Maximizing Submodular Set Function with Connectivity Constraint: Theory and Application to Networks

Tung-Wei Kuo† ‡, Kate Ching-Ju Lin†, and Ming-Jer Tsai‡
†Research Center for Information Technology Innovation, Academia Sinica, Taiwan
‡Department of Computer Science, National Tsing Hua University, Taiwan
twkuo@citi.sinica.edu.tw, katelin@citi.sinica.edu.tw, and mjtsai@cs.nthu.edu.tw

Abstract

In this paper, we investigate the wireless network deployment problem, which seeks the best deployment of a given limited number of wireless routers. We found that many goals for network deployment, such as maximizing the number of covered users or areas, or the total throughput of the network, can be modelled with the submodular set function. Specifically, given a set of routers, the goal is to find a set of locations $S$, each of which is equipped with a router, such that $S$ maximizes a predefined submodular set function. However, this deployment problem is more difficult than the traditional maximum submodular set function problem, e.g., the maximum coverage problem, because it requires all the deployed routers to form a connected network. In addition, deploying a router in different locations might consume different costs. To address these challenges, this paper introduces two approximation algorithms, one for homogeneous deployment cost scenarios and the other for heterogeneous deployment cost scenarios. Our simulations, using synthetic data and real traces of census in Taipei, show that the proposed algorithms achieve a better performance than other heuristics.

I. INTRODUCTION

In recent years, wireless mesh networks have received considerable attention. A wireless mesh network consists of several routers and a gateway that can access the Internet. The routers, which cannot access the Internet, can relay data from their users to the gateway. This type of...
network architecture is a cost-effective solution for expanding the broadband service area, e.g., IEEE 802.16j [1] or rural wireless mesh networks [2]–[7], and flexible to support temporary connectivity for disaster rescue [8]. A major issue in deploying a wireless mesh network is that the resource is limited. For example, as mentioned in [4], some mesh networks are capable only of equipping a single gateway. The connectivity of such a network, i.e., whether each router can find a path to the gateway (possibly in a multi-hop fashion), becomes a fundamental concern for network deployment. Therefore, the gateway and the routers should be placed carefully to meet the connectivity constraint.

In addition to the connectivity requirement, network deployment usually needs to take other optimization goals into consideration. For example, the coverage area and the number of users served by the network can be significantly affected by the network topology. On the other hand, maximizing the network throughput is also an important issue for broadband service protocols, e.g., IEEE 802.16j. Hence, given a limited number of routers (relay nodes) and one gateway (base station), the problem of deploying a network that ensures connectivity and, at the same time, achieves optimization goals is considered in this paper.

Instead of designing a separate algorithm for different types of networks with variant optimization goals, this paper aims at developing a universal algorithm that can be generally applied to different optimization goals. In particular, we tackle the problems with an optimization goal that can be interpreted as a submodular set function [9]. We find that many optimization goals, such as the number of covered users, the network throughput, and the coverage area, can be written as a submodular set function. Fig. 1 shows an example of representing the number of users covered by a set of router locations $S$ as a submodular set function $f(S)$. For example, if only three routers are available for deployment, then the location set $\{C, D, E\}$ is connected. Hence, this is a feasible solution and can cover five users, i.e., $f(\{C, D, E\}) = 5$. Because maximizing such a connected submodular set function is an NP-complete problem, we focus on designing an approximation algorithm to approximate the maximum submodular set problem, while maintaining network connectivity.

In addition, placing a router in different places might incur different deployment costs, which could vary with different types of landforms, buildings, climate, or even health concerns about electromagnetic radiation. We will also design an approximation algorithm for this heterogeneous deployment cost scenario. The following is the contribution of this paper:
Fig. 1: An example of representing the deployment of a rural mesh network as a connected submodular set problem. The router icons in (a) indicate all feasible locations to place routers. Then, the number of covered users \( f(\{A\}) = 2 \), \( f(\{B\}) = 4 \), \( f(\{A, B, F\}) = 5 \), and \( f(\{C, D, E\}) = 5 \). Say three routers are available for deployment; then, deploying them in \( \{C, D, F\} \), as in (b), is the optimal deployment, in which any router can act as the gateway. Note that \( \{B, C, D\} \) is not a feasible solution because they are not connected.

