ROUTING IN THE INTERNET OF THINGS

MARCEAU COUPECHOUX

1. Introduction

Internet of Things (IoT) is characterised by the deployment of a large amount of connected objects. These objects are autonomous devices that use sensors to monitor their environment and communicate through a wireless radio channel. When these sensors are not powerful, they cannot communicate over a long distance, so that additional intermediate relay nodes may be required to connect them together. Sensors can themselves route the messages from other sensors or relays. In this project, we derive the optimal routes between sensors by using the notion of Steiner tree in graph theory.

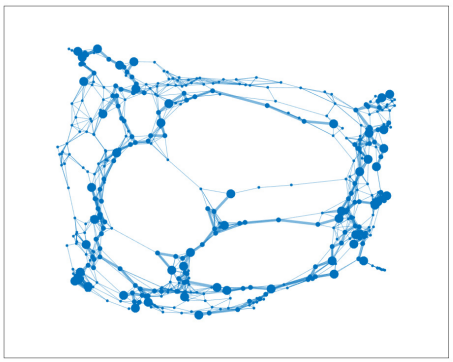

Figure 1. Example of sensor network with terminal nodes and Steiner minimal tree (in bold).

The sensor network is modelled as a undirected weighted graph, whose nodes are sensors and relay nodes. An edge indicates that a wireless communication is possible between two nodes. An edge weight represents the cost of transmitting a message along this edge (either in terms of power or delay). We look for a set of routes of minimal weight connecting all sensors together. In graph theory, this is related to the Steiner tree problem, where sensors are terminals and routes form a Steiner tree.

We start with some definitions, then implement preprocessing steps to reduce the size of the instances. At last, we implement exact and approximation algorithms

Date: 20th Sept. 2022.

to solve the problem. Six instances are provided to test your algorithms: scenarios 1 to 6.

2. Definitions

We consider $G = (V, E, c)$, a connected undirected weighted graph with vertex set V, edge set E and weight function $c : E \mapsto \mathbb{R}_+$. We set $n = |V|$ and $m = |E|$.

Definition 1 (Steiner tree)**.** *Let $K \subseteq V$ be a non-empty subset of k vertices. A subgraph T of G with vertex set $V(T)$ is called a Steiner tree for K, if T is a tree containing all vertices of K, i.e., $K \subseteq V(T)$. The vertices of K are called* terminals *of T, vertices in $V(T) \backslash K$ are called* Steiner points *of T, vertices in $V \backslash K$ are called* non-terminals*. Steiner points with degree at least 3 are called* branching points*.*

Definition 2 (Steiner problem in networks)**.** *Given G and K, find a Steiner minimal tree for K in G, i.e., a Steiner tree T for K such that*

$$c(T) = \min\{c(T') | T' \text{is a Steiner tree for } K \text{ in } G\}$$

where $c(T') = \sum_{e \in T'} c(e)$.

Note that if $K = \{s, t\}$, the Steiner problem reduces to the problem of finding a shortest path between vertices s and t. If $K = V$, then the Steiner tree problem reduces to the problem of finding a minimum spanning tree in G. However, in general, the Steiner tree problem is an NP-hard optimization problem.

The distance network D of G is the complete weighted graph with vertices V and weight function $d_G : V \times V \mapsto \mathbb{R}_+$, where, for $v, w \in V$, $d_G(v, w)$ is the minimum distance between v and w in G. It can be shown that solving an instance of the Steiner tree problem for K in G is equivalent to solving an instance of the Steiner tree problem for K in D. Given a Steiner minimal tree T_D for K in D, a Steiner minimal tree T for K in G is obtained by replacing edges in T_D by the corresponding shortest paths.

3. Pre-processing Steps

In this first part, we apply some pre-processing steps to reduce the size of a given instance of the Steiner tree problem.

Question 1 (Non-terminals of degree 1)**.** Show that the edge connecting a non-terminal of degree 1 in G cannot be in any Steiner minimal tree.

Question 2. Give the pseudo-code of this pre-processing step, implement the algorithm, describe your data structure and show the results on scenarios 1 to 6 with CPU times.

Question 3 (Non-terminals of degree 2)**.** Suppose that a non-terminal v_k is of degree 2, let $e_1 = (v_i, v_k)$ and $e_2 = (v_k, v_j)$ be its two edges and let $e = (v_i, v_j)$. We take the convention that if $e \notin E$, then $c(e) = \infty$. Show that if $c(e_1) + c(e_2) \geq c(e)$, then there is at least one Steiner minimal tree not containing v_k. Hence v_k can be deleted from G. Show that if $c(e_1) + c(e_2) < c_e$, then the edge e cannot belong to any Steiner minimal tree and can be deleted from G.

Question 4. Give the pseudo-code of this pre-processing step, implement the algorithm, describe your data structure and show the results on scenarios 1 to 6 with CPU times.

Question 5 (Long edges)**.** Show that any edge $e = (v_i, v_j) \in E$ such that $c(e) > d(v_i, v_j)$, where $d(v_i, v_j)$ is the minimum distance between v_i and v_j, can be deleted from G.

Question 6. Give the pseudo-code of this pre-processing step, implement the algorithm, describe your data structure and show the results on scenarios 1 to 6 with CPU times.

4. Exact Algorithms

Although there is no known polynomial time algorithm for the Steiner tree problem, there are exact algorithms that solve the problem, at least for small instances.

