Exact Algorithms for Maximizing Lifetime of WSNs using Integer Linear Programming

Xinshu Ma*†, Xiaojun Zhu*†‡§, Bing Chen*†
* College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
† Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210023, China
‡ State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

***Abstract*—In wireless sensor networks, maximizing the lifetime of a data gathering tree is known to be NP-hard, so various (exponential-time) exact algorithms are designed to find the best data gathering tree. The state-of-the-art exact algorithm recursively performs graph decomposition to reduce search space. However, it essentially enumerates all possibilities in adjacent graph decompositions, and cannot handle moderately large sensor networks. In this paper, we propose an exact algorithm using integer linear programming. The challenge is that some constraints are not linear in the optimization problem. To address this challenge, instead of the optimization problem, we formulate the decision problem, and solve each decision problem by integer linear programming. The optimal value is then found by binary search over all possible lifetimes. To reduce the running time, we preprocess the graph by decomposing it into bi-connected subnetworks, and propose various rules to reduce the number of candidate lifetimes. Numerical results on simulated networks show that, within two hours, our algorithm can solve more problem instances than previous algorithms.**

I. Introduction

In wireless sensor networks, each sensor usually operates on limited battery energy and consumes it when transmitting and receiving messages. In many scenarios such as wild forests, batteries are neither replaceable nor rechargeable, so maximizing the functional lifetime of the network is one of the most important problems for wireless sensor networks. In this work, we consider wireless sensor networks that need to collect all original data, so in-network processing such as aggregation cannot be applied. Our focus is on designing exact algorithms to find a maximum lifetime data gathering tree. Unless $P = NP$, such algorithms run in exponential time, since lifetime maximization is known to be NP-hard [1].

A straightforward approach is to enumerate all spanning trees using back-tracking algorithms [2]. Clearly, it runs very slow due to large solution space. To reduce the search space, reference [3] proposes to incorporate a graph decomposition procedure into the back-tracking algorithm. Whenever the graph cannot be decomposed, it considers two subproblems on an arbitrary edge. In one subproblem, the edge appears in the final tree, and in the other subproblem, the edges does not. Graph decomposition is applied to each subproblem. This method essentially enumerates all possibilities between two adjacent graph decompositions. Though it runs much faster than brute-force enumeration algorithms, we find that it cannot handle moderately large sensor networks. Specifically, we find that, in quite a few randomly generated network instances with more than 40 nodes, graph decomposition based algorithm cannot find the optimal solution within two hours.

In this paper, we propose a new exact algorithm that combines the benefit of graph decomposition and integer linear programming. It decomposes the network graph into biconnected subgraphs only once, which could significantly reduce search space as before. Then, it solves every subproblem by performing a binary search over possible lifetimes, and formulates each comparison against a candidate lifetime as a feasibility problem in integer linear programming formulation. By using existing ILP solvers that has incorporated ingenious ideas in the operations research community, our algorithm could explore the solution space in a more efficient manner. In addition, we find that the order for solving the subproblems has a non-negligible impact on the overall running time. This is because, previously solved subproblems could shrink the range of possible lifetimes, reducing the number of steps in binary search. Thus, we propose to solve subproblems in ascending order of problem size.

We conduct extensive simulations on randomly generated networks. Simulation results indicate that our algorithm runs much faster than previous exact algorithms. When the time budget for a network instance is constrained to 2 hours, our algorithm can solve more networks than previous algorithms, especially when the network contains more than 45 nodes.

The rest of the paper is organized as follows. Section II summarizes related works. Section III formulates the problem. Section IV gives the proposed algorithm. Section V evaluates the proposed algorithm by simulations. Finally, Section VI concludes the paper.

II. Related Works

Maximum lifetime routing in wireless sensor networks has been proved to be NP-hard [1], and it cannot be approximated within a factor of 2/3 unless P=NP [3]. Thus, various (polynomial-time) approximation algorithms and (exponential-time) exact algorithms are proposed over the years (e.g., [1], [3], [4]). References [1], [4] all focus on approximation algorithms. Naturally, these algorithms only find suboptimal

§ corresponding author. Email: xzhu@nuaa.edu.cn.

