Maximum Lifetime Continuous Query Processing in Wireless Sensor Networks

Konstantinos Kalpakis[*,a], Shilang Tang[a]

[a]*Computer Science & Electrical Engineering Department*
University of Maryland Baltimore County

Abstract

Monitoring applications emerge as one of the most important applications of wireless sensor networks (WSNs). Such applications typically have long–running complex queries that are continuously evaluated over the sensor measurement streams. Due to the limited energy of the sensors in WSNs, energy efficient query evaluation is critical to prolong the system lifetime — the earliest time that the network can not perform its intended task anymore.

We model complex queries by expression trees and consider the problem of maximizing the lifetime of a wireless sensor network for the continuous in–network evaluation of an expression trees $\mathcal{T}$, so the value of its root is available at the base station. In–network evaluation means that the evaluation of the operators of $\mathcal{T}$ may be pushed to the network nodes, and continuous means the repeated evaluation of $\mathcal{T}$ (once at each round). Continuous in–network evaluation of $\mathcal{T}$ entails addressing the following two coupled aspects of the problem: (a) the placement of the operators, variables, and constants of $\mathcal{T}$ to network nodes, and (b) the routing of their values to the appropriate network nodes that needed them to evaluate the operators.

We analyze the complexity and provide a simple and effective algorithm for the placement of the nodes of $\mathcal{T}$ onto the sensor nodes of a WSN. Our algorithm of operator placement attempts to minimize the total amount of data that need to be communicated. A placement of $\mathcal{T}$ induces a certain Maximum Lifetime Concurrent–Flow (MLCF) problem. We provide an efficient algorithm that finds near–optimal integral solutions to the MLCF problem, where a solution is a collection of paths on which certain amount of integral flow is routed. Our approach to the continuous in–network evaluation of $\mathcal{T}$ consists of utilizing both our placement and routing algorithms above.

[*]Corresponding author.
Email addresses: kalpakis@csee.umbc.edu (Konstantinos Kalpakis), stang2@csee.umbc.edu (Shilang Tang)

We demonstrate experimentally that our approach consistently and effectively find the maximum lifetime solutions for the continuous in–network evaluation of expression trees in wireless sensor networks.
Key words: Wireless sensor networks, in–network query processing, energy management, lifetime maximization, operator placement, routing

---

1. Introduction

Remote monitoring applications are one of the most attractive applications of wireless sensor networks. Such applications, like environmental monitoring and building surveillance, normally have long running queries over the data streams that are continuously generated by sensors near the points of interest. For example, one such query can be found in volcano monitoring application — report the current activity level every five minutes, which is measured by processing and correlating the data streams generated by sensors on surface vibration, air pressure and temperature, gas density change, magnetic variance, and etc. How to energy efficiently process these long–running queries is a critical problem to the success of the deployment and operation of wireless sensor networks, since often replenishing the energy of the sensors by replacing their batteries is cost prohibitive or even infeasible.

In this paper, we consider the task of the continuous evaluation of long–running complex queries in wireless sensor networks. Such queries have multiple function–dependent operators and require repeated evaluation once per each round. Due to the disparity between the amount of data generated by the sensors and the amount of data the network can communicate before the sensors deplete their energy, we aim to push the queries into the network for processing [18]. We model a query with a rooted expression directed acyclic graph (DAG) $\mathcal{Q}$, whose internal nodes are operators (functions) with children as their operands, and its leaves are constants or variables. Each vertex of $\mathcal{Q}$ has a size for its value and a set of candidate network nodes to which it may be placed. Each variable vertex of $\mathcal{Q}$ has a set of source sensor nodes, whose measurements are used to assign values to that variable.

The continuous in–network evaluation of a rooted expression DAG $\mathcal{Q}$ entails addressing the following two aspects of the task: (a) the placement of the operators, variables, and constants of $\mathcal{Q}$ to network nodes, and (b) the routing of the operand values to the appropriate network nodes that needed them to evaluate the operators. These two aspects are coupled because the placement of $\mathcal{Q}$ imposes certain source–destination routing requirements among the sensor nodes, and the manner in

which routing is performed can have a major impact on placement decisions.

While there are many important optimization goals for the continuous in–network evaluation of queries (eg. response time, reliability, etc), we focus on maximizing the system lifetime — the time until the sensor network losses its ability to perform its intended task due to depletion of energy at (some of) its sensors, and analyze how to decouple the two aspects of the task at hand. We find, as shown in our experimental evaluation, that having a near optimal solution to the routing aspect effectively decouples the routing and placement aspects, and therefore allows us to solve these two aspects one at a time.

To find a near optimal solution to the placement aspect of the task, we consider the minimum communication cost placement (MCP) problem. The MCP problem is that of minimizing the total amount of data communicated among network nodes, which have been assigned one or more vertices of $\mathcal{Q}$, during a single evaluation of $\mathcal{Q}$. We show that the MCP problem is MAX–SNP hard even when $\mathcal{Q}$ is a tree of height 1 with unit cost edges. We describe a simple and efficient greedy heuristic, which we call the GREEDYMCP algorithm, for the MCP problem, and show practically useful cases under which GREEDYMCP finds provably optimal solutions to the MCP problem.

To find a near optimal solution to the routing aspect of the task, we solve a maximum lifetime concurrent flow (MLCF) problem. The MLCF problem is the problem of maximizing the lifetime of a system that concurrently pushes flow to satisfy the data rate demands for a given set of source–destination pairs. We provide an efficient and simple algorithm for the MLCF problem, which we call the ALGRSM–MLCF algorithm, that finds at most $n + N$ paths that maximize the fractional system lifetime T_o for satisfying the concurrent flow data demands for N source–destination pairs in a network with n nodes. By rounding down that fractional solution, we get an α–optimal integral concurrent flow solution to the MLCF problem, where $\alpha = (T_o - n - N + 1)/T_o$. Since often in practice $T_o \gg n + N$, $\alpha \approx 1$. We experimentally show that ALGRSM–MLCF outperforms existing routing algorithms that could be applied to the MLCF problem, in terms of system lifetime and energy overhead. ALGRSM–MLCF is an iterative algorithm based on the Revised Simplex Method (RSM).

Our approach for the continuous in–network evaluation of query $\mathcal{Q}$ consists of using both GREEDYMCP and ALGRSM–MLCF. First, we use GREEDYMCP to find a placement of $\mathcal{Q}$ on the network, and we use ALGRSM–MLCF for routing all the data values that need to be communicated. We show, through an extensive experimental evaluation, that our approach consistently finds the maximum

lifetime solution for the continuous in–network evaluation of complex queries in wireless sensor networks. Although we take a centralized approach to tackle the task at hand, we only require the knowledge of two network metadata — the network topology and the initial energy of sensors, which are very useful to many other network tasks as well. Note that the small size of the routing solution found by our approach limits the overhead of distributing the routing information to the sensors.

In summary, for the task of continuous in–network processing of complex queries in wireless sensor networks (WSNs), the original contributions of this paper are as follows:

- theoretically analyze the complexity of the MCP problem, the problem of placing expression DAGs on WSNs with minimum total communication cost.
- provide a greedy heuristic GREEDYMCP, for the MCP problem. GREEDYMCP finds provably optimal solutions in practical useful cases.
- provide a simple and effective algorithm, ALGRSM–MLCF, that finds near–optimal integral solutions to the maximum lifetime concurrent flow (MLCF) problem in WSNs. ALGRSM–MLCF outperforms existing routing methods.
- we find that having near optimal solutions to the MLCF problem enables the decoupling of the placement and routing aspects of the task at hand.
- our approach, consisting of using GREEDYMCP and ALGRSM–MLCF together is both effective and efficient at maximizing the system lifetime.

The rest of the paper is organized as follows. We review related work in section 2, and then in section 3 we give the necessary preliminaries. We describe our GREEDYMCP algorithm for the placement of expression DAGs into WSNs in section 4, and show that GREEDYMCP finds optimal solutions to MCP problem instances under certain conditions. We analyze the complexity of the MCP problem in section 4.2 and show that the MCP problem is MAX SNP–hard even for trees of height 1 and unit cost edged provided they have restricted vertices. We then turn our attention to the routing of operands, and we present our ALGRSM–MLCF algorithm for the MLCF problem in section 5. We discuss the results from our experimental evaluation of the proposed approach in section 6. We conclude in section 7.

2. Related work

Pietzuch et al [26] consider network-aware operator placement in conventional distributed stream processing systems. In similar network settings, Ahmad et al [1] give three operator placement algorithms for constructing a query processing overlay network and compare their performance. The network considered by them is internet–style and consists of nodes with ample computational power, which is very different from the energy–limited wireless sensor networks we consider.

In wireless sensor networks, the notion of in–network processing was first introduced by Intanagonwiwat et al [18] to opportunistically eliminate duplicates in the context of directed diffusion. Gehrke and Madden et al [11, 15, 24] are among the first to integrate query processing and sensor networks so tasking sensor networks can be easily done through declarative queries. In the Cougar project [11], a layered architecture of sensor data management is proposed for presenting the sensor network as a distributed database system. In TinyDB [24], a framework of query processing in WSN is introduced for addressing issues of when, where, and how often data is sampled and which data is delivered in wireless sensor networks. Energy efficiency is one of the major factors considered in [11, 24], but not with the goal of maximizing the system lifetime. In addition, query operators in [11, 24] are modeled from a functionality perspective and often are rather simple operators (aggregation, filter, etc), while in our work we model the operators from a communication perspective with the consideration of their optimal placement.

Ren et al [27] consider quality aware processing of simple aggregate queries (e.g. compute the average, min, max of measurements of sensors in a rectangular area of interest). A centralized algorithm is proposed to find a subset of sensors whose measurements are collected using reactive routing to the base station to compute a probabilistic answer. Hu et al [17], expanding upon the work of Olston and Widom [25], are concerned with approximate answers to continuous aggregate queries (sum, mean, count, etc). They provide a method to allocate a user–specified acceptable tolerance to a query's answer as tolerance ranges for the sensors. Subsequently, a sensor sends its measurement to the base station if it falls outside its tolerance range. These two works are different from ours in many respects: we consider complex queries with various operators besides min, max, avg, we provide accurate answers to queries, and we seek to optimize the system lifetime directly.

Many researchers have advocated the use of data–centric techniques that allow for efficient in–network storage and retrieval of named data using queries [16]. A number of data–centric push–pull query processing techniques have been proposed and examined [6, 8, 23, 28, 29, 31], which can

be categorized to two main approaches: structured and unstructured, which can be represented by the geographic hash–based data centric storage technique [29] and the comb–needle method [23] respectively. Kapadia and Krishnamachari [20] present a comparative mathematical analysis of these two approaches based on two types of simple one–shot queries (ALL–type and ANY–type) in single–sink square–grid sensor networks, and later on, Ahn and Krishnamachari [2] find that the scalability of a data–centric sensor networks performance depends upon whether or not the increase in energy and storage resources with more nodes is outweighed by the resulting application–specific increase in event and query loads.

Bonfils et al [5] consider the placement of operators of a query expression tree to the nodes of a sensor network to minimize the total communication cost of evaluating such a tree. For any pair of parent–child operators in the query tree, the induced communication cost is the product of the length of a shortest path between the nodes where these operators are placed and the data rate from the child to its parent. They provide a distributed protocol that attempts to refine the placement by continuously walking through neighbor nodes to accommodate the data rate change. The overhead of message exchange generated by walking through neighbor nodes is not considered in [5]. Our ALGRSM–MLCF algorithm differs from theirs in that we allow data to be routed over multiple paths rather than a single path, and that we seek to optimize the system lifetime rather than the total communication cost. Restricting routing of child–parent data to a single path instead of allowing multiple paths can have a detrimental effect on the lifetime. As can been seen in Fig. 10, our approach achieves much better lifetime in all the instances over the best placement when using shortest path routing.

Srivastava et al [30] consider the problem of placing operators onto a hierarchy of network nodes with progressively increasing computational power and network bandwidth, such that the total cost of computation and communication is minimized. We assume a different network model in which sensors are homogeneous and are energy limited, and a different goal of optimizing the system lifetime, which does not necessarily result from minimizing the total cost of computation and communication.

Garg and Konemann [14] describe an iterative algorithm with provable approximation ratio for solving the maximum concurrent multicommodity flow problem, whose LP formulation is different from MLCF. Their objective is to maximize the total network flow under limited capacity of edges, while ours is to maximize network lifetime under limited energy of nodes. In addition, the number of routing paths used in the solution of our ALGRSM–MLCF algorithm is bounded, while the

algorithm in [14] finds solutions which may use as many routing path as the number of iterations. Having fewer routing paths is important in practice, since the overhead of distributing the relevant routing information to the sensors is kept smaller with fewer paths.

Chang and Tassiulas [7] propose a shortest path routing algorithm for maximum lifetime data gathering using link costs that reflect both the communication energy consumption rates and the residual energy levels at the two end nodes of each link. Though they also formulate the routing problem as a linear programming problem, they only use it to obtain the optimal lifetime to compare with the lifetime achieved by their proposed algorithm. We formulate that optimal routing problem with a different linear program, and our ALGRSM–MLCF algorithm effectively solves the LP to obtain the optimal routing directly. In addition, the performance of the algorithm in [7] is dependent on the parameters used by the algorithm, while ALGRSM–MLCF is non–parametric.

Wu at al [32] consider constructing transmit/receive schedules for the sensors to collect data using a given routing tree . They describe a method for allocating transmit and receive slots to the sensors so that the collisions during transmits are reduced. This work is complementary to our work. Wu et al [32] assume a routing tree, while our approach constructs a placement and routing to be used for evaluation of the continuous queries. On the other hand, our ALGRSM–MLCF algorithm does not consider conflicts during transmissions by the sensors. Deriving a detailed transmit/receive schedule for our placement/routing is important, and their method could be a step towards that goal.

3. Preliminaries

We provide definitions and notation we utilize throughout the paper, including a simple model for wireless sensor networks and the concept of expression trees.

3.1. Common definitions and notations

Given a graph G, we denote its vertex set with $V[G]$ and its edge set $E[G]$. For brevity, we often write $v \in G$ instead of $v \in V[G]$ for a vertex v, and $ij \in G$ instead of $ij \in E[G]$ for an edge ij. Let $G[V'] = (V', E[G] \cap V' \times V')$ be the subgraph of G induced by a subset of its vertices $V' \subseteq V[G]$. We denote with $G[A] = (V, A)$ the subgraph of G induced by a subset of its edges $A \subseteq E[G]$.

