Computers & Operations Research 78 (2017) 557–563

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Variable neighborhood search variants for Min-power symmetric connectivity problem

A.I. Erzin [a,b], N. Mladenovic [b,c], R.V. Plotnikov [a,*]

[a] Sobolev Institute of Mathematics, Novosibirsk, Russia
[b] Novosibirsk State University, Novosibirsk, Russia
[c] University of Valenciennes and Hainaut-Cambresis, Famars, France

ARTICLE INFO

Article history:
Received 12 April 2015
Received in revised form
4 March 2016
Accepted 17 May 2016
Available online 20 May 2016

Keywords:
Wireless sensor network
Energy efficiency
NP-hard problem
Variable neighborhood search

ABSTRACT

We consider the problem of optimal communication tree construction in a given undirected weighted graph. Such a problem occurs while minimizing the power consumption of data transmission in different distributed networks in the case when network elements are able to adjust their transmission ranges. In this paper, the most general strongly NP-hard formulation, when edge weights have arbitrary non-negative values, is considered. We propose new heuristics, mostly based on variable neighborhood search, for getting an approximate solution of the problem. Extensive comparative analysis between the proposed methods was performed. Numerical experiments demonstrated the high efficiency of the proposed heuristics.

© 2016 Elsevier Ltd. All rights reserved.

1. Introduction

1.1. Min-power symmetric connectivity problem

Elements of many communication networks use wireless communication for data exchange. Herewith, the energy consumption of a network's element is proportional to d^s, where $s \geq 2$ and d is the transmission range [1]. In some networks, e.g., in wireless sensor networks, each element (sensor) has a limited energy stored, and its efficient use allows extending the lifetime of the whole network [10–12]. For a rational energy usage, a modern sensor can adjust its transmission range. The problem then is how to find a transmission range (the transmitter power) for each element that supports a connected subgraph to minimize the energy consumed. If one supposes an equal signal propagation in all directions, then all elements inside a disk, whose radius is equal to the transmission range, receive the data. In this case, we can suppose that the communication network (a spanning subgraph whose edges are used for data transmission) is a complete graph [1,3,9,10]. However, the signal is not always spread equally in all directions and at any distance. Thus, in the general case, it is necessary to consider an arbitrary communication graph $G = (V, E)$. The communication energy consumption over each edge could be arbitrary too. If $c_{ij} \geq 0$ is a transmission-related energy consumption needed for sending data from $i \in V$ to $j \in V$, then in the connected subgraph $T = (V, E')$, $E' \subseteq E$ the energy consumption of node $i \in V$ equals to $E_i(T) = \max_{j:(i,j)\in E'} c_{ij}$. The goal of this paper is the development of algorithms for the construction of a spanning subgraph T that minimizes $\sum_{i\in V} E_i(T)$. Without loss of generality, we assume that subgraph T is a spanning tree. The problem can then be formulated in the following way.

1.2. Combinatorial formulation

Given a simple undirected weighted graph $G = (V, E)$ with a vertex set V, $|V| = n$, and an edge set E, find a spanning tree T^* of G, which is the solution to the following problem:

$$W(T) = \sum_{i\in V} \max_{j\in V_i(T)} c_{ij} \to \min_{T}, \tag{1}$$

where $V_i(T)$ is the set of vertices adjacent to vertex i in tree T and $c_{ij} \geq 0$ is the weight of the edge $(i, j) \in E$.

Any feasible solution of (1), i.e., a spanning tree of G, will be called a *communication tree* (subgraph). It is known that (1) is strongly NP-hard [1,5,6,9], and if N ≠ NP, then the problem is inapproximable within $1 + \frac{1}{260}$ [6]. Therefore, the construction and analysis of efficient approximation algorithms are some of the most important issues regarding the research on this problem.

1.3. Literature review

It is shown in [1] that a minimal spanning tree (a spanning tree with minimal total edge weights) is a 2-approximation solution to

* Corresponding author.
E-mail addresses: adilerzin@math.nsc.ru (A.I. Erzin), nenadmladenovic12@gmail.com (N. Mladenovic), nomad87@ngs.ru (R.V. Plotnikov).

http://dx.doi.org/10.1016/j.cor.2016.05.010
0305-0548/© 2016 Elsevier Ltd. All rights reserved.

the problem (1). In [5], a more precise ratio estimate for a minimal spanning tree was reported. In the same paper, a set of heuristic algorithms were proposed and their a posteriori analysis was performed. Since we were not completely satisfied with the quality of the results obtained, in [4] we proposed a hybrid heuristic that combines genetic algorithm (GA) with variable neighborhood search (VNS) [8]. Two new local search heuristics for the problem were proposed in [4]. These two heuristics are used then as a mutation operator in the GA. For the sake of completeness, we will recall both these local search heuristics in this paper. Computational results showed the high efficiency of the proposed hybrid heuristic.