978-1-5090-4183-1/17/$31.00 ©2017 IEEE

spanning trees and none of them can guarantee to find an optimal tree. It is difficult to evaluate the empirical performances of these algorithms since the optimal value is unknown. A straightforward approach is to perform exhaustive search using back-tracking algorithms such as [2], but a similar attempt could only consider networks with 10 nodes [5].

To find the optimal solution, reference [3] proposes an exact algorithm using graph decomposition to reduce search space. It repeatedly decomposes graph into biconnected subgraphs, speeding up the search process significantly. However, it still tries all possibilities when the graph is already biconnected. Our experiments indicate that this algorithm begins to fail occasionally when sensor nodes exceed 40. Our work in this paper has several improvements by incorporating the graph decomposition process with an integer linear programming formulation. The proposed algorithm runs faster than all previous exact algorithms.

When every sensor node can compress all received data into a single message, the problem appears to be simpler. Though it is still NP-hard, there exist efficient approximation algorithms [5]–[7]. Other metrics besides lifetime have also been considered [8]–[10].

III. System Model and Problem Formulation

We assume that there are n sensor nodes v_1, v_2, ..., v_n in a wireless sensor network with an additional sink node v_0. All sensors continuously monitor the environment and send data periodically to the sink node. The network is modeled as a connected undirected graph $G(V, E)$, where $V = \{v_0, v_1, ..., v_n\}$ represents the set of nodes and E is the set of edges in the network. An edge connecting v_i and v_j indicates that v_i and v_j can receive messages from each other.

Each sensor node $v_i \in V$ except the sink is initialized with a different and non-replenishable battery energy $e_i (e_i > 0)$, and the sink node has unlimited power supply, i.e., $e_0 = \infty$. We assume that nodes consume the same energy T_x to transmit a message and R_x to receive a message. In each round, each node generates one message consisting of a certain number of bits of data and transmits the messages received from its descendants together with its own message to sink via a single hop or a multi-hop path. Therefore, the total energy consumption in each round of a node can be represented as $d_T(i)(R_x + T_x) + T_x$, where $d_T(i)$ is the number of descendants of node v_i in tree T.

The lifetime of a node v_i in tree T is defined as the number of rounds from its deployment to its death:

$$l(T, i) = \frac{e_i}{d_T(i)(R_x + T_x) + T_x}$$

The lifetime of a tree T is defined as the lifetime of the first dead node, i.e., the shortest lifetime among all the nodes:

$$l(T) = \min_{i=1,\ldots,n} l(T, i)$$

Problem 1. *Given a network $G(V, E)$ and nodes' initial energies $e_1, e_2, \ldots, e_n$, find a data gathering tree that has the maximum lifetime, i.e.,*

$$\max_{T \in S(G)} l(T),$$

where $S(G)$ is the set of spanning trees of graph G.

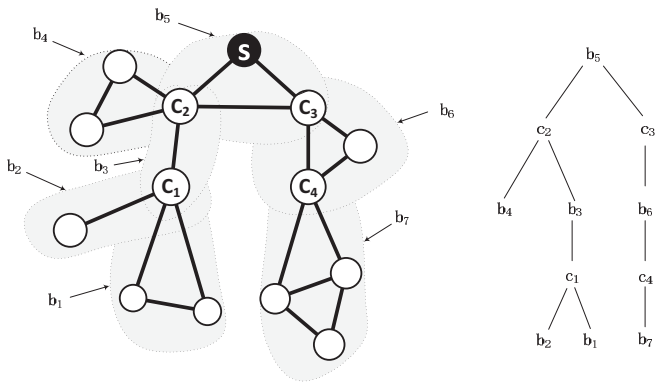

Fig. 1. A sensor network (left) and its block graph (right).

It has been proved that this problem is not only NP-hard, but also NP-hard to be approximated within a factor greater than 2/3 [3]. It is worth mentioning that all spanning trees in the formulation are implicitly rooted trees, because different roots produce different lifetimes. The formulation is consistent with existing formulations in [1], [3], [4].

