Reproducibility of Superposed Data Uploading Problem in Networks With Smart Devices Paper to Optimize The Energy Communication Cost

Documentation

Israa Moustafa

Abstract—**The project focuses on reproducing of "Complexity and Algorithms for Superposed Data Uploading Problem in Networks With Smart Devices" paper which applies Dijkstra's and Prim's algorithms to optimize the energy communication cost between smart deceives and vehicle base stations in dangerous environments. The project was implemented using Python to deploy the D-SDUP and P-SDUP algorithms applying scenarios mentioned in the paper as experimental setups to analyze the effect of graph density and VBs capacity constraint on both the cost and execution time of both algorithms. After the analysis of the results, we found that the P-SDUP algorithm shows better performance in most cases regarding execution time and cost.**

I. Introduction

The extensive application of smart devices (SDs) enhanced by edge computing, particularly in settings without or with hazardous human intervention, has greatly improved the performance of toxic gas leaks, where SDs use sensor and AI applications to collect and process data to help decision-making and intervention in execution time. Still, there is the problem of how communication between SDs and vehicle-based stations (VBSs) can be used in mobile edge computing to strike a balance between energy consumption and efficient transmission of data.

In this project we'll continue to dive deep into paper [1] which shows the energy problem in superposed data uploading networks with SDs, Optimal communication links and transmission strategy for SDs to VBSs were taken into consideration under the power resource constraints and system effective constraint. The primary objective is to implement and further investigate the problem and algorithms proposed in the paper.

The introduction provides an overview of the scenario, highlighting the critical importance of SDs and VBSs in hazardous environments and the need for energy-efficient communication protocols. The following sections of the paper will explore related work, define the problem, outline the methodology, present results, and analyze the findings, with the aim of replicating and building upon the original research.

II. Related Work

Optimizing communication energy consumption is a widely searched topic in the fields of wireless sensor networks (WSNs), IoT, and mobile edge computing (MEC). In their work, Mao et al. [2] developed online algorithms for MEC that effectively minimize device energy consumption by offloading computation workloads. The SMECN [3] project aimed to identify the most energy-efficient paths in WSNs. They compare SMECN and MECN algorithms that were made for energy-efficient communication optimization in wireless networks by iteratively determining neighbor sets and transmission powers based on signal strengths and distances. In SMECN, neighbor sets are computed using circular search regions while in MECN, they are computed from adjacent nodes that aren't in each other's neighbor sets. Lazarescu [4] introduced an energy-saving environmental monitoring scheme for WSN platforms, highlighting the need to balance latency and energy consumption. Abedin et al. [5] proposed a system model for energy-efficient networks that encompassed various types of devices.

In the realm of device-to-device (D2D) communication, efforts have been directed toward finding the tradeoffs between energy consumption and spectral efficiency [6] [7], as well as resource sharing [8] and resource allocation [9].

Rasheed et al. [10] proposed connected target coverage algorithms specifically designed for energy-efficient industrial WSNs. Amer et al. [11] focused on minimizing energy consumption in clustered D2D networks. Patil and Mishra [12] concentrated on energy-efficient data collection in 3-D underwater sensor networks.

III. Problem Definition

For some environments that are hard or dangerous to explore by humans, such as toxic gas leakage in a factory, It is useful to use smart devices (SDs) like IoT sensors. Environmental Data collected from those smart devices are further processed by AI-based applications installed on the edging computing servers deployed on moving robots called vehicle base stations (VBSs). Edge computing is used to ensure that SD and AI applications are computationally intensive and latency-sensitive. Although the coverage area of VBSs is smaller, they are more practical to deploy in certain dangerous environments than fixed BSs.