1.4. Contribution

In this paper, we propose different heuristics based on a variable neighborhood search metaheuristic [2,7,8]. The contribution of this paper may be summarized as follows:

- A new local search heuristic that is based on elementary tree transformation (ETT) is proposed. In terms of solution quality, it significantly outperforms the previous one (named as local improvement (LI)), but uses more computation time.
- Several basic VNS- and general VNS-based heuristics are proposed and tested. Some of these new heuristics give results with better quality than recent state-of-the-art (hybrid heuristic [4]) techniques, especially for solving more realistic large-size problems.

1.5. Outline

In the next section, rules on new VNS-based heuristics are given.

In Section 3, an extensive computational analysis is described, while Section 4 concludes the paper.

2. Variable neighborhood based heuristics

2.1. Basic VNS and general VNS

As mentioned earlier, we use the VNS metaheuristic idea to get an approximate solution of (1). We use two well-known schemas: basic VNS, wherein only one neighborhood structure is used in the local search phase, and general VNS, which uses a variable neighborhood descent (VND) approach in the local search phase. Detailed descriptions of both these methods can be found in [7,8].

2.2. Local searches

For efficiency purposes, some of the below algorithms require the solution (i.e., a tree) to be an *oriented* tree $T = (v_0, V, A)$, where v_0 is the root and A is a set of arcs oriented from the root. Although the choice of the root vertex may affect the result of the proposed algorithms, we do not declare any special conditions for this choice, as we always take the first vertex of the entire array of vertices as the root.

To further clarity of exposition, let us introduce the notion of an arc's (or edge's) *drop*. Suppose that an arc (or an edge) a does not belong to the graph $G' \subset G$. Let us call the *drop of a in relation to* G' as the value of $W(G' \cup \{a\}) - W(G')$. In the same manner, let us define the drop of an arc (or edge) set. Suppose that an arc (or edge) set A does not belong to the graph $G' \subset G$. Let us call the *drop of A in relation to* G' as the value of $W(G' \cup A) - W(G')$.

The first local search procedure we used in basic VNS and in general VNS is LI from [4]; it is presented in Algorithm 1. In this algorithm, sorting is performed using all arcs in decreasing values of the drop in relation to the rest of the tree in step 5. After that, for each arc, an attempt to remove it from the tree (step 8) and add an arc with the minimum drop (steps 9–10) is performed. This procedure is repeated while the solution is improved.

The second algorithm used in basic VNS and general VNS as the local search procedure is ETT (see Algorithm 2). Let us briefly describe ETT. In steps 4–12, within one iteration of the **while** loop, all non-adjacent vertex pairs are considered. For each such pair $\{i, j\}$ an attempt to add an arc (i, j) to a tree (see step 6) and remove an arc with the maximum drop from the obtained cycle (see steps 7 and 8) is performed. This procedure is repeated while the solution is improved.

Algorithm 1. LI

1: *Input*: initial solution $T = (v_0, V, A)$ is a tree rooted in v_0;
2: *improved*← **true**;
3: **while** *improved* **do**
4: *improved*← **false**;
5: Sort A by decreasing of the drops;
6: **for all** arcs (i, j) in A **do**
7: Let B be a subtree of T where j is the root;
8: $R \leftarrow T \backslash (B \cup \{(i, j)\})$;
9: Find a vertex k from all vertices of R such that the drop of the arc (k, j) is minimum;
10: $T' \leftarrow (T \backslash \{(i, j)\}) \cup \{(k, j)\}$;
11: **if** T' is better than T
12: $T \leftarrow T'$;
13: *improved*← **true**;
14: **end if**
15: **end for**
16: **end while**

Algorithm 2. ETT.

1: *Input*: initial solution $T = (v_0, V, A)$ is a tree rooted in v_0;
2: *improved*← **true**;
3: **while** *improved* **do**
4: *improved*← **false**;
5: **for all** non-adjacent vertex pairs i and j of T **do**
6: Set $T' \leftarrow T \cup \{(i, j)\}$;
7: Find an arc a of a cycle in T' with the maximum drop;
8: $T' \leftarrow T' \backslash \{a\}$;
9: **if** T' is better than T **then**
10: $T \leftarrow T'$; *improved*← **true**;
11: **end if**
12: **end for**
13: **end while**

In [4] a VND approach with three neighborhood structures for local search (N_1, N_2, and N_3) was proposed; we also use this procedure as part of general VNS. Let us define these neighborhood structures. Let T be a spanning tree of G. Then

$$N_k(T) = \{T' = (v_0, V, A') \subset G | T - \text{spanning tree on } V;\ |A' \backslash A| \le k\},$$
$$k = 1, 2, 3.$$