IV. The Proposed Exact Algorithm

Our algorithm incorporates graph decomposition with integer linear programming as follows. First, the original network graph is decomposed into biconnected subgraphs, leading to subproblems. Then, each subproblem is solved by binary search over possible lifetimes, where each question is answered by solving a feasibility problem in integer linear program formulation. Third, we provide approaches to reduce the number of possible lifetimes, thus speeding up binary search. Finally, we give the overall algorithm.

A. Obtaining Subproblems by Graph Decomposition

The basic idea of graph decomposition can be illustrated by Figure 1. The left side shows a sensor network with circles representing sensor nodes and solid circle representing the sink. Four cut vertices c_1, c_2, c_3, and c_4 divide the graph into seven biconnected subgraphs $b_1 \ldots, b_7$, where subgraphs are indicated by dashed areas. Each subgraph is either a 2-connected subgraph or a bridge, and it is referred to as a block in graph theory. If we treat blocks and cutvertices as vertices, and connect two vertices if a block contains a cutvertex, then we can get the tree in the right side of Figure 1, which is called the block graph. This decomposition can be done in linear time [11].

It has been proved that the original lifetime maximization problem can be divided to subproblems, where each subproblem is to find a maximum lifetime tree in a biconnected subgraph [3]. However, the lifetime of a tree in a subgraph

needs to be re-defined, because we have to consider nodes in other subgraphs. For example, in Figure 1, nodes in block b_1 are descendants of c_2 in all spanning trees. Thus, when we compute the lifetime of a spanning tree of block b_3, we need to take nodes in b_1 into account.

Formally, suppose the graph is decomposed into blocks $b_1, \ldots, b_t$ with block sinks $s_1, \ldots, s_t$. Let d_j denote the number of common descendants of node j in all spanning trees. For example, in Figure 1, we have $d_1 = 3$, $d_2 = 6$, $d_3 = 5$ and $d_4 = 3$. Other d_js are all zero. Then we get t subproblems $P_1, \ldots, P_t$ corresponding to the t blocks and t block sinks, such that in subproblem P_i, we need to find a spanning tree of subgraph b_i rooted at s_i that has the maximum lifetime, where node j's lifetime is defined as $e_j/(d_T(j)(R_x + T_x) + (d_j + 1)T_x)$. Thus, the difference of a subproblem with the original problem is that the lifetime in subproblems considers additional traffic d_j. Note that all d_j can be computed during the graph decomposition process, and only cutvertices have non-zero d_j.

B. Solving Subproblems by Integer Linear Programming

The challenge of formulating lifetime maximization problem as an integer linear program is that some constraints are inherently non-linear. To this end, instead of the original optimization problem, we propose to formulate decision problems as integer linear programs, and perform binary search to find the maximum lifetime. In the following, we first give a non-linear integer program formulation, and then show how to solve it by multiple integer linear programs together with binary search.

Given a subproblem P with block b and block sink s, we introduce binary variables x_{ij} to indicate whether node i's parent is node j, and variables y_{ij} to represent the number of messages sent from node i to node j. It is easy to see that x_{ij} and y_{ij} are undefined if v_i and v_j are not adjacent in b. Borrowing the standard network flow formulation in [12], we have the following non-linear integer program.

$$\max_{\mathbf{x},\mathbf{y}} \quad l \tag{1}$$

subject to

$$\sum_j x_{ij} = 1, \quad \forall v_i \neq s \tag{2}$$

$$\sum_j y_{ij} = \sum_j y_{ji} + d_i + 1, \quad \forall v_i \neq s \tag{3}$$

$$x_{ij} \leq y_{ij} \leq \left\lfloor \frac{e_i/l + R_x}{R_x + T_x} \right\rfloor x_{ij}, \quad \forall v_i, v_j \tag{4}$$

$$y_{ij} \geq 0, x_{ij} \in \{0, 1\}, \quad \forall v_i, v_j \tag{5}$$

In the formulation, l represents the lifetime, equality (2) ensures that every node except the sink has one parent, equality (3) states that the number of outgoing messages is equal to the number of incoming messages plus d_i+1. Inequality (4) states that messages can only be sent to parent node. The maximum number of sent messages follows from the lifetime definition that $l \geq e_i/((y_{ij} - 1)(R_x + T_x) + T_x)$ and from the fact that y_{ij} is an integer. The correctness of this formulation can be easily proved.