The problem discussed revolves around optimizing the utilization of resources within a network comprising VBS servers and SDs. SD devices, which have computing capabilities, can interact with each other in device-to-device (D2D) communication proximity or send data to VBSs within their coverage range. Initially, these facilities weren't aware of each other's existence, so they have to exchange information like location and signal strength to facilitate communication. Communication cost tables store signal specifics between facilities, which are regularly updated to accommodate any changes. SD devices execute applications for data processing and transmission, this data can't be retrieved once merged with the next node's data, and the merged data has the same size as the original ones. VBSs suffer from limitations, so they can't control all SD decives due to the limitations of their computation capacity constraints. That's why SDs can't always be connected to the nearest VBS with the least required power to communicate with.The primary objective is to minimize energy consumption during data uploads by identifying communication links from each SD device to VBSs and establishing the transmission sequence for SD devices accordingly. Energy expenses are impacted by factors such, as signal specifics, topography conditions, environmental elements, and device distances.

The project's objective is to identify efficient communication links and data transmission order from each SD to some VBSs that minimize the total energy cost for communication.

IV. Problem Model

Let G graph as $G = (V = (D \cup S), E, w)$, so that D is the set of all SDs, S is the set of all VBSs, E is the edge between any node (SD or VBS), and W is the set of weights of communication between any node. Additionally, $c(s)$ the capacity of VBSs for each VBS $s \in S$ is the number of SDs it can control. The Figure 1 present example of graph G.

$$\text{Let } x_{i,d} = \begin{cases} 1 & \text{if element } d \text{ is included in tree } T_i, \forall d \in D \\ 0 & \text{otherwise} \end{cases}$$

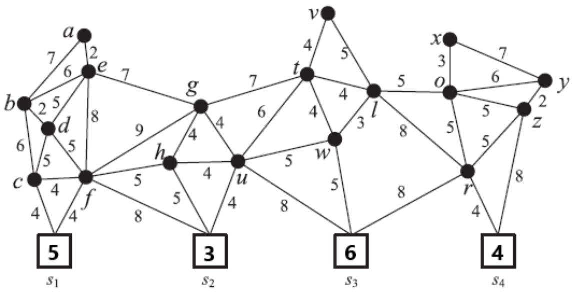

Fig. 1: Example graph G.

Objective Function:

The objective is to minimize the total communication cost of all quasi-tree T_i:

$$\text{Minimize:} \quad \sum_{i=1}^{n} w(c_{t_i})$$

Non negative constraints:

$$|D| > 0 \tag{1}$$

$$|S| > 0 \tag{2}$$

Constraints:

1) Each SD must be connected to one VBS (quasi-tree T_i):
$$\sum_{i=1}^{n} x_{i,d} = 1 \quad \forall d \in D$$
2) VBS capacity constraint:
$$s(c_{t_i}) \leq c(s_i) \quad \forall i \in [1, |S|]$$
3) Communication cost weights are positive integers:
$$w(e) > 0 \quad \forall e \in E$$

V. Methodology

We implemented 2 Algorithms that were in the paper to solve the Superposed Data Uploading Problem (SDUP):

A. *Algorithm 1(D-SDUP)*

The first Algorithm uses **Dijkstra's algorithm** [13] to generate a communication path from any SD to any VBS. The algorithm is presented in **Algorithm 1**.

First, we create subtrees for each VBS and let D be the set of SDs that aren't assigned to any subtree. We go through an iteration process until D is empty, where we pick any random node $v \in D$ and use Dijkstra's algorithm to find the shortest path from v to some tree T_j so that the number of added nodes from D is less or equal to the remaining capacity of T_j. Then, add the nodes to T_j and remove them from D. If it is larger than the capacity, remove nodes from the shortest path starting with node v until it fits the capacity. Add the remaining nodes in the path to T_j and remove them from D. The output is the subtrees containing the SDs connected to each VBS.

B. *Algorithm 2(P-SDUP)*

Besides **Dijkstra's algorithm**, The Second Algorithm uses **Prim's algorithm** [13] to hold the global optimal property in finding the minimum spanning trees. First Prim's algorithm is used to assign as many DC nodes as possible to the VBSs, After that Algorithm 1 is used to assign the rest. The algorithm is presented in **Algorithm2**.