In other words, the number of different edges in T and T' is less than or equal to k. Regarding the efficiency of the search, the question is whether to enumerate all the trees that belong to $N_k(T)$ or not, since their number increases exponentially with respect to k. Another question also is how to perform a search in these neighborhoods. There are some obvious strategies: (i) add k edges

to T and then remove another k edges to get tree T'; (ii) remove k edges from T to get $k+1$ subtrees and then add another k edges to get spanning tree T'; (iii) repeat the add/remove actions for k times (i.e., k consecutive ETT moves); (iv) repeat the remove/add sequence for k times. In [4], a VND approach that uses three neighborhood structures (N_1, N_2, and N_3) was proposed and used in the GA. Search through these three neighborhoods is organized as in variant (ii) of the above, i.e., k edges are removed from T and then another k edges are added to the obtained graph. Moreover, the removed edges are taken at random, whereas the best possible edges to add are selected thereafter. Thus, procedures that explore N_1,N_2, and N_3 in a semi-random fashion are not classical local search procedure since they do not guarantee that the obtained solution is a local optimum. Nevertheless, a VND algorithm that uses those three algorithms appears to be efficient. In this paper, the same VND-based method is used as the local search procedure in the general VNS as well.

Let us briefly describe the procedures that explore the neighborhoods N_1,N_2, and N_3 in a semi-random way (sometimes called *intensified shaking*). In the procedure of "Move in N_1", which is presented in Algorithm 3, one randomly chosen edge is excluded from the tree in step 3 and then the best edge connecting the two obtained components is found in step 4. In the same manner, within the procedure of "Move in N_2", which is presented in Algorithm 4, two randomly chosen edges are excluded from the tree, splitting it into three connected components (steps 2 and 3). In the remaining steps 4–16, the two best edges connecting these components are found: in step 4, an attempt to find the best pair of adjacent edges is performed; in step 9, the three best edges connecting the components are found; and in steps 11–16, the best two of them are chosen. In the procedure of "Move in N_3", which is presented in Algorithm 5, three edges are removed from the initial tree in step 2, and then in step 3, for each pair of the four obtained components, the connecting edge with the minimum value of drop is found; after that, by solving the minimal spanning tree problem, we can find the three edges connecting the obtained components in steps 4–6.

Algorithm 3. Move in N_1

1: *Input*: initial solution $T = (v_0, V, A)$ is a tree rooted in v_0;
2: Choose an arc $a = (i, j)$ from T with a probability proportional to the drop of the objective function;
3: By excluding (i, j) from T, divide T into two components: a root component R, where i is a leaf, and a branch component B, where j is the root;
4: Find a vertex k from all the vertices of R such that the drop of the arc (k, j) is minimum;
5: **Return** $(T\backslash\{(i, j)\}) \cup \{(k, j)\}$.

Algorithm 4. Move in N_2

1: *Input*: initial solution $T = (V, E)$ is a tree (orientation is not taken into account);
2: Choose two random edges, e_1 and e_2, with a probability proportional to drop;
3: Find three connected components, C_0, C_1, and C_2, after removing the two chosen edges;
4: Find a pair of edges, $bestEdge_1$ and $bestEdge_2$, that connect the three components, have a common vertex in one component, and have the minimum drop in relation to $C_0 \cup C_1 \cup C_2$. Set $T' = C_0 \cup C_1 \cup C_2 \cup \{bestEdge_1\} \cup \{bestEdge_2\}$;
5: **If** $W(T') < W(T)$ **then**
6: Set $T \leftarrow T'$;
7: **End if**
8: **For all** $i \in \{0, 1, 2\}$ **do**
9: Find an edge, $candidateEdge_i$, that connects C_i and $C_{[(i+1)\bmod 3]}$ and that has the minimum drop in relation to $C_i \cup C_{[(i+1)\bmod 3]}$.
10: **End for**
11: **For all** $i \in \{0, 1, 2\}$ **do**
12: Set $T' = C_0 \cup C_1 \cup C_2 \cup \{candidateEdge_i\} \cup \{candidateEdge_{[(i+1)\bmod 3]}\}$;
13: **If** $W(T') < W(T)$ **then**
14: Set $T \leftarrow T'$;
15: **End if**
16: **End for**
17: **Return** T.

Algorithm 5. Move in N_3

1: Choose three edges, e_1, e_2, and e_3, with a probability proportional to the drop in relation to the remaining graph;
2: Construct four connected components after the deletion of these edges;
3: Find all six edges that connect these components with minimum drops;
4: Construct a complete edge-weighted graph on four vertices whose vertices correspond to the connected components and whose edges have the minimum drops in relation to the components;
5: Find a minimal spanning tree (MST) on the constructed graph;
6: Let e_4, e_5, and e_6 be the edges of the obtained MST;
7: **Return** $(T\backslash\{e_1, e_2, e_3\})U\{e_4, e_5, e_6\}$.