Consider an edge–weighted directed graph G with edge weights $\mathbf{w} \in \mathbb{R}^{|E[G]|}$. The *edge–contraction* of G by an edge $ij \in E[G]$ is the graph obtained from G by collapsing (merging) vertex i into vertex j. Collapsing a vertex i into vertex j entails the following three steps: (a) for each edge $ki \in E[G]$

with weight w_{ki}, if $kj \notin E[G]$ then add the edge kj with weight w_{ki} to G, otherwise increase the weight of edge kj by w_{ki}, (b) for each edge $ik \in E[G]$ with weight w_{ik}, if $jk \notin E[G]$ then add the edge jk with weight w_{ik} to G, otherwise increase the weight of edge jk by w_{ik}, (c) delete vertex i and any self–loop (edge jj) from G. Consider a partition $\Pi = \langle V_1, V_2, \ldots, V_m \rangle$ of the vertex set $V[G]$ of G into m blocks V_i, for some $m \geq 1$. We define the *contraction* graph of G by Π to be the following edge–weighted directed graph G_Π. The vertices of G_Π are the blocks of Π, i.e. $V[G_\Pi] = \{V_1, V_2, \ldots, V_m\}$. There is an edge $ij \in E[G_\Pi]$ iff there exists a cut–edge from V_i to V_j in G, i.e. an edge with $uv \in E[G]$ with $u \in V_i$ and $j \in V_j$. The weight of each edge $ij \in E[G_\Pi]$ is equal to the sum of the weights of the cut–edges of G from V_i to V_j. Note that G_Π is isomorphic to the graph obtained from G by contracting all of its intra–block (uncut) edges.

Given an instance $\mathfrak{I}$ of an optimization problem, let $\text{OPT}(\mathfrak{I})$ and $\text{SOL}(\mathfrak{I})$ be an optimal and a feasible solution to $\mathfrak{I}$, respectively. For brevity, $\text{OPT}(\mathfrak{I})$ and $\text{SOL}(\mathfrak{I})$ will also indicate the value of the corresponding solution. The relative error of a solution $\text{SOL}(\mathfrak{I})$ is equal to $|\text{OPT}(\mathfrak{I}) - \text{SOL}(\mathfrak{I})| / \text{OPT}(\mathfrak{I})$ and its approximation ratio is equal to $\text{SOL}(\mathfrak{I})/\text{OPT}(\mathfrak{I})$. We refer to continuous (integral) solutions to instances of optimization problems, whenever fractional values for their unknowns are allowed (not allowed).

We denote vectors and matrices with lower and upper case bold letters, e.g. $\mathbf{x}$ and $\mathbf{A}$ is a vector and matrix respectively. The support of a vector $\mathbf{x}$ is defined as $\mathbb{I}(\mathbf{x}) = \{i \ : \ x_i \neq 0\}$. Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, we say that $\mathbf{x}$ *dominates* $\mathbf{y}$ and write $\mathbf{x} \geq \mathbf{y}$, if $x_i \geq y_i$ for $i = 1, 2, \ldots, m$. We say that $\mathbf{x}$ is *lexicographically larger (smaller)* than $\mathbf{y}$ if the smallest index non–zero component of $\mathbf{x} - \mathbf{y}$ is positive (negative). Since a matrix with a single column is a vector, many notations/operations on matrices naturally extend to vectors.

We denote the transpose of a matrix $\mathbf{A}$ with $\mathbf{A}^T$. Given an $n \times m$ matrix $\mathbf{A}_{n\times m}$ and two index sequences $I \subseteq \{1, 2, \ldots, n\}$ and $J \subseteq \{1, 2, \ldots, m\}$, we denote by $\mathbf{A}_{I,J}$ the sub–matrix $\mathbf{X}_{|I|\times|J|} = (x_{ij})$ of $\mathbf{A}$ where $x_{ij} = a_{I_i,J_j}$, and by $\mathbf{A}_J$ the sub–matrix $\mathbf{X}_{n\times|J|} = (x_{ij})$ of $\mathbf{A}$ where $x_{ij} = a_{iJ_j}$. Since a set can always be converted into a sequence by listing its elements in increasing order, we extend the notations $\mathbf{A}_{X,Y}$ and $\mathbf{A}_Y$ to index sets.

3.2. A model for wireless sensor networks

Consider a wireless sensor network (WSN) with n nodes. One node, denoted b, is designated as the base station, with the remaining nodes being sensors. Sensors are identified by unique IDs and are

assumed to have limited non–replenishable energy while the base station has no energy limitations. Sensors close to the points of interest are called data sources, and they monitor and generate sensing data at a predefined data rate. A long running query that acquires and processes the data generated by the data sources is posed at the base station and is parsed as an expression DAG. Time is discrete, and the data rate is defined as the number of packets transmitted by the sensor node during each time period. For simplicity, we assume the data packet size is fixed. The system lifetime T is the earliest time at which the wireless sensor network losses its ability to perform its intended task (e.g. query evaluation) due to the depletion of the available energy at one or more sensors.

The topology of the wireless sensor network is modeled as a directed graph (digraph) $G = (V, E)$, with $V = \{1, 2, \ldots, n\}$ and $E \subseteq V \times V$. There exists an edge $ij \in E$ whenever i can successfully send a packet to j. Let d_{ij} be the distance between i and j. Let τ_{ij} be the energy consumed by node i in order to transmit a single packet to node j, and let r_j be the energy needed to receive such a packet at node j. Let ϵ_i be the energy available at node i. We assume that $\epsilon_b = \infty$. For simplicity we consider sensor networks with a single base station, and assume a single source node near each point of interest. In a real deployment there can be multiple sensors near the same point of interest, and it will be ideal for them to share the data acquisition duties. As long as all the sensors around the same point of interest can hear the communications to any one of them, this can can be easily done as follows. Introduce a new node, i' with $r_{i'} = 0$, to serve as the data source, and then append to G, for each sensor j near the point of interest, two new edges $i'j$ and ji' with both $\tau_{i'j}$ and $\tau_{ji'}$ equal to 0. Sensor networks with multiple base stations can be handled similarly — introduce a new node, b', to serve as the new single base station, and then append to G, for each current base station i, a new edge ib' with $\tau_{ib'} = 0$.

In this paper, we do not consider issues related to signal and channel interference, and the scheduling of transmissions to avoid or reduce such interference. Scheduling of transmissions can be done after the routing decisions have been made, which can increase the latency of query evaluation.

3.3. A model for complex queries and their continuous evaluation

We describe a simple model we will use for the continuous (repeated) processing of complex queries. A query with only one operator (eg add, multiply, etc) or only one kind of an operator (eg. sum, count, min, max, etc) is called simple, otherwise it is called complex. We are interested in complex queries over measurements of sensors in a WSNs. At each round (sensing period), a query is evaluated using the current measurements of the sensors (and possibly prior measurements

as well).

We model queries using rooted expression directed acyclic graphs (DAGs). A rooted expression DAG $\mathcal{Q}$ is a DAG with a single root (vertex without any parents), internal vertices that correspond to operators, and leaves (vertices without any children) that correspond to constants or variables. Each vertex $v \in V[\mathcal{Q}]$ has an associated value of constant but varying size $size(v)$, which is measured in units of packet sizes. The weight of each edge $uv \in E[\mathcal{Q}]$ is equal to $size(u)$. Each operator vertex has one or more operands (children) and its value is a function of its operands. In the AND–operator (OR–operator) model, the evaluation of the function of an operator requires the values of all (any one) of its operands. [1]

Such expression DAGs arise in various domains, e.g. in the continuous evaluation of the TinyDB SQL query

```
SELECT floor, room, AVG(temperature)
FROM sensors
WHERE floor<6
GROUP BY floor, room
HAVING AVG(temperature) > 70
SAMPLE PERIOD 30s;
```

which has a tree DAG representation whose operators are relational algebra operators and leaves are the sensor samples (measurements).

An operator can be evaluated when the values of its required operands are available, in an eager (as soon as possible) or lazy (when its output is requested) way. The evaluation of $\mathcal{Q}$ at a certain round (time) t requires the assignment (binding) of values to (possibly not all) its variables and the evaluation of (possibly not all) its operators in an appropriate order so that the value of its root becomes available to the user.

Let $\mathcal{H}$ be a wireless sensor network in which an expression DAG $\mathcal{Q}$ is to be evaluated. We call $\mathcal{H}$ the *host* graph and $\mathcal{Q}$ the *guest* graph. The value assigned to a variable $v \in V[\mathcal{Q}]$ at time t depends on the measurements up to time t of a set of host network nodes (sensors) $src(v) \subseteq V[\mathcal{H}]$, called

[1] Multiple queries can be modeled by a DAG with multiple roots. Further, more general operator models are possible, e.g. k–out–of–m, etc.

the *data source set* of v. Typically, the data source set of a variable is a singleton or a small set of sensors located nearby each other.

In order to evaluate a guest $\mathcal{Q}$ in a host $\mathcal{H}$, we need to place all the guest vertices to one or more host nodes. Each vertex $v \in V[\mathcal{Q}]$ can be placed at a specified set of candidate host nodes $cands(v) \subseteq V[\mathcal{H}]$. If v is a variable then $cands(v) = src(v)$. Each candidate host node in $cands(v)$ is capable of computing the value of v, once the values of all its required operands, if any, are available to it. A guest vertex v is called *free* if $cands(v) = V[\mathcal{H}]$, *pinned* if $|cands(v)| = 1$, and *restricted* otherwise. We extend the data source set and candidate hosts set to any subset of guest vertices $X \subseteq V[\mathcal{Q}]$ as $cands(X) = \cap_{v \in X} cands(v)$ and $src(X) = \cup_{v \in X} src(v)$. We call an $X \subseteq V[\mathcal{Q}]$ *admissible* iff $cands(X) \neq \emptyset$. Fig. 1 gives an example query expression DAG (tree).

A placement of the guest DAG $\mathcal{Q}$ onto the host network $\mathcal{H}$ is the placement of each vertex $v \in V[\mathcal{Q}]$ to a non–empty set $host(v) \subseteq cands(v)$ of host nodes. Whenever the value of a vertex $v \in V[\mathcal{Q}]$ is needed, a host node $i \in host(v)$ is asked to provide that value – doing so may require sensing or computing at the network node i, or even communication between i and other host network nodes. Fig. 2 provides an example placement of an expression DAG onto a wireless sensor network.

Hereafter, we only consider placements where each guest vertex is placed at exactly one host node, i.e. there is no replication of the guest vertices in the host network.

Consider a placement of the guest DAG $\mathcal{Q}$ onto the host network $\mathcal{H}$. The intersection of candidate sets of the set of guest vertices placed at the same host node is always non–empty. Consequently, by removing the edges between guest vertices placed at different host nodes (cut edges), we get a collection of connected components of the guest graph, such that all the vertices V_i of each such connected component are placed at the same host node u_i. In other words, a placement of the guest $\mathcal{Q}$ onto the host $\mathcal{H}$ induces a partition $\Pi = \langle V_1, \ldots, V_i, \ldots \rangle$ of $V[\mathcal{Q}]$, such that all the guest vertices in V_i are placed at a host node $u_i \in cands(V_i)$. Note that each block V_i of Π is admissible – we call such partitions of $V[\mathcal{T}]$ *admissible*. Furthermore, the amount of data communicated at each round of evaluating $\mathcal{Q}$ equals the total cost of the cut edges induced by the placement at hand. In other words, the total amount of communication required for a single evaluation of $\mathcal{Q}$ for the given placement is equal to the total weight of the contraction $\mathcal{Q}_\Pi$ of $\mathcal{Q}$ by the partition Π. In fact, the total amount of communication required from host node u_i to host node u_j is equal to the weight of the edge V_iV_j in $\mathcal{Q}_\Pi$. Define the cost of a placement to be equal to the total amount of data communicated in each round of evaluating $\mathcal{Q}$. Define the *communications demand graph* $\mathcal{R}$ of the placement at hand to be

the graph obtained from $\mathcal{Q}_\Pi$ by relabeling each vertex $V_i \in V[\mathcal{Q}_\Pi]$ with the host node u_i to which the guest vertices in V_i are placed. The weight of an edge $u_iu_j \in V[\mathcal{R}]$ is equal to the total amount of data that needs to be communicated in one round from host node u_i to host node u_j due to the guest vertices placed at u_i and u_j.

We define the *Minimum communication Cost Placement* (MCP) problem of $\mathcal{Q}$ onto $\mathcal{H}$ as the problem of finding a placement of the guest vertices onto candidate host vertices with minimum cost (amount of data communicated). Since the cost of a placement of $\mathcal{Q}$ onto the host $\mathcal{H}$ is equal to the total weight of cut edges induced by that placement, it follows that the MCP problem is essentially equivalent to the following graph partitioning problem. We define the *Minimum Cost Constrained Partitioning* (MCCP) problem $\langle\, G, f \,\rangle$ as follows: given any edge–weighted graph G and a function $f : V[G] \longrightarrow U$, find a minimum cost set of cut edges (edge–cut) A such that $f(V[G_i]) = \cap_{v \in V[G_i]} f(v)$ is non–empty for each connected component G_i of $G - A$. An optimal solution to the MCCP problem instance $\langle\, \mathcal{Q}, cands \,\rangle$ is also an optimal solution to the MCP problem for the guest $\mathcal{Q}$ onto the host $\mathcal{H}$, and vice versa. We study the complexity of the MCP and MCCP problems in section 4.2.

Given a placement of the guest $\mathcal{Q}$ onto the host $\mathcal{H}$, we now need to find an energy efficient way to satisfy the data routing requirements indicated by the communication demands graph $\mathcal{R}$ in order to maximize the system lifetime. In other words, we need to find the maximum lifetime T together with routes that satisfy the communication demands $T\lambda_i$ for the collection of source–destination pairs (edges) $(s_i, d_i) \in E[\mathcal{R}]$, where the demand λ_i is equal to the weight (amount of data to be communicated in one round) of the edge from host node s_i to host node d_i. This is the Maximum Lifetime Concurrent Flow (MLCF) problem which we consider in section 5.

In this paper, we assume expression DAGs with the AND–operator model. We also assume that the data source set of a variable $v \in V[\mathcal{Q}]$ is a singleton, and thus v is pinned at its single data source host network node. Furthermore, we assume that the root of $\mathcal{Q}$ is pinned at the base station and that there is no replication in the placement of the vertices of $\mathcal{Q}$, i.e. $|host(v)| = 1$ for all $v \in V[\mathcal{Q}]$. Moreover, we assume that the energy consumed to either compute the value of an operator v, or take a measurement and assign a value to a variable v, or furnish the value of a constant v, is negligible compared to the energy consumed for communication for all vertices $v \in V[\mathcal{Q}]$.

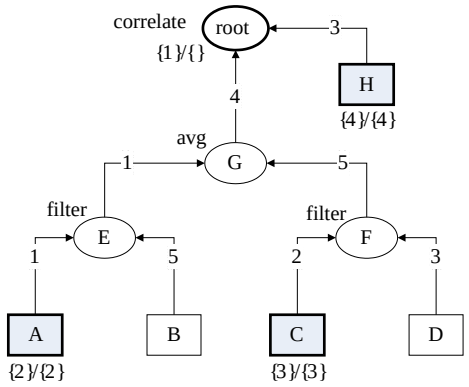


Figure 1: An example expression DAG (tree) $\mathcal{Q}$. Ellipses denote operators, while (shaded) squares denote (variables) constants. We tag each vertex v with $cands(v)$, unless it contains all network nodes – for example vertex B and E. In addition, we tag operators by their associated function, and variables their data source sets ($\{cands(v)\}/\{src(v)\}$). We tag each edge $u \rightarrow v$ with the size of u's value. Pinned vertices are indicated with heavy outlines.