After creating subtrees for each VBS and letting D be the set of SDs that aren't assigned to any subtree as the first algorithm, we add another step where we apply Prim's algorithm to apply the growing strategy on the nodes $\in D$ that are adjacent to any subtree T_j which satisfies the condition that $V(T_j) \leq c(v_i)$. More preciously, the algorithm selects an edge with the minimum weight and adds it to some tree so that one node is in D, and the other is in that tree. We repeat this step until the remaining nodes in D aren't adjacent to any of the subtrees that still have free capacity. Then, we apply the steps of Algorithm 1.

Algorithm 1 D-SDUP

```
Require: An edge-weighted undirected graph $G = (V = D \cup S, E, w)$.
Ensure: $|S|$ subtrees of $G$.
1: Let subtree $T_i = v_i$ for each $v_i \in S$, and $D = V \backslash S$ be the set of DC nodes not in any subtree.
2: while $D \neq \emptyset$ do
3:     Randomly pick a node $p_1$ from $D$.
4:     Find the shortest path $P = (p_1, p_2, \ldots, p_k)$ from $v$ to $\bigcup_{|V(T_i)| \leq c(v_i)} V(T_i)$ using Dijkstra's algorithm. Assume $p_k$ is in the tree $T_j$.
5:     if $|V(P) \cap D| < c(v_j) - |V(T_j)| + 1$ then
6:         Add $P$ into $T_j$ and $D = D \backslash V(P)$.
7:         Mark the nodes in $V(P) \cap D$ as the DC nodes.
8:     else
9:         Add $P' = (p_q, \ldots, p_k)$ into $T_j$ and $D = D \backslash V(P')$ where $p_q \in D$ and $|V(P') \cap D| = c(v_j) - |V(T_j)| + 1$.
10:        Mark the nodes in $V(P') \cap D$ as DC nodes.
11:    end if
12: end while
13: Output $T_1, T_2, \ldots, T_{|S|}$.
```

Algorithm 2 P-SDUP

```
Require: An edge-weighted undirected graph $G = (V = D \cup S, E, w)$.
Ensure: $|S|$ subtrees of $G$.
1: Let subtree $T_i = v_i$ for each $v_i \in S$, and $D = V \backslash S$ be the set of DC nodes not in any subtree.
2: while $N_G \left( \bigcup_{|V(T_i)| \leq c(v_i)} V(T_i) \right) \cap D \neq \emptyset$ do
3:     Pick an edge $(u, w)$ with minimum weight in $G$ such that $u \in V(T_j)$, $w \in D$, and $c(v_j) - |V(T_j)| > 0$, $1 \leq j \leq |S|$.
4:     Add edge $(u, w)$ into $T_j$,$D = D \backslash w$ and mark $w$ as a DC node.
5: end while
6: while $D \neq \emptyset$ do
7:     Randomly pick a node $p_1$ from $D$.
8:     Find the shortest path $P = (p_1, p_2, \ldots, p_k)$ from $v$ to $\bigcup_{|V(T_i)| \leq c(v_i)} V(T_i)$ using Dijkstra's algorithm.
9:     Assume $p_k$ is in the tree $T_j$.
10:    if $|V(P) \cap D| < c(v_j) - |V(T_j)| + 1$ then
11:        Add $P$ into $T_j$ and $D = D \backslash V(P)$.
12:        Mark the nodes in $V(P) \cap D$ as the DC nodes.
13:    else
14:        Add $P' = (p_q, \ldots, p_k)$ into $T_j$ and $D = D \backslash V(P')$ where $p_q \in D$ and $|V(P') \cap D| = c(v_j) - |V(T_j)| + 1$.
15:        Mark the nodes in $V(P') \cap D$ as DC nodes.
16:    end if
17: end while
18: Output $T_1, T_2, \ldots, T_{|S|}$.
```

VI. Implementation

I implemented the optimization problem by myself using Python without a solver, utilizing object-oriented programming (OOP) principles to create a graph class. I also created functions for the Dijkstra algorithm, the D-SDUP and P-SDUP algrithms were implemented as functions.

I choose to implement the problem by myself instead of using a solver to enable flexibility to adjust the algorithms to ensure a proper reproducibility of the paper. solvers are very powerful in certain optimization problems but lack a high degree of customization, dynamic decision-making, or specialized graph manipulations which is needed to implement specific algorithms.