The drawback of this formulation is that it is nonlinear. In inequality (4), there exists a term x_{ij}/l that is division of two variables. The floor operation is also non-linear. Consequently, using this formulation, we need to rely on general purpose solvers that can deal with both integer variables and non-linear inequalities, which are slow.

To this end, we propose to formulate the decision version of the problem, i.e., whether the optimal lifetime can be greater than or equal to a given value l_0. Define

$$cap(i, l_0) = \left\lfloor \frac{e_i/l_0 + R_x}{R_x + T_x} \right\rfloor.$$

Then, the decision problem has a positive answer if and only if the following integer linear program ILP(l_0) is feasible.

$$\begin{aligned} \max_{\mathbf{x},\mathbf{y}} \quad & 0 \\ \text{subject to} \quad & (2), (3), (5) \\ & x_{ij} \leq y_{ij} \leq cap(i, l_0)x_{ij}, \forall v_i, v_j \end{aligned} \tag{6}$$

Note that $cap(i, l_0)$ is a constant for fixed l_0, so inequality (6) is linear. Consequently, ILP(l_0) is an integer linear program, and we can use existing solvers such as lp_solve, CPLEX, and Gurobi, to solve it.

To find the optimal spanning tree for the subproblem, we can perform binary search over all possible lifetimes. However, a lifetime is real number, making it impossible to enumerate. Fortunately, the following observation can narrow down the number of possibilities to within n^2 values.

Observation 1. *Given a sensor network with n sensor nodes (excluding the sink), and a subproblem P on block b consisting of n_1 sensor nodes, the lifetime of any spanning tree for subproblem P is in the set*

$$L = \left\{ \frac{e_i}{j(R_x + T_x) + T_x} \mid v_i \in b, j = 0, 1, \ldots, n - 1 \right\}.$$

To see this, simply note that the lifetime is taken at some vertex in b, which is enumerated by v_i, and the number of possible descendants of any node is less or equal to $n - 1$, which is enumerated by j. Since $|L| \leq nn_1$, we can sort L, and then perform binary search on L to find the optimal lifetime.

Note that, for a given subproblem, constraints (2), (3), (5), and the left side of (6) (i.e., $x_{ij} \leq y_{ij}$) are the same for ILP(l_0) with different l_0 values. This observation can speed up the model creation process across iterations.

The formulation in this section alone can already solve the lifetime maximization problem. We can treat the original network as one subproblem and use the formulation to find the optimal value. Its performance will be evaluated in Section V with the name ILP-B.

C. Reducing the Number of Possible Lifetimes

Observation 1 shows that there are at most nn_1 possible lifetimes when the network size and block size are n and n_1 respectively. We can further reduce this number by the following methods. For ease of presentation, we refer to the optimal lifetime of the original problem as global optimal lifetime.

First, we can find a lower bound of the global optimal lifetime to delete some small candidate values. This is done by randomly generating a spanning tree for the original problem, computing its lifetime as lower bound, and deleting all values strictly smaller than the lower bound.

Second, we can maintain an upper bound of the global optimal lifetime to delete large candidate values. The upper bound can be initialized to $e_{\min}/T_x$, where $e_{\min}$ is the minimum initial energy of all nodes. The reason is that the node with minimum initial energy needs to at least send out one message, so the global optimal lifetime cannot be greater than $e_{\min}/T_x$. In addition, the optimal lifetime of any subproblem is an upper bound of the global optimal lifetime. Thus, after each subproblem is solved, we can update the upper bound of the global optimal lifetime. The benefit of maintaining an upper bound is that we can delete all values in L greater than the currently estimated upper bound during solving a subproblem.