Algorithm 6. Shaking

1: *Input*: initial solution $T = (v_0, V, A)$ is a tree rooted in v_0;
2: $i \leftarrow 1$;
3: **while** $i \le k$ **do**
4: Select two different vertices i and j of V at random;
5: **If** $(i, j) \in A$ **then**
6: continue;
7: **End if**
8: $A' \leftarrow A \cup \{(i, j)\}$;
9: Let $C \subseteq A'$ be a cycle containing (i, j); select at random an arc $a \in C$;
10: $A' \leftarrow A'\backslash\{e\}$;
11: $T \leftarrow (V, A')$;
12: **End while**

2.3. Shaking

For the procedure Shaking, which is used in basic VNS and general VNS, we propose a new algorithm, a pseudo code of which is presented in Algorithm 6. In step 4, two di erent vertices are selected; then, in case they are not adjacent, the arc connecting these vertices is added to the tree in step 8; after that, a randomly selected arc is excluded from the obtained cycle in step 10. Such attempt to replace an arc is repeated k times. Let $k_{\max}$ be the maximum number of arc replacements by the *Shaking* procedure. Note that $k_{\max}$ is a free parameter of all the considered VNS heuristics. The best value of this parameter is estimated experimentally.

2.4. VNS-based heuristics

We propose two basic VNS approaches based on LI and ETT:

- BVNS_LI: LI is used as a local search procedure;

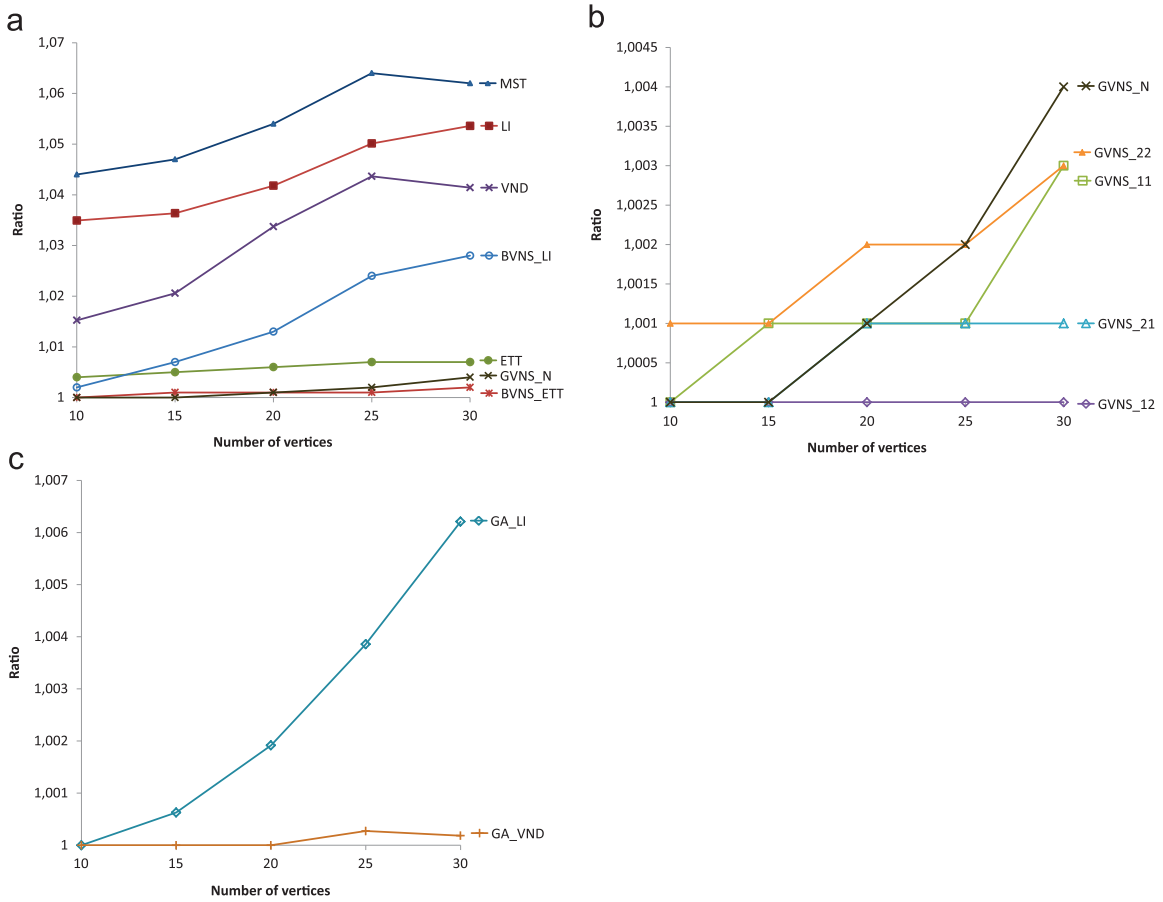

Fig. 1. Ratio in the case of the small dimension ($n \leq 30$). (a) Ratio of local search procedures LI, ETT, and VND and basic VNS with these local search procedures: BVNS_LI, BVNS_ETT, and GVNS_N. (b) Ratio of the general VNS: GVNS_11, GVNS_12, GVNS_21, GVNS_22, and GVNS_N. (c) Ratio of the hybrid genetic algorithms GA_LI and GA_VND.

