25 July 2022; Accepted 16 August 2022; Published 12 November 2022

Academic Editor: Jun He

The Maximum Lifetime Coverage Problem (MLCP) requires heuristic optimization methods due to its complexity. A real-world problem model determines how a solution is represented and the operators applied in these heuristics. Our paper describes adapting a local search scheme and its operators to MLCP optimization. The operators originate from three local search algorithms we proposed earlier: LS_{HMA}, LS_{CAIA}, and LS_{RFTA}. Two steps of the LS scheme's main loop can be executed in three different ways each. Hence, nine versions of the LS approach can be obtained. In experimental research, we verified their effectiveness. Test cases come from three benchmarks: SCP1, proposed and used in our earlier research on the three LS algorithms mentioned above, and two others found in the literature. The results obtained with SCP1 showed that the algorithm based on the hypergraph model approach (HMA) is the most effective. The remaining results of the other algorithms divide them into two groups: effective ones and weak ones. However, other benchmarks showed that the more redundant the coverage of points of interest (POIs) by sensors, the more effective the perturbation method from the approach inspired by cellular automata (CAIA). The findings expose the strengths and weaknesses of the problem-specific steps applied in the LS algorithms.

1. Introduction

Wireless sensor networks (WSNs) are essential parts of IT solutions in many applications: military, like battlefield surveillance, and civil ones, including forecast systems, environment observation, or habitat monitoring [1]. Sensors can be integrated into numerous electronic devices and machines. Moreover, due to the advances in semiconductor and microelectromechanical technologies and the miniaturization of computing and sensing technologies, sensors and microcontrollers are tiny, consume low power, and are inexpensive. Thus, WSNs may consist of large numbers of small yet powerful devices cooperating in large areas. WSNs aim to monitor a region or a set of targets for collecting valuable information for modeling and forecasting situations in the area or controlling the usage of resources. In WSNs consisting of miniature devices, monitoring quality becomes an energy efficiency issue due to limited battery capacities.

WSN lifetime maximization techniques depend on two main components of a problem model. The first is objective, e.g., network lifetime optimization, coverage, connectivity, or transmission parameters. The second one represents WSN design constraints, for example, communication medium, resource limits, fault tolerance and self-organization, QoS requirements, or mobility and deployment [2]. In our research, we focus on the issues concerning network lifetime maximization, that is, the maximization of an uninterrupted interval when the network satisfies the level of coverage above a certain threshold under some resource-limited constraints. Our research concerns a simplified network model, where sensors remain immobile once deployed, and their connectivity is always guaranteed. We assume that individual sensor placement is infeasible due to environmental conditions, such as monitoring a disaster area or a battlefield. Therefore, sensors are randomly scattered over the monitored field. Sensors have a finite battery capacity. Thus, after the drain, they should be replaced or

recharged. However, we assume that neither of these options is applicable due to the operating conditions. So, once the energy reserve is depleted, the sensor is regarded as irretrievably lost to the network. The network is equipped with an external solid computational unit responsible for optimally scheduling and distributing tasks among the network devices.

We assume that sensors have uniform batteries and sensing ranges and have to monitor a set of targets also called points of interest (POIs). When the number of sensors is extensive and their monitoring ranges overlap, some sensors can be off while the required level of coverage is still satisfied. In this case, the optimization aims to find a sensor activity schedule with the longest makespan and guarantee a sufficient coverage level. This class of problems is called the Maximum Lifetime Coverage Problem (MLCP). The approaches to solving MLCP are based on different models of real-world circumstances; thus, they have different structures of the solution and the complexity of the problem.

In this paper, we develop local search strategies enriched by problem-specific procedures originating from three algorithms we proposed earlier: LS_{HMA}, LS_{CAIA}, and LS_{RFTA}. The strategies share the same execution scheme. In the beginning, they find a first feasible schedule that becomes a preliminary solution to the problem. Then, iteratively, they generate its neighbor and try to refine it. The new solution takes the place of its ancestor when we gain an improvement. The advantage of these strategies lies in the neighbor generation (so-called perturbation) and refinement procedures, which take advantage of some model-specific properties.

In recent years, many heuristic approaches have been proposed; thus, quite naturally, an idea arose to verify the efficiency of hybrid heuristic algorithms composed of components of various origins. Unfortunately, some approaches do not share the same model of the real-world problem and the same set of constraints. Therefore, their integration into one method is a nontrivial task and sometimes even questionable. However, this is not the case for LS_{HMA}, LS_{CAIA}, and LS_{RFTA}, where the approaches share the same model of the real-world problem and the structure of the solution representation. Hence, one can easily use selected problem-specific steps as exchangeable building blocks. Eventually, we constructed nine local search strategies by swapping the two steps: perturbation and refinement, from each of the three approaches. Then, we evaluated their performance experimentally.

The main contribution of this paper lies in generating nine heuristics, based on the building blocks originating from three other existing approaches, and experimentally verifying their efficiency. We used three benchmarks for testing the performance of the heuristics: SCP1, provided in our earlier publications [3–6]; the benchmark from [7, 8]; and the benchmark from [9].

The paper is organized as follows. Related work is briefly discussed in Section 2. Section 3 formally defines the Maximum Lifetime Coverage Problem (MLCP). The local search approach is introduced in Section 4. Section 5 describes our experiments with LS algorithms for MLCP. Our conclusions are given in Section 6.

2. Related Work

The majority of MLCP solving methods are based on two-stage approaches. In the first stage, we solve the Target Coverage Problem by finding the maximum number of sensor sets so that every set can perform the coverage task individually. In the second stage, we solve the Set Coverage Problem by finding the optimal scheduling for the sets of covers obtained in the first stage. The constraint for the sets concerning full coverage of POIs proposed in the first publications, for example, in [10], was later relaxed in [11] by introducing the required minimum percentage of coverage constraint.

The sets of covers are of minimum cardinality and can be disjoint or non-disjoint. In the disjoint sets, every sensor can be included in one of the sets of covers at most, whereas in the non-disjoint sets, there is no such restriction. In [12], the authors introduce the Disjoint Set Cover (DSC) problem, that is, the problem of finding the maximum number of disjoint covers, where every cover is a set of sensors that together monitor all the POIs, and prove its NP-completeness. The complexity of the case with non-disjoint sets of covers is analyzed in [13], where the Maximum Set Cover (MSC) problem is defined. It is a problem of determining a number of p non-disjoint covers to maximize the network lifetime $t_1 + \cdots + t_p$, where t_j, $j = 1, \ldots, p$, is the time interval while the j-th cover is active. The authors of [13] prove that the MSC problem is also NP-complete.

Selected papers on MLCP optimization with Disjoint and Non-Disjoint Set Cover based approaches are presented in Table 1 (DSC) and Tables 2 and 3 (NDSC). For information about sensor activity scheduling strategies for other WSN lifetime optimization definitions, the reader is referred to surveys and monographs, for example, [1, 2, 35, 36].

3. Maximum Lifetime Coverage Problem (MLCP)

3.1. Model of the Real-World Sensor Network. The subject of our research is a network of immobile homogenous sensors monitoring a number of points of interest (POIs). These sensors are randomly distributed over a monitored area. Sensors' batteries have limited capacity. For energy saving purposes, some sensors can be turned off from time to time. We propose a model of such a network activity where we assume that time is discrete. During every time slot, a sensor can be on or off. When a sensor is off, its energy consumption is negligible. When a sensor is active, it consumes one unit of energy during every time slot. The number of time slots when the sensor can be active, that is, the initial battery load of a sensor, is denoted by T_{batt}.

The model of a sensor considered here is simplified. In real life, the working time of a battery depends on the surrounding temperature and how the battery is used. When we turn it on and off frequently, it discharges more than when we keep it on and off for extended periods. We ignore such problems in this research.

We assume that every POI can be covered by at least one sensor. Some POIs can be within the sensing range of

Table 1: Selected papers on MLCP optimization with Disjoint Set Cover based approaches.

No.	Reference	Year	Brief information about the approach and its optimization goal
1	S. Slijepcevic and M. Potkonjak [10]	2001	The most constrained–minimally constraining covering heuristic maximizes the cardinality of the set of covers
2	M. Cardei and D. Z. Du [12]	2005	The problem of set cover maximization converted into maximum-flow problem is modeled and solved by mixed integer programming (MIP), and then a heuristic computes the covers based on the solution returned by MIP
3	C.-C. Lai et al. [14]	2007	A genetic algorithm finds the maximum number of covers using the idea of critical sensors (the sensors covering the most sparsely covered POIs) to find an upper bound of the number of covers
4	X.-M. Hu et al. [15]	2010	The schedule transition hybrid genetic algorithm with the forward encoding scheme for the representation of chromosomes and problem-specific operators maximizes the number of covers and schedules them for makespan maximization
5	N. Ahn and S. Park [16]	2011	a binary integer programming formulation and heuristics compute the maximum number of covers
6	J.-M. Gil and Y.-H. Han [17]	2011	A genetic algorithm using a two-dimensional chromosome and problem-specific operators finds and schedules a given number of covers for makespan maximization in directional sensor networks
7	R. Cerulli et al. [18]	2012	A greedy heuristic and exact approach based on the column generation technique find the maximum number of covers
8	Y. Lin et al. [19]	2012	An ant colony optimization-based approach finds the maximum number of covers
9	M. E. Keskin et al. [20]	2014	The period iteration heuristic and the sequential assignment heuristic maximize four design issues: sensor locations, activity scheduling, data, and mobile sink routes integrated into an integer linear programming model
10	L. Wang et al. [21]	2017	Whale group algorithm optimizes the cover in terms of overall network performance: coverage, node utilization, and energy consumption
11	S. Balaji et al. [22]	2020	a cuckoo search algorithm finds the maximum number of covers

multiple sensors. Since an active sensor covers every POI in its sensing range, this means that not all sensors must be active all the time. Moreover, many applications do not need to monitor all POIs all the time. It is often sufficient to monitor 80 or 90% of them anytime. We call this value the required level of coverage and denote it by $\mathrm{cov} \in [0, 1]$. We want to maintain this level of coverage all the time, but to save sensor batteries, we want to keep the coverage level as close to cov as possible. It should not exceed cov by more than the tolerance factor δ (usually 2–5%).

In the real world, some sensors not needed for monitoring may be necessary to assure communication within the network. Our simplified model assumes steady sensor connectivity over the network lifetime and negligible energy spent on communication.

3.2. Formal Description. Let us formulate the network makespan maximization regarding a scheduling problem as proposed in [37]. We consider N_S parallel machines which represent sensors from $S = \{s_1, \ldots, s_{N_S}\}$ and N_P POIs from $P = \{p_1, \ldots, p_{N_P}\}$. Each machine has assigned its task $t(\cdot)$ consisting of a subset of POIs. The i-th task contains POIs located in the monitoring range of the i-th sensor: $t(s_i) = \{p_k, \ldots, p_l\}$. An m-th job J_m, where $m \in \{1, \ldots, N_J\}$, consists of some tasks scheduled for being executed simultaneously in a single time slot. The tasks included in a job are not chosen randomly. They identify a set of machines active when this job is executed. That is, they are a set of sensors for which the selection criterion is to cover the requested number of POIs. Each slot has its duration time sdt_m; hence, the schedule makespan M_{sch} is a sum of jobs' duration times: $M_{\mathrm{sch}} = \sum_{m=1}^{N_J} sdt_m$. In further considerations, for simplicity, we assume that sdt equals a time unit, and thus the schedule makespan is the same as the total number of jobs in the schedule N_J.

In our experiments, a network activity schedule H is represented as a 0–1 matrix with rows corresponding to machines and columns corresponding to jobs. The element in row i and column m is equal to 1 (resp., 0) when the machine i is on (resp., off) in job J_m.

The required level of coverage constraint says that the cardinality of the sum of POIs in tasks assigned to active machines in the job J_m should be equal to or exceed the $\mathrm{cov} \cdot \mathbf{card}(P)$; that is,t

$$\mathrm{cov} \cdot \mathbf{card}(P) \leq \mathbf{card}\left(\bigcup_{i=1}^{N_S} \begin{cases} t(s_i) \text{ if } H[i,m] = 1 \\ \varnothing \text{ otherwise} \end{cases} \right). \tag{1}$$

Moreover, for all machines, the processing time should not exceed the limit T_{batt}:

$$\sum_{m=1}^{N_J} H[i,m] \leq T_{\mathrm{batt}}. \tag{2}$$

The objective is the maximization of the schedule makespan M_{sch} with the required level of coverage and without exceeding the maximum processing time of machines T_{batt}.