Both P-SDUP and D-SDUP algorithms functions were implemented from scratch. They take the graph, VBS source nodes, and their capacities as input. Initially, we create a dictionary for each subtree that contains information about each node in the subtree like id, its state (forwarding or data collecting node), its mother node, and the cost of adding this node to the subtree, which we can sum at the end to get the total cost of each tree. Then I implemented the proposed algorithms. In the end, they return the updated subtrees.

The Graph class and Dijkstra's algorithm are inspired from [14] with minor alterations to fit my problem. The Graph class is a data structure that are initialized by a set of vertices and the connecting edges. It has some functions to deal with the graph like adding a new node or edge to the graph. also getting the neighbors of a specific node. Another method that was helpful to calcualte the communication cost is the value which return the value of the edge between any two nodes in the graph. Dijkstra algorithm takes a graph as an adjacency matrix and the start source node. It returns an array of shortest distances from the source node to each other node in the graph, and all the nodes between them in this shortest path. //

As all the code was made from scratch, all the used libraries are basic libraries for plotting for example, below is a list of the Python libraries utilized in the project :

- sys
- random
- tqdm
- matplotlib.pyplot
- time

VII. Experimental Set-up

A. *Hardware*

the used hardware for project implementation is NVIDIA GeForce RTX 3070 GPU using anaconda. The used language will be Python.

B. *Data and implementation*

To test my model, I implemented the two setup scenarios described in the paper to compare the results, the 2 scenarios were :

Number of Servers	Capacity Ranges
10	[20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
15	[20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
20	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
25	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

TABLE I: Capacity Ranges for Each Number of Servers, k, for Experimental Setup for Scenario 2

1) ***Experimental Setup for Scenario 1:*** 200 SDs are randomly deployed and VBSs are randomly located in the same area. Depending on probability p the edges are deployed and a randomly generated positive integer is assigned as the edge weight. the range of probability p is [0.05, 0.10, ..., 0.50], so the higher values of p represent higher communication probabilities between all nodes. k is the number of servers and it varies in the set [10, 15, 20, 25], then we calculate the capacity of each server as n/k. D-SDUP and P-SDUP algorithms are implemented 100 times for each combination of parameters.

2) ***Experimental Setup for Scenario 2:*** 200 SDs are randomly deployed and VBSs are randomly located in the same area. the communication probability is fixed as $p = 0.20$. k varies in the set [10, 15, 20, 25], then the capacity is assigned depending on k value as in the table I. D-SDUP and P-SDUP algorithms are implemented 100 times for each combination of parameters.

VIII. Results

The paper implemented scenarios that deployed random nodes, edges, and weights with specific ranges and compared the results for both algorithms. I implemented the scenarios they deployed and compare the results to ensure that the algorithms were implemented properly. Also, I added execution time analysis which wasn't in the original paper. I think it added a lot to the comparison.

After trying different p values, My results show a similar curve pattern to the paper results shown in figures 4 as the cost will decrease gradually when p values increase. also, both algorithms D-SDUP and P-SDUP cost for each p-value are very close to each other however most of the time D-SDUP shows better performance with a slight difference between the cost of the two models. I also studied the mean required execution time of both models in each case. and we can see in 11 that P-SDUP always took a couple of seconds and the p-value increase has nearly zero effect on the execution time while the D-SDUP algorithm started with about 30 seconds at p-value = 0.05 and the execution time taken by the algorithm increased linearly until it reached 70 sec at p-value = 0.5 which is more than the double.

In the results of the testing of different capacities 7, we can see that cumulative capacity is getting larger and larger than the number of nodes the gap between the performance of the two algorithms increases as the P-SDUP gives a higher drop in the cost with capacity increase compared to D-SDUP. For the execution time, We can see in figure 12 a similar pattern of time linear increase in P-SDUP until some point where the curve started to saturate.