Third, we suggest to solve subproblems in ascending order of problem size. This is because, generally, the upper bound becomes better when more subproblems are solved.Large subproblems lead to large integer problems, and it is more important to make the corresponding search range smaller.

D. The Overall Algorithm

The above designs are implemented in Algorithms 1 and 2. Algorithm 1 implements the problem decomposition step, and Algorithm 2 implements binary search and integer linear programming. The techniques for reducing search range is implemented in both algorithms.

Specifically, Algorithm 1 decomposes the network graph, creating subproblems. It sorts subproblems in ascending order of the number of edges, to implement techniques introduced in Section IV-C. Then, it initializes the lower bound and upper bound of the global optimal lifetime. Finally, it considers subproblems one by one and solves each one in the **for** loop, updates the upper bound whenever possible, and merges solutions of the subproblems to obtain the global optimal solution. To solve a subproblem, if the subproblem is a bridge, it can be trivially solved. Otherwise, Algorithm 1 calls BinarySearch to find the optimal solution.

Algorithm 2 constructs the set of possible lifetimes, and then performs binary search. Besides filtering out impossible optimal values in line 5, it adds two additional values up and $up + 1$ in line 7. Including up into L is necessary, because it may happen that the optimal lifetime of the subproblem is strictly larger than up. In this case, if up is not in L, the algorithm will find a value slightly smaller than up. The reason for including $u + 1$ is related to the subsequent **while** loop. It ensures that the largest number in L is greater than the global optimal lifetime, so that the loop invariant holds prior to the **while** loop. By induction on the number of iterations, we can prove that $L[j]$ is greater than the global optimal lifetime and $L[i]$ is not, which is the loop invariant. When the **while** loop terminates, we have $i = j - 1$, so $L[i]$ is the desired lifetime.

Combining the above analysis, we have the following theorem.

Algorithm 1: Constructing maximum lifetime data gathering tree

Input: graph G, sink s
Output: an optimal spanning tree T and its lifetime l

1 decompose graph G into blocks using the algorithm in [11];
2 sort the blocks in ascending order of the number of edges, and let the sorted blocks be $b_1, b_2, \ldots, b_t$ with sink nodes $s_1, s_2, \ldots, s_t$;
`// initialize the lower and upper bound of the global optimal lifetime`
3 $low \longleftarrow$ lifetime of an arbitrary spanning tree of G;
4 $up \longleftarrow e_{\min}/T_x$;
5 **for** $i \longleftarrow 1$ **to** t **do**
6 **if** b_i *contains only one edge* **then** `// a bridge`
7 $T_i = b_i$;
8 $l_i \longleftarrow \frac{e_j}{d_j \cdot R_x + (d_j+1)T_x}$, where $v_j \neq s_i$;
 `/* d_j is calculated during graph decomposition */`
9 **else**
10 $[T_i, l_i]$ = BinarySearch(b_i, s_i, low, up);
 `// update upper bound`
11 **if** $l_i < up$ **then**
12 $up \longleftarrow l_i$;
13 $l \longleftarrow \min_{i=1}^{t} l_i$;
14 $T \longleftarrow \bigcup_{i=1}^{t} T_i$;
15 return T, l;

Theorem 1. *Algorithm 1 solves the lifetime maximization problem defined in Problem 1.*

V. Simulations

We compare the running time of our algorithm with previous algorithms on randomly generated sensor networks. All sensors are randomly distributed in a $100m \times 100m$ square field with the sink node located at the center. The initial energy from each sensor is uniformly drawn from $[1, 10]J$. We set $R_x = 3.33 \times 10^{-4}J$ and $R_x = 6.66 \times 10^{-4}J$ according to [3], [5], [6]. An edge connects two nodes if their distance is not greater than $20m$. We implement the following three algorithms for comparison.