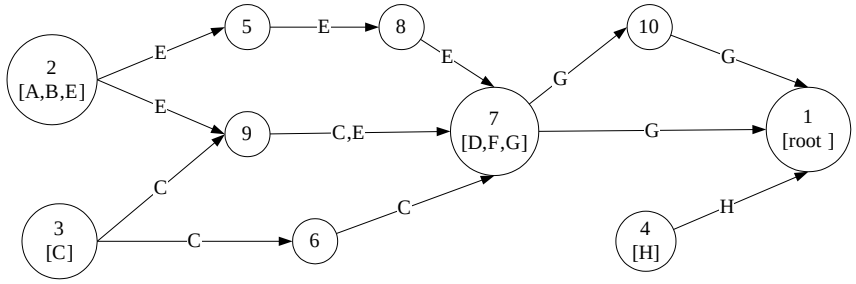


Figure 2: An example placement and operand routing for the expression DAG $\mathcal{Q}$ in Fig. 1 to a WSN with 10 nodes. We indicate the vertices of $\mathcal{Q}$ placed at each node within the brackets, and label the links of the WSN with the operand they are tasked to route. For example, node 2 is hosting the operators/variables A, B, and E, the operand (value of the operator) E is routed from node 2 to node 7 using the two paths $(2, 9, 7)$ and $(2, 5, 8, 7)$, and the operand C is routed from node 3 to node 7 using the two paths $(3, 6, 7)$ and $(3, 9, 7)$.

4. The Minimum communication Cost Placement (MCP) problem

We present a greedy approximation algorithm for the Minimum communication Cost Placement (MCP) problem of a guest query expression DAG $\mathcal{Q}$ onto a host network $\mathcal{H}$, and analyze the computational complexity of the general MCP problem. Our algorithm works by finding an admissible small–cutsize partition of $\mathcal{Q}$, and then placing each one of its blocks to nodes of the host $\mathcal{H}$. Utilizing the idea of edge contractions from Karger's randomized min–cut algorithm [21], we find a small–cutsize admissible partition of $\mathcal{Q}$ by contracting the edges of $\mathcal{Q}$ in decreasing order of their weight, excluding any non–contractible edges. An edge $uv \in \mathcal{Q}$ is contractible iff $cands(u) \cap cands(v)$ is not empty. See Algorithm 1 (GREEDYMCP) for details.

During the edge–contraction process, we maintain for each remaining guest vertex v the following three quantities: the set $Block[v]$ of guest vertices that merged (collapsed) into v, the set $Hosts[v] = \cap_{u \in Block[v]} cands(u)$ of candidate host nodes for all the vertices in $Block[v]$, and the total energy $e[v]$ of the host nodes in $Hosts[v]$. Because we are interested in admissible partitions of $\mathcal{Q}$, we only contract edges $uv \in \mathcal{Q}$ for which $Hosts[u] \cap Hosts[v]$ is not empty.

GREEDYMCP resolves ties between contractible edges $uv \in E[\mathcal{Q}]$ with equal maximum weight based on $e[u]$ and $e[v]$. Other than resolving such ties, GREEDYMCP does not consider the available energy of host nodes nor how host nodes spend their energy — those considerations are left to the MLCF problem. In GREEDYMCP, we assume that a guest vertex of $\mathcal{Q}$ can not be placed (replicated) at multiple host nodes, which we plan to relax in future work. We also assume that there is no limit on how many guest vertices (number of operators) may be placed at each host node. When such limitations exist, one more condition can be added to line 8 to also check if $|Block[u] \cup Block[v]|$ exceeds the load limit of the candidate host nodes of v.

Next, we analyze the running time of GREEDYMCP. The repeat–loop in lines 7–15 is executed $O(V[\mathcal{Q}])$ times, and each iteration takes $O(V[\mathcal{Q}]+V[\mathcal{H}])$ time. Lines 1–6 and 16–20 take $O(V[\mathcal{Q}]+E[\mathcal{Q}])$ time. Hence, the running time of GREEDYMCP is $O(V[\mathcal{Q}] \times (V[\mathcal{Q}] + V[\mathcal{H}]))$.

4.1. Optimality of GREEDYMCP

We show conditions under which GREEDYMCP finds an optimal solution to the MCCP problem, and thus the MCP problem as well. In particular, we show conditions under which the MCCP problem is a matroid (see definitions below), which implies that the greedy approach of GREEDYMCP finds an optimal solution to both the MCCP and MCP problems.

First, we need a few definitions. A set system $\langle U, \mathcal{F} \rangle$, with $\mathcal{F} \subseteq 2^U$, is called an *independence system* if $\emptyset \in \mathcal{F}$ and it has the inheritance property, i.e. $A \subseteq B \in \mathcal{F}$ implies $A \in \mathcal{F}$. The elements of $\mathcal{F}$ are called independent. A *matroid* is an independence system that has the exchange property, i.e. for any two maximal independent sets A and B, if $x \in A - B$ then there exists $y \in B - A$ so that $B + x - y$ is also independent.

Second, we define a particular set system that is useful in studying the MCCP problem. Consider the MCCP problem instance $\langle G, cands \rangle$. Consider the following set system $\mathcal{I}_G = \langle E[G], \mathcal{F}_G \rangle$, where an edge–set $A \subseteq E[G]$ belongs to $\mathcal{F}_G$ if each connected component G_i of $G[A]$ has a non–

Algorithm 1 The GREEDYMCP algorithm for placing a guest $\mathcal{Q}$ to a host $\mathcal{H}$

Input: A guest query expression DAG $\mathcal{Q}$, host network $\mathcal{H}$, and host node initial energies $\boldsymbol{\epsilon}$.

Output: A placement of the guest vertices of $\mathcal{Q}$ to host nodes.

1. **for** each vertex $v \in V[\mathcal{Q}]$ **do**
2. $\quad Hosts[v] \longleftarrow cands(v)$
3. $\quad Block[v] \longleftarrow \{v\}$
4. $\quad e[v] \longleftarrow \sum_{x \in Hosts[v]} \epsilon_x$
5. **for** each edge $uv \in E[\mathcal{Q}]$ **do**
6. $\quad weight[uv] \longleftarrow size(u)$
7. **repeat**
8. $\quad$ find edge $uv \in E[\mathcal{Q}]$ with maximum $(weight[uv], \frac{1}{e[u]}, e[v])$ and $Hosts[u] \cap Hosts[v] \neq \emptyset$
9. $\quad$ **if** no such edge uv exists **then**
10. $\quad\quad$ **break**
11. $\quad Hosts[v] \longleftarrow Hosts[v] \cap Hosts[u]$
12. $\quad Block[v] \longleftarrow Block[v] \cup Block[u]$
13. $\quad e[v] \longleftarrow \sum_{x \in Hosts[v]} \epsilon_x$
14. $\quad$ contract edge uv by collapsing u into v
15. **until true**

    // Place the guest vertices of $\mathcal{Q}$ to host network nodes
16. **for** each remaining vertex $v \in V[\mathcal{Q}]$ **do**
17. $\quad$ let u be a host network node in $Hosts[v] \cap V[\mathcal{H}]$
18. $\quad$ **for** each vertex $x \in Block[v]$ **do**
19. $\quad\quad host(x) \longleftarrow u$
20. **return** placement $\{host(v) \ : \ v \in V[\mathcal{Q}]\}$.

empty set of candidate nodes $cands(V[G_i])$. [2] Note that $A \subseteq E[G]$ belongs to $\mathcal{F}_G$ iff $G[A]$ has no path between any two subsets of vertices $X, Y \subseteq V[G]$ whenever $cands(X) \cap cands(Y)$ is empty.

Lemma 1. *The set system $\mathcal{I}_G$ is an independence system.*

Proof. Since when $A = \emptyset$, each connected component of $G[A]$ is a singleton, it follows that $\emptyset \in \mathcal{F}_G$. Next, we show that $\mathcal{I}_G$ has the inheritance property. Consider two subsets $A \subseteq B$ where B is independent. Since B is a superset of A, each connected component G_i of $G[B]$ consists of one or more connected components G_j of $G[A]$. Since B is independent, each candidate set $cands(V[G_i])$ is non–empty. Consequently, since $V[G_j] \subseteq V[G_i]$, we also have that $cands(V[G_j])$ is non–empty. Hence, A is also independent and it belongs to $\mathcal{F}_G$. ■

Third, we show that the set system $\mathcal{I}_G$ is a matroid when G is a tree, and $cands$ is such that there are no distinct guest vertices pinned to the same element (host node).

Lemma 2. *Suppose that the guest G is a tree with all guest vertices free or pinned, and without any two distinct guest vertices pinned to the same host node. Then, the set system $\mathcal{I}_G$ is a matroid.*

Proof. Since $\mathcal{I}_G$ is already shown to be an independence system, we only need to prove the exchange property.

Consider two maximal independent sets $A, B \subseteq E[G]$. Suppose that there exists an edge $e = uv$ that belongs to $A - B$. Since $e \notin B$, the edge e is between two different connected components G_u and G_v of $G[B]$. Each connected component G_u and G_v contains exactly one pinned vertex, say x and y respectively. Note that $cands(x) \cap cands(y) = \emptyset$, and at least one of x or y is not incident to e. If we were to add the edge e to B, we would establish a path between the two pinned vertices x and y, rendering $B + e$ non–independent. Since G is a tree and A is independent, the vertices x and y belong to different connected components of $G[A]$. Since x or y is not incident to e, there exists another edge $e' \in B - A$ along the single x–y path in G. Moreover, since $G[B + e - e']$ has no paths between pinned vertices, it follows that $B + e - e'$ is also independent. ■

Lemma 3. *Suppose that the guest G is a path or a cycle, and all guest vertices free or pinned. Then, $\mathcal{I}_G$ is a matroid.*

Proof. Since $\mathcal{I}_G$ is already shown to be an independence system, we only need to prove the exchange

[2] The connected components of a directed graph G correspond to the connected components of undirected graph obtained by ignoring the direction on G's edges.

property.

Consider two maximal independent sets $A, B \subseteq E[G]$. Suppose that there exists an edge $e = uv$ that belongs to $A - B$. Since $e \notin B$, the edge e is between two different connected components G_u and G_v of $G[B]$. Each connected component G_u and G_v contains a set of pinned vertices, say X and Y respectively. Note that $cands(X) \cap cands(Y)$ is empty, while both $cands(X)$ and $cands(Y)$ are non–empty.

Consider two vertices $x \in X$ and $y \in Y$ such that x–y is a shortest X–Y path in G that contains e. Since $e \in A$ and $e \notin B$, at most one of the vertices x or y is incident to e. Then, the single x–e–y path in G has two or more edges. Let e' be an edge on the x–e–y path which is different than e. Then, the subgraph $G[B + e - e']$ has no paths between pinned vertices with disjoint candidate sets, and hence $B + e - e'$ is independent. ■

The conditions in the two lemmas above are as general as possible, in the sense that relaxing them in any way renders the independence system $\mathcal{I}_G$ a non–matroid.

Lemma 4. *Suppose that the guest G has all its vertices free or pinned. The independence system $\mathcal{I}_G$ is not a matroid in the following cases:*

(i) *G is a tree with two or more distinct guest vertices pinned to the same host node.*

(ii) *G's vertices are free or pinned to distinct hosts, and*

- **(ii.1)** *G becomes a tree by removing a single edge, or*
- **(ii.2)** *G becomes a simple cycle by removing a single edge.*

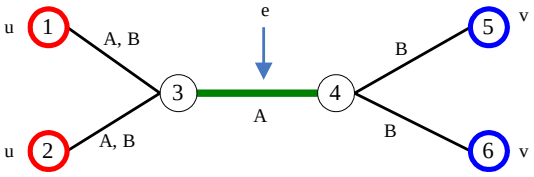
(iii) *G is isomorphic to ii.1 or ii.2 after a sequence of edge contractions.*

Proof. Figure 3 (a), (b), and (c) show counter–examples for the cases (i), (ii.1), and (ii.2) respectively. We indicate the edges in a set A or B by placing the label A or B next to those edges. In each case, we demonstrate maximal independent sets $A, B \in \mathcal{F}_G$, such that there exists an edge $e \in A - B$ for which there is no **single** edge $e' \in B - A$ so that $B + e - e'$ is independent.

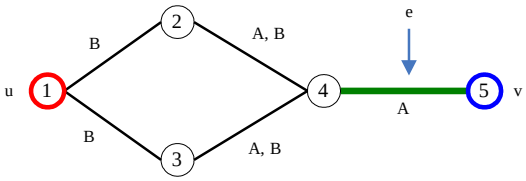
Since edge contractions preserve the matroid structure of any set system, (iii) follows from (ii). ■

Lemma 2 together with Theorem 13.20 in [22], shows that our GREEDYMCP algorithm finds a minimum cost placement of a guest expression DAG G onto a host network, when (i) G is a tree with

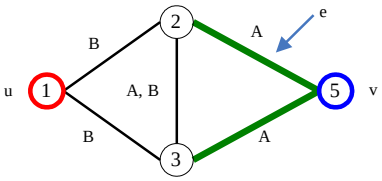
all the guest vertices either pinned or free, and there are no two distinct guest vertices pinned to the same host node, or (ii) G is a path or a cycle with all guest vertices free or pinned. GREEDYMCP may not optimally solve the MCP and MCCP problems in the other cases, such as those mentioned in Lemma 4. In the next subsection, we consider the computational complexity of the general MCP problem.



(a) Case i.



(b) Case ii.1



(c) Case ii.2

Figure 3: Counterexamples used in the proof of Lemma 4. Each edge is labeled with the independent set (A or B) to which it belongs. Pinned vertices are indicated with heavy outline (with different color) and labeled with the node to which they are pinned to. All other vertices are free. An edge $e \in A - B$ is shown as a heavy line in each subfigure.

4.2. Complexity of the MCP *problem*

We showed in the previous section that GREEDYMCP finds optimal solutions to MCP problem instances under certain conditions, and that there are MCP problem instances for which GREEDYMCP does not find optimal solutions. In this section, we analyze the computational complexity of the MCP problem, and show that the MCP problem is NP–hard and MAX–SNP–hard even for trees DAGs with height 1 and unit costs edges as long as they have restricted vertices.

We analyze the complexity of the MCP problem by studying the complexity of the simpler MCCP problem. Recall that an optimal solution to the MCCP problem instance $\langle G, cands \rangle$ provides us with an optimal solution to the minimum communication cost placement (MCP) problem of a guest G onto a host network H and vice versa.

Consider an instance of the MCCP problem $\langle G, f \rangle$, where $f : V[G] \longrightarrow U$. Call a vertex $v \in V[G]$ free if $f(v) = U$, pinned if $|f(v)| = 1$, and restricted otherwise.