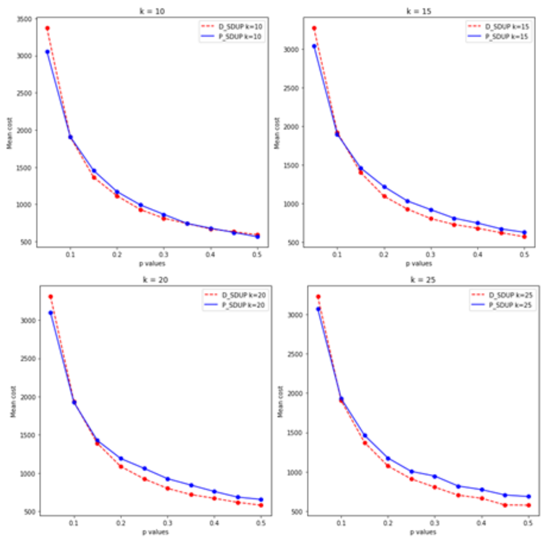

Fig. 2: My project experimental results

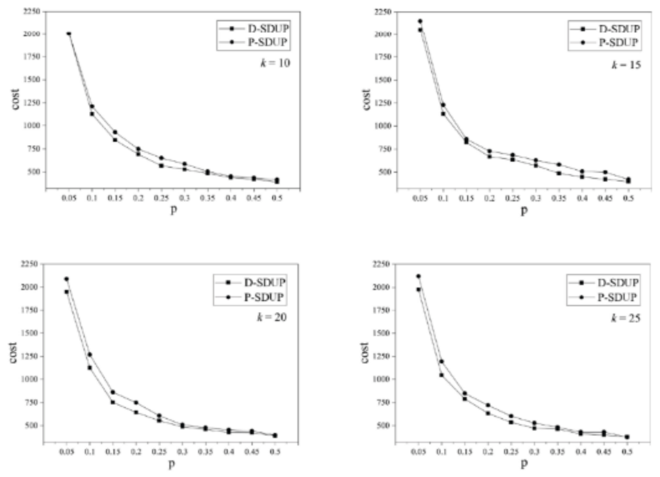

Fig. 3: The paper experimental results

Fig. 4: Performance comparison of two algorithms D-SDUP and P-SDUP with different values of k and p. The capacity of each server is n/k , where n = 200 is the number of SDs and k is the number of servers. Experimental results for the cases that the values of k are 10, 15, 20, and 25, respectively.

IX. Analysis

The algorithm D-SDUP slightly outperforms P-SDUP when the density of the graph is relatively small. When the value of p (the probability of nodes being connected by an edge) is near 0.5, the performance is almost the same. Increasing the density of the graph increases the effect of Prim's algorithm in P-SDUP.

When the value of c reaches a certain threshold, as the cumulative capacity is way larger than the number of nodes,