4.1. **Enumeration Algorithm.** The Enumeration Algorithm, as its name suggests, enumerates all the Steiner trees in the graph and chooses one with the minimum weight. In order to restrict the number of inspected trees, we rely on some properties of Steiner trees (we don't ask for proofs):

- Looking for a Steiner minimal tree in G is equivalent to find a Steiner minimal tree in D.
- There exists a Steiner minimal tree for K in D where each Steiner vertex is a branching point.
- There exists a Steiner minimal tree for K in D with at most $k-2$ Steiner vertices.

As a consequence, if we were given the set S of branching points, a Steiner minimal tree in D would be a minimum spanning tree in the subgraph induced by $K \cup S$ of cardinality $|K \cup S| \leq 2k - 2$.

Question 7. Based on these observations, give the pseudo-code of the Enumeration Algorithm, implement it, describe your data structure and show the obtained Steiner tree and its weight on scenarios 1 to 6 with CPU times.

Question 8. What is the complexity of the Enumeration Algorithm?

4.2. **The Dreyfus-Wagner Algorithm.** We now study a dynamic programming approach for the Steiner tree problem. We will first compute Steiner minimal trees for all 2-element subsets of K. We will then use these results to compute Steiner minimal trees for all 3-element subsets of K, and so on and so forth.

For $X \subseteq K$ and $v \in V \backslash X$, let $s(X \cup v)$ denote the length of a Steiner minimal tree for $X \cup v$ (with no restriction on the degree of v). Let $s_v(X \cup v)$ denote the length of a Steiner minimal tree for $X \cup v$ in which v has degree at least 2. These values can be computed recursively.

Let T be a Steiner minimal tree for $X \cup v$ with no restriction on the degree of v. For a leaf u in T, we denote P_u the longest path in T starting in u in which all interior points have degree 2 in T and are non-terminals. We consider the following cases:

- v has degree at least 2 in T. Then $s(X \cup v) = s_v(X \cup v)$. In this case, T is the union of two Steiner minimal trees, one spanning $X' \cup v$ and the other spanning $X \backslash X' \cup v$ where $\emptyset \neq X' \subsetneq X$.
- v is a leaf in T and P_v ends at a vertex $w \in X$. Then, T is the union of a Steiner minimal tree for X and a shortest path from v to w. Hence, $s(X \cup v) = s(X) + d(v, w)$, where $d(v, w)$ is the minimum distance between v and w.

Algorithm 1: Distance Network Heuristic

1: **Input:** A graph $G = (V, E, c)$ and a terminal set $K \subseteq V$.
2: **Output:** A Steiner tree $T_G(K)$.
3: Compute the distance network $D(K)$ between terminals
4: Compute a MST T_D in $D(K)$
5: Transform T_D into a subgraph of G by replacing every edge in T_D by the corresponding shortest path
6: Compute a MST T for the subgraph
7: Transform T into a Steiner tree for G by deleting leaves which are not terminals (one at a time).
8: **return**

- v is a leaf in T and P_v ends at a vertex $w \notin X$. Then w has degree at least 3 in T. T is the union of a Steiner minimal tree for $X \cup w$ and a shortest path from v to w, i.e., $s(X \cup v) = s_w(X \cup w) + d(v, w)$.

Question 9. Based on these observations, derive two coupled recursive dynamic programming equations for $s_v(X \cup v)$ and for $s(X \cup v)$.

Question 10. Give the pseudo-code of the Dreyfus-Wagner Algorithm, implement it, describe your data structure and show the obtained Steiner tree and its weight on scenarios 1 to 6 with CPU times.

Question 11. What is the complexity of the Dreyfus-Wagner Algorithm?

5. Approximation Algorithms

Because of the exponential complexity of exact algorithms, it is preferable to rely on polynomial heuristics for large instances.

5.1. **Distance Network Heuristic.** We start with an heuristic based on the computation of a minimum spanning tree (MST) in the distance network. This heuristic has a competitive ratio of 2. A rough pseudo-code is shown in Algorithm 1.

Question 12. Give a more detailed pseudo-code of the Distance Network Heuristic, implement it, describe your data structure and show the obtained Steiner tree and its weight on scenarios 1 to 6 with CPU times.

5.2. **Shortest-Path Heuristic.** Another heuristic, also with competitive ratio 2, called the Shortest-Path Heuristic, consists in starting with an arbitrary terminal in T, add to T a terminal not in T closest to T, determine a MST for the subgraph of G induced by T, delete from the MST non-terminals of degree 1 and repeat this procedure until there is no more terminal in $G \backslash T$.

Question 13. Give a pseudo-code of the Shortest-Path Heuristic, implement it, describe your data structure and show the obtained Steiner tree and its weight on scenarios 1 to 6 with CPU times.

Question 14. Show that the Distance Network Heuristic and the Shortest-Path Heuristic are polynomial-time algorithms.

5.3. **Statistical Analysis.** We consider random connected networks with the following characteristics. There are $n = 500$ nodes, among which $k = 50$ terminals. The network area is a square of side $L = 1$. Node locations are uniformly distributed in the network area. Assume $\delta = 0.025$. A communication between two nodes is possible with cost 1, 2, 3, if their Euclidian distance is respectively less than δ, 2δ and 3δ. If their distance is more than 3δ, there is no possible communication.

Question 15. Compare the statistical performance of the Distance Network Heuristic and the Shortest-Path Heuristic.