- BVNS_ETT: ETT is used as a local search procedure.

Both algorithms (LI and ETT) can be combined in the general VNS algorithm. Moreover, in each step of the general VNS, the best neighbor instead of the local optimum can be found. For this purpose, let us introduce algorithms LI_1 and ETT_1, which are the same as LI and ETT, respectively, except that they stop after the first improvement. Thus, we propose five algorithms based on the general VNS schema:

- GVNS_11: Within a local search phase, LI is performed first, then — ETT;
- GVNS_12: Within a local search phase, ETT is performed first, then — LI;
- GVNS_21: Within a local search phase, LI_1 is performed first, then — ETT_1;
- GVNS_22: Within a local search phase, ETT_1 is performed first, then — LI_1;
- GVNS_N: N_1, N_2, and N_3 from the VND proposed in [4] are used as neighborhood structures.

3. Simulation

All the proposed algorithms have been implemented in C++ using the Visual Studio 2010 Integrated Development Environment. A simulation was executed for $n = 10, 15, 20, 25, 30, 50, 150, 200$. For the same dimension, 100 different instances were generated. For each instance, a required number of points were scattered on a square area. After this, a complete edge-weighted graph whose vertices corresponded to the points and whose edge weights were equal to the squared distances between points was defined. Then a minimal spanning tree for use as an initial approximate solution in all the algorithms was constructed. As a data structure for storing a feasible solution of the problem, we used a tree wherein each element stores a pointer to its parent vertex (or a null pointer in the case of the root) and the list of its direct successors. The experiment was performed on an Intel Core i5–3470 (3.2 GHz) 8 Gb machine.

For the small dimension ($n \leq 30$), we defined the parameters of the problem formulation as an integer linear programming problem (ILP), as proposed in [5], and then we obtained the optimal solution using the IBM ILOG CPLEX package. Thus, for $n \leq 30$, we computed the exact value of the *ratio*, which was expressed by the value of $W_A(T)/W(T^*)$, where $W_A(T)$ is the value of the objective function of the solution constructed by algorithm A and $W(T^*)$ is the optimal value of the objective function. In the case of larger dimensions, the upper bound of the ratio was calculated by the formula $W_A(T)/LB(W(T^*))$, where $LB(W(T^*))$ is the lower bound for $W(T^*)$. As a value of $LB(W(T^*))$, the sum of the weights of the minimal spanning tree edges was taken (see [5]).

For the VNS-based heuristics, it is necessary to define the parameter k_{max}. For this goal, each algorithm was run on the same instances with different values of k_{max}. It appeared that, beginning from $k_{max} = 30$, on average, the ratio of the obtained solution did not decrease significantly, whereas the runtime of some algorithms increased up to twice, while k_{max} increased by 10. Moreover, on average, the runtime of all the algorithms remained accessible for $k_{max} = 30$. Therefore, in all the VNS-based algorithms, we set $k_{max} = 30$.