- DEnum [3], the previous exact algorithm that incorporates graph decomposition in a back-tracking algorithm. If a graph is already 2-connected, DEnum tests whether an edge belongs to the final spanning or not, to reduce the problem size, followed by graph decomposition on smaller problems.
- ILP-B, integer linear programming with binary search. This is Algorithm 2 with the network graph and the sink

Algorithm 2: BinarySearch

Input: block b, sink s, lower bound low, upper bound up
Output: an optimal spanning tree T of b, and lifetime l

```
// construct the set of possible lifetimes
L ⟵ ∅;
foreach v_i ∈ b do
    for j ⟵ 0 to n do
        t ⟵ e_i / (j(R_x+T_x)+T_x);
        if t ≥ low ⋀ t ≤ up then // refine the range
            L ⟵ L ∪ {t};
L ⟵ L ∪ {up, up+1};        // two more lifetimes
sort L in ascending order;
// initialize i, j, mid for the binary search
i ⟵ 1, j ⟵ |L|, mid ⟵ ⌊(i+j)/2⌋;
prepare ILP, add constraints (2), (3), (5), and the left side of (6);
// loop invariant: L(j) is greater than the
global optimal lifetime, and L(i) is not
while i < j − 1 do
    l_0 ⟵ L[mid];
    compute cap(k, l_0) for all v_k ∈ b;
    add the right side of (6) into the ILP model, finishing the
    construction of ILP(l_0);
    T' ⟵solve ILP(l_0);        // return a non-empty
    solution if feasible
    if T' ≠ ∅ then // feasible
        i ⟵ mid;
        T ⟵ T';
    else // infeasible
        j ⟵ mid
    mid ⟵ ⌊(i+j)/2⌋;
    delete the constraints added in line 13 from the ILP model;
return T and its lifetime L[i];
```

node as input. The lower bound is $-\infty$ and the upper bound is $+\infty$.

- ILP-BD, the proposed algorithm in Algorithm 1. It uses graph decomposition, and integer linear programming with binary search.

The difference between ILP-B and ILP-BD is that ILP-BD has three more designs. First, ILP-BD contains a graph decomposition step to decompose the original problem into subproblems. Second, ILP-BD solves subproblem in ascending order of problem size. Third, ILP-BD maintains lower and upper bound of the global optimal lifetime to reduce the number of possible lifetimes. The improvements of performance come from all three aspects. All algorithms are written in Java as single-thread programs. The environment for running them is listed in Table I. To implement ILP-B and ILP-BD, we use the popular open-source software lp_solve [13] as the underlying ILP solver.

We vary the number of nodes from 30 to 50 with increments of 5. For each node number, we generate 20 network instances, giving a total of 100 networks. Note that only connected networks are considered. For every network instance, we use three algorithms to find the optimal, and compare their execution times. To finish the experiments within reasonable time, we set the maximum execution time to be 2 hours. If an algorithm fails to solve a network instance within two hours, we kill the thread and try the next network instance.

TABLE I
SIMULATION ENVIRONMENT

Operating System	Windows 7 64 bit
CPU	Intel(R) Core(TM) i5-3230m
Memory	4GB
Java Runtime Environment	JRE 1.8.0
ILP solver	lp_solve 5.5 [13]

Figure 2 shows the average running time of all three algorithms. Note that for failed networks, we assume that the running time is 2 hours. We can see from the figure that the average running time of all three algorithms increase rapidly with the increase of node number. Indeed, with the increase of node number, DEnum needs to examine more solutions, and both ILP-B and ILP-BD need to consider larger integer linear programs. Among the three algorithms, ILP-BD runs significantly faster than both DEnum and ILP-B. The superiority of ILP-BD becomes more obvious when the number of nodes is greater than 40. At this point, ILP-BD only takes one third of the time needed by DEnum. By comparing ILP-BD with ILP-B, we can see that adding the decomposition indeed helps in saving computation time.