4. General Scheme of the Local Search

In our earlier papers [4–6], we proposed three heuristic algorithms for solving MLCP. They all follow the general

Table 2: Selected papers on MLCP optimization with Non-Disjoint Set Cover based approaches—Part I.

No.	Reference	Year	Brief information about the approach and its optimization goal
1	P. Berman et al. [23]	2004	For a given series of covers, the method maximizes the sum of activation times of the covers by formulating the problem as a packing linear program
2	M. Cardei et al. [13]	2005	(1) The problem of set cover maximization is modeled and solved by the mixed integer programming (MIP), and then a linear programming based heuristic computes the covers based on the solution returned by MIP. (2) A greedy heuristic solves the problem of set cover maximization
3	D. Zorbas et al. [24]	2010	A greedy heuristic reinforced by the idea of critical targets (POIs with small neighbor-sensor sets) finds the maximum number of covers which is equal to the schedule makespan since sets provide coverage for the same unit of time
4	K. Deschinkel [25]	2011	The column generation based heuristic finds covers and maximizes the sum of activation times of the covers for the problem modeled with a linear programming formulation
5	Manju and A. K. Pujari [26]	2011	High-energy-first heuristic finds covers using the highest remaining battery states; the selection priorities change after every execution of the selected set, hence the variety of sets over the network lifetime
6	H. Mohamadi et al. [27]	2013	Three learning automata-based scheduling algorithms find both disjoint and non-disjoint covers and then optimize their cardinality
7	A. Tretyakova and F. Seredynski [28]	2013	Two approaches—genetic algorithm and memetic algorithm with a two-dimensional representation of schedules (columns: covers; rows: sensors)—maximize the number of feasible covers satisfying the battery capacity restriction in the rows
8	F. Castaño et al. [29]	2014	The problem of set cover maximization is solved by the column generation based framework having embedded the greedy randomized adaptive search procedure and the variable neighborhood search
9	A. Tretyakova and F. Seredynski [7]	2015	Simulated annealing with a two-dimensional representation of schedules (columns: covers; rows: sensors) maximizes the number of feasible covers satisfying the battery capacity restriction in the rows
10	Y. E. E. Ahmed et al. [30]	2016	A genetic algorithm maximizes the number of covers and schedules them for makespan maximization
11	Y. E. E. Ahmed et al. [31]	2016	A genetic algorithm with problem-specific operators maximizes the number of covers and schedules them for makespan maximization
12	A. Tretyakova et al. [8]	2016	Graph cellular automata approach for cover generation and makespan maximization where cells correspond to sensors, and the neighborhood relation between cells maps the existence of targets or areas commonly monitored by these sensors
13	K. Trojanowski et al. [4]	2017	Local search-based approach with problem-specific perturbation operators (LS_{RFTA}) for cover generation and makespan maximization

Table 3: Selected papers on MLCP optimization with Non-Disjoint Set Cover based approaches—Part II.

No.	Reference	Year	Brief information about the approach and its optimization goal
14	K. Trojanowski et al. [5]	2017	Local search-based approach with a hypergraph model of the WSN and problem-specific perturbation operators (LS_{HMA}) for cover generation and makespan maximization
15	J. Roselin et al. [32]	2017	Heuristic for cover generation and makespan maximization where the sensor priority depends on its sensing coverage/connectivity and its remaining energy
16	Manju et al. [9]	2017	Minimal heuristic finds covers and maximizes the makespan
17	A. Tretyakova et al. [33]	2017	Stochastic greedy heuristic and simulated annealing algorithm maximize the number of feasible covers and optimize their execution times in the way satisfying the battery capacity restriction in the rows
18	K. Trojanowski et al. [6]	2018	Local search-based approach with perturbation operator inspired by cellular automata (LS_{CAIA}) for cover generation and makespan maximization
19	Y. E. E. Ahmed et al. [34]	2018	An exact method of cover generation based on the sensor coverage relation matrix and using the integer linear programming model
20	The approach presented in this publication	2022	Local search-based approaches with the neighbor generation (so-called perturbation) and refinement procedures taking advantage of some model-specific properties and originating from LS_{HMA}, LS_{CAIA}, and LS_{RFTA} for cover generation and makespan maximization

approach of local search presented in Algorithm 1. Each of them uses different methods in the three steps of local search: initialization (Step #1: line 1), perturbation (Step #2: line 3), and refinement (Step #3: line 4).

The methods used at the stage of initialization of the proposed algorithms have the following names: the random and fine-tuning approach (RFTA), the cellular automata inspired approach (CAIA), and the hypergraph model

```
(1)  Initialize x ∈ 𝒟 ▷ step #1
(2)  Repeat
(3)     x' = perturbation(x) ▷ step #2
(4)     x'' = refinement(x') ▷ step #3
(5)     If F(x'') > F(x) then
(6)        x = x'' ▷ x is replaced by its neighbor
(7)     End if
(8)  Until termination condition met
(9)  Return x
```

Algorithm 1: Local search (for the schedule maximization context).

approach (HMA). Thus, we denote the LS algorithms proposed in the corresponding papers, respectively, by LS_{RFTA}, LS_{CAIA}, and LS_{HMA}. This notation appeared for the first time in [3], not in original papers [4–6]. In all initialization step methods, we start with an empty schedule and add to it new slots, one by one. Just the methods used to obtain a new slot are different. Adding a new slot to a schedule decreases the battery levels of the sensors active in this slot.

When the initialization step terminates (Step #1 in Algorithm 1), some sensors are usually left with non-empty batteries. However, turning on even all these sensors does not guarantee a sufficient coverage level. Thus, we cannot regard this set as a configuration for yet another slot. These sensors contribute to the perturbation and refinement steps (Steps #2 and #3 in Algorithm 1). In the perturbation step, we either remove some slots or modify sets of active sensors in some slots, adding sensors from the set mentioned above and removing others. When we remove an entire slot from the schedule or turn off one or more sensors in a slot, we recover energy. This way, the set of sensors with non-empty batteries grows, and we hope to obtain one or more new slots from this set. In the refinement step, we employ one of the methods used in the initialization step to create new slots and add them to the modified schedule. If the new schedule is longer than the initial one, it replaces the initial one. The perturbation and refinement steps are repeated until some termination condition is met.

Removing some slots from the schedule in the perturbation step can be done in two ways. In [4], such slots are selected randomly. In [6], first, we try to turn off every active sensor in every slot. The decision to turn it off is made with a low probability threshold. Next, if removing an active sensor from a slot makes this slot unfeasible, the whole slot is deleted. Our experiments show that more than one slot is removed even for a very low probability, such as 0.0005. Thus, this method of perturbation is stronger than the former.

In [5], we used a completely different perturbation approach. We add the sensors from the set with non-empty batteries to randomly selected slots but only when adding an additional sensor increases the coverage level. If this is not the case, we must choose another slot for this sensor. When we use up all the sensors, we attempt to remove from the modified slots other sensors to make their coverage level as close to cov as possible.

Thus, for each of the three steps of local search, we have three different methods of proceeding. By selecting one of these methods at every step, we get 27 different variants. We will call these algorithms using the names of every step's origins. For example, [HMA, RFTA, CAIA] denotes an algorithm where the initialization procedure from the LS_{HMA} algorithm generates the initial schedule, the perturbation step originates from LS_{RFTA}, and the refinement step originates from LS_{CAIA}. [RFTA, RFTA, RFTA] is identical with the LS_{RFTA} algorithm because all three problem-specific steps come from this algorithm.

5. Experiments

To assess the performance of new algorithms, we conducted experiments with some of them. We decided to always use the same method to generate an initial schedule. This way, the set of 27 algorithms described in Section 4 has been reduced to 9 because our algorithms differ only in the steps of the main loop in Algorithm 1. For the initialization step, a method producing schedules of moderate quality would be appropriate. It should give the main loop more room to improve these schedules and thus highlight performance differences between algorithms. Since HMA usually generates the most extended schedules, which are often hard to improve, we chose between the remaining two methods—RFTA and CAIA. We decided to use CAIA, and this choice was somewhat arbitrary. Eventually, we compared experimentally nine versions of LS algorithms of the form [CAIA,*,*] where * stands for one of HMA, CAIA, and RFTA. Consequently, for a problem instance, all versions of the main loop start with the same initial schedule. The termination condition of the loop is a limit of 500 iterations.

From now on, we skip the initial CAIA in notation given at the end of Section 4 and refer to tested algorithms giving only the origins of the perturbation step and the refinement step. The compared nine algorithms are [HMA, HMA], [HMA, RFTA], [HMA, CAIA], [RFTA, HMA], [RFTA, RFTA], [RFTA, CAIA], [CAIA, HMA], [CAIA, RFTA], and [CAIA, CAIA].

In the experimental part, we used the benchmark SCP1 proposed in our earlier publications but also did tests with two external benchmarks: one from [7, 8] and the other from [9], selected due to compatible problem definitions and optimization criteria but differently defined test cases.

Experiments were conducted on HP Workstation Z2 G4 SFF with Intel® Core™ i7-8700 CPU @ 3.20 GHz and 16 GB RAM and Windows 10 Pro. Application for simulations was implemented in MS Visual Studio *C++*.

5.1. Measurement Methodology. The best measure of the efficiency of our LS algorithms would be a comparison of the obtained problem solution with an optimal solution. In this case, we would compare the length of the schedule returned by an LS algorithm with the length of an optimal schedule for the given problem instance. Unfortunately, we do not know optimal solutions because it is impossible to compute them in a reasonable time due to the problem's computational complexity. Therefore, we decided to compare our solutions to the best-obtained suboptimal solution of the problem instance. These best-obtained suboptimal solutions were produced by LS algorithms using HMA to get initial schedules. Since initial schedules produced by HMA are longer than those obtained

by other approaches, we hope the resulting schedules are genuinely close to optimal after applying LS to improve them. The lengths of best-obtained suboptimal schedules were used as reference values to evaluate the percentage quality of the schedules produced by our nine LS algorithms.

Moreover, we measured the mean percentage improvement of the initial schedule's length, obtained in the main loop of Algorithm 1. It was calculated by subtracting the length of the initial schedule from the length of the final schedule and dividing the difference by the length of the initial schedule. For every class of problems, the obtained values were averaged over the number of problem instances.

5.2. Performance Comparisons Using Benchmark SCP1. In our first group of experiments, we used the benchmark SCP1 (Sensor Coverage Problem, Set No. 1) introduced in our earlier publications [3–6].

5.2.1. Benchmark SCP1. SCP1 consists of eight classes of problems. There are 2000 sensors with the sensing range of one abstract unit in all of them. The monitored area is a square. Its side size varies from 13 to 28 abstract units (possible values: 13, 16, 19, 22, 25, and 28). POIs are placed in nodes of a rectangular or a triangular grid. Since the distance between grid nodes grows together with the area side size, the number of nodes is similar for all classes of SCP1. However, only about 80% of grid nodes have a POI. A POI is located in the node only if a randomly generated number between 0 and 1 is less than 0.8.

Consequently, instances of the same test case can have different numbers of POIs. For the triangular grid, this number is between 199 and 240, while for the rectangular grid, it is between 166 and 221. Coordinates of sensor locations are obtained using either a random generator or a Halton generator. Eventually, the test classes have the following configurations: 1. $[13 \times 13, \Delta\mathscr{R}]$, 2. $[13 \times 13, \diamond\mathrm{H}]$, 3. $[16 \times 16, \Delta\mathscr{H}]$, 4. $[19 \times 19, \diamond\mathscr{R}]$, 5. $[19 \times 19, \Delta\mathscr{R}]$, 6. $[22 \times 22, \Delta\mathscr{H}]$, 7. $[25 \times 25, \Delta\mathscr{R}]$, 8. $[28 \times 28, \Delta\mathscr{H}]$, where Δ means a triangular grid of POI locations, $\diamond$ means a rectangular grid, and $\mathscr{H}$ and $\mathscr{R}$ mean a Halton and a random generator of sensor locations. We generated 40 instances of every class.