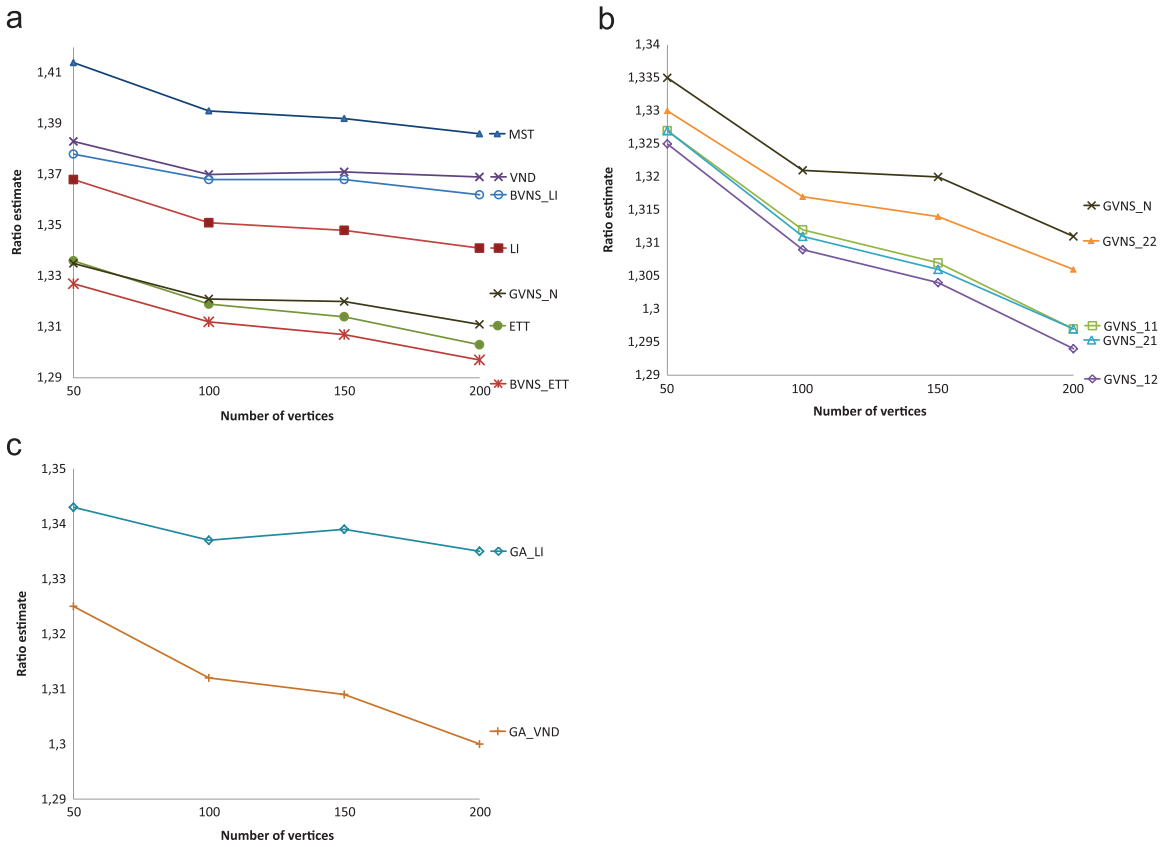

Fig. 2. Ratio upper estimates in the case of the large dimension ($50 \le n \le 200$). (a) Ratio upper estimate of local search procedures LI, ETT, and VND and basic VNS with these local search procedures: BVNS_LI, BVNS_ETT, and GVNS_N. (b) Ratio upper estimate of the general VNS: GVNS_11, GVNS_12, GVNS_21, GVNS_22, and GVNS_N. (c) Ratio upper estimate of the hybrid genetic algorithms GA_LI and GA_VND.

Let us introduce the algorithms that were considered in the graphics. Algorithms LI, ETT, VND, BVNS_LI, BVNS_ETT, GVNS_11, GVNS_12, GVNS_21, GVNS_22, and GVNS_N were described in Section 2; the hybrid genetic algorithms GA_LI and GA_VND were described in [4]; a minimal spanning tree is denoted as MST.

In Fig. 1, the ratios of the solutions yielded by the algorithms for $n \le 30$ are presented. On average, in cases when the exact values of the ratio could be computed (i.e., when $n \le 30$), algorithms GA_VND and GVNS_* yielded solutions whose objectives differed from the optimal one by not more than 0.6%. At the same time, the solutions closest to the optimal solutions were those constructed by GVNS_12 and GA_VND: they both yielded ratios that did not exceed 1.0003. It was found that a hybrid GA, GA_VND, which uses VND as a mutation operator, on average, yielded a significantly better solution than GVNS_N, which is based on the same VND procedure (the second one is faster, however; see Fig. 3). It should be noted that the algorithm ETT, on average, yielded a solution whose objective differed from the optimal one by at most 0.7% (for comparison, on average, the objective of the solution yielded by another local search LI differed by about 3% from the objective of the optimal solution when $25 \le n \le 30$).

The experiment results for the case of $50 \le n \le 200$ (see Fig. 2) showed that the main trends found in the case of small values of n regarding the majority of the algorithms persisted for the larger dimensions. However, one can distinguish algorithm GA_LI, whose solution quality became significantly worse in relation to algorithms GVNS_*, ETT, and BVNS_ETT while n grew. The ratio estimate graphics of the last ones were almost parallel to each other, which means that they had similar dynamics of ratio estimate that changed with growing n. The most accurate solution was again constructed by GVNS_12. Furthermore, in decreasing order, the graphics of the ratio estimates of BVNS_ETT, GVNS_21, GVNS_11 (these three algorithms had almost confluent graphics of ratio estimate), GA_VND, ETT, GVNS_22, and GVNS_N are depicted in Fig. 2. Herein the difference between the maximum value of the estimates of the aforementioned algorithms and the minimum one did not exceed 2%, e.g., when $n=200$, the ratio estimate of GVNS_12 was 29% and that of GVNS_N was 31%.

