Heuristic Optimization of a Sensor Network Lifetime Under Coverage Constraint

Krzysztof Trojanowski[1], Artur Mikitiuk[1(✉)], Frédéric Guinand[1,2], and Michał Wypych[1]

[1] Cardinal Stefan Wyszyński University in Warsaw, Warsaw, Poland
{k.trojanowski,a.mikitiuk}@uksw.edu.pl
[2] Normandy University of Le Havre, Le Havre, France
frederic.guinand@univ-lehavre.fr

Abstract. Control of a set of sensors disseminated in the environment to monitor activity is a subject of the presented research. Due to redundancy in the areas covered by sensor monitoring ranges a satisfying level of coverage can be obtained even if not all the sensors are on. Sleeping sensors save their energy, thus, one can propose schedules defining activity for each of sensors over time which offer a satisfying level of coverage for a period of time longer than a lifetime of a single sensor. A new heuristic algorithm is proposed which searches for such schedules maximizing the lifetime of the sensor network under a coverage constraint. The algorithm is experimentally tested on a set of test cases and effectiveness of its components is presented and statistically verified.

1 Introduction

Different applications of sensor networks are a subject of growing interest. In our research we focus on one of them, that is, the problem of effective monitoring of activity in an environment by a set of sensors disseminated randomly over it. We define this activity with use of a set of points of interest (POIs) located in a selected area. The sensor network is regarded as operational when it is able to monitor sufficient number of POIs at a time, that is, satisfies a coverage constraint. In a considered model of the problem, we assume that a monitoring range of sensors as well as their battery capacity are limited. Fortunately, due to the high number of sensors, an operational network does not necessary have them all in active state all the time. Thus, the solution of the problem we want to address is maximizing the lifetime of an operational sensor network. Precisely, we search for the longest schedule defining sensor states over time under the coverage constraint.

For generation of sensor activity schedules we propose two new algorithms. The first one builds a schedule starting from the empty one and then constructs its subsequent components in an iterative manner. A method of a single component construction is not deterministic, however, it guarantees that the proposed settings make the sensor network operational. When another component of a schedule cannot be created, the first algorithm finishes. The outcome of this

N.T. Nguyen et al. (Eds.): ICCCI 2017, Part I, LNAI 10448, pp. 422–432, 2017.
DOI: 10.1007/978-3-319-67074-4_41

algorithm is an input for a local search algorithm which goal is to improve the obtained initial schedule. The main novelty is represented by a perturbation operator which generates neighbour schedules. Effectiveness of the proposed approach is experimentally verified on a set of newly proposed test cases.

The paper consists of six sections. Section 2 gives definition of the solved problem and all the necessary constraints. The proposed algorithms are presented in Sect. 3. A benchmark is defined in Sect. 4 and the results of experiments are discussed in Sect. 5. Section 6 concludes the paper.

2 Maximum Lifetime Coverage Problem (MLCP)

In [1] authors define a sensor coverage problem, where a homogeneous sensor network $\mathbf{S}$ comprised of $N_{\mathbf{S}}$ immobile sensors is randomly deployed over an area to cover (monitor) $N_{\mathbf{P}}$ points of interest. Each sensor has a sensing range r_{sens} and starts off its service with a fully loaded battery, that is, an initial energy E_{sens}. We assume a sensor node consumes a unit of energy per time unit for its activity. It is assumed that time is discrete, thus, the lifetime of a sensor T_{batt} is the maximal number of consecutive, or not, time steps when the sensor can be on. Once the sensors are deployed, they schedule their activity.

An active sensor monitors all POIs located within its sensing range. On the other hand, according to its position, a POI can be monitored by several active sensors during the same time period. A full coverage of a set of POIs can be obtained only if all of them lie in a range of a sensor. However, for effective monitoring full coverage of POIs is not necessary. A satisfying coverage level *cov* represents the percentage level of the number of POIs being monitored. Precisely, the satisfying coverage level is obtained if the number of monitored POIs fits in the range $[cov, cov + \delta]$, where δ represents a tolerance factor.