Figure 3 shows the number of failed networks of each algorithm. We can see that DEnum performs well on networks with 30 and 35 nodes, but fails on most networks with 50 nodes. ILP-B performs worse than DEnum when $n = 40$, but becomes better when $n \geq 45$. ILP-BD is consistently better than the other two algorithms. To compare the running time of three algorithms on fixed network instance, we draw scatter plots in Figures 4, 5 and 6. We can observe that longer running time of one algorithm does not necessarily imply longer running time of the other algorithm. This indicate that it is difficult to identify 'hard' problem instances. For example, there exists an instance in Figure 5 where ILP-BD takes roughly 100 seconds, but DEnum takes 1 second. This observation coincides with that made in [3].

Finally, we study whether we can terminate an algorithm early. For example, if the algorithm has run for 1 hour and has not found the optimal solution, can we terminate it? To this end, we study the probability of events that an algorithm has run for x seconds on a network without solving the problem, and it will finally solve the network if we allow it to use up its two hours budget. Figure 7 shows the probability of three algorithms. Note that for clarity, we calculate probabilities every 5 minutes. We can see that, it is safe to terminate DEnum at 80 minutes, ILP-B at 40 minutes and ILP-BD at 70 minutes, because they will not solve the problem anyway.

VI. CONCLUSIONS

In this paper, we propose an exact algorithm using integer linear programming to find an optimal spanning tree of wireless sensor network. Our algorithm incorporates the idea of graph decomposition, integer linear programming, and binary

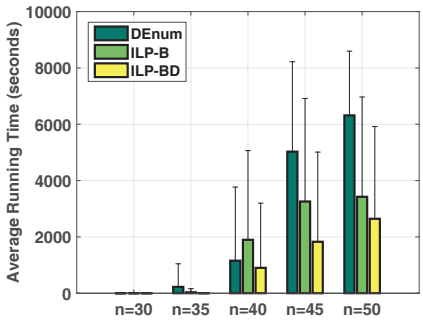

Fig. 2. Average running time of three algorithms.

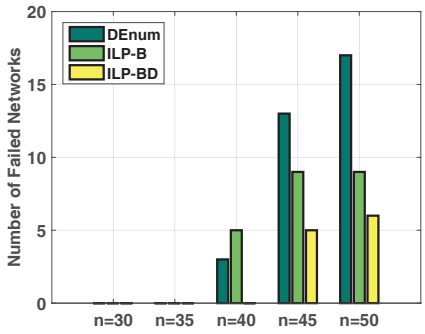

Fig. 3. Failed networks of three algorithms.

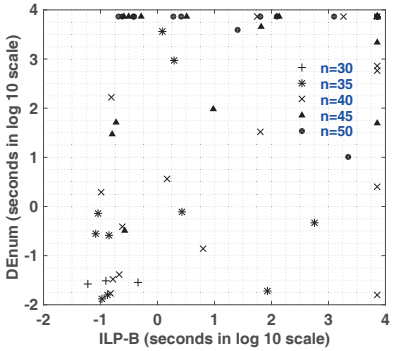

Fig. 4. Scatter plot of running time of DEnum and ILP-B.

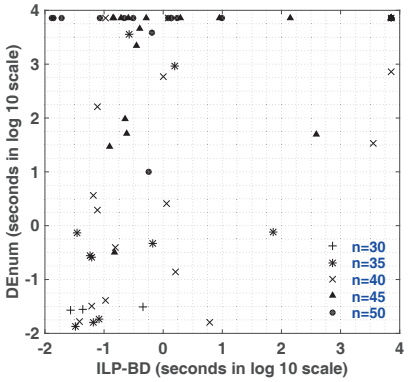

Fig. 5. Scatter plot of running time of DEnum and ILP-BD.

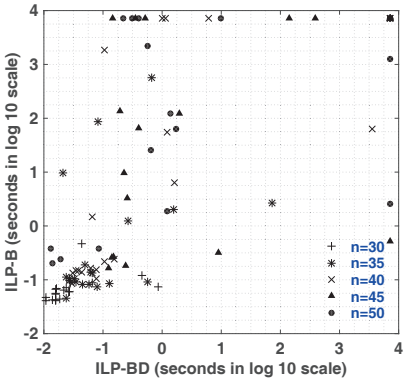

Fig. 6. Scatter plot of running time of ILP-B and ILP-BD.