Figure 1 depicts boxplots (minimum, lower quartile, median, upper quartile, and maximum) of Maximum–Minimum Distance (MMD) [38] values for the sensors and POIs in the instances sets. Precisely, the boxplots show the MMD of the evenness measure and are calculated as follows:

$$\mathrm{MMD} = \max_{x \in S}\left(\min_{v \in V}\left(d(x, v)\right)\right), \tag{3}$$

where $d(\mathbf{x}, \mathbf{v})$ is an Euclidean distance between $\mathbf{x}$ and $\mathbf{v}$, S is the set of sensors' locations, and V is the set of vertices in the Voronoi polygons for the set of POIs locations. The values in the diagram grow as the side size of the monitored area grows, which is reasonable. Moreover, for the areas of the same side size, values for the cases where POIs are located on the nodes of the triangular grid are smaller than those for the cases applying a rectangular grid for the POI distribution.

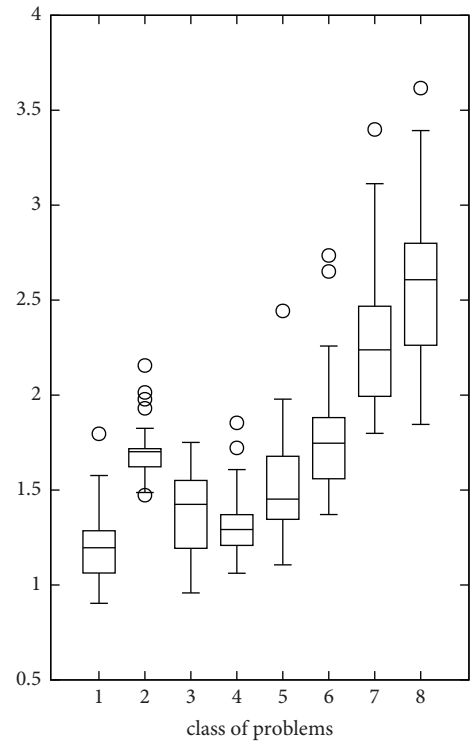

Figure 1: Boxplots of MMD values (minimum, lower quartile, median, upper quartile, maximum, and outliers) for the sets of 40 instances for the eight classes of SCP1.

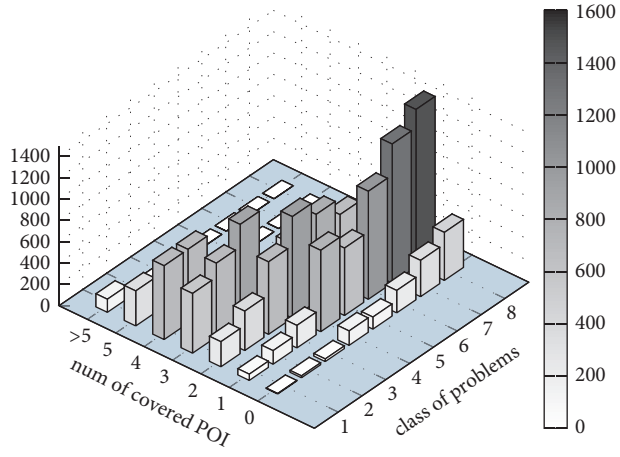

Figure 2: Mean numbers of sensors covering 0, 1, 2, 3, 4, 5, and more than 5 POIs for the eight test cases of SCP1.

Figure 2 shows the mean numbers of sensors covering 0, 1, 2, 3, 4, 5, and more than 5 POIs for the eight test cases of SCP1. One can see that the number of sensors covering a more significant number of POIs decreases as the side size of the monitored area grows. When the side sizes increase, the overlapping sections of neighbor sensor monitoring areas shrink, and some even disappear, so the

Table 4: Mean makespans of schedules returned by the LS algorithms for the SCP1 benchmark with *cov* = 80%. Codes in column headers: C: CAIA, H: HMA, R: RFTA; e.g., HR represents the version [HMA, RFTA].

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1	1076.95	1013.42	1064.47	992.50	630.75	645.21	623.29	627.43	623.02
2	1123.49	1049.56	1115.98	1030.01	653.39	670.46	644.69	649.18	644.50
3	741.78	706.81	739.25	698.68	428.38	429.47	419.91	421.79	419.72
4	490.89	493.20	456.59	483.83	282.14	273.80	268.23	269.27	268.05
5	524.84	520.62	498.60	514.87	300.56	295.96	290.57	291.67	290.32
6	423.40	418.12	412.82	415.20	241.59	233.57	230.17	231.02	229.92
7	308.84	311.64	291.18	308.24	183.09	175.99	172.92	173.57	172.67
8	272.86	273.61	263.97	272.79	161.22	152.76	150.51	150.94	150.24

Table 5: Mean percentage qualities of the best-found schedules returned by the LS algorithms (the top table) and mean percentage improvement of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table) for each of the eight classes of SCP1 with cov = 80%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1	97.29	91.55	96.16	89.66	56.98	58.29	56.30	56.68	56.28
2	97.23	90.83	96.58	89.14	56.55	58.02	55.80	56.18	55.78
3	97.45	92.85	97.12	91.79	56.28	56.42	55.16	55.41	55.14
4	95.01	95.45	88.37	93.64	54.61	52.99	51.91	52.12	51.88
5	96.65	95.87	91.82	94.82	55.35	54.50	53.51	53.71	53.46
6	98.34	97.11	95.89	96.44	56.12	54.25	53.46	53.66	53.41
7	94.24	95.09	88.86	94.06	55.88	53.71	52.77	52.97	52.69
8	97.36	97.63	94.18	97.33	57.53	54.50	53.71	53.86	53.61
No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1	73.45	63.22	71.42	59.85	1.58	3.91	0.38	1.04	0.33
2	74.93	63.42	73.75	60.37	1.73	4.39	0.38	1.07	0.35
3	77.63	69.26	77.01	67.31	2.58	2.83	0.54	0.99	0.50
4	84.53	85.41	71.64	81.88	6.07	2.93	0.83	1.23	0.77
5	82.07	80.61	72.96	78.62	4.27	2.67	0.80	1.18	0.71
6	85.75	83.43	81.10	82.15	5.99	2.46	0.97	1.34	0.86
7	81.05	82.69	70.70	80.69	7.35	3.17	1.37	1.76	1.23
8	84.04	84.54	78.04	83.99	8.75	3.03	1.52	1.81	1.33

numbers of shared POIs decrease. In the last class of problems, almost 75% of the sensors cover only one POI.

In our experiments, we assumed the required level of coverage cov as either 80 or 90%, while the tolerance factor δ was 5%. The same set of experiments was repeated for five different values of T_{batt}—10, 15, 20, 25, and 30.

5.2.2. Mean Normalized Lengths of Schedules and Percentage Quality of Schedules. The lengths of the schedules returned by the LS algorithms in question are the natural outcome of our experiments. However, the average length returned for all problem instances from a particular class may not measure the algorithm quality well. The optimal schedule lengths may differ for subsequent instances and values of T_{batt}. For the purpose of compatibility of output values, we normalized the schedule lengths. We assumed that the lifetime of batteries is the same, but different numbers of sensor activity intervals represented by T_{batt} can be available for scheduling. We assumed that, for $T_{\text{batt}} = 30$, the battery lifetime is divided into 30 intervals and a slot takes one unit. Thus, the schedule makespan equals exactly the number of slots in a schedule. Consequently, for $T_{\text{batt}} = 10$, one slot takes three time units; for $T_{\text{batt}} = 15$, two units; for $T_{\text{batt}} = 20$, 1.5 units; and for $T_{\text{batt}} = 25$, 1.2 units. Eventually, normalized lengths of schedules represent the schedule makespans, that is, the total number of slots multiplied by the number of time units per slot.

Another measure is to compute for every result its percentage quality with respect to the best-known suboptimal schedule and then average these percentages over all the problem instances in a class.

Table 4 presents the mean makespans of schedules for particular classes of test cases. The mean makespan is computed as the sum of the makespans obtained for all instances and all values of T_{batt} divided by 40×5 (the number of instances for a class of problems multiplied by the number of different values of T_{batt}). Table 5 shows, for each of the eight classes of SCP1, the mean percentage qualities of the best-found schedules returned by the LS algorithms (the top table). It also shows the mean percentage improvement of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table). Tables 4 and 5 present results for cov = 80%. Tables 6 and 7 show the same types of results as Tables 4 and 5 but assuming *cov* = 90%.

One can see from these results that [HMA, HMA] is usually the best version of LS. Three other LS algorithms,

Table 6: Mean makespans of schedules returned by the LS algorithms for the SCP1 benchmark with cov = 90%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1	905.12	887.02	814.68	862.19	678.27	456.90	432.13	434.15	431.50
2	975.69	946.18	861.55	905.32	746.50	480.18	450.46	452.91	449.92
3	636.98	626.42	576.43	605.41	523.98	305.81	293.27	294.35	292.79
4	365.33	371.18	322.44	366.09	302.65	191.88	181.82	181.75	180.70
5	418.08	420.40	372.41	414.85	358.20	208.96	200.04	200.36	199.42
6	356.69	358.54	331.46	352.69	317.19	168.57	162.84	163.17	162.12
7	236.60	239.14	214.89	237.34	182.97	125.54	120.13	119.72	119.12
8	228.63	230.24	213.71	229.33	201.44	112.21	108.55	108.24	107.75

Table 7: Mean percentage qualities of the best-found schedules returned by the LS algorithms (the top table) and mean percentage improvement of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table) for each of the eight classes of SCP1 with cov = 90%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1	96.83	94.89	87.16	92.24	72.59	48.88	46.23	46.45	46.16
2	97.74	94.78	86.31	90.69	74.80	48.10	45.12	45.37	45.07
3	97.41	95.79	88.16	92.57	80.16	46.76	44.85	45.01	44.77
4	92.81	94.30	81.93	93.00	76.92	48.74	46.20	46.18	45.91
5	94.03	94.55	83.77	93.30	80.60	47.00	45.00	45.07	44.86
6	97.26	97.77	90.39	96.17	86.52	45.97	44.41	44.50	44.21
7	92.81	93.81	84.30	93.10	71.80	49.24	47.13	46.96	46.73
8	95.58	96.25	89.34	95.87	84.23	46.91	45.38	45.25	45.05
No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1	110.97	106.77	89.87	100.98	58.14	6.49	0.71	1.19	0.57
2	118.13	111.54	92.58	102.40	66.91	7.35	0.70	1.25	0.58
3	119.27	115.65	98.41	108.41	80.41	5.27	0.95	1.32	0.79
4	104.58	107.86	80.58	105.01	69.47	7.48	1.83	1.79	1.20
5	112.08	113.26	88.92	110.44	81.73	6.00	1.48	1.64	1.17
6	122.97	124.13	107.21	120.47	98.32	5.38	1.79	2.00	1.34
7	102.55	104.72	83.97	103.18	56.63	7.49	2.86	2.50	1.99
8	116.57	118.09	102.43	117.23	90.86	6.28	2.83	2.53	2.07

[HMA, RFTA], [CAIA, HMA], and [HMA, CAIA], are also effective, while the remaining five LS versions give much worse results—we will call them weak approaches. However, for cov = 90%, [RFTA, HMA] is better than the other weak approaches but not so good as effective ones. Thus, all LS algorithms with the perturbation method from LS_{HMA} produce a relatively good schedule, no matter what way is used to refine the schedule obtained after the perturbation.

However, our results also show that [HMA, HMA] is not always the best method. In Table 4, in three cases, the values for [HMA, RFTA] are slightly better than those for [HMA, HMA]. Moreover, in the first three lines of Tables 4 and 5, the values for [CAIA, HMA] are better than the corresponding values for [HMA, RFTA]. In Tables 6 and 7, the values for [HMA, CAIA] are always better than those for [CAIA, HMA] and, in three cases, better than the corresponding values for [HMA, HMA]. Moreover, for the last five classes of SCP1, the values for [HMA, RFTA] are better than those for [HMA, HMA]. From these observations, we can only say that one group of algorithms is better than the other group, but we cannot claim that one specific algorithm is always better than another one.

Comparing the results produced by [CAIA, HMA] and [RFTA, HMA], one could ask why the first approach belongs to the group of effective ones, whereas the second one can be weak. A possible explanation is that the perturbation method from LS_{CAIA} is much stronger than the one from LS_{RFTA} in terms of the number of slots deleted from the original schedule. Thus, in the case of [CAIA, HMA], the refinement step begins with higher energy levels in the batteries of available sensors. This allows an efficient HMA method to improve a shorter input schedule much more than in the case of a more extended input schedule but less energy in the batteries of available sensors.