The MCCP with all vertices either free or pinned to distinct elements of U is equivalent to the MULTITERMINAL CUT problem in graphs: given a graph G and a set of K terminals, find a minimum cost edge–cut A so that $G - A$ has no path between any two terminals. To see this observe that, by treating the pinned vertices as terminals, any edge–cut A that ensures that $G - A$ has no path between terminals also ensures that each connected component G_i of $G - A$ has at most one pinned vertex (and hence non–empty $f(V[G_i])$).

Dahlhaus et al [10] show that the MULTITERMINAL CUT problem is NP-Hard even for planar graphs with unit cost edges, and provide a simple 2-approximate algorithm based on the "*isolation heuristic*". For each terminal u, compute a min–cost cut that isolates u from all other terminals. Since the union of any $K - 1$ such isolating cuts is a multiterminal cut, by taking the union of all but the heaviest of these isolating cuts, they show that the resulting multiterminal cut is optimal within a factor of 2. Dahlhaus et al [10] also show that the problem is MAX SNP–hard for any fixed number of terminals $K \geq 3$. [3] Chopra and Rao [9] provide a polynomial–time dynamic programming algorithm for the MULTITERMINAL CUT problem for trees. Erdos and Szekely [13] provide a polynomial–time dynamic programming algorithm for a generalized multiterminal cuts in trees.

We study the complexity of the MCCP problem for trees with restricted vertices by linearly reducing the K–HITTING SET problem to it.

The K–HITTING SET problem $\langle V, \mathcal{C} \rangle$ is defined as follows: given collection $\mathcal{C}$ of K-subsets of a set V, find a minimum–size subset $S \subseteq V$ that intersects (hits) each subset in $\mathcal{C}$. A K–subset is a subset of size $\leq K$. We assume, w.l.o.g., that each $v \in V$ appears in at least one subset in $\mathcal{C}$. The K–HITTING SET problem is equivalent to the K–SET COVER problem (to see this, simple switch the roles of elements and sets in the two problems) [3]. Consequently, the K–HITTING SET problem is NP-hard, MAX SNP-hard, APX-complete, and approximable within a factor of $\sum_{i=1}^{K} \frac{1}{i} - \frac{1}{2} \approx \lg K + \frac{1}{2}$ (see [12, 3]).

Given a K–HITTING SET instance $\langle V, \mathcal{C} \rangle$, we construct an MCCP instance $\langle G, \phi \rangle$ as follows. The graph G is a tree with root a new vertex $r \notin V$, leaves V, height 1, and unit cost edges. The function $\phi : V \longrightarrow 2^V$ is defined so that the root r is free and all the leaves are restricted. In particular, for each leaf $v \in V$, $\phi(v)$ consists of all the K-subsets X of V that contain v and either are **not** in $\mathcal{C}$ or $|X| < K$.

[3] MAX SNP–hard problems do not have polynomial–time approximation schemes unless $P = NP$.

Lemma 5. *Consider any subset $X \subseteq V[G]$ with at most K vertices. Then,*

- $\phi(X) = X$ *if* $|X| < K$.
- $\phi(X) = X$ *if* $|X| = K$ *and* $X \notin \mathcal{C}$.
- $\phi(X) = \emptyset$ *if* $|X| = K$ *and* $X \in \mathcal{C}$.

Proof. By construction. ■

Lemma 6. *The* K–HITTING SET *linearly reduces to* MCCP*, i.e. the* MCCP *instance* $\langle G, \phi \rangle$ *has a feasible edge–cut* A *of size* m *iff the* K–HITTING SET *instance* $\langle V, \mathcal{C} \rangle$ *has a feasible hitting set* S *of size* m.

Proof. Consider the K–HITTING SET instance $\langle V, \mathcal{C} \rangle$ and the corresponding MCCP instance $\langle G, \phi \rangle$.

Consider a feasible edge–cut A for the MCCP instance. We show that the set $S = \{ u : ur \in A \}$ hits each subset in $\mathcal{C}$, i.e. S is a feasible hitting set. First, observe that, for any $X \subseteq V$ with an empty $\phi(X)$, the edge–cut A cuts the subtree $G[rX]$. [4] Second, consider any subset $X \in \mathcal{C}$. By Lemma 5, the set $\phi(X)$ of X is empty. Thus, A contains an edge xr from the subtree $G[rX]$, which implies that $x \in S \cap X$ and $S \cap X \neq \emptyset$. Since for each $X \in \mathcal{C}$, $S \cap X$ is not empty, S hits each X, and S is a feasible hitting set, and $|S| = |A|$.

Consider now a feasible hitting set S for the K–HITTING SET instance $\langle V, \mathcal{C} \rangle$. We show that $A = \{ ur : u \in S \}$ is a feasible edge–cut to the MCCP instance $\langle G, \phi \rangle$. Consider any subtree $G[rX]$ such that $\phi(rX)$ is empty. Since r is a free vertex, $\phi(X)$ is empty. There are three cases to consider.

Case 1: $|X| = K$. By Lemma 5, since $\phi(X) = \emptyset$, we have that X is in the collection $\mathcal{C}$. Since S is a hitting set, $S \cap X$ is not empty, and hence there exists $x \in X \cap S$. By construction, xr is in A. Thus, A cuts the subtree $G[rX]$.

Case 2: $|X| < K$. Since $|X| < K$, for each vertex $x \in X$, $\phi(x)$ contains X. Consequently, the set $\phi(X)$, which by definition is equal $\cap_{x \in X} \phi(x)$, is non–empty as it contains at least one element (namely X). Contradiction, since $\phi(X)$ is supposed to be empty.

[4] For brevity, $G[rX]$ indicates the subtree of G spanning the vertices $\{r\} \cup X$.

Table 1: Complexity of the MCCP problem $\langle G, \phi \rangle$, and its corresponding MCP problem $\langle G, H, cands \rangle$.

Structure G	**Vertices**	**Complexity class**	**Available Algorithms**
General graph	free or pinned	NP-hard	2-Approx [10]
Planar graph with unit cost edges	free or pinned	NP-hard	2-Approx [10]
Tree with unit cost edges	restricted	NP-hard	MAX–SNP–hard, APX–complete (this paper)
Tree	free or pinned	P	Dynamic Programming [9, 13]
Tree	free or pinned to distinct nodes	P	GREEDYMCP (this paper)

Case 3: $|X| > K$. If X hits S, then A cuts subtree $G[rX]$. If some K–subset Y of X is in $\mathcal{C}$, then A cuts the subtree $G[rY]$, and subtree $G[rX]$ as well. Otherwise, X is disjoint from S and no K-subset Y of X is in $\mathcal{C}$. Hence, none of the elements of X appear in any subset in the collection $\mathcal{C}$. Contradiction, since we assume that each element of V appears in at least one subset in $\mathcal{C}$.

Hence, A is a feasible edge–cut for the MCCP instance at hand, and $|A| = |S|$. ■

Theorem 1. *The* MCCP *problem* $\langle G, \phi \rangle$ *is NP-hard even for trees of height 1 and unit cost edges. In fact,* MCCP *is MAX SNP-hard.*

Proof. Follows from Lemma 6 and the complexity of the K–HITTING SET. ■

We summarize the complexity of the MCP and MCCP problems in Table 1.

5. The Maximum Lifetime Concurrent Flow (MLCF) Problem

In the previous two sections we considered the problem of placing the vertices of an expression tree DAG to the nodes of a wireless sensor network so that the total amount of operand data that needs to be communicated in each round of evaluating that DAG is minimized. We now turn our attention in how to route those operand data in order to maximize the number of rounds that DAG can be evaluated by the wireless sensor network.

Given a placement of the guest query expression DAG onto the host network, we need to find an energy efficient way to satisfy the data routing requirements indicated by the communication demands graph. In other words, we need to find a solution to an MLCF problem instance.

Consider a wireless sensor network $G = (V, E)$ with initial energy at the nodes $\boldsymbol{\epsilon} \in \mathbb{R}^{|V[G]|}$, and with unit transmit and receive costs $\boldsymbol{\tau}, \mathbf{r} \in \mathbb{R}^{|V[G]| \times |V[G]|}$. Let $\mathcal{S}$ be a set of n source–destination pairs of nodes $\langle s_i, d_i \rangle$, and let $\boldsymbol{\lambda} \in \mathbb{Z}^{|\mathcal{S}|}$ be such that the demand data rate for pair $\langle s_i, d_i \rangle \in \mathcal{S}$ is λ_i.

The MLCF problem is concerned with finding the maximum lifetime T such that we have total data flow $\lambda_i T$ for each pair $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$, while respecting the initial energy $\boldsymbol{\epsilon}$ of all the nodes. An instance of the MLCF problem is given as $\mathfrak{I} = \langle\, G, \boldsymbol{\epsilon}, \boldsymbol{\tau}, \mathbf{r}, \mathcal{S}, \boldsymbol{\lambda} \,\rangle$.

Let $\mathcal{P}$ be a set of directed paths $\mathbf{p}_j$ in G, where each path $\mathbf{p}_j$ is an $s_i \rightsquigarrow d_i$ path for some pair $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$. We assume, w.l.o.g., that, for each pair $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$, the set $\mathcal{P}$ contains all paths $s_i \rightsquigarrow d_i$ in G. Let $\mathbf{H} = (h_{ij}) \in \{0,1\}^{|\mathcal{S}|\times|\mathcal{P}|}$, where $h_{ij} = 1$ iff the path $\mathbf{p}_j \in \mathcal{P}$ is an $s_i \rightsquigarrow d_i$ directed path for some pair $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$. Let $\mathbf{Q} = (q_{kj}) \in \mathbb{R}^{|V[G]|\times|\mathcal{P}|}$ be such that q_{kj} is the energy consumed by node k due to the flow of one data unit along the path $\mathbf{p}_j \in \mathcal{P}$. Let $\mathbf{F} = (f_{ij}) \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{P}|}$ be such that f_{ij} is the amount of data flow for the pair $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$ along the path $\mathbf{p}_j \in \mathcal{P}$. Observe that even though $\mathcal{P}$ can be very large, as we will see shortly, we never store or otherwise explicitly consider each element of $\mathcal{P}$, or any of the other large objects $\mathbf{F}$, $\mathbf{Q}$, or $\mathbf{H}$.

Our approach to the MLCF problem is based on the Revised Simplex method of linear programming. We provide an overview of linear programming concepts in Appendix A. The reader is referred to a text such as [4, 22] for further details on linear programming.

We formulate the MLCF problem as the following integer program (IP)

$$\begin{array}{ll} \max T & \text{such that} \\ \sum_{j=1}^{|\mathcal{P}|} h_{ij} f_{ij} \geq \lambda_i T, & \text{for each pair } \langle\, s_i, d_i \,\rangle \in \mathcal{S}, \\ \sum_{i=1}^{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{P}|} f_{ij} q_{kj} \leq \epsilon_k, & \text{for each node } k \in V[G], \\ \text{and non–negative integral } \mathbf{F} = (f_{ij}). & \end{array} \tag{1}$$

Since solving integer programs is in general computationally intractable, we focus our attention to the linear relaxation of this IP. We consider the standard form of the linear relaxation of the IP above,

$$\begin{array}{ll} \min \mathbf{c}^T \cdot \mathbf{x} & \text{such that} \\ \mathbf{A} \cdot \mathbf{x} = \mathbf{b} & \text{and} \\ \mathbf{x} \geq \mathbf{0}, & \end{array} \tag{2}$$

by appropriately defining $\mathbf{A}, \mathbf{b}, \mathbf{c}$ and $\mathbf{x}$. Observe that there always exists an optimal solution where there is no excess data flow for any source–destination pair in $\mathcal{S}$. We account for the residual energy of the nodes using the vector of slack variables $\mathbf{z} = (z_j) \in \mathbb{R}^{|V[G]|}$. We collect all the $K = 1 + |\mathcal{S}| \cdot |\mathcal{P}| + |V[G]|$ unknowns into a single vector $\mathbf{x} \in \mathbb{R}^K$,

$$\mathbf{x}^T = (T\ \mathbf{f}^T\ \mathbf{z}^T) \tag{3}$$

where $\mathbf{f}$ is obtained by a row–major traversal of the matrix $\mathbf{F}$. Let $\psi[v]$ be the index an unknown v maps to in the vector $\mathbf{x}$. Further, the first $|\mathcal{S}|$ rows of the matrix $\mathbf{A} \in \mathbb{R}^{(|\mathcal{S}|+|V[G]|)\times K}$ account

for the $|\mathcal{S}|$ constraints due to the demand data rates for the source–destination pairs in $\mathcal{S}$, while the remaining $|V[G]|$ rows account for the energy constraints at the nodes. The vectors $\mathbf{c}$ and $\mathbf{b}$, and the columns of $\mathbf{A}$ are defined as follows:

$\mathbf{c}$	$\mathbf{b}$	$\mathbf{A}_{\psi[T]}$	$\mathbf{A}_{\psi[f_{ij}]}$	$\mathbf{A}_{\psi[z_j]}$
$-\mathbf{e}^{(\psi[T])}_{K\times 1}$	$\mathbf{O}_{\|\mathcal{S}\|\times 1}$	$-\boldsymbol{\lambda}$	$h_{ij}\mathbf{e}^{(i)}_{\|\mathcal{S}\|\times 1}$	$\mathbf{O}_{\|\mathcal{S}\|\times 1}$
	$\boldsymbol{\epsilon}$	$\mathbf{O}_{\|V[G]\|\times 1}$	$\mathbf{Q}_j$	$\mathbf{e}^{(j)}_{\|V[G]\|\times 1}$

(4)

where $\mathbf{e}^{(i)}_{j\times 1} \in \mathbb{R}^j$ denotes the vector with all zeros except that its ith entry is equal to 1, and $\mathbf{O}_{i\times j}$ denotes an $i \times j$ matrix of all zeroes.

We show that the Revised Simplex method can be used to find an optimal solution to the LP in Eq. (2) for the MLCF problem by showing how to efficiently find (a) an initial basic feasible solution (BFS), and (b) a variable that enters the BFS at each pivoting step, without explicitly manipulating the entire matrix $\mathbf{A}$ or vector $\mathbf{c}$. The details of our iterative algorithm for solving the LP for the MLCF problem, which is based on the Revised Simplex method, are given in Algorithm 2 (ALGRSM–MLCF).