A schedule consisting of sensor controls giving the satisfying coverage level is a solution to this problem. During one time step a sensor can be either in a *working* mode when it uses one unit of its battery capacity, or in a *sleeping* state when the energy consumption is negligible. A schedule H is a matrix of 0 s and 1 s representing states of sensors *off* and *on* in the time slots. In the presented research communication between sensors is not a part of a solved problem, thus the energy consumption necessary for communication is not present in the model. Moreover, it is assumed that communication graph always provides connectivity independently from the landform and localization of sensors even if some sensors are off.

The optimization aim is to find a schedule for the sensor network activity which gives the longest time period of uninterrupted monitoring of the given set of POIs. This is called the Maximum Lifetime Coverage Problem. The value of a schedule equals the length of the longest sequence of slots during which the satisfying coverage requirement is met.

In spite of large number of publications devoted to theory and applications of sensor networks, publications concerning MLCP are not so numerous. One can mention, e.g., sensor control methods [3] or schedule optimization approaches based on evolutionary algorithms [4], simulated annealing [5], or graph cellular automata [6].

3 Search Algorithm

A method of searching for an effective schedule of sensor activity is presented in Algorithm 1. After the first schedule is created by *generateSchedule* (see Algorithm 2), the algorithm iteratively tries to improve it by the problem specific neighbour operator *generateNeighbour* (see Algorithm 4). When a newly found schedule is longer than the current one, the new one takes place of the current and the process continues. Both procedures never generate unfeasible solutions like, for example, solutions including time slots with insufficient coverage of POIs, or impossible to execute due to battery load limitations. Therefore, the search process can be interrupted in any moment and the current schedule is ready to implement.

Algorithm 1. Main Algorithm

Require: $\mathbf{S}, T_{\text{batt}}, cov, \delta$;
Ensure: schedule H
1: $H \leftarrow generateSchedule(\mathbf{S}, T_{\text{batt}}, cov, \delta)$ ▷ *initial schedule*
2: **repeat**
3: $H' \leftarrow generateNeighbour(H)$
4: **if** $F(H') > F(H)$ **then** ▷ *if a new schedule has more slots ...*
5: $H \leftarrow H'$ ▷ *... the current schedule is replaced by the new one*
6: **until** termination condition met
7: **return** H

3.1 Generation of the Initial Schedule

We create an initial schedule by generation of time slots one by one as presented in Algorithm 2. First, all battery levels in the set of all sensors $\mathbf{S}$ are set to max value and a max effective coverage $cov_{S(max)}$ of the set of active sensors obtained from the *filter* procedure is calculated (lines 2–5). An outcome of $filter(\mathbf{S}, t)$ (used also in Algorithm 4) is a subset of sensors from a set $\mathbf{S}$ which still have a remaining battery lifetime in the time step t and are able to monitor at least one POI. In line 5 a procedure $covPoi(\cdot)$ is called, which returns the percentage of POIs covered by a given set of sensors. Then, the main loop starts. First, a single time slot for sensor activities is generated by *generateSingleTimeSlot* taking into account requested minimum coverage level cov, the tolerance δ and a current set of active sensors. The slot is added to the schedule and the sensor battery levels $batt(S)$ are updated respectively to the sensor activity setting proposed in the slot (lines 7–9). Next, the set of active sensors is updated and the max effective coverage $cov_{S(max)}$ is reevaluated. The loop stops when the set of remaining active sensors is not able to cover a sufficient number of POIs even if all of them are on, that is, $cov > cov_{S(max)}$.