Figures 3 and 4 present boxplots of makespans of schedules returned by the four good LS algorithms for the SCP1 benchmark with cov of 80% and 90%, respectively. These graphs show that all statistical parameter values (minimum, lower quartile, median, upper quartile, maximum) for [HMA, HMA] are usually better than the corresponding values for the three other algorithms.

In Figure 3, only for class 7, the values of minimum, maximum, and quartiles for [HMA, RFTA] are better than the ones for [HMA, HMA]. For class 8, the values of the minimum, the lower quartile, and the median for [HMA, RFTA] are better than the ones for [HMA, HMA], while the values of the upper quartile and the maximum are equal. For class 4, the median and the upper quartile for [HMA, RFTA] are better than the ones for [HMA, HMA], but the minimum, the lower quartile, and the maximum are equal. Thus,

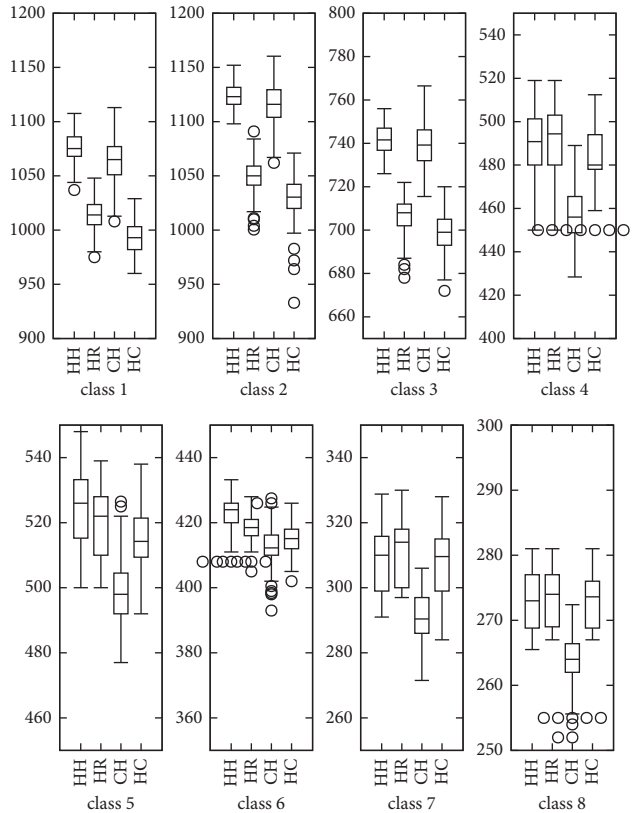

Figure 3: Boxplots of makespans of schedules (minimum, lower quartile, median, upper quartile, maximum, and outliers) returned by the LS algorithms for the SCP1 classes nos. 1, 2, 3, and 4 (top figures) and classes nos. 5, 6, 7, and 8 (bottom figures) with cov = 80%.

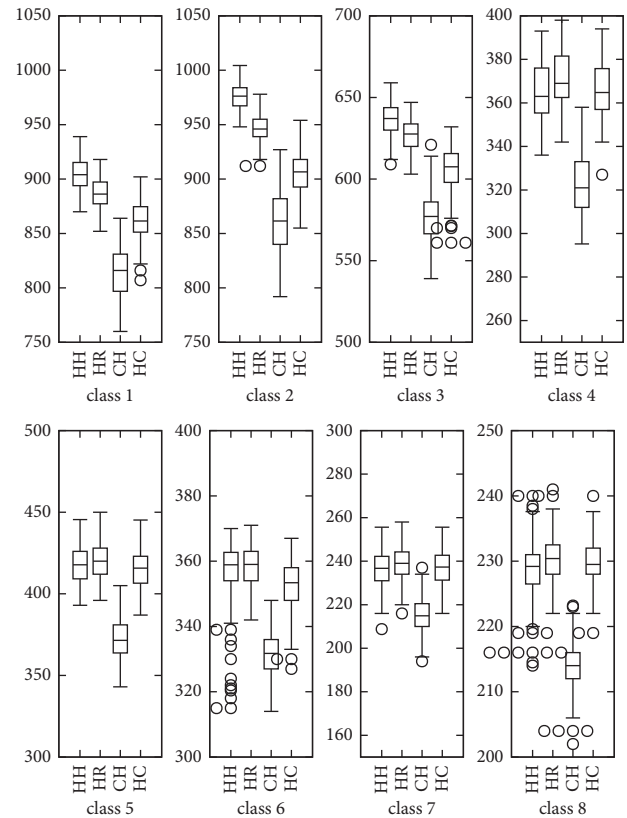

Figure 4: Boxplots of makespans of schedules (minimum, lower quartile, median, upper quartile, maximum, and outliers) returned by the LS algorithms for the SCP1 classes nos. 1, 2, 3, and 4 (top figures) and classes nos. 5, 6, 7, and 8 (bottom figures) with cov = 90%.

Table 8: Mean lengths of schedules returned by the LS algorithms for each of the four classes of the benchmark given in [7, 8] with cov = 90%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
#1	165.97	153.47	134.63	147.07	110.20	104.40	101.10	102.07	100.23
#2	337.97	297.37	284.47	272.10	219.80	200.87	191.20	193.53	190.00
#3	510.53	423.50	441.43	373.33	319.07	301.10	282.20	287.87	281.07
#4	157.00	145.03	127.60	139.17	104.63	100.97	98.63	99.53	97.63

Table 9: Mean percentage qualities of the best-found schedules returned by the LS algorithms (the top table) and mean percentage improvements of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table) for each of the four classes of the benchmark given in [7, 8] with cov = 90%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
#1	97.06	89.75	78.73	86.00	64.44	61.05	59.12	59.69	58.62
#2	98.53	86.70	82.93	79.33	64.08	58.56	55.74	56.42	55.39
#3	99.13	82.23	85.72	72.49	61.95	58.47	54.80	55.90	54.58
#4	98.13	90.65	79.75	86.98	65.40	63.10	61.65	62.21	61.02
No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
#1	71.35	58.47	39.03	51.87	13.77	7.78	4.36	5.35	3.46
#2	80.85	59.12	52.22	45.59	17.64	7.47	2.29	3.55	1.64
#3	83.94	52.57	59.03	34.48	14.98	8.47	1.66	3.71	1.25
#4	65.95	53.35	34.88	47.12	10.63	6.71	4.24	5.18	3.18

Table 10: Mean lengths of schedules returned by the LS algorithms for the benchmark given in [7, 8] with cov = 80%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
#1	193.90	166.07	168.80	162.27	141.03	141.33	138.77	140.03	138.10
#2	388.53	308.73	355.70	290.93	271.33	276.40	267.37	271.00	267.03
#3	573.10	443.87	553.23	425.07	402.53	413.93	397.50	402.23	397.10
#4	183.30	160.37	163.17	155.40	138.50	139.00	136.40	137.27	135.77

Table 11: Mean percentage qualities of the best-found schedules returned by the LS algorithms (the top table) and mean percentage improvements of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table) for each of the four classes of the benchmark given in [7, 8] with cov = 80%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
#1	95.99	82.21	83.56	80.33	69.82	69.97	68.70	69.32	68.37
#2	96.89	76.99	88.70	72.55	67.66	68.93	66.67	67.58	66.59
#3	95.68	74.10	92.36	70.96	67.20	69.10	66.36	67.15	66.29
#4	94.48	82.66	84.11	80.10	71.39	71.65	70.31	70.76	69.98
No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
#1	43.00	22.46	24.46	19.67	3.99	4.21	2.32	3.25	1.82
#2	47.13	16.90	34.68	10.16	2.73	4.65	1.23	2.60	1.10
#3	45.24	12.48	40.21	7.73	2.01	4.90	0.74	1.94	0.63
#4	37.58	20.37	22.46	16.62	3.94	4.32	2.36	3.01	1.88

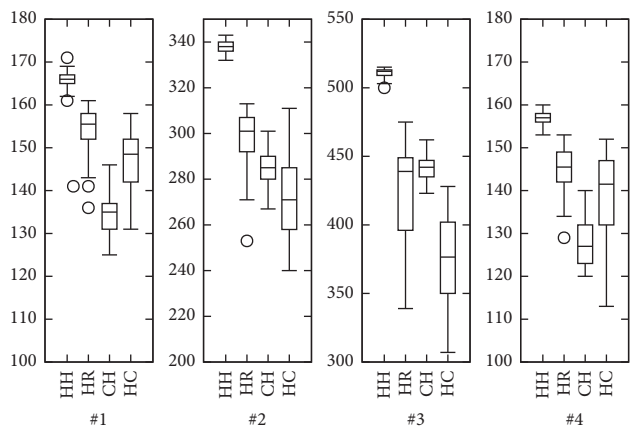

Figure 5: Boxplots of lengths of schedules (minimum, lower quartile, median, upper quartile, maximum, and outliers) returned by the LS algorithms for the benchmark given in [7, 8] with cov = 90%.

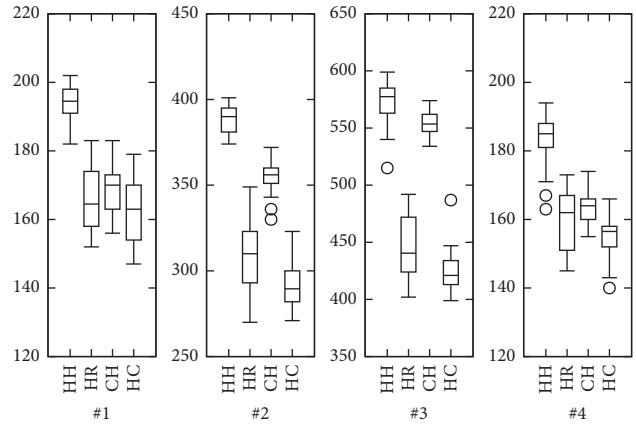

Figure 6: Boxplots of lengths of schedules (minimum, lower quartile, median, upper quartile, maximum, and outliers) returned by the LS algorithms for the benchmark given in [7, 8] with cov = 80%.

Table 12: Mean lengths of schedules returned by the LS algorithms for the benchmark given in [9] with cov = 100%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	548.27	506.38	487.45	406.88	392.13	452.05	374.50	435.50	370.63
1.2	1189.92	925.00	983.15	759.95	775.80	892.48	744.90	848.02	741.75
1.3	1756.40	1328.17	1444.80	1099.90	1125.38	1299.63	1088.63	1222.15	1085.40
1.4	2509.07	1730.05	1979.03	1487.80	1516.35	1763.38	1476.35	1651.33	1473.05
1.5	3100.28	2009.63	2495.50	1861.13	1892.33	2212.93	1854.70	2059.82	1850.70
2.1	2121.10	1711.03	1787.53	1442.90	1459.33	1655.03	1437.53	1583.55	1434.22
2.2	1984.28	1513.63	1637.90	1271.30	1282.83	1481.33	1264.70	1410.90	1259.92
2.3	1808.70	1405.95	1558.40	1196.83	1207.88	1403.97	1186.65	1328.50	1182.95
2.4	1829.63	1311.08	1520.03	1154.17	1173.42	1362.00	1143.47	1284.97	1140.05
2.5	1850.97	1317.22	1487.13	1128.25	1144.10	1331.42	1116.95	1256.55	1113.05