An initial base $\mathcal{B}$ and its corresponding BFS $\mathbf{x}$ is obtained as follows. Initialize $\mathbf{x}$ to $\mathbf{0}$. For each $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$, choose one path $\mathbf{p}_{j_i} = s_i \rightsquigarrow d_i$ in G; if there is no path for some pair $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$ then this instance of the MLCF problem is infeasible. Choose an initial lifetime $x_{\psi[T]} = \epsilon_{k_o}/\hat{e}_{k_o}$, where

$$k_o = \arg \min_{1\le k\le |V[G]|} \{\epsilon_k/\hat{e}_k\}\,, \tag{5}$$

and $\hat{e}_k = \sum_{i=1}^{|\mathcal{S}|} \lambda_i q_{kj_i}$, $1 \le k \le |V[G]|$. Push flow $x_{\psi[f_{ij_i}]} = \lambda_i \cdot x_{\psi[T]}$ on each chosen path $\mathbf{p}_{j_i}$, $i = 1, 2, \ldots, |\mathcal{S}|$; note that there is no flow on any other paths. Set $x_{\psi[z_k]}$ to the residual energy of each node $k \in V$ for the current flow assignment $\mathbf{F}$; note that node k_o exhausts all its initial energy, i.e. $x_{z_{k_o}} = 0$. Observe that the $\mathbf{x}$ constructed above satisfies the equation $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$. By taking the initial base, which corresponds to $\mathbf{x}$, to be

$$\mathcal{B} = \{\ \psi[T], \psi[f_{ij_i}], \psi[z_k]\ :\ i = 1, 2, \ldots, |\mathcal{S}| \text{ and } k \in V[G] - \{k_o\}\ \}. \tag{6}$$

we have that $|\mathcal{B}| = |\mathcal{S}| + |V[G]|$ and $\mathbf{A}_{\mathcal{B}}$ has full–rank, which implies that $\mathbf{x}$ is a BFS.

At each pivot iteration (lines 13–29 of Algorithm 2), we choose a variable x_m that should enter the current base $\mathcal{B}$ and a variable x_l that should exit $\mathcal{B}$, where $m = \psi[f_{ij}]$, $1 \le i \le |\mathcal{S}|$ and $1 \le j \le |\mathcal{P}|$. No other variable ever needs to enter $\mathcal{B}$, since all other variables outside $\mathcal{B}$ are slack variables. By the theory of linear programming, a variable should enter the current base, if its current relative cost is

negative, and if no such variable is found then $\mathcal{B}$ corresponds to an optimal solution and the algorithm terminates. The relative cost of a variable $x_{\psi[f_{ij}]}$ outside $\mathcal{B}$ is

$$\overline{c}_{\psi[f_{ij}]} = c_{\psi[f_{ij}]} - (\boldsymbol{\rho}^T \ \boldsymbol{\pi}^T) \begin{pmatrix} h_{ij}\mathbf{e}^{(i)}_{|\mathcal{S}|\times 1} \\ \mathbf{Q}_j \end{pmatrix} = c_{\psi[f_{ij}]} - h_{ij}\boldsymbol{\rho}^T\mathbf{e}^{(i)}_{|\mathcal{S}|\times 1} - \boldsymbol{\pi}^T\mathbf{Q}_j = -h_{ij}\rho_i - \boldsymbol{\pi}^T\mathbf{Q}_j. \quad (7)$$

where the shadow prices vector is $(\boldsymbol{\rho}^T_{|\mathcal{S}|\times 1} \ \boldsymbol{\pi}^T_{|V[G]|\times 1}) = \mathbf{c}^T_{\mathcal{B}}\mathbf{A}^{-1}_{\mathcal{B}}$. In other words, $x_{\psi[f_{ij}]}$ should enter $\mathcal{B}$, if the energy cost of the path $\mathbf{p}_j$ under the energy prices $-\boldsymbol{\pi}$ is at most $h_{ij}\rho_i$. Such a path can be found by running a shortest path algorithm on the graph G where the weight of each edge $ij \in E[G]$ is given by $w_{ij} = -\pi_i\tau_{ij} - \pi_j r_{ij}$. A variable x_l that exits the current base $\mathcal{B}$ is chosen according to the lexico–min rule, which relies on $\mathbf{A}_{\mathcal{B}}$ and $\mathbf{A}_{\psi[f_{ij}]}$. The lexico–min rule guarantees termination in a finite number of pivoting steps [4]. By the theory of the Revised Simplex method of linear programming, ALGRSM–MLCF is correct and terminates with an an optimal fractional solution, whenever such a solution exists.

The ALGRSM–MLCF algorithm finds a fractional solution that maximizes the system lifetime T_o and consists of at most $n+N$ paths for satisfying the concurrent flow data demands for N source–destination pairs in a network with n nodes. By rounding down that fractional solution, we get an α–optimal integral concurrent flow solution to the MLCF problem, where $\alpha = (T_o - n - N + 1)/T_o$. Since often in practice $T_o \gg n + N$, it follows that $\alpha \approx 1$.

NOTE: It is possible, though rare, to have negative weight cycles in G under the weights $\mathbf{w}$ in lines 18–21 of ALGRSM–MLCF, which may lead into a scenario where the shortest path called upon in line 21 does not exist. In such a case, we consider walks instead of simple paths, and this consideration preserves the correctness of the algorithm – a variable $x_{\psi[f_{ij}]}$ enters the base $\mathcal{B}$ if it corresponds to a walk $\mathbf{p}_j = s_i \rightsquigarrow d_i$ in G of weight at most $h_{ij}\rho_i$ under the edge weights $\mathbf{w}$. At termination, without increasing the energy consumed by the sensors, each walk in the base $\mathcal{B}$ is converted to a simple path by eliminating all of its cycles.

Next, we consider the running–time of ALGRSM–MLCF. Lines 1–14 take $O(V[G]+E[G]+\mathcal{S})$ time, while lines 34–39 take $O((\mathcal{S}+V[G])^3)$ time. The running–time of each pivot iteration in lines 16–32 of ALGRSM–MLCF. is $O((\mathcal{S}+V[G])^3)$. Thus the running time of Algorithm 2 is $\#iterations \times O((\mathcal{S}+V[G])^3)$. Typically, in our experimental results, we find that the number of iterations is quite small.

Algorithm 2 The ALGRSM–MLCF algorithm for finding an optimal fractional solution to the MLCF problem.

Input: MLCF instance $\mathfrak{I} = \langle\, G, \boldsymbol{\epsilon}, \boldsymbol{\tau}, \mathbf{r}, \mathcal{S}, \boldsymbol{\lambda} \,\rangle$.

Output: $\text{OPT}(\mathfrak{I}) = \langle\, T, \mathcal{P}_o, \mathbf{f} \,\rangle$, where T is the lifetime, $\mathcal{P}_o$ is a collection of at most $|V[G]| + |\mathcal{S}|$ paths, and $\mathbf{f}$ is the vector of path flows.

// find an initial basis

1. $path \longleftarrow$ a vector with $|\mathcal{S}| + |V[G]|$ **null**s.
2. let $\mathcal{B} \longleftarrow \langle\ \rangle$ and $\hat{e} \longleftarrow \mathbf{0}_{|V[G]|\times 1}$
3. **for** each $1 \le i \le |\mathcal{S}|$ **do**
4. find a directed path $\mathbf{p}_j = s_i \rightsquigarrow d_i$ in G for the pair $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$
5. **if** no path $\mathbf{p}_j$ exists **then**
6. **return** *instance $\mathfrak{I}$ is infeasible*
7. append $\psi[f_{ij}]$ to $\mathcal{B}$
8. $path[|\mathcal{B}|] \longleftarrow \mathbf{p}_j$
9. increase $\hat{e}_k$ by $\lambda_i q_{kj}$ for each node k on $\mathbf{p}_j$
10. $k_o \longleftarrow \arg\min_{1\le k\le |V[G]|} \{\epsilon_k/\hat{e}_k\}$
11. $T \longleftarrow \epsilon_{k_o}/\hat{e}_{k_o}$
12. append $\psi[T]$ to $\mathcal{B}$
13. **for** each $1 \le k \le |V[G]|$, $k \ne k_o$ **do**
14. append $\psi[z_k]$ to $\mathcal{B}$

// Pivoting

15. **repeat**
16. $done \longleftarrow$ **true**
17. $(\boldsymbol{\rho}^T\ \boldsymbol{\pi}^T) \longleftarrow \mathbf{c}_{\mathcal{B}}^T \cdot \mathbf{A}_{\mathcal{B}}^{-1}$
18. **for** each edge $ij \in E[G]$ **do**
19. $w_{ij} \longleftarrow -\pi_i \cdot \tau_{ij} - \pi_j \cdot r_{ij}$
20. **for** each pair $\langle\, s_i, d_i \,\rangle \in \mathcal{S}$ **do**
21. find a shortest path $\mathbf{p}_j = s_i \rightsquigarrow d_i$ in G with edge weights $\mathbf{w}$
22. **if** the cost of $\mathbf{p}_j$ is less than ρ_i **then**
23. $done \longleftarrow$ **false**
24. $\mathbf{u} \longleftarrow \mathbf{A}_{\mathcal{B}}^{-1} \cdot \mathbf{A}_{\psi[f_{ij}]}$
25. **if** the support $\mathbb{I}(\mathbf{u})$ is empty **then**
26. **return** *instance $\mathfrak{I}$ is unbounded*
27. let $\mathbf{a}_i$ be the ith row of the matrix $\mathbf{A}_{\mathcal{B}}$, $1 \le i \le |\mathcal{B}|$
28. $l \longleftarrow \arg\,\text{lexico–min}\{[x_i, \mathbf{a}_i]/u_i \ :\ i \in \mathbb{I}(\mathbf{u})\}$
29. $\mathcal{B}(l) \longleftarrow \psi[f_{ij}]$

5.1. Computing an upper bound on the lifetime

We compute an upper bound on the system lifetime during each iteration of ALGRSM–MLCF using the duality theory of linear programming. Such an upper bound allows us to continuously assess the distance of the current feasible solution to MLCF from the optimal, and hence enables us to terminate ALGRSM–MLCF early if that distance is below some desired threshold.

Consider the primal linear program for the MLCF problem. The dual linear program of this primal linear program is

$$\begin{aligned} &\min \mathbf{y}^T \cdot \mathbf{b} && \text{such that} \\ &\mathbf{y}^T \cdot \mathbf{A}_J \geq \mathbf{c}_J^T && \text{and} \\ &\mathbf{y}_I \geq \mathbf{0}, \end{aligned} \tag{8}$$

where $J = \{\ \psi[T],\ \psi[f_{ij}]\ :\ 1 \leq i \leq |\mathcal{S}| \text{ and } 1 \leq j \leq |\mathcal{P}|\ \}$ and $I = \{|\mathcal{S}| + 1, \ldots, |\mathcal{S}| + |V[G]|\}$. To see this observe that for any primal feasible solution $\mathbf{x}$, we have $\mathbf{x}_J \geq \mathbf{0}$, $\mathbf{A}_{I,J} \cdot \mathbf{x}_J \leq \mathbf{b}_I$, and $\mathbf{A}_{\overline{I},J} \cdot \mathbf{x}_J = \mathbf{b}_{\overline{I}}$. By duality theory, $\mathbf{c}^T \cdot \mathbf{x} \leq \mathbf{y}^T \cdot \mathbf{b}$ for any pair of primal–dual feasible solutions $(\mathbf{x}, \mathbf{y})$. We show how to find a dual feasible solution for any primal BFS, and thus provide an effective mechanism for estimating the sub–optimality of the BFS computed at each iteration of ALGRSM–MLCF.

Consider a primal BFS $\mathbf{x}$ with basis $\mathcal{B}$ such that $x_{\psi[T]} > 0$. Let the corresponding shadow prices vector be

$$\mathbf{y}^T = (\boldsymbol{\rho}^T_{|\mathcal{S}|\times 1}\ \boldsymbol{\pi}^T_{|V[G]|\times 1}) = \mathbf{c}^T_{\mathcal{B}} \cdot \mathbf{A}^{-1}_{\mathcal{B}}. \tag{9}$$

Vector $\mathbf{y}$ may fail to be a dual feasible solution because $\mathbf{y}_I \ngeq \mathbf{0}$ or because $\mathbf{y}^T \cdot \mathbf{A}_J \ngeq \mathbf{c}_J$. We rectify any such shortcomings of $\mathbf{y}$ by constructing a feasible dual solution based on $\mathbf{y}$.

Since $x_{\psi[T]} > 0$, it follows that $\psi[T] \in \mathcal{B}$ and

$$1 = c_{\psi[T]} = \mathbf{y}^T \cdot \mathbf{A}_{\psi[T]} = (\boldsymbol{\rho}^T\ \boldsymbol{\pi}^T) \cdot \mathbf{A}_{\psi[T]} = \boldsymbol{\rho}^T \cdot (-\boldsymbol{\lambda}) \tag{10}$$

Further, since $\boldsymbol{\lambda} > \mathbf{0}$, it follows that $\boldsymbol{\rho}$ has at least one negative component. Moreover, for any $\psi[f_{ij}] \in \mathcal{B}$, we have

$$0 = c_{\psi[f_{ij}]} = \mathbf{y}^T \cdot \mathbf{A}_{\psi[f_{ij}]} = \boldsymbol{\pi}^T \cdot \mathbf{Q}_j + h_{ij}\boldsymbol{\rho}^T \cdot \mathbf{e}^{(i)}_{n\times 1} = \boldsymbol{\pi}^T \cdot \mathbf{Q}_j + h_{ij}\rho_i. \tag{11}$$

Define the vector $\tilde{\boldsymbol{\pi}} = \max(\boldsymbol{\pi}, \mathbf{0}) \geq \mathbf{0}$. Let $\tilde{\mu}_i \geq 0$ be the length of a shortest path $s_i \rightsquigarrow d_i$ in G under the energy prices $\tilde{\boldsymbol{\pi}}$. Let $\gamma_{\min} = \min\{\gamma\ :\ \gamma\tilde{\mu}_i \geq -\rho_i\}$. Since $\boldsymbol{\rho}$ has at least one negative

Table 2: Node locations for network topologies with n nodes. All nodes whose location is not explicitly indicated are uniformly distributed in a $50m \times 50m$ field. Node 1 is the base station.

node	Topology I	Topology II	Topology III	Topology IV
1	$(45m, 45m)$	$(45m, 45m)$	$(45m, 45m)$	$(45m, 45m)$
2	$(20m, 20m)$	$(10m, 30m)$	$(10m, 30m)$	$(10m, 30m)$
3		$(30m, 10m)$	$(20m, 20m)$	$(10m, 20m)$
4			$(30m, 10m)$	$(20m, 10m)$
5				$(30m, 10m)$
6–n				

component, $\gamma_{\min} > 0$. Let $\hat{\boldsymbol{\pi}} = \gamma_{\min} \cdot \tilde{\boldsymbol{\pi}}$. Since $\tilde{\boldsymbol{\pi}}^T \cdot \mathbf{Q}_j \geq \tilde{\mu}_i$ for any path $\mathbf{p}_j \in s_i \rightsquigarrow d_i$, it follows that $(\gamma_{\min}\tilde{\boldsymbol{\pi}})^T \cdot \mathbf{Q}_j \geq \gamma_{\min}\tilde{\mu}_i \geq -\rho_i$. Therefore, for all $\psi[f_{ij}] \in J$, we have

$$c_{\psi[f_{ij}]} = 0 \leq \hat{\boldsymbol{\pi}}^T \cdot \mathbf{Q}_j + h_{ij}\boldsymbol{\rho}^T \cdot \mathbf{e}^{(i)}_{n\times 1} = (\boldsymbol{\rho}^T\ \hat{\boldsymbol{\pi}}^T) \cdot \mathbf{A}_{\psi[f_{ij}]}. \tag{12}$$

Consequently, the vector $\hat{\mathbf{y}}^T = (\boldsymbol{\rho}^T\ \hat{\boldsymbol{\pi}}^T)$ is a dual feasible solution with value $\hat{\boldsymbol{\pi}}^T \cdot \boldsymbol{\epsilon} \geq \mathbf{c}^T \cdot \mathbf{x}$, where $\mathbf{x}$ is a primal BFS.