The *generateSingleTimeSlot* procedure is presented in Algorithm 3. The procedure consists of two main phases: first, splits the set of sensors roughly

Algorithm 2. *generateSchedule*

1: **procedure** $generateSchedule(\mathbf{S}, T_{\text{batt}}, cov, \delta)$
2: $H \leftarrow \emptyset$; $t \leftarrow 0$ ▷ *initialize a schedule to empty and a time counter to 0*
3: **for all** $S \in \mathbf{S}$ **do** $batt(S) \leftarrow T_{\text{batt}}$ ▷ *sensor batteries level initialization*
4: $\mathbf{S}_{\text{work}} \leftarrow filter(\mathbf{S}, t)$ ▷ *select subset of active sensors having POIs in range*
5: $cov_{S(max)} \leftarrow covPoi(\mathbf{S}_{\text{work}})$ ▷ *evaluate coverage when all active sensors are on*
6: **repeat**
7: $H^t \leftarrow generateSingleTimeSlot(\mathbf{S}_{\text{work}}, cov, \delta)$
8: **for all** $S \in \mathbf{S} \mid H_S^t = 1$ **do**
9: $batt(S) \leftarrow batt(S) - 1$ ▷ *sensor batteries level update*
10: $t \leftarrow t + 1$ ▷ *time counter update*
11: $\mathbf{S}_{\text{work}} \leftarrow filter(\mathbf{S}, t)$ ▷ $\mathbf{S}_{\text{work}}$ *update*
12: $cov_{S(max)} \leftarrow covPoi(\mathbf{S}_{\text{work}})$ ▷ $cov_{S(max)}$ *update*
13: **until** $cov > cov_{S(max)}$
14: **return** H

into two subsets (lines 3–7), and next, fine tunes collections of sensors in these subsets (lines 9–17).

In the first phase an important role plays $selectRand(\mathbf{S}, n)$ which selects a subset of sensors from a set $\mathbf{S}$. For each of the sensors in $\mathbf{S}$ a newly generated independent random value r from $U[0, 1]$ is compared with a threshold value n, and in the case $r < n$ the sensor becomes selected. The selected sensors are removed from the set $\mathbf{S}$ and returned as an outcome of the procedure. The number of $n \times 100\%$ sensors are returned on average. In the beginning, all sensors are off, and the *selectRand* procedure is called to set statistically 50% of them to on (line 5). Then the coverage level of activated sensors is compared with the requested level *cov*. When the coverage level of activated sensors does not fit the range $[cov, cov + \delta]$, the set is extended or decreased with use of *selectRand* until the coverage fits the range.

If we had a single sensor covering more than δ% POIs, the loop in lines 3–7 could run forever turning on this sensor in one iteration (when the coverage level is below *cov*) and turning it off in next iteration (when the coverage level is above $cov + \delta$). However, this does not seem likely in real life situations and it never happens in the selected benchmark described in the next Section.

In the second phase we try to fine-tune the collection of active sensors keeping the coverage in the range $[cov, cov + \delta]$ and decreasing it as close to the lower bound of this range as possible. To obtain this, small numbers of sensors from the set of active ones are randomly selected with procedure *select* and set to off as long as the coverage of the other active sensors satisfies requirements.

3.2 Iterative Improvement of the Schedule

A schedule obtained from the *generateSchedule* procedure undergoes iterative improvement in the main loop of the algorithm (Algorithm 1, lines 2–5) where the *generateNeighbour* procedure presented in Algorithm 4 is called. In this