Table 13: Mean percentage qualities of the best-found schedules returned by the LS algorithms (the top table) and mean percentage improvements of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table) for each of the ten classes of the benchmark given in [9] with cov = 100%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	78.44	72.44	69.74	58.21	56.10	64.67	53.58	62.30	53.02
1.2	85.36	66.36	70.53	54.52	55.65	64.02	53.44	60.83	53.21
1.3	84.40	63.82	69.43	52.85	54.08	62.45	52.31	58.73	52.16
1.4	89.23	61.52	70.38	52.91	53.92	62.71	52.50	58.72	52.38
1.5	87.80	56.91	70.67	52.71	53.59	62.67	52.53	58.34	52.41
2.1	81.96	66.11	69.07	55.75	56.39	63.95	55.55	61.19	55.42
2.2	84.33	64.33	69.61	54.03	54.52	62.95	53.75	59.96	53.55
2.3	85.88	66.76	74.00	56.83	57.35	66.67	56.35	63.08	56.17
2.4	83.32	59.70	69.22	52.56	53.43	62.02	52.07	58.51	51.91
2.5	86.25	61.38	69.30	52.57	53.31	62.04	52.05	58.55	51.87
No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	51.17	40.23	34.22	12.63	8.12	24.50	3.17	19.99	2.05
1.2	62.11	26.78	33.67	3.41	5.55	21.40	1.29	15.35	0.86
1.3	62.90	23.99	33.89	1.93	4.30	20.46	0.88	13.28	0.58
1.4	71.34	18.66	35.03	1.53	3.51	20.37	0.75	12.69	0.52
1.5	68.14	9.27	35.25	0.86	2.58	19.95	0.51	11.62	0.29
2.1	48.76	20.38	25.38	0.98	2.15	16.03	0.60	10.98	0.36
2.2	58.57	21.10	30.72	1.40	2.34	18.21	0.86	12.56	0.46
2.3	54.23	19.63	32.42	1.68	2.61	19.33	0.82	12.90	0.50
2.4	61.92	16.08	34.06	1.86	3.58	20.19	0.89	13.38	0.58
2.5	67.54	19.56	34.42	2.01	3.43	20.39	0.96	13.60	0.61

Table 14: Mean lengths of schedules returned by the LS algorithms for the benchmark given in [9] with cov = 90%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	755.92	728.50	826.48	724.63	725.30	802.63	723.38	786.83	722.48
1.2	1490.42	1469.13	1664.13	1464.28	1465.50	1599.30	1463.42	1559.72	1462.22
1.3	2206.97	2140.10	2444.95	2137.32	2139.20	2325.25	2136.70	2267.32	2135.47
1.4	2976.40	2921.50	3361.13	2920.65	2921.30	3181.55	2919.97	3095.38	2918.68
1.5	3743.65	3654.90	4226.15	3651.30	3652.18	3977.00	3650.05	3860.32	3649.28
2.1	2523.25	2338.20	2584.65	2305.63	2308.55	2480.97	2303.38	2422.95	2301.88
2.2	2293.40	2230.90	2521.25	2214.82	2215.18	2399.50	2213.63	2344.30	2212.05
2.3	2253.45	2197.20	2508.68	2195.40	2197.35	2388.50	2194.78	2332.57	2193.78
2.4	2259.72	2195.90	2512.95	2192.75	2194.93	2382.68	2192.40	2327.03	2191.45
2.5	2242.60	2191.82	2508.32	2190.00	2191.20	2388.40	2188.88	2321.75	2187.60

Table 15: Mean percentage qualities of the best-found schedules returned by the LS algorithms (the top table) and mean percentage improvements of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table) for each of the ten classes of the benchmark given in [9] with cov = 90%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	78.91	76.04	86.27	75.64	75.71	83.78	75.51	82.13	75.41
1.2	80.30	79.16	89.66	78.89	78.96	86.17	78.85	84.04	78.78
1.3	81.86	79.38	90.69	79.28	79.35	86.25	79.25	84.10	79.21
1.4	80.25	78.77	90.62	78.74	78.76	85.78	78.73	83.46	78.69
1.5	81.05	79.13	91.49	79.05	79.07	86.10	79.02	83.57	79.01
2.1	72.74	67.40	74.51	66.46	66.55	71.52	66.40	69.85	66.36
2.2	78.33	76.19	86.11	75.64	75.65	81.95	75.60	80.06	75.55
2.3	78.60	76.64	87.50	76.57	76.64	83.31	76.55	81.36	76.52
2.4	77.04	74.87	85.68	74.76	74.84	81.24	74.75	79.34	74.72
2.5	78.39	76.61	87.67	76.55	76.59	83.48	76.51	81.15	76.46
No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	5.57	1.50	15.27	0.91	1.01	11.89	0.73	0.73	0.61
1.2	2.27	0.79	14.28	0.45	0.53	9.81	0.39	0.39	0.31
1.3	3.79	0.41	14.82	0.27	0.36	9.15	0.24	0.24	0.18
1.4	2.24	0.20	15.38	0.17	0.19	9.19	0.15	0.15	0.10
1.5	2.87	0.27	16.02	0.16	0.18	9.15	0.12	0.12	0.10
2.1	10.29	1.86	12.73	0.36	0.49	8.12	0.26	0.26	0.19
2.2	3.80	1.00	14.27	0.27	0.29	8.70	0.22	0.22	0.15
2.3	2.91	0.32	14.64	0.23	0.33	9.11	0.21	0.21	0.16
2.4	3.32	0.37	14.96	0.23	0.33	8.96	0.21	0.21	0.17
2.5	2.79	0.37	14.97	0.28	0.34	9.43	0.23	0.23	0.17

Table 16: Mean lengths of schedules returned by the LS algorithms for the benchmark given in [9] with cov = 80%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	973.10	969.85	1030.22	968.80	968.67	1020.33	968.08	1009.98	967.63
1.2	1969.78	1968.40	2083.75	1967.95	1967.92	2056.05	1967.60	2032.17	1966.92
1.3	2890.60	2886.07	3059.93	2885.40	2885.35	3009.22	2884.97	2971.97	2884.50
1.4	3938.82	3934.85	4197.38	3934.70	3934.40	4110.50	3934.45	4051.63	3933.53
1.5	4931.93	4930.52	5277.35	4930.23	4930.88	5149.18	4930.25	5072.93	4929.88
2.1	3062.82	3039.93	3205.32	3038.50	3039.25	3155.60	3038.45	3120.10	3037.85
2.2	2963.93	2961.72	3136.97	2960.53	2960.65	3086.78	2960.82	3050.03	2959.80
2.3	2980.10	2978.78	3161.28	2977.65	2976.97	3102.93	2977.10	3067.32	2976.32
2.4	2961.70	2959.60	3140.32	2958.55	2958.30	3086.32	2958.10	3047.95	2957.43
2.5	2968.40	2966.47	3145.07	2965.95	2965.97	3093.15	2966.03	3055.30	2965.28

Table 17: Mean percentage qualities of the best-found schedules returned by the LS algorithms (the top table) and mean percentage improvements of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table) for each of the ten classes of the benchmark given in [9] with cov = 80%.

No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	82.05	81.77	86.87	81.69	81.68	86.03	81.63	85.16	81.59
1.2	85.57	85.51	90.52	85.49	85.49	89.32	85.47	88.28	85.44
1.3	82.47	82.34	87.30	82.32	82.32	85.86	82.31	84.79	82.30
1.4	83.73	83.65	89.23	83.65	83.64	87.38	83.64	86.13	83.62
1.5	84.71	84.69	90.64	84.68	84.69	88.44	84.68	87.13	84.68
2.1	81.26	80.66	85.04	80.62	80.64	83.73	80.62	82.78	80.60
2.2	86.94	86.88	92.02	86.84	86.85	90.55	86.85	89.47	86.82
2.3	86.38	86.34	91.63	86.31	86.29	89.94	86.29	88.91	86.27
2.4	86.75	86.69	91.98	86.66	86.65	90.40	86.65	89.28	86.63
2.5	83.20	83.14	88.15	83.13	83.13	86.69	83.13	85.63	83.11
No.	HH	HR	CH	HC	RH	CR	RR	CC	RC
1.1	0.96	0.61	6.98	0.50	0.48	5.93	0.42	4.86	0.37
1.2	0.29	0.22	6.17	0.19	0.19	4.74	0.18	3.51	0.14
1.3	0.34	0.18	6.28	0.15	0.15	4.50	0.14	3.20	0.12
1.4	0.22	0.11	6.84	0.11	0.10	4.62	0.10	3.11	0.08
1.5	0.09	0.06	7.16	0.06	0.07	4.53	0.06	2.98	0.05
2.1	0.98	0.17	5.78	0.12	0.15	4.09	0.12	2.89	0.10
2.2	0.25	0.18	6.18	0.14	0.14	4.46	0.15	3.20	0.11
2.3	0.25	0.20	6.39	0.16	0.14	4.41	0.15	3.21	0.12
2.4	0.27	0.20	6.38	0.16	0.16	4.54	0.15	3.23	0.13
2.5	0.20	0.13	6.21	0.11	0.11	4.45	0.11	3.16	0.09

none of these two algorithms can produce a schedule longer than the best schedule given by the other, but [HMA, RFTA] more often returns better results. For classes 1–3, the maximum for [CAIA, HMA] is better than the maximum for [HMA, HMA], but the other boxplot parameters for [CAIA, HMA] are worse than those for [HMA, HMA]. [CAIA, HMA] has a more extensive interquartile range and distribution of the results. This is due to the properties of the perturbation method from LS_{CAIA} which is more random than the method used by LS_{HMA}. Hence sometimes, it can produce better usage of the sensors than the more systematic approach of LS_{HMA}.

In Figure 4, for classes 4–8, the values of minimum, maximum, and quartiles for [HMA, RFTA] are better than the ones for [HMA, HMA]. Moreover, for classes 4 and 8, the values of almost all parameters for [HMA, CAIA] are better than the ones for [HMA, HMA] (the only exception is the maximum in class 8). For class 8, [HMA, CAIA] has smaller interquartile range than [HMA, HMA]. For class 7, the values of the corresponding parameters for [HMA, CAIA] and [HMA, HMA] are almost equal. Thus, boxplot graphs confirm conclusions from mean values.

5.3. Performance Comparisons Using Two Other Benchmarks. To validate our findings from Section 5.2, we decided to conduct additional experiments using benchmarks provided by other authors. We selected the same two benchmarks we used for experiments with our original algorithms LS_{HMA}, LS_{CAIA}, and LS_{RFTA} [3].

5.3.1. The Benchmark Proposed by Tretyakova et al. in [7, 8]. We selected four classes of problems proposed in [7, 8]. In all cases, the monitored area is a square with a side size of 100 abstract units. In three classes from [8], there are 100 POIs while the number of sensors is 100, 200, and 300 (cases #1, #2, and #3). In the class from [7], we have 400 POIs and 100 sensors (case #4). The remaining parameters have also the same values as in [7, 8]; that is, cov = 90%, the sensing range is 20 abstract units, and $T_{batt} = 20$. Table 8 presents the mean measured lengths of schedules. Table 9 shows the mean percentage qualities of the best-found schedules returned by our LS algorithms (the top table) and mean percentage improvements of the lengths of schedules returned by the LS phase with respect to the lengths of schedules returned by the initialization phase (the bottom table) for each of the four classes of the benchmark in question, assuming cov = 90%.

This set of experiment results are similar to our earlier results with the benchmark SCP1. [HMA, HMA] always gives the most extended schedules. Algorithms [HMA, RFTA], [CAIA, HMA], and [HMA, CAIA] are worse than [HMA, HMA] but significantly better than the remaining five approaches.

Since in experiments with SCP1, we performed computations for two values of cov, 80% and 90%, we did the same for the test cases from [7, 8]. Tables 10 and 11 show the set of results similar to the ones in Tables 8 and 9 but obtained for cov = 80%.

For this set of benchmarks, lowering cov from 90% to 80% has not changed the relative performance of the three groups of our LS algorithms. However, the relative performance of the three effective algorithms has changed.