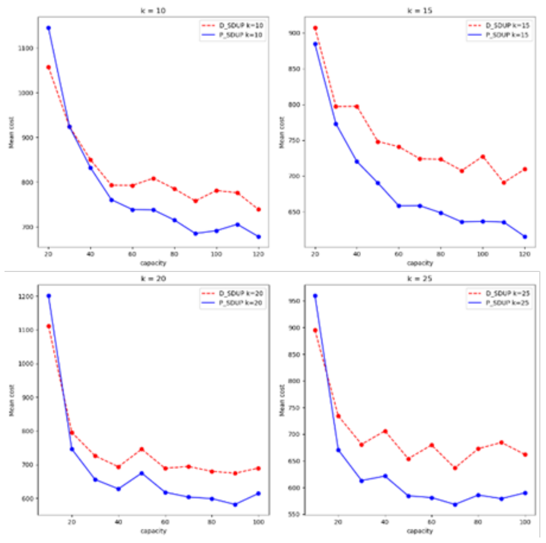
Fig. 5: My project experimental results

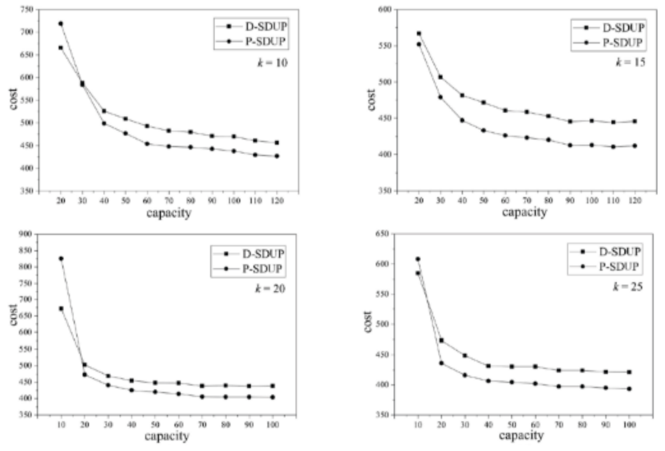
Fig. 6: The paper experimental results

Fig. 7: Performance comparison of two algorithms D-SDUP and P-SDUP with p = 0.20 and n = 200. The capacity of each server is the same, and the sum of the capacities of all servers is larger than the number of SDs. Experimental results for the cases that the values of k are 10, 15, 20, and 25, respectively

P-SDUP outperforms D-SDUP as the size of the subtree for each VBS is hardly constrained by its capacity, and the sub-algorithm Prim could find almost all the local optimal edges.

In the case of execution time, we can see that the prim algorithm is extremely faster than the Dijkstra algorithm, as the Dijkstra algorithm has to go through all nodes to recalculate the optimal path, while the prim algorithm only looks at the neighboring edges and there are no calculations. We can conclude that in the prim algorithm, the near quick gain has the whole attention rather than the global optimal gain. Although the Dijkstra algorithm is more considered in the global optimal gain, the first selected random points get more attention.

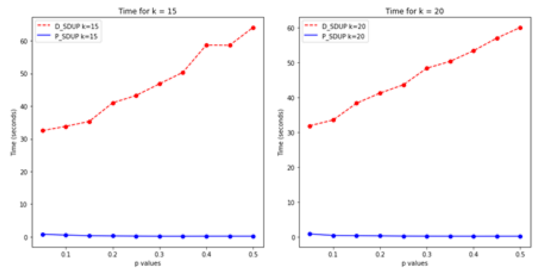
Fig. 8: Execution time change with respect to p values

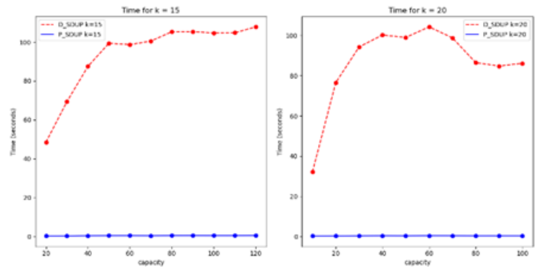
Fig. 9: Execution time change with respect to capacity values

Fig. 10: Requried execution time comparison of the two algorithms D-SDUP and P-SDUP for different p values and different capacities

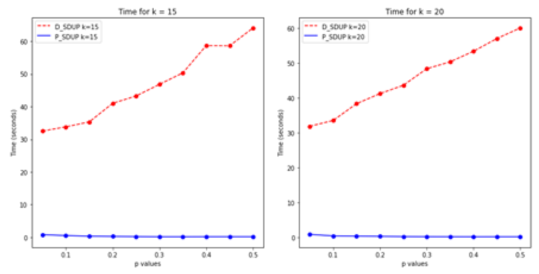
Fig. 11: Execution time change with respect to p values

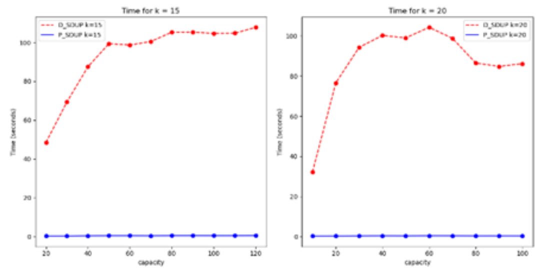
Fig. 12: Execution time change with respect to capacity values

Fig. 13: Requried execution time comparison of the two algorithms D-SDUP and P-SDUP for different p values and different capacities

Also, the execution time increases in the D-SDUP algorithm with the increase in p-value as the required calculations increase with the increase of edges. I think it increases with capacity rise for two reasons. First, when the cumulative capacity is exactly the number of nodes, you will be removing whole sub-trees with their nodes from the calculation of the Dijkstra algorithm when the capacity is reached, which reduces the complexity of the algorithm. but when increasing the capacity, you will have all the sub-trees until the end. Another reason is that increasing capacity can increase the complexity of some sub-trees increasing the execution time required for calculations. This complexity reaches a specific point where it doesn't matter anymore as it reaches almost its max, this is where it saturates.