Algorithm 3. *generateSingleTimeSlot*

procedure $generateSingleTimeSlot(\mathbf{S}_{\text{work}}, cov, \delta)$
  $\mathbf{S}_{\text{on}} \leftarrow \emptyset$; $\mathbf{S}_{\text{off}} \leftarrow \mathbf{S}_{\text{work}}$ ▷ *initialization of* $\mathbf{S}_{\text{on}}$ *and* $\mathbf{S}_{\text{off}}$
  while $\neg(cov \leq covPoi(\mathbf{S}_{\text{on}}) \leq cov + \delta)$ **do**
    if $covPoi(\mathbf{S}_{\text{on}}) < cov$ **then**
      $\mathbf{S}_{\text{on}} \leftarrow \mathbf{S}_{\text{on}} \cup selectRand(\mathbf{S}_{\text{off}}, 0.5)$ ▷ *selection and transfer of sensors*
    else if $covPoi(\mathbf{S}_{\text{on}}) > cov + \delta$ **then**
      $\mathbf{S}_{\text{off}} \leftarrow \mathbf{S}_{\text{off}} \cup selectRand(\mathbf{S}_{\text{on}}, 0.5)$ ▷ *selection and transfer of sensors*
  $n \leftarrow 1$; $\mathbf{S}_{\text{tOn}} \leftarrow \mathbf{S}_{\text{on}}$ ▷ *initialization of* n *and* $\mathbf{S}_{\text{tOn}}$
  while $(cov \leq covPoi(\mathbf{S}_{\text{on}}) \leq cov + \delta) \wedge (n \geq 1)$ **do**
    $\mathbf{S}_{\text{tOff}} \leftarrow \mathbf{S}_{\text{off}} \cup select(\mathbf{S}_{\text{tOn}}, n)$ ▷ *selection and transfer of* n *sensors*
    if $cov \leq covPoi(\mathbf{S}_{\text{tOn}}) \leq covPoi(\mathbf{S}_{\text{on}})$ **then**
      $\mathbf{S}_{\text{on}} \leftarrow \mathbf{S}_{\text{tOn}}$ ▷ *update* $\mathbf{S}_{\text{on}}$
      $\mathbf{S}_{\text{off}} \leftarrow \mathbf{S}_{\text{tOff}}$ ▷ *update* $\mathbf{S}_{\text{off}}$
      $n \leftarrow n + 1$
    else
      $\mathbf{S}_{\text{tOn}} \leftarrow \mathbf{S}_{\text{on}}$ ▷ *restore* $\mathbf{S}_{\text{tOn}}$
      $n \leftarrow \frac{n}{2}$
  for all $S \in \mathbf{S}_{\text{on}}$ **do** $H_S^t \leftarrow 1$
  for all $S \in \mathbf{S}_{\text{off}}$ **do** $H_S^t \leftarrow 0$
  return H^t ▷ *return single time slot for current time* t

Algorithm 4. *generateNeighbour*

procedure $generateNeighbour(\mathbf{S}, \mathbf{H}, \mathbf{T}_{\max}, cov, n, \delta)$
  for $j \leftarrow 1, n$ **do** ▷ *remove* n *time slots from a schedule*
    $k \leftarrow rand(0, T_{\max})$ ▷ *select a slot to remove*
    for all $S \in \mathbf{S} \mid H_S^k = 1$ **do**
      $batt(S) \leftarrow batt(S) + 1$ ▷ *sensor batteries level update*
    $H \leftarrow H - H^k$; $T_{\max} \leftarrow T_{\max} - 1$ ▷ *update of a schedule and its length*
  $\mathbf{S}_{\text{work}} \leftarrow filter(\mathbf{S}, T_{\max})$ ▷ *select a subset of active sensors having POIs in range*
  $cov_{S(max)} \leftarrow covPoi(\mathbf{S}_{\text{work}})$ ▷ *evaluate coverage when all active sensors are on*
  while $cov < cov_{S(max)}$ **do**
    $H^{T_{\max}+1} \leftarrow generateSingleTimeSlot(\mathbf{S}_{\text{work}}, cov, \delta)$
    for all $S \in \mathbf{S} \mid H_S^{T_{\max}+1} = 1$ **do**
      $batt(S) \leftarrow batt(S) - 1$ ▷ *sensor batteries level update*
    $T_{\max} \leftarrow T_{\max} + 1$ ▷ *schedule length update*
    $\mathbf{S}_{\text{work}} \leftarrow filter(\mathbf{S}, T_{\max})$ ▷ $\mathbf{S}_{\text{work}}$ *update*
    $cov_{S(max)} \leftarrow covPoi(\mathbf{S}_{\text{work}})$ ▷ $cov_{S(max)}$ *update*
  return H ▷ *return the neighbour schedule*