In Table 1, a percentage of the cases when optimal solutions were constructed is presented. One can see that, in the case of the small dimension, both algorithms GVNS_12 and GA_VND almost always constructed an optimal solution: in the worst-case scenario (when $n=30$), the algorithm GVNS_12 constructed an optimal solution in 95% of the cases and GA_VND in 93% of the cases (again, when $n=30$). It can be seen that the percentage of optimality of the other algorithms dropped down while n grew. These algorithms can be conventionally divided into two groups: (a) GVNS_21, BVNS_ETT, GVNS_11, GVNS_22, GVNS_N, ETT, and GA_LI, whose optimality percentage was not less than 45%; and (b) MST, VND, LI, and BVNS_LI, whose optimality percentage in the worst case (i.e., when $n=30$) did not exceed 5%. In general, it matches the results presented in Fig. 1.

The graphics in Fig. 3 show the change in the runtime with growing n. Notice that the algorithms GVNS_N and ETT worked rather fast (when $n=200$, on average, they solved the problem in less than 0.5 seconds), and, at the same time, as mentioned previously, on average, they yielded rather accurate solution. The most accurate algorithm, GVNS_12, worked for about 43 seconds when $n=200$, but the algorithm GA_VND, whose ratio was close to that of GVNS_12, solved the problem more than ten times faster in the case of $n=200$. Here we should note that all the GAs were well

Table 1
Percentage of the cases when the optimal solution was obtained by the algorithm.

n	MST	LI	ETT	VND	GA_LI	GA_VND	BVNS_LI	BVNS_ETT	GVNS_11	GVNS_12	GVNS_21	GVN_22	GVNS_N
10	23	57	84	51	98	98	90	99	98	99	98	97	**100**
15	11	37	74	25	97	**100**	67	94	93	99	99	96	96
20	3	22	61	13	79	**100**	29	89	86	98	86	85	88
25	1	4	48	2	59	95	13	88	85	**97**	84	77	71
30	0	4	47	2	46	93	5	82	80	**95**	89	69	51