6. Experimental Evaluation

In this section we experimentally evaluate the performance of our method, which uses GREEDYMCP for the placement of the operators and then uses ALGRSM–MLCF for the routing of the operands, for the continuous evaluation of complex queries. We use the first order radio model used in [19] as our sensor energy model with the following parameter values: the energy consumed to run the transmitter and receiver circuitry is $50\,nJ/bit$, the energy consumed to run the transmitter amplifier is $100\,pJ/bit/m^2$, and the packet size is 1000 bits. With these parameters, the energy consumed by a sensor i to receive a packet is $r_i = 5\,mJ$ and the energy to transmit a packet to a node j at distance d_{ij} meters is $\tau_{ij} = 5 + 0.1 \cdot d_{ij}^2\,mJ$. Each sensor has $1\,J$ initial available energy. In our experiments here, in order to better demonstrate the benefits of our routing algorithm, we avoid taking advantage of the base station's unlimited available energy when routing operands by setting both its initial energy ϵ_b and its receive energy r_b to 0.

We consider wireless sensor networks with n nodes and four different topologies I–IV. In each instance of a network topology the location of certain nodes is fixed while all other nodes are randomly distributed in the field. Node 1 is always the base station. See Table 2 for more details. We

show example wireless sensor networks with these four topologies in Fig. 4. In our experiments, we generate 20 instances of network topology I with $n = 20$ and $n = 40$ nodes, and 10 instances for each network topology II, III, and IV with $n = 30$ nodes.

All of our experiments were done in Matlab running on a standard desktop PC.

6.1. ALGRSM–MLCF *vs. FA for maximum lifetime single source–destination routing*

First we compare the performance of ALGRSM–MLCF with that of the Flow Augment algorithm (FA) by Chang and Tassiulas [7] for solving the MLCF problem. We do this evaluation because a maximum lifetime routing of all the operand values during the continuous evaluation of an expression tree gives rise to an instance of the MLCF problem. Here we compare the achieved system lifetime of ALGRSM–MLCF with that of the FA algorithm for routing one unit demand (one packet) in each round for the single source–destination pair, the pair 2–1. The parameters (x, λ) of the FA algorithm are set to the values $(x = 30, \lambda = 5000\text{bits})$, suggested as their optimal values in [7]. The FA algorithm collects the current residual energy information of every sensor for every λ bits transmitted from source to destination. The amount of energy consumed by the sensors to collect that information from them is not taken into consideration, despite the fact that it can be quite significant.

Fig. 5 gives a scatter plot of the lifetimes achieved by ALGRSM–MLCF and the FA algorithm, with each point corresponding to an instance of network topology I with n nodes. We use 20 instances of network topology I with $n = 20$ and 40 nodes. ALGRSM–MLCF achieves better lifetime than the FA algorithm for all the network instances. For $\lambda = 5000bits$ and packets of size 1000 bits, the FA algorithm needs to collect the residual energy information of every sensor once every five packet transmissions from source to destination. The energy consumed to collect all that information is at least 20% of the energy used to transmit packets with operand values. Taking the energy required to collect all that information from the sensors would significantly reduce the lifetime achieved by the FA algorithm, leading to significant benefits of using ALGRSM–MLCF instead of FA.

6.2. Performance of our placement and routing approach for the continuous evaluation of queries

For evaluating the performance, in terms of system lifetime, of our placement and routing algorithms for the continuous evaluation of query expression trees, we use the expression trees shown in Fig. 6.

The tree DAGs we experiment with have operands of various sizes (small, medium, large). The values 1, 2, and 5 are chosen as sample values for small, medium, and large size operands. For example, the tree DAG $\mathcal{T}_{13}$ could model the TinyDB [24] SQL query

```
SELECT windowA.light, windowA.temperature, windowB.temperature
FROM windowA, windowB
WHERE windowA.light = windowB.light
SAMPLE PERIOD 10s FOR 100s;
```

where

```
CREATE STORAGE POINT windowB SIZE 32
AS (SELECT nodeid, light, temperature FROM sensors
    WHERE nodeid='B' SAMPLE PERIOD 10s);
CREATE STORAGE POINT windowA SIZE 32
AS (SELECT nodeid, light, temperature FROM sensors
    WHERE nodeid='A' SAMPLE PERIOD 10s);
```

and where the operator C of $\mathcal{T}_{13}$ corresponds to the natural join specified in the SQL SELECT statement above, while the operands A and B correspond to the windowA and windowB storage points respectively. Moreover, we present results only for some combinations of operand sizes for the given tree DAGs. Our choices of the different operand size combinations were guided by the desire to induce different placements of the free operators of the tree DAGs (while exploiting symmetries in the DAGs and the locations of the fixed operators), and to have tree DAGs that model common (SQL) queries.

We consider the evaluation of all expression trees $\mathcal{T}_{1j}$ in instances of network topology II, $\mathcal{T}_{2j}$ and $\mathcal{T}_{3j}$ in instances of network topology III, and $\mathcal{T}_{4j}$ in instances of network topology IV. We use 10 instances of each network topology II–IV with $n = 30$ nodes. For each expression tree $\mathcal{T}_{ij}$ and each instance $\mathcal{G}$ of the corresponding network topology, we compute the lifetime $\mathcal{P}_{\mathrm{O}}(\mathcal{T}_{ij}, \mathcal{G})$ achieved when placing $\mathcal{T}_{ij}$ on $\mathcal{G}$ using GREEDYMCP, together with the sequence $\mathcal{P}_{\mathrm{ALL}}(\mathcal{T}_{ij}, \mathcal{G})$ of lifetimes achieved by each one of all possible placements of $\mathcal{T}_{ij}$ on $\mathcal{G}$, indexed by those placements of $\mathcal{T}_{ij}$, while using ALGRSM–MLCF to route the required operand values during the continuous evaluation of $\mathcal{T}_{ij}$ in $\mathcal{G}$.

Fig. 7 shows $\mathcal{P}_{\rm ALL}(\mathcal{T}_{1j}, \mathcal{G})$, indexed by the placement of the only non–pinned vertex of $\mathcal{T}_{1j}$ (vertex C), for one instance $\mathcal{G}$ of network topology II. GREEDYMCP places vertex C of $\mathcal{T}_{11}$, $\mathcal{T}_{12}$, and $\mathcal{T}_{13}$ at network nodes 1, 2, and 1, respectively. Observe that these placements found by GREEDYMCP provide the optimal lifetime, and all the other placements of $\mathcal{T}_{1j}$ achieve comparable lifetimes to each other, which are all substantially smaller than the one achieved by our GREEDYMCP placement. that, in maximizing the system lifetime for the continuous evaluation of an expression tree, (a) the system lifetime seems to be less sensitive to the placement of the tree vertices, provided a near–optimal routing is used to route the operand values, and (b) a placement that strives to minimize the total size of all operands communicated, tends to provide the largest system lifetime.

Similar observations can be made when looking at the results for each expression tree $\mathcal{T}_{ij}$ in each instance $\mathcal{G}$ of the corresponding network topology (e.g. see Fig 8). Due to limited space, we cannot plot and present all of these results like in Fig. 7 and 8. Instead, we compute and present in Table 3 the average and standard deviation of each one of the following measures $\mathcal{P}_{\rm O}$, $\max(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, $\min(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, $\text{avg}(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, $\text{median}(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, and $\text{pct}_{95\%}(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, , for each $\mathcal{T}_{ij}$ over all 10 relevant network topology instances $\mathcal{G}$. We also show these measures as boxplots in Fig. 9. Table 3 and Fig. 9 demonstrate that our placement and routing algorithms provide near optimal system lifetimes for the continuous evaluation of expression trees in wireless sensor networks. [5]. In fact, as can be seen from both Table 3 and Fig. 9, GREEDYMCP achieves significantly better lifetime than the 95% percentile of the lifetime achieved over all possible placements.

Observe that since all the expression trees in our experiments have their nodes fixed or free, it follows from Lemma 2, that GREEDYMCP solves the MCP placement problem optimally for all our expression trees. Also, recall (see section 5) that ALGRSM–MLCF computes an integral solution to the MLCF problem that is optimal within a factor of $\alpha = (T_o - n - N + 1)/T_o$, where T_o is the optimal fraction lifetime, n is the number of network nodes, and N is the number of edges of the DAG. Since in all our experiments $n \leq 40$ and $N \leq 7$, it follows that ALGRSM–MLCF computes a solution to the MLCF problem that is always within 47 rounds of the optimal lifetime. Furthermore, since the average $T_o \geq 4905$ (see Table 3), it follows that ALGRSM–MLCF computes integral solutions to the MLCF problem that are within 1% of the optimal lifetime on the average.

To further demonstrate the performance of our placement and routing approach for the continuous

[5]The boxplot for each expression tree $\mathcal{T}_{ij}$ shows the average $\min(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, $\text{pct}_{25\%}(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, $\text{pct}_{50\%}(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, $\text{pct}_{75\%}(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, and $\max(\mathcal{P}_{\rm ALL}(\mathcal{T}_{ij}, \mathcal{G}))$, over all the relevant network topologies $\mathcal{G}$.

Table 3: Average $\pm$ std. deviation of $\mathcal{P}_{\mathrm{O}}(\mathcal{T}_{ij}, \mathcal{G})$ and certain other measures (e.g. maximum, minimum, median, etc) based on $\mathcal{P}_{\mathrm{ALL}}(\mathcal{T}_{ij}, \mathcal{G})$, with the summarizing done over all relevant network topology instances $\mathcal{G}$.

Tree	$\mathcal{P}_{\mathrm{O}}$	$\max(\mathcal{P}_{\mathrm{ALL}})$	$\min(\mathcal{P}_{\mathrm{ALL}})$	mean($\mathcal{P}_{\mathrm{ALL}}$)	median($\mathcal{P}_{\mathrm{ALL}}$)	$\mathrm{pct}_{95\%}(\mathcal{P}_{\mathrm{ALL}})$
$\mathcal{T}_{11}$	17531 ± 1448	17531 ± 1448	6145 ± 107	7106 ± 64	6589 ± 25	9556 ± 353
$\mathcal{T}_{12}$	6503 ± 162	6503 ± 162	2422 ± 16	2680 ± 14	2489 ± 3	3290 ± 46
$\mathcal{T}_{13}$	9034 ± 774	9034 ± 774	2140 ± 55	2642 ± 37	2426 ± 20	2686 ± 134
$\mathcal{T}_{21}$	16967 ± 1422	16967 ± 1422	4601 ± 324	5483 ± 54	4963 ± 14	6537 ± 124
$\mathcal{T}_{22}$	4905 ± 90	4905 ± 90	2137 ± 77	2373 ± 11	2215 ± 2	2832 ± 29
$\mathcal{T}_{31}$	16968 ± 1287	16968 ± 1287	4588 ± 113	6279 ± 31	6464 ± 49	6639 ± 10
$\mathcal{T}_{32}$	6534 ± 136	6534 ± 136	2143 ± 20	2616 ± 8	2483 ± 5	3299 ± 42
$\mathcal{T}_{33}$	6534 ± 136	6534 ± 136	1928 ± 33	2547 ± 5	2482 ± 5	3299 ± 42
$\mathcal{T}_{34}$	8710 ± 705	8710 ± 705	1859 ± 100	2390 ± 16	2402 ± 26	2495 ± 2
$\mathcal{T}_{35}$	6532 ± 134	6532 ± 134	1302 ± 22	2301 ± 16	2391 ± 30	2482 ± 7
$\mathcal{T}_{41}$	16928 ± 1746	16928 ± 1746	2960 ± 277	5854 ± 53	6367 ± 91	6600 ± 30
$\mathcal{T}_{42}$	6577 ± 97	6577 ± 97	1697 ± 98	2572 ± 9	2484 ± 4	3289 ± 30
$\mathcal{T}_{43}$	6482 ± 291	6482 ± 291	1186 ± 49	2322 ± 10	2471 ± 8	2496 ± 1
$\mathcal{T}_{44}$	6482 ± 291	6482 ± 291	942 ± 78	2200 ± 15	2459 ± 13	2495 ± 1

evaluation of expression trees, we compare our ALGRSM–MLCF routing to a naturally alternative routing: whenever an operand value needs to be routed from a network node x to network node node y, we use an x–y shortest path in the network, where the cost of each network link uv is equal to the energy $\tau_{uv} + r_v$ consumed to send a packet from u to v. We re–run the simulation for each expression tree $\mathcal{T}_{ij}$ and each instance $\mathcal{G}$ of the corresponding network topology, and we find the best lifetime among all possible placements of $\mathcal{T}_{ij}$ onto $\mathcal{G}$, this time using shortest paths to route the required operand values during the continuous evaluation of $\mathcal{T}_{ij}$ in $\mathcal{G}$.

Fig. 10 shows the scatter plot of the optimal lifetimes achieved by using ALGRSM–MLCF vs. fixed (shorest) paths for the expression tree $\mathcal{T}_{i1}$. Each point in Fig. 10 represents a topology instance $\mathcal{G}$. We see that ALGRSM–MLCF achieved much better lifetime (over 20% on average) in all the instances. Again, this demonstrates that our method provides near optimal system lifetimes for the continuous evaluation of expression trees in wireless sensor networks.

7. Conclusions

In this paper we consider the problem of maximum lifetime continuous in-network evaluation of complex queries $\mathcal{Q}$ in wireless sensor networks. This task includes two coupled aspects that impact each other as well as the system lifetime — to which network nodes should one place the operators, variables, and constants of $\mathcal{Q}$, and how to route the values of the operands among network nodes during the continuous evaluation of $\mathcal{Q}$. While most of the existing work focuses on the placement aspect and treats the routing as a substrate, we address both the placement and routing aspects of the task together.

We present a simple, effective, and efficient algorithm for the maximum lifetime concurrent flow (MLCF) problem in wireless sensor networks that facilitates the decoupling of the placement and routing aspects of the task at hand. We also present a simple greedy heuristic, GREEDYMCP, for the minimum communication cost placement (MCP) problem of an expression DAG to a network, and show that it finds provable optimal solutions on certain practical useful cases, while we prove that the MCP problem is MAX–SNP hard even for trees with unit costs edges. Through an extensive experimental evaluation, we show that our approach consistently finds the maximum lifetime solution for the continuous in–network evaluation of complex queries in wireless sensor networks.

Designing fault–tolerant algorithms for the maximum lifetime continuous in-network evaluation of complex queries in highly dynamic wireless sensor networks remains an important open problem

to be addressed in future work.