procedure a randomly selected set of slots is removed from the schedule and for the sensors being active in these slots their battery levels are restored. Then the procedure *generateSingleTimeSlot* is called as many times as possible. The perturbation is based on observation, that the set of sensors activated again can

produce new time slots which have not been present in the modified schedule and due to different distribution of sensor activities their number may be higher.

4 Benchmark SCP1

For experimental evaluation of the proposed algorithm a set of test cases has been prepared. For brevity the set is called SCP1 (Sensor Coverage Problem, Set No. 1). Every test case is controlled by a set of parameters. The parameters fixed for all test cases are: (1) number of sensors: 2000, (2) sensing range r_{sens}: 1 unit, (3) the satisfying coverage level *cov*: 80%, (4) the coverage level tolerance δ: 5%. The remaining parameters may vary between test cases. These are:

- distribution of POIs — two types of distribution: nodes of a triangular grid and nodes of a rectangular grid,
- method of sensor distribution — coordinates of sensor localization originate from (1) random, (2) Halton generator [2].
- area size — the area is a square of the side size: 13, 16, 19, 22, 25, 28 units.
- battery max capacity, that is, max time of sensor activity T_{batt}: varies from 10 to 30 time steps with step 5 (value is the same for all sensors).

It has to be stressed, that due to non-empirical nature of proposed test cases the distance unit does not represent particular number of meters or inches but its role is just to show proportion between the area size, sensing range of sensors, and distances between POIs.

Additionally, it was assumed, that the number of POIs is the same for different area sizes, which means, that the grid of POIs stretches as the square side grows. To avoid full regularity in the POIs distribution, a small fraction of nodes in the grid, precisely, 20%, is not filled with POIs. The number of POIs in subsequent test cases is similar to each other but varies slightly (from 199 to 240 for the triangular grid and from 166 to 221 for the rectangular grid). The grid nodes which do not represent POIs are randomly selected in the grid for every instance of the test case.

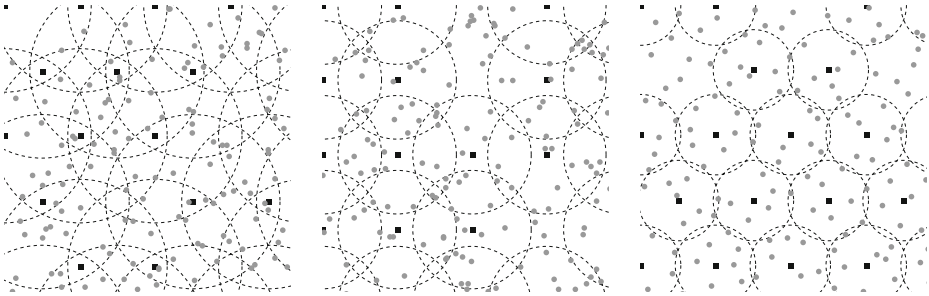
Eventually, 8 configurations of the test case have been proposed, which differ in the area side size, the type of a grid for POIs, and generator of sensor locations. For every test case a set of 40 instances was generated. In practice, every instance consists of two files: one with a set of POIs and one with a set of sensors. Mean numbers of sensors covering different numbers of POIs for each of the test cases are presented in Table 1.

Table 1 presents numbers of sensors which cover 1, 2, 3, 4, 5 and more POIs. One can see, that in the test case No. 1 all fractions of sensors are present, whereas, in the test case No. 8 majority of sensors (almost three fourths of them) cover just one POI and the mean number of sensors covering two POIs is really small and equals 74.7.