Figures 5 and 6 show boxplots of lengths of schedules returned by the four good LS algorithms for the benchmark proposed in [7, 8] with cov of 90% and 80%, respectively. In Figure 5, one can see that the minimum for [HMA, HMA] is always greater than or equal to the maximum for the other

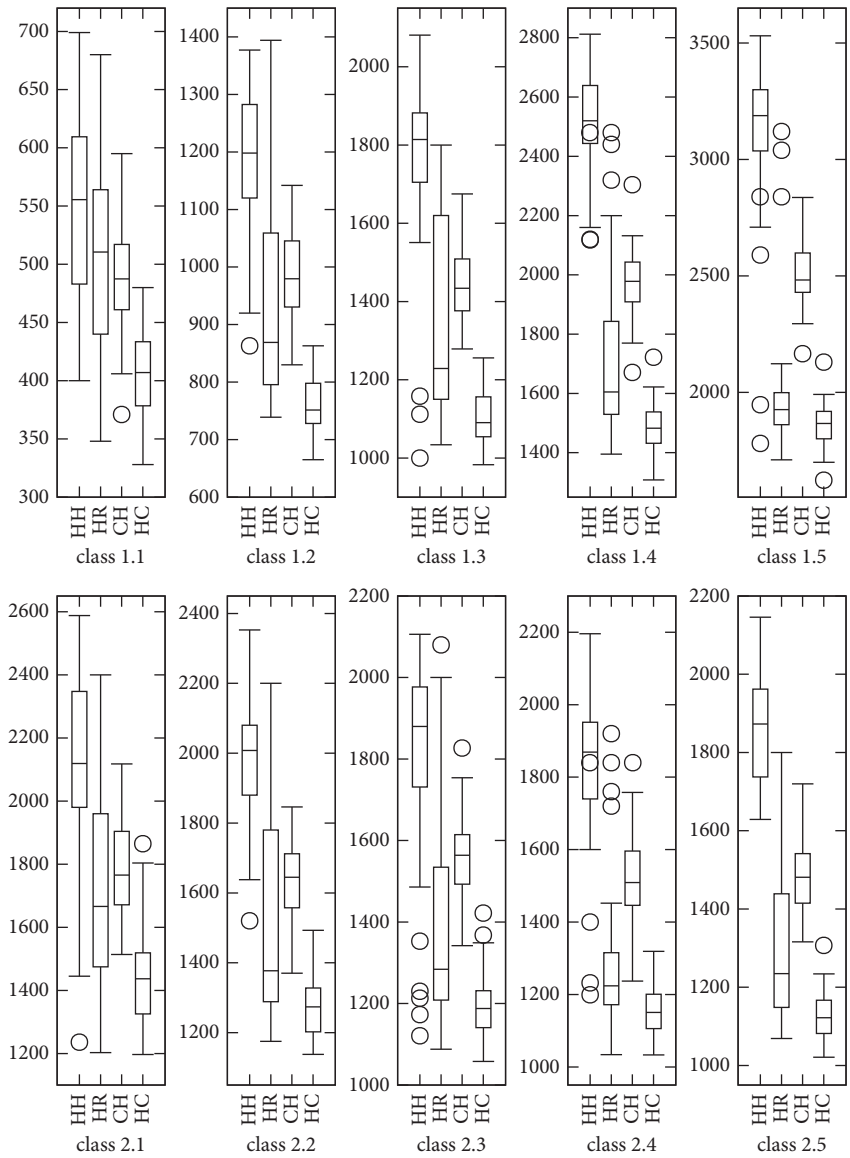

Figure 7: Boxplots of lengths of schedules (minimum, lower quartile, median, upper quartile, maximum, and outliers) returned by the LS algorithms for the benchmark given in [9] with cov = 100%.

three algorithms. Moreover, the results for [HMA, HMA] have a much smaller interquartile range than those for the other algorithms. The schedules' lengths are less dispersed, proving the higher stability of the [HMA, HMA] variant for cov = 90%. In Figure 6, the values of all parameters for [HMA, HMA] are better than those of the corresponding parameters for the other algorithms. However, for three out of four classes of the benchmark, the results for [CAIA, HMA] have the smallest interquartile range. Less dispersed schedules' lengths prove the higher stability of the [CAIA, HMA] variant for cov = 80%. Again, boxplot graphs confirm conclusions from mean values.

5.3.2. The Benchmark Proposed by Manju et al. in [9]. The authors of the minimal-heuristic approach [9] proposed the following set of benchmarks for experiments with algorithms solving MLCP. The monitored area is a square with a side size of 150 abstract units. POIs and sensors are distributed randomly over this area. In the first five test cases (1.1–1.5), we have 100 POIs while the sensors' numbers are 50, 100, 150, 200, and 250, respectively. In the next five test cases (2.1–2.5), there are 150 sensors, while the numbers of POIs are 20, 40, 60, 80, and 100. As in [9], the sensing range is 70 abstract units and cov = 100%. Requiring full coverage is reasonable because this set of benchmarks gives much more redundant coverage of POIs by sensors; that is, larger numbers of sensors cover individual POIs than in SCP1 and the benchmark from [7, 8] (see the next subsection for details). We used $T_{\text{batt}} = 40$.

Table 12 gives the results of our LS strategies for the benchmarks from [9]. The top part of Table 13 shows the mean percentage qualities of the best-found schedules returned by the LS algorithms. In the bottom part of

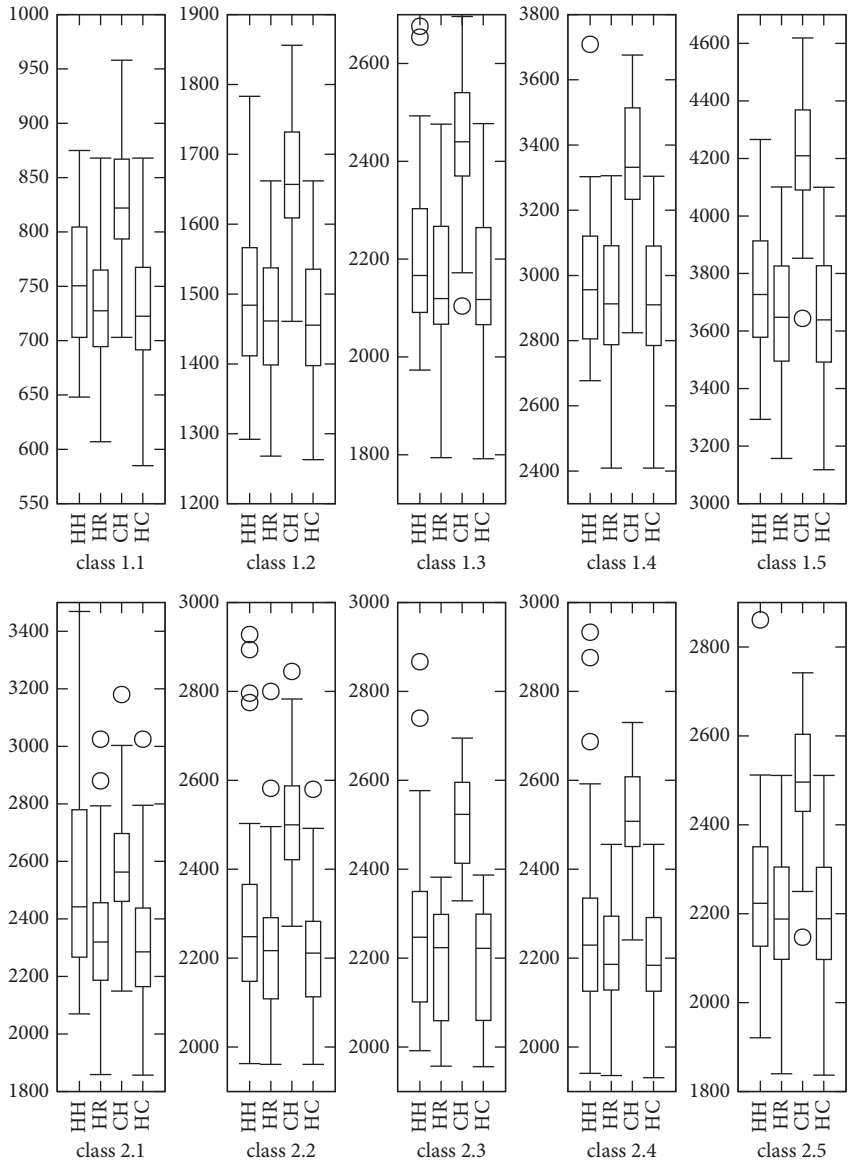

Figure 8: Boxplots of lengths of schedules (minimum, lower quartile, median, upper quartile, maximum, and outliers) returned by the LS algorithms for the benchmark given in [9] with cov = 90%.

Table 13, we present mean percentage improvements of the lengths of schedules produced by the LS phase with respect to the lengths of schedules returned by the initialization phase.

One can see from these results that [HMA, HMA] is still the winner. [CAIA, HMA] is the second best LS approach in 9 out of 10 cases. As the number of sensors grows (which means that the redundancy of coverage also increases), [CAIA, RFTA] becomes better than [HMA, RFTA].

Since in our experiments with the benchmark set SCP1 and with the benchmarks from [7, 8] we used cov equal to 80% and 90%, we also performed tests with the data sets from [9] using the same values of cov. For these test cases, the number of alternative sensor activity configurations satisfying coverage level is relatively high due to high redundancy in POI coverage. Lowering *cov* makes the number of these configurations even greater. The results for *cov* = 90% are given in Tables 14 and 15 while the results for cov = 80% are provided in Tables 16 and 17.

In this set of experiments, [CAIA, HMA] is the best LS algorithm in all cases. For cov = 90%, [HMA, HMA] is the second best in test Case 2.1. In all other test cases, for both cov equal to 80% and that equal to 90%, the second best is [CAIA, RFTA]. The third is [CAIA, CAIA]. Thus, for cov equal to 80% or 90%, the perturbation method used in CAIA always gives results better than the other two methods. Interestingly, for cov = 80%, the remaining six LS algorithms (including [HMA, HMA]) give a relative improvement of less than 1%.

Figures 7, 8, and 9 show boxplots of lengths of schedules returned by the four good LS algorithms for the benchmark proposed in [9] with cov of 100%, 90%, and 80%, respectively. In Figure 7 [HMA, HMA] always gives the largest values of

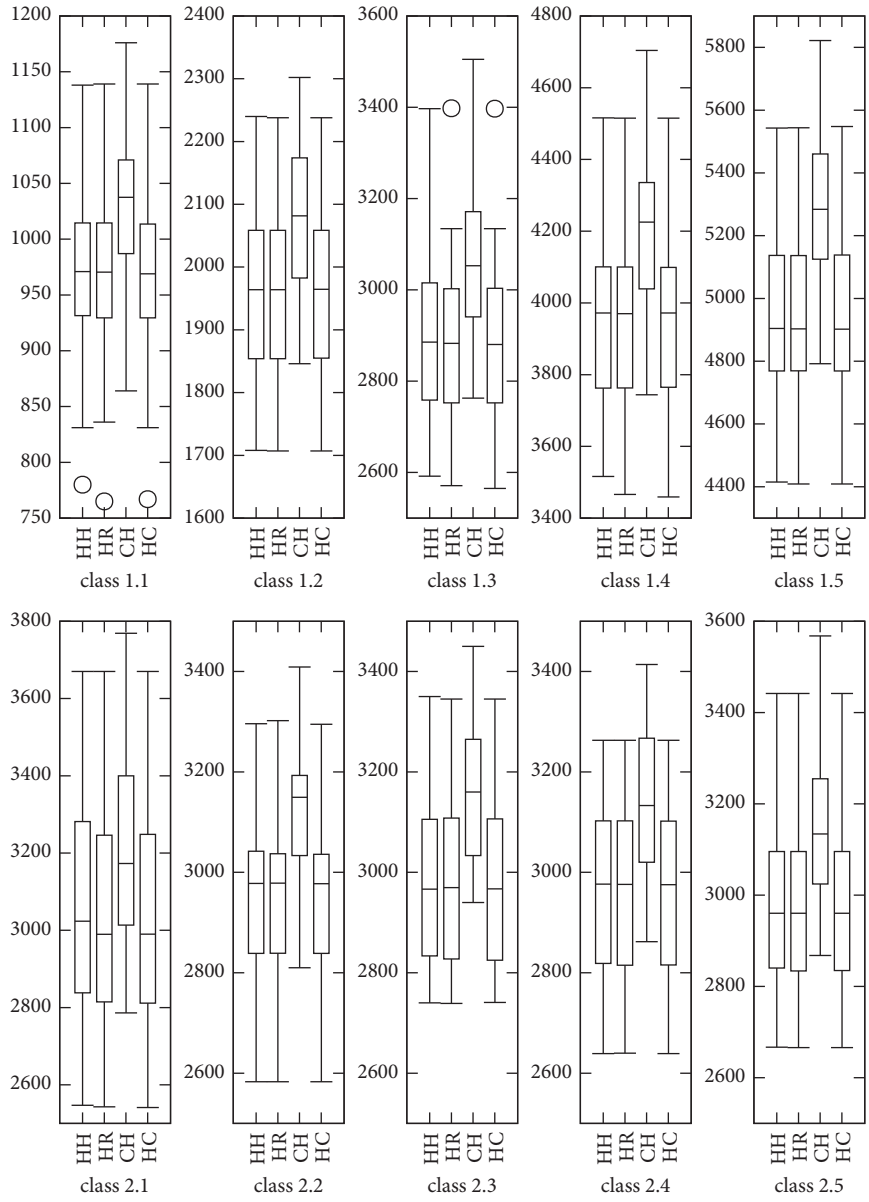

Figure 9: Boxplots of lengths of schedules (minimum, lower quartile, median, upper quartile, maximum, and outliers) returned by the LS algorithms for the benchmark given in [9] with cov = 80%.

minimum, maximum, and quartiles. [CAIA, HMA] and [HMA, CAIA] have usually smaller interquartile ranges than [HMA, HMA] and [HMA, RFTA]. The small value of the interquartile range for [HMA, CAIA] with the mean value lower than the mean values of the results returned by other algorithms means that [HMA, CAIA] never gives good results for this benchmark and cov = 100%. In Figure 8, [CAIA, HMA] not only has the highest values of all parameters but also almost always has the smallest interquartile range (only for class 1.1, [HMA, RFTA] has a smaller interquartile range). Decreasing cov from 100 to 90% changed the winner—more random perturbation method from LS_{CAIA} turned out to be better in this case. Finally, in Figure 9, [CAIA, HMA] again has the highest values of all parameters and often has the smallest interquartile range. However, in this case, interquartile ranges for all four algorithms are close to each other. One more time, boxplot graphs confirm conclusions from mean values.