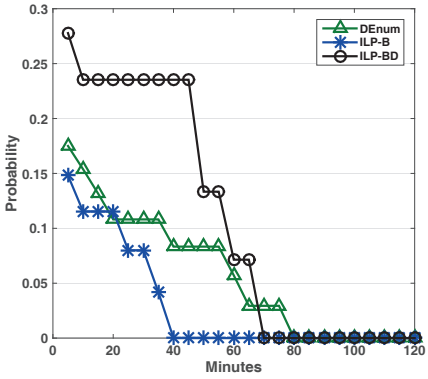

Fig. 7. The probability of a network being solved within two hours, given that it has not been solved by a given time.

search. It applies graph decomposition to reduce problem size, and relies on integer linear programming solvers to perform the search. To deal with non-linear constraints, we propose a binary search step to transform the non-linear optimization problem into multiple feasibility problems in integer linear programming form. The speedup comes from both reduced problem size and the recent progress in operations research literature for faster ILP solvers. Numerical results suggest that the proposed algorithm runs much faster than the previous algorithm. For networks with 50 nodes and a time budget of 2 hours on each network, our algorithm solved 70% networks, while the previous algorithm can only solve 6%.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (61502232, 61502251, 61672283, 61602238), China Postdoctoral Science Foundation (2015M570445, 2016T90457), and the Innovative Project for Undergraduate Students of Nanjing University of Aeronautics and Astronautics (2016CX01627).

REFERENCES

[1] C. Buragohain, D. Agrawal, and S. Suri, "Power aware routing for sensor databases," in *Proceedings of INFOCOM*, 2005.

[2] R. C. Read, "Bounds on backtrack algorithms for listing cycles, paths and spanning trees," *Networks*, vol. 5, pp. 237–252, 1975.

[3] X. Zhu, X. Wu, and G. Chen, "An exact algorithm for maximum lifetime data gathering tree without aggregation in wireless sensor networks," *Wireless Networks*, vol. 21, no. 1, pp. 281–295, 2015.

[4] J. Liang and T. Li, "A maximum lifetime algorithm for data gathering without aggregation in wireless sensor networks," *Applied Mathematics and Information Sciences*, vol. 7, no. 5, pp. 1705–1719, 2013.

[5] Y. Wu, Z. Mao, S. Fahmy, and N. Shroff, "Constructing maximum-lifetime data-gathering forests in sensor networks," *IEEE/ACM Transactions on Networking*, 2010.

[6] X. Zhu, G. Chen, S. Tang, X. Wu, and B. Chen, "Fast approximation algorithm for maximum lifetime aggregation trees in wireless sensor networks," *INFORMS Journal on Computing*, vol. 28, no. 3, pp. 417–431, 2016.

[7] M. Shan, G. Chen, D. Luo, X. Zhu, and X. Wu, "Building maximum lifetime shortest path data aggregation trees in wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 11, no. 1, pp. 11:1–11:24, 2014.

[8] J. Crowcroft, M. Segal, and L. Levin, "Improved structures for data collection in wireless sensor networks," in *Proceedings of INFOCOM*, 2014.

[9] Z. Lu, W. W. Li, and M. Pan, "Maximum lifetime scheduling for target coverage and data collection in wireless sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 2, pp. 714–727, 2015.

[10] L. Xu, X. Zhu, H. Dai, X. Wu, and G. Chen, "Towards energy-fairness for broadcast scheduling with minimum delay in low-duty-cycle sensor networks," *Computer Communications*, vol. 75, pp. 81–96, 2016.

[11] J. E. Hopcroft and R. E. Tarjan, "Efficient algorithms for graph manipulation," *Communications of The ACM*, 1971.

[12] B. Gavish, "Formulations and algorithms for the capacitated minimal directed tree problem," *Journal of the ACM*, vol. 30, no. 1, pp. 118–132, 1983.

[13] lp solve. [Online]. Available: http://lpsolve.sourceforge.net/5.5/