Additionally, Fig. 1 shows, that in the test case No. 1 the intersections between neighbor sensor monitoring areas are greater and consist of a larger number of POIs. For the next test cases these areas decrease and the numbers of common POIs decrease as well.

Table 1. Mean numbers of sensors covering 0, 1, 2, 3, 4, 5 and more that 5 POIs for the eight test cases of SCP1. △ means a triangular grid of POI locations, and ⋄ - a rectangular grid.

No.	Configuration	0	1	2	3	4	5	> 5
1.	13×13, △, rand	5.0	58.2	234.9	559.7	691.6	327.9	122.7
2.	13×13, ⋄, Halton	18.3	126.6	369.4	691.2	693.5	95.4	5.7
3.	16×16, △, Halton	24.6	211.1	679.2	902.7	182.4	0.0	0.0
4.	19×19, ⋄, rand	135.1	763.0	951.2	128.8	21.8	0.0	0.0
5.	19×19, △, rand	112.7	631.7	819.7	435.8	0.0	0.0	0.0
6.	22×22, △, Halton	209.9	1012.6	665.9	111.6	0.0	0.0	0.0
7.	25×25, △, rand	340.1	1303.4	350.9	5.6	0.0	0.0	0.0
8.	28×28, △, Halton	450.1	1475.2	74.7	0.0	0.0	0.0	0.0



(a) No. 1: △, rand, 13 (b) No. 4: ⋄, rand, 19 (c) No. 8: △, Halton, 28

Fig. 1. Visualizations of a monitored area for selected instances of a three among the eight test cases in SCP1: squares represent POIs, dots — sensors, circles around POIs — which sensors have in its range the POI located in the circle center.

5 Results of Experiments

For experimental evaluation of the proposed algorithm the SCP1 benchmark was used. For every test case from SCP1 40 instances were generated and the algorithm was executed once for each of the instances. Table rows present mean lengths of schedules obtained for respective 40 instances, as well as min and max lengths among them. The experiments were performed for five different values of T_{batt} varying from 10 to 30 with step 5.

Results of experiments with the proposed algorithm and the SCP1 benchmark are presented in Table 2. They are divided into two groups: the first one (the top half of Table 2) presents quality of initial schedules returned by *generateSchedule* (Algorithm 2), and the second one (the bottom half of Table 2) presents quality of schedules returned by the main algorithm (Algorithm 1). The values in the series are paired because output schedules of Algorithm 2 were an input for the main loop in Algorithm 1.

Table 2. Mean, min and max lengths of schedules returned by *generateSchedule* (Algorithm 2) – top part – and by the main algorithm (Algorithm 1) – bottom part – for each of the eight test cases in SCP1 and for five values of T_{batt} from 10 to 30

No.	T_{batt}	mean	min	max	No.	T_{batt}	mean	min	max
1	10	210.1	202	215	5	10	95.8	93	99
	15	316.2	309	326		15	144.5	140	149
	20	421.4	410	434		20	192.8	187	200
	25	527.5	513	544		25	241.4	234	249
	30	633.1	617	650		30	290.1	281	299
2	10	216.8	211	222	6	10	75.5	73	77
	15	325.9	319	332		15	113.5	111	116
	20	435.3	426	444		20	151.5	149	155
	25	543.6	534	559		25	189.8	184	194
	30	652.7	643	672		30	228.1	223	233
3	10	139.4	137	143	7	10	56.1	54	59
	15	209.9	204	217		15	85.0	81	88
	20	280.4	274	287		20	113.3	107	118
	25	350.5	342	360		25	141.9	135	148
	30	420.9	410	431		30	170.8	163	177
4	10	88.5	85	92	8	10	48.6	47	50
	15	133.2	126	138		15	73.6	72	76
	20	178.2	170	186		20	98.3	97	101
	25	222.7	210	230		25	123.1	120	125
	30	267.7	257	277		30	148.2	145	151
1	10	214.8	209	222	5	10	97.8	95	101
	15	321.5	312	331		15	146.9	142	152
	20	427.8	415	440		20	195.7	190	203
	25	534.2	519	548		25	244.5	237	252
	30	640.1	623	656		30	293.4	284	302
2	10	221.9	218	227	6	10	77.2	76	79
	15	332.2	327	342		15	115.8	113	118
	20	441.9	432	453		20	154.2	151	158
	25	551.6	543	567		25	192.5	187	196
	30	661.2	650	680		30	231.2	226	236
3	10	142.2	140	145	7	10	57.9	55	60
	15	213.2	207	220		15	87.0	83	90
	20	284.1	276	291		20	115.8	110	121
	25	354.7	346	365		25	144.7	137	150
	30	425.3	414	436		30	173.8	166	180
4	10	90.6	86	94	8	10	50.1	49	51
	15	135.5	129	140		15	75.4	74	77
	20	181.2	174	188		20	100.6	99	103
	25	226.0	214	234		25	125.8	123	128
	30	271.4	261	280		30	151.2	148	154