To conclude, in general P-SDUP is better than the D-SDUP algorithm especially when in cases of having numerous SD nodes, the capacity constraints are loose and the graph has high density having better connection between all SDs. First, the execution time taken by the algorithm doesn't change by increasing complexity, whether the number of nodes, density (p-value), or the capacity of VBSs. Second, it shows a very similar performance to D-SDUP or even outperforms it in many cases. The D-SDUP algorithm would be useful when we have very tight constraints, the optimal is critical, the time parameter is not the main concern and the number of nodes is not large.

References

[1] W. Li, H. Xu, H. Li, Y. Yang, P. K. Sharma, J. Wang, and S. Singh, "Complexity and algorithms for superposed data uploading problem in networks with smart devices," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5882–5891, 2020.

[2] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[3] L. Li and J. Y. Halpern, "Minimum-energy mobile wireless networks revisited," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2001, pp. 278–283.

[4] M. T. Lazarescu, "Design of a wsn platform for long-term environmental monitoring for iot applications," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 3, no. 1, pp. 45–54, Mar. 2013.

[5] S. F. Abedin, M. G. R. Alam, R. Haw, and C. S. Hong, "A system model for energy efficient green-iot network," in *Proc. IEEE Int. Conf. Inf. Netw. (ICOIN)*, 2015, pp. 177–182.

[6] Z. Zhou, M. Dong, K. Ota, J. Wu, and T. Sato, "Energy efficiency and spectral efficiency tradeoff in device-to-device (d2d) communications," *IEEE Wireless Commun. Lett.*, vol. 3, no. 5, pp. 485–488, Oct. 2014.

[7] L. Wei, R. Q. Hu, Y. Qian, and G. Wu, "Energy efficiency and spectrum efficiency of multihop device-to-device communications underlaying cellular networks," *IEEE Trans. Veh. Technol.*, vol. 65, no. 1, pp. 367–380, Jan. 2016.

[8] D. Wu, J. Wang, R. Q. Hu, Y. Cai, and L. Zhou, "Energy-efficient resource sharing for mobile device-to-device multimedia communications," *IEEE Trans. Veh. Technol.*, vol. 63, no. 5, pp. 2093–2103, Jun. 2014.

[9] C. Yang, X. Xu, J. Han, and X. Tao, "Energy efficiency-based device-to-device uplink resource allocation with multiple resource reusing," *Electron. Lett.*, vol. 51, no. 3, pp. 293–294, May 2015.

[10] M. B. Rasheed, N. Javaid, M. Imran, Z. A. Khan, U. Qasim, and A. V. Vasilakos, "Delay and energy consumption analysis of priority guaranteed mac protocol for wireless body area networks," *Wireless Netw.*, vol. 23, no. 4, pp. 1249–1266, 2017.

[11] R. Amer, M. M. Butt, H. ElSawy, M. Bennis, J. Kibiłda, and N. Marchetti, "On minimizing energy consumption for d2d clustered caching networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2018, pp. 1–6.

[12] M. S. A. Patil and M. P. Mishra, "Improved mobicast routing protocol to minimize energy consumption for underwater sensor networks," *Int. J. Res. Sci. Eng.*, vol. 3, no. 2, pp. 197–204, 2017.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 1990, mIT Elect. Eng. Comput. Sci.

[14] A. Klochay, "Implementing Dijkstra's algorithm in Python," https://www.udacity.com/blog/2021/10/implementing-dijkstras-algorithm-in-python.html, October 2021.