Acknowledgments.

We thank the referees for their valuable comments and suggestions for improving the presentation of the paper.

References

[1] Y. Ahmad and U. Çetintemel. Network-aware query processing for stream-based applications. In *Proceedings of the 30th VLDB Conference*, pages 456–467, 2004.

[2] J. Ahn and B. Krishnamachari. Fundamental scaling laws for energy–efficient storage and querying in wireless sensor networks. In *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 334–343, New York, NY, USA, 2006. ACM.

[3] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21:136–153, 1980.

[4] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[5] B. J. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. *LNCS 2634*, pages 47–62, 2003.

[6] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31, New York, NY, USA, 2002. ACM.

[7] J.-H. Chang and L. Tassiulas. Maximum lifetime routing in wireless sensor networks. *IEEE/ACM Transaction on Network*, 12(4):609–619, 2004.

[8] N. Chang and M. Liu. Revisiting the ttl-based controlled flooding search: optimality and randomization. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 85–99, New York, NY, USA, 2004. ACM.

[9] S. Chopra and M. R. Rao. On the multiway cut polyhedron. *Networks*, 21(1):51–89, 1991.

[10] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

[11] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. The cougar project: A work-in-progress report. *ACM SIGMOD Record*, 32:53–59, 2003.

[12] R. Duh and M. Frer. Approximation of k-set cover by semi-local optimization. pages 256–265, 1997.

[13] P. L. Erdős and L. A. Székely. On weighted multiway cuts in trees. *Mathematical Programming*, 65(1):93–105, 1994.

[14] N. Garg and J. Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[15] J. Gehrke and S. Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004.

[16] R. Govindan. Data–centric routing and storage in sensor networks. pages 185–205, 2004.

[17] W. Hu, A. Misra, and R. Shorey. Caps: Energy-efficient processing of continuous aggregate queries in sensor networks. In *PERCOM '06: Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 190–199, Washington, DC, USA, 2006. IEEE Computer Society.

[18] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Network (Mobicom)*, Boston, August 2000.

[19] K. Kalpakis, K. Dasgupta, and P. Namjoshi. Maximum lifetime data gathering and aggregation in wireless sensor networks. In *Proceedings of the 2002 IEEE International Conference on Networking (ICN'02)*, August 2002.

[20] S. Kapadia and B. Krishnamachari. Comparative analysis of push–pull query strategies for wireless sensor networks. In *DCOSS 2006, LNCS 4024*, pages 185–201, 2006.

[21] D. Karger. Random sampling in cut, flow, and network design problems. In *Proc. of 25 th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 648–657, 1994.

[22] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Verlag, 1991.

[23] X. Liu, Q. Huang, and Y. Zhang. Combs, needles, haystacks: balancing push and pull for discovery in large-scale sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 122–133, New York, NY, USA, 2004. ACM.

[24] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: The design of an acquisitional query processor for sensor networks. ACM Trans. Database Syst. 30 (1), pp. 122–173, 2005.

[25] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 144–155, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[26] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *Proceedings of the 22nd International Conference on Data Engineering*, Atlanta, GA, April 2006.

[27] Q. Ren and Q. Liang. Energy and quality aware query processing in wireless sensor database systems. *Inf. Sci.*, 177(10):2188–2205, 2007.

[28] N. Sadagopan, B. Krishnamachari, and A. Helmy. Active query forwarding in sensor networks (acquire). *Ad Hoc Networks Journal-Elsevier Science*, 3(1):91–113, 2005.

[29] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev.*, 33(1):137–142, 2003.

[30] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *PODS*, pages 250–258, 2005.

[31] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Hybrid push–pull query processing for sensor networks. In *In Proceedings of the Workshop on Sensor Networks as part of the GI–Conference Informatik 2004*, pages 370–374, 2004.

[32] H. Wu, Q. Luo, and W. Xue. Distributed cross-layer scheduling for in-network sensor query processing. In *PERCOM '06: Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 180–189, Washington, DC, USA, 2006. IEEE Computer Society.

Appendix A. Linear Programming Primer

This appendix provides an overview of some concepts in linear programming used in this paper. Further details can be referred in a text such as [22, 4].

Consider a linear program in standard form

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \quad && \text{such that} \\ &\mathbf{A}\cdot\mathbf{x} = \mathbf{b} \quad && \text{and} \\ &\mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{13}$$

where $\mathbf{A} \in \mathbb{R}^{n\times m}$, $\mathbf{c}, \mathbf{x} \in \mathbb{R}^m$, $\mathbf{b} \in \mathbb{R}^n$, and $n \leq m$. The linear program above defines a convex polyhedron $\mathcal{P} = \{\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$. For convenience, and without loss of generality, suppose that the constraint matrix $\mathbf{A}$ is of full–rank n and that $\mathbf{b} \geq \mathbf{0}$. The case where $\mathbf{A}$ has rank less than n leads to degeneracies requiring special handling, see [4]. We further assume, w.l.o.g., that the polyhedron $\mathcal{P}$ is bounded and non–empty, i.e. the linear program has a bounded optimal solution.

Let $\mathcal{B}$ be a sequence (ordered set) of n column indexes in $\{1, \ldots, m\}$. Let $\mathbf{A}_\mathcal{B}$ be the $n \times n$ sub–matrix of $\mathbf{A}$ whose ith column is $\mathbf{A}_{\mathcal{B}(i)}$. A sequence $\mathcal{B}$ is called a base if $\mathbf{A}_\mathcal{B}$ is of full-rank (invertible). It is called a feasible basis if $\mathbf{A}_\mathcal{B}^{-1}\mathbf{b} \geq \mathbf{0}$. Since $\mathbf{A}$ is of full–rank and the linear program is feasible, a feasible basis always exists. A variable x_i (column $\mathbf{A}_i$) with index in $\mathcal{B}$ is called a *basic variable (basic column)*, otherwise it is called a *non–basic variable (non–basic column)*.

Construct a feasible solution $\mathbf{x}$ corresponding to a feasible base $\mathcal{B}$ by taking $\mathbf{x}_\mathcal{B} = \mathbf{A}_\mathcal{B}^{-1}\mathbf{b}$ and $\mathbf{x}_{\overline{\mathcal{B}}} = \mathbf{0}$. Such a solution is called a *basic feasible solution* (BFS). There is a bijection between basic feasible solutions and vertices (extreme points) of the polytope defined by $\mathbf{A}$. Furthermore, an optimal solution always occurs at one of its vertices.

Associate with each constraint a *shadow price* (or *dual variable*). The shadow prices $\boldsymbol{\pi} \in \mathbb{R}^n$

corresponding to a base $\mathcal{B}$ is given by

$$\boldsymbol{\pi}^T = \mathbf{c}_{\mathcal{B}}\mathbf{A}_{\mathcal{B}}^{-1}. \tag{14}$$

The *relative cost* $\bar{c}_j$ of each non–basic column $\mathbf{A}_j$ is given by

$$\bar{c}_j = c_j - \boldsymbol{\pi}^T\mathbf{A}_j \tag{15}$$

The Simplex method, discovered by Dantzig, systematically explores the set of basic feasible solutions, starting from an initial BFS, until an optimal BFS is found. The process of moving from a BFS to an adjacent BFS is called *pivoting*. In pivoting, we exchange a basic column with a non–basic column, without increasing the cost of the best feasible solution so far.

We describe next a variant of the Simplex method, the Revised Simplex Method (RSM) with the lexico–min rule. An arbitrary non–basic column $\mathbf{A}_j$ enters the current base $\mathcal{B}$ if its relative cost $\bar{c}_j < 0$. If all non–basic columns have relative cost ≥ 0, then the current BFS is optimal and Simplex terminates. Otherwise, a basic column to exit the current base $\mathcal{B}$ needs to be selected. There are multiple approaches to do so. We describe the lexico–min approach for choosing the basic column to exit the current basis $\mathcal{B}$, since it guarantees termination in a finite number of pivoting steps [4]. Let $\mathbf{a}_i$ denote the ith row of the matrix $\mathbf{A}_{\mathcal{B}}$. Let l be the index of the lexicographically smallest row $[x_i, \mathbf{a}_i]/u_i$ with $u_i > 0$,

$$l = \arg \text{lexico–min} \left\{ \frac{[x_i, \mathbf{a}_i]}{u_i} \ : \ u_i > 0 \right\}, \tag{16}$$

where $\mathbf{u} = \mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_j$ and $\mathbf{x} = \mathbf{A}_{\mathcal{B}}^{-1}\mathbf{b}$. Column $\mathbf{A}_j$ enters the base $\mathcal{B}$ replacing column $\mathbf{A}_{\mathcal{B}(l)}$, i.e. $\mathcal{B}(l) \longleftarrow j$. An index l always exists, since otherwise $u_i \leq 0$ for all i and the problem is unbounded.

Extensive computational experience since the discovery of the Simplex method demonstrated that in practice it is an efficient algorithm. The Revised Simplex method offers computational advantages for linear programs with sparse constraint matrices. Moreover, observe that RSM allows us to solve linear programs with exponentially many variables by performing few pivots in practice, provided that we can either find, in polynomial–time, a non–basic column with negative relative cost or show that no such column exists.

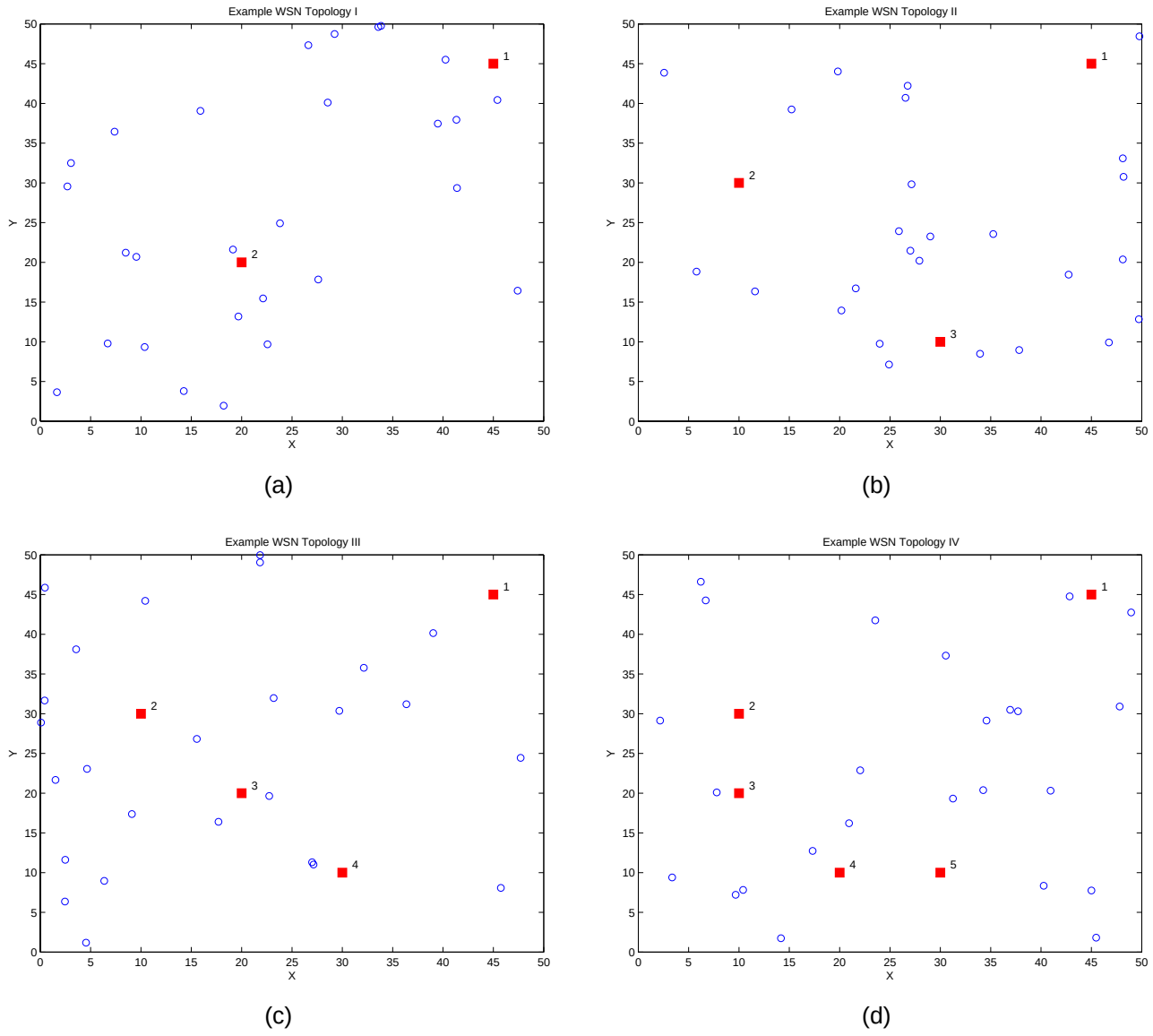


Figure 4: Example wireless sensor networks with topology I, II, III, and IV. Fixed nodes are indicated with squares, while random nodes are indicated with circles. Node 1 is always the base station.

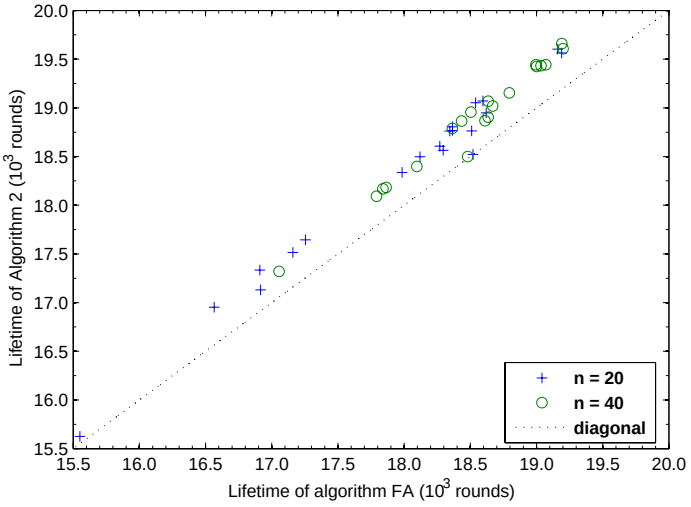
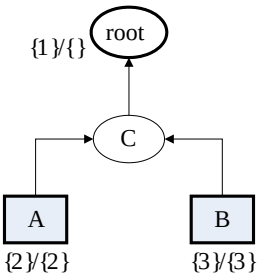
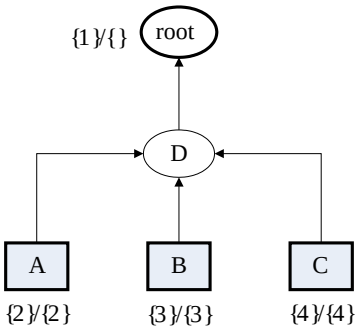


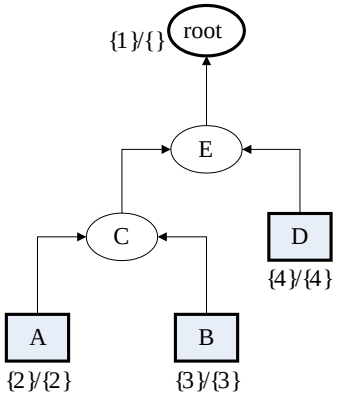
Figure 5: Lifetime achieved by ALGRSM–MLCF and the FA algorithm for the source–destination pair 2–1 with unit demand, for each instance of network topology I.