To determine whether application of LS improves quality of the schedules, statistic t-tests for paired data were performed where the null hypothesis is that any differences in schedule lengths before and after LS are due to chance. Obtained p-values are presented in Table 3. In every case the value is less than 0.001, which means that the null hypothesis can be rejected, that is, one can conclude with 99.9% confidence that the differences in schedule length before and after LS are not due solely to chance.

Additionally, for comparisons with results of algorithms presented in [5,6] another set of experiments was performed. In these experiments means and standard dev. of lengths of schedules obtained from LS are compared with means and standard dev. obtained by algorithms presented in these two papers. A set of four test cases has been selected. From [6] we selected three cases where 100 POIs are located in the form of a rectangular grid on an area of size 100×100, and the number of randomly deployed sensors is: case#1 – 100, case#2 – 200, case#3 – 300. From [5] one problem was selected where 400 POIs are located

Table 3. Results of statistical tests for paired samples obtained from Algorithm 2 and the main algorithm (Algorithm 1): p-values, std.dev.#1 for the results from Algorithm 2 and std.dev.#2 for the main algorithm obtained for each of the eight test cases in SCP1 and for five values of T_{batt} from 10 to 30

No.	T_{batt}	p-value	s.d.#1	s.d.#2	No.	T_{batt}	p-value	s.d.#1	s.d.#2
1	10	$2.47E^{-22}$	2.97	3.14	5	10	$3.04E^{-24}$	1.62	1.58
	15	$7.2E^{-22}$	4.55	4.52		15	$1.58E^{-20}$	2.46	2.49
	20	$8.22E^{-24}$	6.01	6.15		20	$4.93E^{-27}$	3.16	3.34
	25	$1.05E^{-21}$	8.02	7.83		25	$1.96E^{-23}$	3.92	4.09
	30	$4.29E^{-23}$	9.22	9.05		30	$2.89E^{-25}$	4.75	4.82
2	10	$1.24E^{-21}$	2.50	2.60	6	10	$1.11E^{-20}$	0.90	0.90
	15	$1.86E^{-21}$	3.50	3.52		15	$5.38E^{-23}$	1.38	1.17
	20	$1.88E^{-23}$	4.64	4.62		20	$6.38E^{-25}$	1.43	1.53
	25	$1.79E^{-22}$	5.36	5.82		25	$1.9E^{-26}$	2.09	2.03
	30	$1.02E^{-22}$	6.28	6.84		30	$8.93E^{-27}$	2.46	2.32
3	10	$3.01E^{-22}$	1.41	1.46	7	10	$7.29E^{-22}$	1.22	1.21
	15	$7.78E^{-23}$	2.69	2.54		15	$5.25E^{-21}$	1.92	1.83
	20	$1.78E^{-19}$	3.42	3.34		20	$3.17E^{-22}$	2.47	2.43
	25	$3.06E^{-29}$	4.06	4.10		25	$1.41E^{-23}$	2.74	2.86
	30	$4.88E^{-25}$	4.61	4.76		30	$9.55E^{-26}$	3.48	3.43
4	10	$3.06E^{-19}$	1.84	1.75	8	10	$2.82E^{-19}$	0.62	0.55
	15	$3.77E^{-20}$	2.57	2.49		15	$2.21E^{-19}$	1.03	0.84
	20	$2.37E^{-22}$	3.51	3.28		20	$5.38E^{-24}$	1.17	1.17
	25	$8.77E^{-24}$	4.45	4.57		25	$1.9E^{-26}$	1.19	1.18
	30	$6.35E^{-25}$	4.73	4.60		30	$1.2E^{-26}$	1.24	1.38