5.4. What Affects Algorithms' Effectiveness?

5.4.1. The Redundancy in Coverage of POIs by Sensors in the Benchmarks. To explain differences in the relative performance of our nine LS approaches on various benchmarks, we investigated the numbers of sensors able to control particular POIs in SCP1 and the benchmarks from [7–9]. Our findings are presented in Figures 10–12.

Figures 10 and 12 show that in the test cases from SCP1 and from [7, 8], every POI can be monitored by not more than 50 sensors and, in some of these cases, even by not more than 20

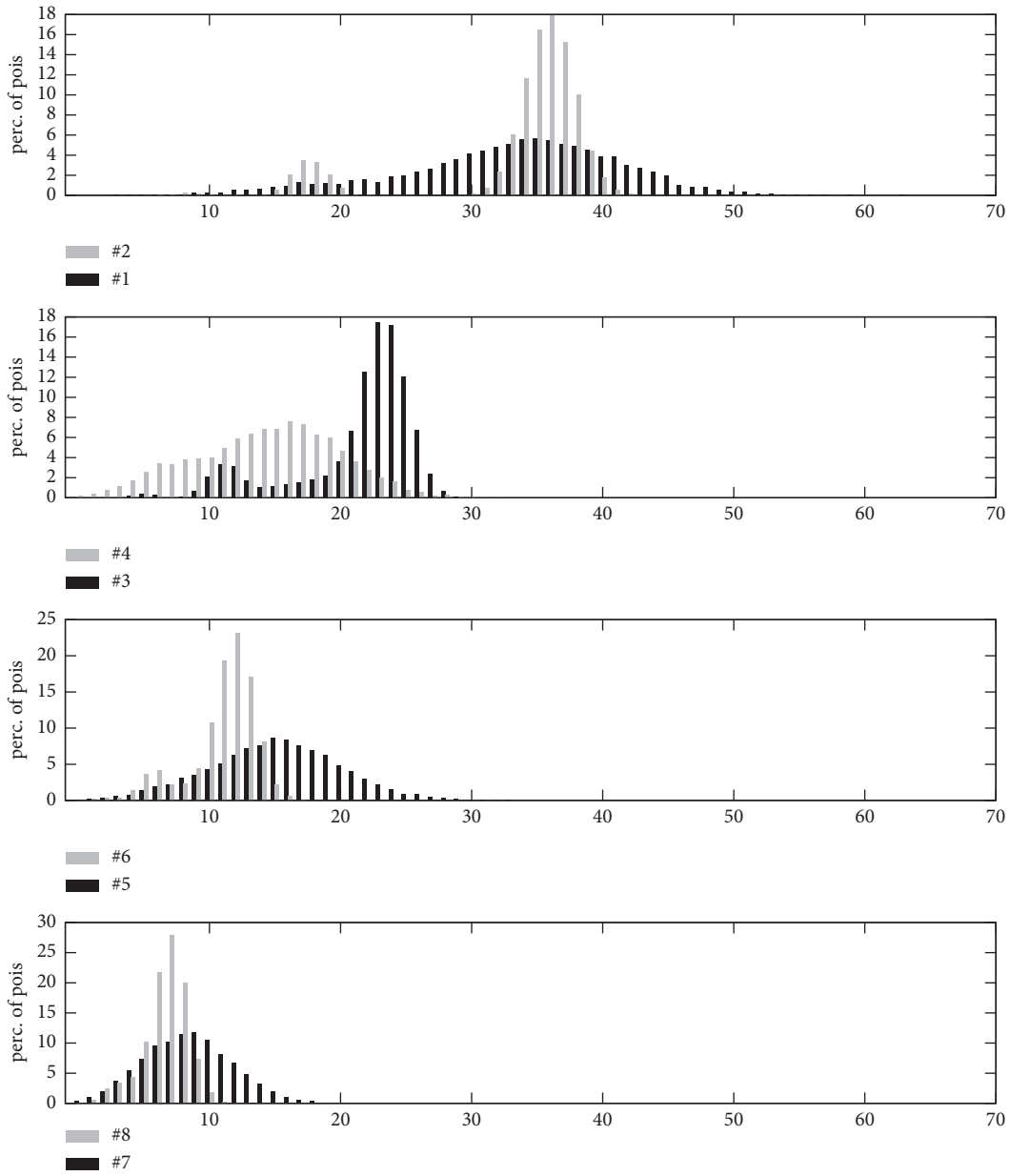

Figure 10: Percentage numbers of POIs covered by different numbers of sensors for SCP1.

sensors. On the other hand, Figure 11 shows that in the test cases from [9], many POIs can be monitored by more than 100 sensors. This high monitoring redundancy makes finding a cover for POIs much more manageable, even for cov = 100%.

We conclude from the above considerations that the perturbation method used in LS_{CAIA} works better than the other two methods when there are many possibilities of forming a required cover for a given set of POIs. When the number of such options decreases due to a higher value of cov or a smaller number of sensors, the perturbation method from LS_{HMA} becomes better. The advantage of the perturbation method from LS_{CAIA} over the one from LS_{HMA} when there is much redundancy in coverage of POIs is probably that the former removes more slots from the original schedule than the latter. When the number of similar legal covers becomes large, there are many possibilities of creating a valid schedule. Removing more slots and restoring more power to sensor batteries gives the refinement method more space to show its effectiveness and improve the original schedule. On the other hand, when the redundancy in coverage of POIs decreases, a more systematic approach of HMA works better.

Let us also notice that too much redundancy in coverage of POIs by sensors is not desirable in real life due to increased costs. Of course, the ultimate redundancy level depends mainly on a specific application.

5.4.2. Strengths and Weaknesses of the Perturbation Operators. The good performance of the perturbation

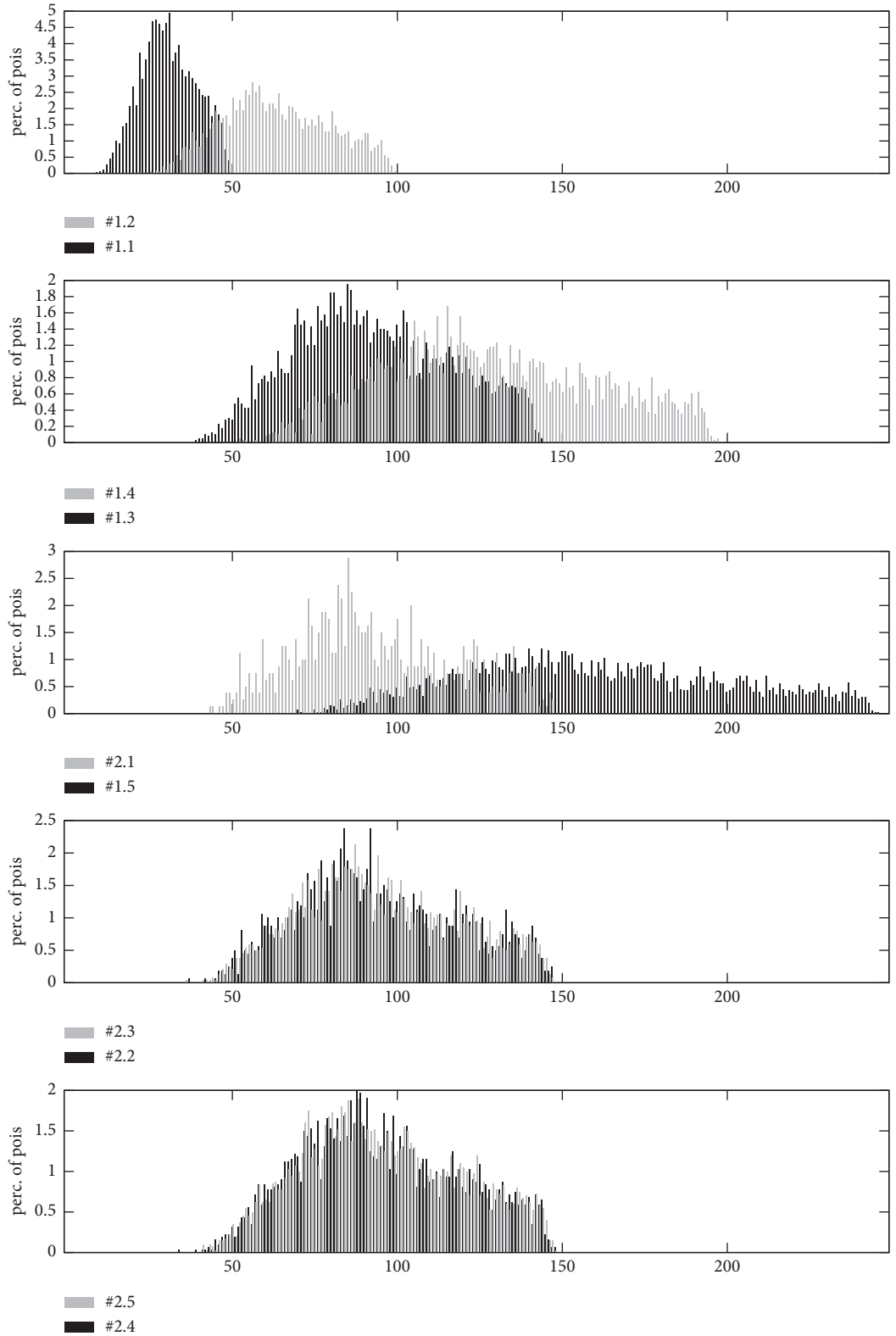

Figure 11: Percentage numbers of POIs covered by different numbers of sensors for the benchmark given in [9].

method from LS_{HMA} comes from the fact that this method does not remove a single slot from the original schedule. Instead, the method only changes sets of active sensors in selected slots. Consequently, the set of sensors with non-empty batteries changes too. It opens up new possibilities for producing one additional slot in the refinement step, which is sufficient to obtain a more extended schedule. On the other hand, the perturbation methods from LS_{RFTA} and

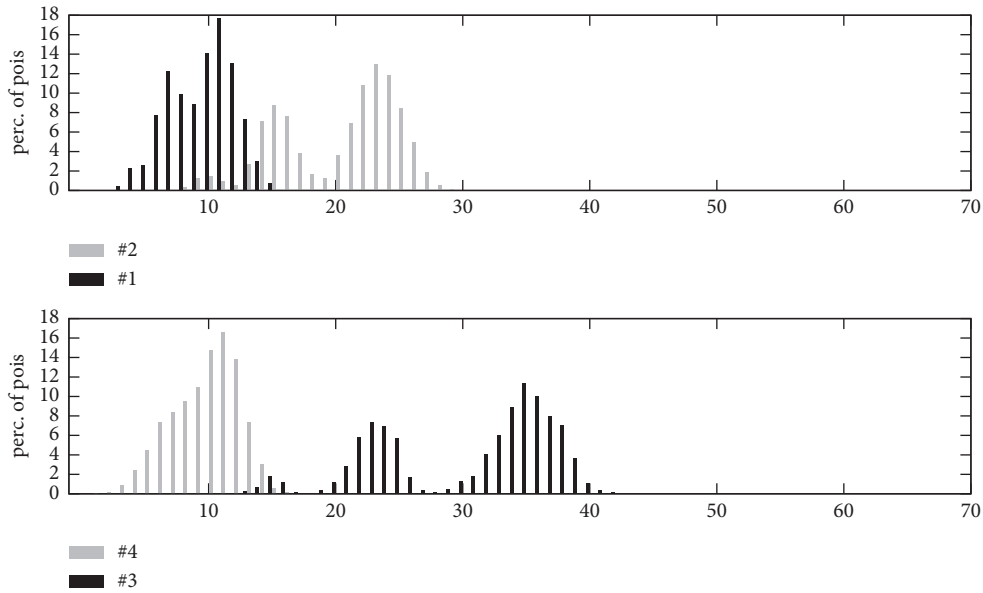

Figure 12: Percentage numbers of POIs covered by different numbers of sensors for the benchmark given in [7, 8].