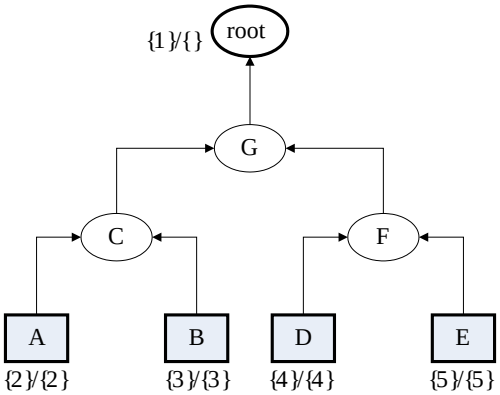
(a) $\mathcal{T}_1$



(b) $\mathcal{T}_2$



(c) $\mathcal{T}_3$



(d) $\mathcal{T}_4$

tree structure	$\mathcal{T}_1$			$\mathcal{T}_2$		$\mathcal{T}_3$					$\mathcal{T}_4$			
vertex size	$\mathbf{s}_{11}$	$\mathbf{s}_{12}$	$\mathbf{s}_{13}$	$\mathbf{s}_{21}$	$\mathbf{s}_{22}$	$\mathbf{s}_{31}$	$\mathbf{s}_{32}$	$\mathbf{s}_{33}$	$\mathbf{s}_{34}$	$\mathbf{s}_{35}$	$\mathbf{s}_{41}$	$\mathbf{s}_{42}$	$\mathbf{s}_{43}$	$\mathbf{s}_{44}$
root	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	1	5	1	1	5	1	5	5	1	5	1	5	5	5
B	1	2	2	1	2	1	2	2	2	2	1	2	2	2
C	1	1	5	1	1	1	1	1	5	1	1	1	1	1
D				1	1	1	1	2	2	2	1	1	5	5
E						1	1	1	1	5	1	1	2	2
F											1	1	1	1
G											1	1	1	5

(e) Vertex size vectors

Figure 6: Expression trees used in the experiments. Each expression tree structure $\mathcal{T}_i$ in (6a)–(6d) and each operand (vertex) size vector $\mathbf{s}_{ij}$ in (6e) provides an expression tree $\mathcal{T}_{ij}$, where the vertices of $\mathcal{T}_i$ have their size specified by the vector $\mathbf{s}_{ij}$. The values 1, 2, 5 are proxies for small, medium, and large size operands.

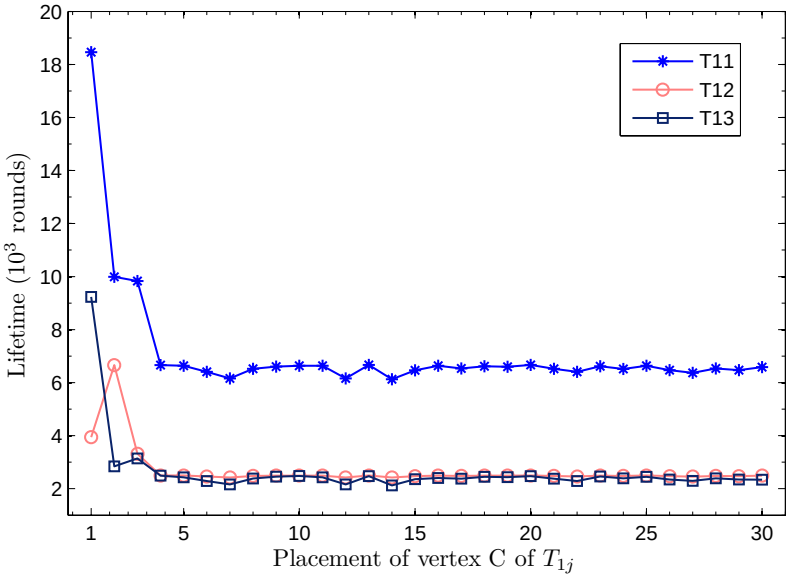
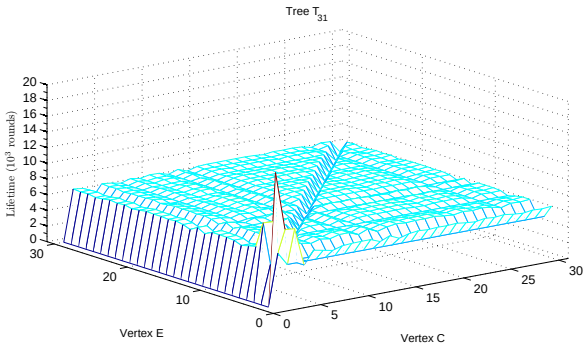
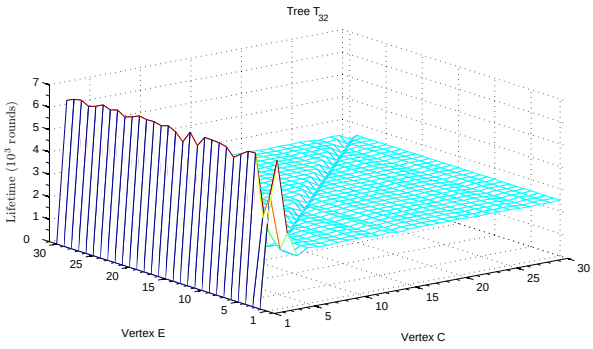


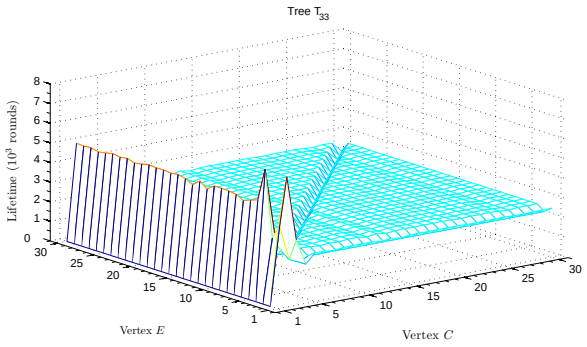
Figure 7: System lifetime vs. placement of vertex C of $\mathcal{T}_1$ in a network instance with $n = 30$ nodes, for different operand size vectors $\mathbf{s}_j$. All other vertices of $\mathcal{T}_1$ are pinned. GREEDYMCP places vertex C of $\mathcal{T}_{11}$, $\mathcal{T}_{12}$, and $\mathcal{T}_{13}$ at network nodes 1, 2, and 1, respectively. All of them are optimal placement.



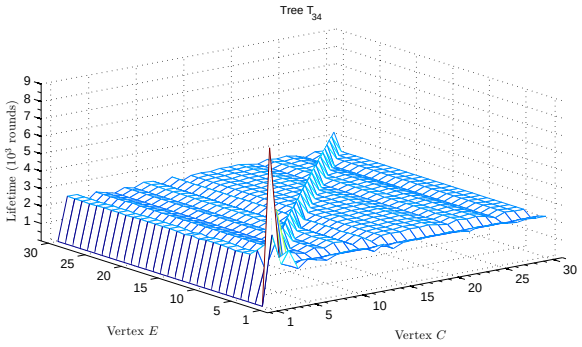
(a) $\mathcal{T}_{31}$



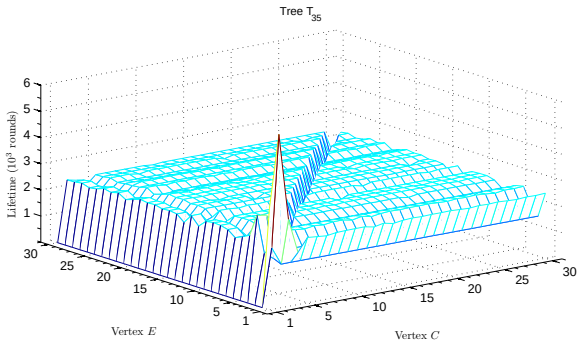
(b) $\mathcal{T}_{32}$



(c) $\mathcal{T}_{33}$



(d) $\mathcal{T}_{34}$



(e) $\mathcal{T}_{35}$

Figure 8: System lifetime sequence $\mathcal{P}_{\mathrm{ALL}}(\mathcal{T}_{3j}, \mathcal{G})$, indexed by the placement of $C, E \in \mathcal{T}_{3j}$, and $\mathcal{P}_{\mathrm{O}}(\mathcal{T}_{3j}, \mathcal{G})$, for an instance $\mathcal{G}$ of network topology IV with $n = 30$ nodes.

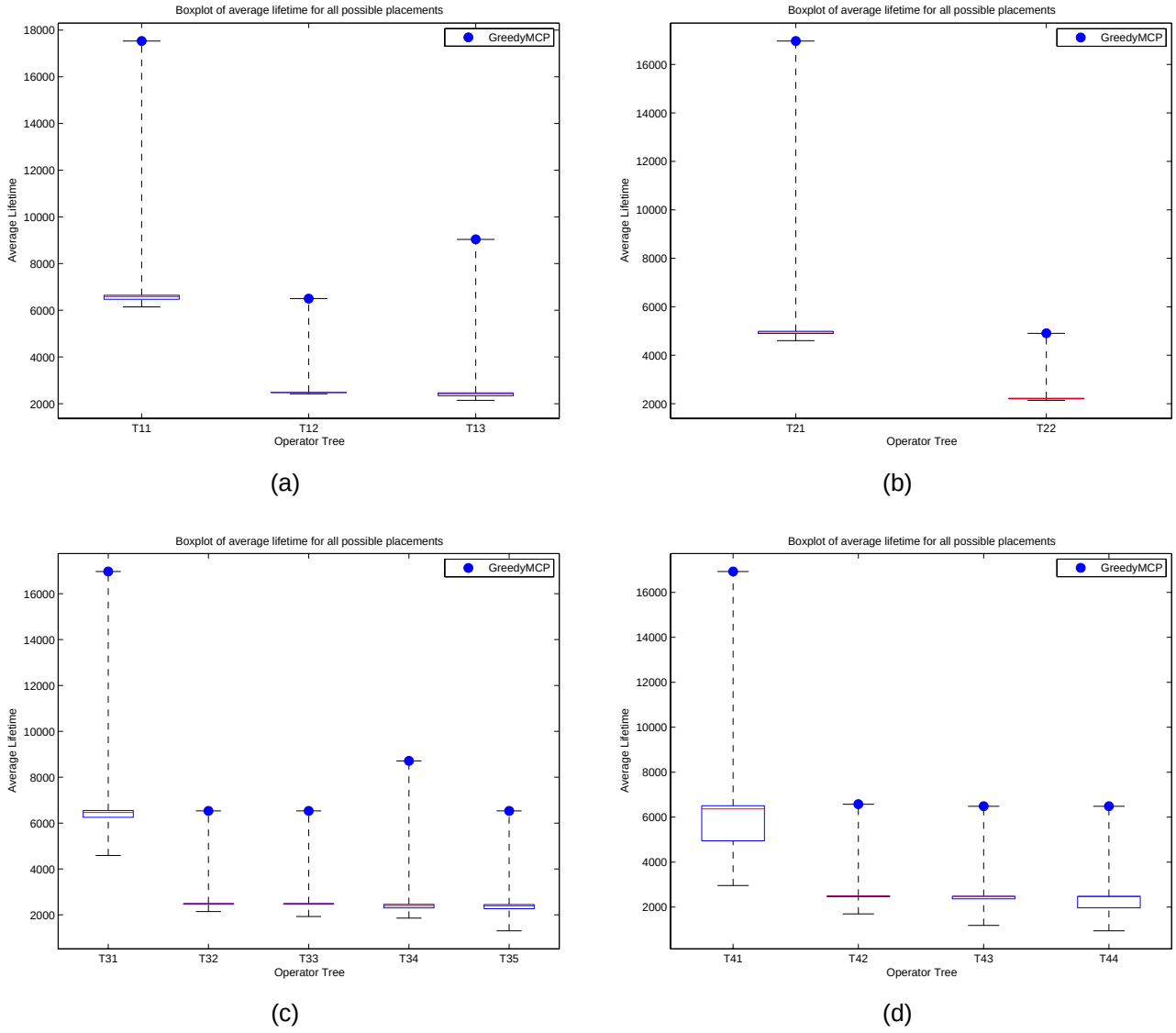


Figure 9: Boxplot of the average system lifetime achieved for each expression tree for all possible placements, and across all the relevant random network topologies. We also show the lifetime achieved by the GREEDYMCP placement.

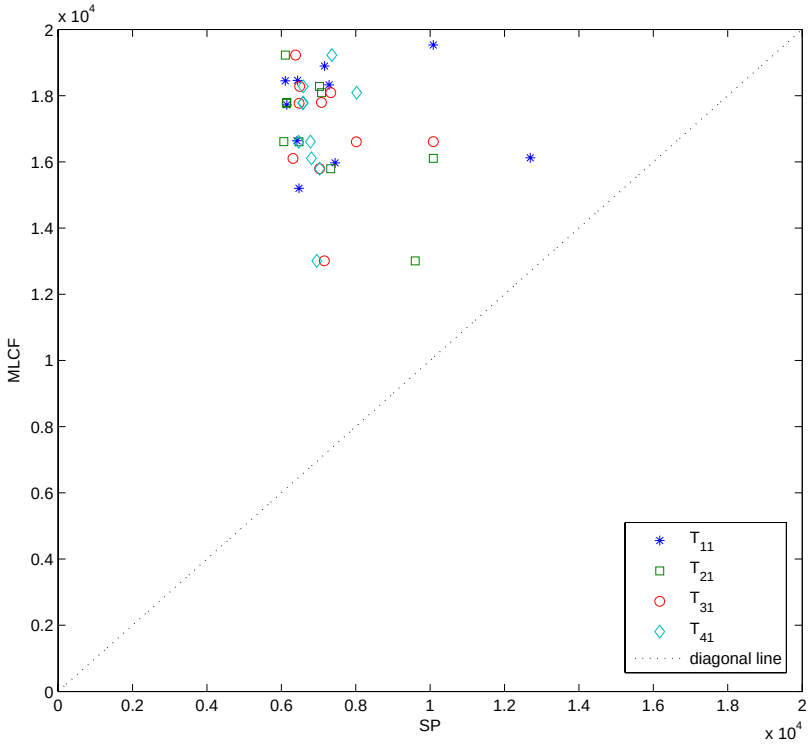


Figure 10: Optimal lifetime achieved by using fixed (shortest) paths vs. using ALGRSM–MLCF to route the required operand values during the continuous evaluation of $\mathcal{T}_{i1}$.