Table 4. Mean and standard dev. of lengths of schedules returned by the main algorithm (Algorithm 1) and presented in publications (here as the reference values)

No.	LS	ref. value	No.	LS	ref.value
case#1	123.5 ± 1.38	83.0 ± 2.23	case#3	358.4 ± 2.75	248.0 ± 2.82
case#2	241.3 ± 2.29	165.0 ± 2.44	case#4	118.6 ± 1.69	49

in the form of a rectangular grid on an area of size 100×100, and 100 sensors are randomly deployed. In every case $cov = 0.9$, a sensing range $r_{\text{sens}} = 20$ and $T_{\text{batt}} = 20$. 30 instances were generated for each of the four cases. Results from Table 4 show that Algorithm 1 gives schedules almost 50% longer than schedules obtained using methods from [6]. In case #4 our schedule is more than 100% longer than the reference value.

6 Conclusions

The paper presents a new heuristic approach to the Maximum Lifetime Coverage Problem. The proposed algorithm is build on the frame of the Local Search approach, and novelty is represented by two problem specific procedures. The first one generates an initial solution for the algorithm, whereas the second one is a perturbation procedure. Schedules generated by these procedures always satisfy coverage constraint.

For experimental evaluation of our algorithm a set of eight test cases is also proposed. Results of our experiments, that is, lengths of obtained schedules are divided into two groups: in the first one we evaluate effectiveness of the procedure generating initial solution, whereas in the second one effectiveness of the entire algorithm. Differences between schedule lengths obtained in the first and the second group demonstrate influence of the perturbation procedure on the quality of the results. Statistical tests confirmed our hypothesis that differences between means obtained for schedules before and after application of the iterative improvement are not due to chance, that is, the proposed perturbation can significantly improve obtained schedules. Moreover, our approach gives results much better than methods described in [5,6]

Further research will concern development of the process of a schedule iterative improvement on one hand and on the other – experimental comparisons of the proposed algorithm based on benchmarks presented in other publications.

References

1. Cardei, I., Cardei, M.: Energy-efficient connected-coverage in wireless sensor networks. IJSNet **3**(3), 201–210 (2008)
2. Halton, J.H.: Algorithm 247: radical-inverse quasi-random point sequence. Commun. ACM **7**(12), 701–702 (1964)

3. Tian, D., Georganas, N.D.: A coverage-preserving node scheduling scheme for large wireless sensor networks. In: Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA-02), pp. 32–41. ACM Press (2002)
4. Tretyakova, A., Seredynski, F.: Application of evolutionary algorithms to maximum lifetime coverage problem in wireless sensor networks. In: IPDPS Workshops, pp. 445–453. IEEE (2013)
5. Tretyakova, A., Seredynski, F.: Simulated annealing application to maximum lifetime coverage problem in wireless sensor networks. In: Global Conference on Artificial Intelligence, GCAI, vol. 36, pp. 296–311. EasyChair (2015)
6. Tretyakova, A., Seredynski, F., Bouvry, P.: Graph cellular automata approach to the maximum lifetime coverage problem in wireless sensor networks. Simulation **92**(2), 153–164 (2016)