LS_{CAIA} delete slots from the initial schedule. It means that during the refinement step, the minimum goal is to create at least as many new slots as were removed during the perturbation. When we succeed, we can go further and try to generate more slots to make the new schedule longer. It can happen, nevertheless, that we are not able to get as many additional slots as we removed. Despite the refinement step execution, we get a shorter schedule. Slots or sensors in slots are selected for removal randomly, so our success depends on luck. As we mentioned above, when there is much redundancy in coverage of POIs by sensors and we remove many slots from the original schedule, the chances of removing the right slots and eventually getting a more extended schedule grow.

6. Conclusions

In this paper, we proposed 27 local search algorithms solving MLCP and studied the relative performance of 9 of them. The starting point in this research was three LS algorithms presented in our earlier papers: LS_{HMA}, LS_{CAIA}, and LS_{RFTA}.

The local search approach consists of two major steps: generating an initial solution to the problem and looking for its neighbor. The second step can be divided into two substeps executed iteratively: perturbation of the original solution and refinement of the perturbation's result. This way, we have three problem-specific steps in our LS algorithms for MLCP. By swapping these three steps between three basic algorithms, we can get 27 different versions of LS solving MLCP.

Our research studied the relative performance of just nine of these versions—the initial problem solution was always generated using CAIA. For experimental research, we used the benchmark SCP1 (proposed in our earlier papers) and the benchmarks proposed in [7–9].

We computed the mean lengths of schedules returned by the LS algorithms, mean percentage qualities of the best-found schedules returned by the LS algorithms, and mean percentage improvements of the lengths of schedules produced by the LS phase with respect to the lengths of schedules returned by the initialization phase. Moreover, we analyzed boxplots of lengths of schedules returned by the four best LS algorithms.

The results of our experiments show that for the SCP1 benchmark and the benchmarks from [7, 8], the best pair of perturbation and refinement methods is usually the one used in LS_{HMA}, i.e., [HMA, HMA]. Approaches [HMA, RFTA], [CAIA, HMA], and [HMA, CAIA] are also effective, while the remaining combinations give much worse results. However, experiments with the benchmarks from [9] gave different results. When we assume cov = 100%, [HMA, HMA] is still the best approach, but for cov equal to 80% or 90%, the perturbation method used in LS_{CAIA} gives better results. We conclude that the more redundant the coverage of POIs by sensors and the lower the value of cov, the more effective the perturbation method from LS_{CAIA}. When this redundancy is lower or cov is higher, the perturbation method used in LS_{HMA} becomes better. One could ask about a threshold value of cov for which the perturbation method from LS_{CAIA} becomes better than the method from LS_{HMA}. The results show that this value is problem-dependent. It depends on the redundancy in coverage of POIs by sensors. However, no rule defining this threshold value can be determined based on the presented experiments. It can be figured out only by conducting experiments with a given class of problems.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

References

[1] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*, Wiley, Hoboken NY, USA, 2010.

[2] H. Yetgin, T. K. Cheung, M. El-Hajjar, and L. Hanzo, "A survey of network lifetime maximization techniques in wireless sensor networks," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 2, pp. 828–854, 2017.

[3] A. Mikitiuk and K. Trojanowski, "Maximization of the sensor network lifetime by activity schedule heuristic optimization," *Ad Hoc Networks*, vol. 96, Article ID 101994, 2020.

[4] K. Trojanowski, A. Mikitiuk, F. Guinand, and M. Wypych, "Heuristic optimization of a sensor network lifetime under coverage constraint," in *Proceedings of the Computational Collective Intelligence: 9th International Conference*, Springer International Publishing, Nicosia, Cyprus, 2017.

[5] K. Trojanowski, A. Mikitiuk, and M. Kowalczyk, "Sensor network coverage problem: a hypergraph model approach," in *Proceedings of the Computational Collective Intelligence: 9th International Conference*, Springer International Publishing, Nicosia, Cyprus, 2017.

[6] K. Trojanowski, A. Mikitiuk, and K. J. M. Napiorkowski, "Application of local search with perturbation inspired by cellular automata for heuristic optimization of sensor network coverage problem," in *Proceedings of the Parallel Processing and Applied Mathematics*, Springer International Publishing, Berlin, Germany, 2018.

[7] A. Tretyakova and F. Seredynski, "Simulated annealing application to maximum lifetime coverage problem in wireless sensor networks," *Global Conference on Artificial Intelligence GCAI*, vol. 36, pp. 296–311, 2015.

[8] A. Tretyakova, F. Seredynski, and P. Bouvry, "Graph cellular automata approach to the maximum lifetime coverage problem in wireless sensor networks," *Simulation*, vol. 92, no. 2, pp. 153–164, 2016.

[9] Manju, D. Singh, S. Chand, and B. Kumar, "Target coverage heuristics in wireless sensor networks," *Advanced Computing and Communication Technologies Proceedings of the 10th ICACCT*, vol. 562, pp. 265–273, 2016.

[10] S. Slijepcevic and M. Potkonjak, "Power efficient organization of wireless sensor networks," in *Proceedings of the IEEE International Conference on Communications*, IEEE, Helsinki, Finland, 2001.

[11] Z Abrams, A. Goel, and S. Plotkin, "Set k-cover algorithms for energy efficient monitoring in wireless sensor networks," in *Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, ACM Press, New York, NY, USA, 2004.

[12] M. Cardei and D.-Z. Du, "Improving wireless sensor network lifetime through power aware organization," *Wireless Networks*, vol. 11, no. 3, pp. 333–340, 2005.

[13] M. Cardei, M. T. Thai, Y. Li, and W. Wu, "Energy-efficient target coverage in wireless sensor networks," in *Proceedings of the INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, Miami, FL, USA, 2005.

[14] C.-C. Lai, C.-K. Ting, and R.-S. Ko, "An effective genetic algorithm to improve wireless sensor network lifetime for large-scale surveillance applications," in *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE, Singapore, 2007.

[15] X.-M. Hu, J. Zhang, Y. Yu et al., "Hybrid genetic algorithm using a forward encoding scheme for lifetime maximization of wireless sensor networks," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 766–781, 2010.

[16] N. Ahn and S. Park, "A new mathematical formulation and a heuristic for the maximum disjoint set covers problem to improve the lifetime of the wireless sensor network," *Ad Hoc and Sensor Wireless Networks*, vol. 13, no. 3-4, 2011.

[17] J. M. Gil and Y. H. Han, "A target coverage scheduling scheme based on genetic algorithms in directional sensor networks," *Sensors*, vol. 11, no. 2, pp. 1888–1906, 2011.

[18] R. Cerulli, R. De Donato, and A. Raiconi, "Exact and heuristic methods to maximize network lifetime in wireless sensor networks with adjustable sensing ranges," *European Journal of Operational Research*, vol. 220, no. 1, pp. 58–66, 2012.

[19] Y. Lin, J. Zhang, H. S.-H. Chung, W. H. Ip, Y. Li, and Y. H. Shi, "An ant colony optimization approach for maximizing the lifetime of heterogeneous wireless sensor networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 3, pp. 408–420, 2012.

[20] M. E. Keskin, İ. K. Altınel, N. Aras, and C. Ersoy, "Wireless sensor network lifetime maximization by optimal sensor deployment, activity scheduling, data routing and sink mobility," *Ad Hoc Networks*, vol. 17, no. 18–36, 2014.

[21] L. Wang, W. Wu, J. Qi, and Z. Jia, "Wireless sensor network coverage optimization based on whale group algorithm," *Computer Science and Information Systems*, vol. 15, no. 3, pp. 569–583, 2018.

[22] S. Balaji, M. Anitha, D. Rekha, and D. Arivudainambi, "Energy efficient target coverage for a wireless sensor network," *Measurement*, vol. 165, Article ID 108167, 2020.

[23] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky, "Power efficient monitoring management in sensor networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference IEEE Cat. No.04TH8733*, IEEE, Atlanta, GA, USA, 2004.

[24] D. Zorbas, D. Glynos, P. Kotzanikolaou, and C. Douligeris, "Solving coverage problems in wireless sensor networks using cover sets," *Ad Hoc Networks*, vol. 8, no. 4, pp. 400–415, 2010.

[25] K. Deschinkel, "A column generation based heuristic for maximum lifetime coverage in wireless sensor networks," in *Proceedings of the SENSORCOMM 2011: The Fifth International Conference on Sensor Technologies and Applications*, Curran Associates, Inc, Red Hook, NY, USA, 2011.

[26] Manju and A. K Pujari, "High-energy-first (HEF) heuristic for energy-efficient target coverage problem," *International Journal of Ad hoc, Sensor & Ubiquitous Computing*, vol. 2, no. 1, pp. 45–58, 2011.

[27] H. Mohamadi, A. S. Ismail, and S. Salleh, "Solving target coverage problem using cover sets in wireless sensor networks based on learning automata," *Wireless Personal Communications*, vol. 75, no. 1, pp. 447–463, 2013.

[28] A. Tretyakova and F. Seredynski, "Application of evolutionary algorithms to maximum lifetime coverage problem in wireless sensor networks," in *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, Workshops*, IEEE, Cambridge, MA, USA, 2013.

[29] F. Castaño, A. Rossi, M. Sevaux, and N. Velasco, "A column generation approach to extend lifetime in wireless sensor networks with coverage and connectivity constraints,"

Computers & Operations Research, vol. 52, pp. 220–230, dec 2014.

[30] Y. E. E. Ahmed, K. H. Adjallah, and S. F. Babikier, "Non disjoint set covers approach for wireless sensor networks lifetime optimization," in *Proceedings of the International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, IEEE, Offenburg, Germany, 2016.

[31] Y. E. E. Ahmed, K. H. Adjallah, R. Stock, and S. F. Babikier, "Wireless sensor network lifespan optimization with simple, rotated, order and modified partially matched crossover genetic algorithms," *IFAC-PapersOnLine*, vol. 49, no. 25, pp. 182–187, 2016.

[32] J. Roselin, P. Latha, and S. Benitta, "Maximizing the wireless sensor networks lifetime through energy efficient connected coverage," *Ad Hoc Networks*, vol. 62, no. 1–10, pp. 1–10, 2017.

[33] A. Tretyakova, F. Seredynski, and F. Guinand, "Heuristic and meta-heuristic approaches for energy-efficient coverage-preserving protocols in wireless sensor networks," in *Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks - Q2SWinet*, ACM Press, New York, NY, USA, 2017.

[34] Y. E. E. Ahmed, K. H. Adjallah, R. Stock, I. Kacem, and S. F. Babiker, "NDSC based methods for maximizing the lifespan of randomly deployed wireless sensor networks for infrastructures monitoring," *Computers & Industrial Engineering*, vol. 115, no. 17–25, 2018.

[35] M. Cardei, "Coverage problems in sensor networks," Springer, Berlin, Germany, 2013.

[36] B. Wang, *Coverage Control In Sensor Networks. Computer Communications and Networks*, Springer, Berlin, Germany, 2010.

[37] Y. E. E. Ahmed, *Modeling, Scheduling and Optimization of Wireless Sensor Networks Lifetime*, Université de Lorraine, Nancy, France, 2016.

[38] X. Shen, "Evenness evaluation in ad-hoc sensor networks," in *Proceedings of the First International Conference on Networking and Distributed Computing*, IEEE, Hangzhou, China, 2010.