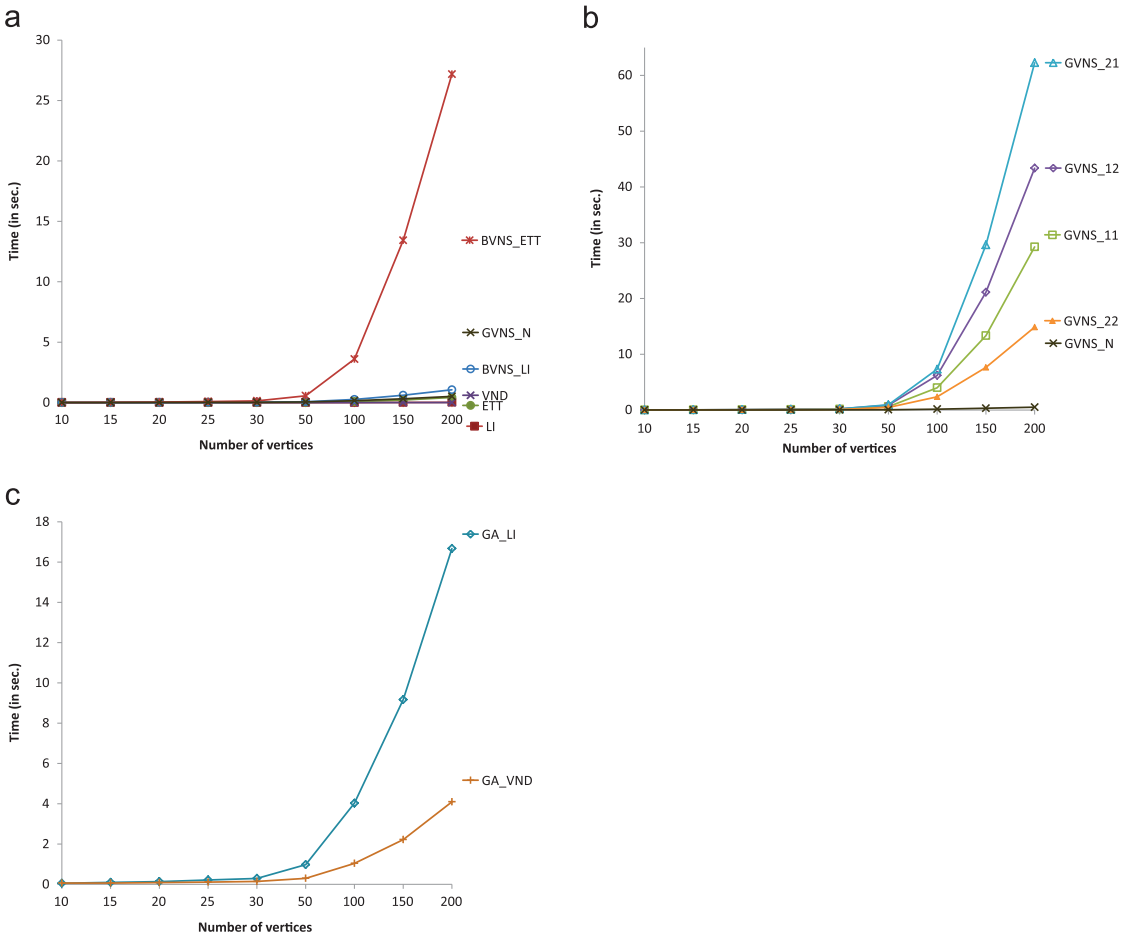

Fig. 3. Running time. (a) Running time of local search procedures LI, ETT, and VND and basic VNS with these local search procedures: BVNS_LI, BVNS_ETT, and GVNS_N. (b) Running time of the general VNS: GVNS_11, GVNS_12, GVNS_21, GVNS_22, and GVNS_N. (c) Running time of the hybrid genetic algorithms GA_LI and GA_VND.

parallelized, i.e., they used four parallel threads. Algorithm GVNS_21 was the slowest one: on average, it worked for about 60 seconds when $n=200$. All the other VNS-based heuristics turned out to be faster than GVNS_12 but slower than GA_VND. Notice that the algorithm BVNS_ETT worked, on average, for about 29 seconds when $n=200$; in the case of large dimensions, it also yielded a more accurate solution than GA_VND.

4. Conclusion

In this paper, we have proposed new heuristics based on the variable neighborhood search (VNS) approach for finding approximate solutions of the optimal communication tree synthesis problem. The proposed algorithms were compared between one another and also with the algorithms recently proposed in [4]. The majority of the proposed VNS-based variants appeared to be rather efficient. However, the most effective, on average, appeared to be the general VNS algorithm named GVNS_12, which sequentially uses elementary tree transformation (ETT) and the so-called local improvement (LI) as local search routines in the VNS. In the case of large dimensions, the algorithms GVNS_21 and BVNS_ETT, on average, yielded a better solution than the hybrid genetic algorithm GA_VND from [4], but spent more computing time. Moreover, an ETT local search procedure proposed in this paper took more time than the known LI, but it constructed solutions of significantly better quality. In general, in the case of successive choice of neighborhood structures for the local search procedure, the VNS approach is justified for the efficient solution of (1).

Acknowledgments

The research of A. Erzin is partly supported by the Russian Foundation for Basic Research (Grant no. 16-07-00552). The

research of N. Mladenovic is partly supported by the Ministry of Education and Science, Republic of Kazakhstan (Institute of Information and Computer Technologies) (Project no. 0115PK00546). The research of R. Plotnikov is partly supported by the Russian Foundation for Basic Research (Grant no. 16-37-60006).

Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.cor.2016.05.010.

References

[1] Althaus E, Calinescu G, Mandoiu I, Prasad S, Tchervenski N, Zelikovsky A. Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks. Wirel Netw 2006;12(3):287–99.
[2] Brimberg J, Urosevic D, Mladenovic N. Variable neighborhood search for the vertex weighted k-cardinality tree. Eur J Oper Res 2006;171:74–84.
[3] Carmi P, Katz M. Power assignment in radio networks with two power levels. Algorithmica 2007;47:183–201.
[4] Erzin A, Plotnikov R. Using VNS for the optimal synthesis of the communication tree in wireless sensor networks. Electron Notes Discret Math 2015;47:21–8.
[5] Erzin A, Plotnikov R, Shamardin Y. On some polynomially solvable cases and approximate algorithms in the optimal communication tree construction problem. J Appl Ind Math 2013;7:142–52.
[6] Fuchs B. On the hardness of range assignment problems, Tech. rep. TR05–113, electronic colloquium on computational complexity; 2005.
[7] Hanafi S, Lazic J, Mladenovic N, Wilbaut C, Crevits I. New variable neighbourhood search based 0-1 MIP heuristics. Yugosl J Oper Res 2015. http://dx.doi.org/10.2298/YJOR140219014H.
[8] Hansen P, Mladenovic N. Variable neighborhood search: principles and applications. Eur J Oper Res 2001;130:449–67.
[9] Kirousis L, Kranakis E, Krizanc D, Pelc A. Power consumption in packet radio networks. Theor Comput Sci 2000;243:289–305.
[10] Pottie G, Kaiser W. Wireless integrated network sensors. Commun ACM 2000;43(5):51–8.
[11] Wu J, Yang S. Energy-efficient node scheduling models in sensor networks with adjustable ranges. Int J Found Comput Sci 2005;16(1):3–17.
[12] Zhang H, Hou J. Maintaining sensing coverage and connectivity in large sensor networks. Ad Hoc Sens Wirel Netw 2005;1(1–2):89–124.