- By proving that several common optimization goals are submodular set functions, we consider the problem of finding the optimal network deployment that maximizes different predefined submodular set functions subject to the connectivity constraint. We design an \( O(\sqrt{k}) \)-approximation algorithm, where \( k \) is the number of available routers. To the best of our knowledge, this is the first approximation algorithm for the maximum connected submodular set problem.
- We also propose an approximation algorithm when different deployment costs are involved.
- We conduct simulations, using synthetic data and real traces, to evaluate the performance of our algorithms.

The remainder of this paper is organized as follows. In Section II, we introduce the submodular set function, describe our proposed approximation algorithm, and demonstrate how to apply it in real applications. In Section III, we study the problem for heterogeneous deployment cost scenarios. Using simulations, we evaluate the performance of the proposed algorithms in Section IV. We summarize related work in Section V. Finally, we conclude the paper in Section VI.
II. CONNECTED SUBMODULAR SET FUNCTION PROBLEM

In this section, we first provide the preliminaries required to describe the maximum connected submodular (MCS) set function problem. We then propose an approximation algorithm to solve the MCS problem, and prove its approximation ratio. Finally, we explore several applications that can be formulated as the MCS problem and solved by our algorithm.

A. Preliminaries

Before defining the maximum connected submodular set function problem, we first introduce the maximum submodular set function problem and one of its variations, called the maximum rooted submodular set function problem.

**Definition 1.** Given a finite set $V$, a real-valued function $f$ on the set of subsets of $V$ is called a submodular set function [9] if $f$ satisfies one of the following two equivalent conditions.

1) $f(A) + f(B) \geq f(A \cup B) + f(A \cap B), \forall A, B \subseteq V.$

2) $f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B), \forall A \subseteq B \subseteq V$ and $v \in V \setminus B.$

Based on the above property, Lovász has shown that submodular functions can be understood as set functions with convexity [10]. Furthermore, $f$ is said to be nondecreasing if $f(A) \leq f(B), \forall A \subseteq B \subseteq V.$

The following lemma is a directed result of the above property and will be used in our later analysis.

**Lemma 1.** If $f$ is a submodular set function on the set of subsets of $V$, then $\sum_{i=1}^{n} f(S_i) \geq f(\bigcup_{i=1}^{n} S_i), \forall S_i \subseteq V, n \geq 1.$

**Proof:** We prove it by induction on $n$. Obviously, the statement holds when $n = 1$ or 2. Assume that the statement holds when $n = k$, i.e., $\sum_{i=1}^{k} f(S_i) \geq f(\bigcup_{i=1}^{k} S_i).$ Adding $f(S_{k+1})$ on
both sides yields the following inequality,

\[
\sum_{i=1}^{k+1} f(S_i) = \sum_{i=1}^{k} f(S_i) + f(S_{k+1}) \geq f(\bigcup_{i=1}^{k} S_i) + f(S_{k+1}) \\
\geq f(\bigcup_{i=1}^{k} S_i \cup S_{k+1}) = f(\bigcup_{i=1}^{k+1} S_i).
\]

Hence, the statement also holds when \( n = k + 1 \). The proof then follows by mathematical induction.

Many combinatorial optimization problems involve submodular set functions, and can be modeled as follows:

**Problem 1.** Given a nondecreasing submodular function \( f \) on the set of subsets of \( V \) and a positive integer \( k \), the **Maximum Submodular set function (MS)** problem is to find a subset \( S \subseteq V \) such that

1) \( |S| \leq k \),

2) \( f(S) \) is maximized.

Let \( MS(f, V, k) \) denote an instance of the MS problem.

Then, when \( f(\emptyset) = 0 \), Nemhauser et. al. proposed a greedy incremental algorithm for the MS problem with an approximation ratio of \( e/(e-1) \) [9]. Note that an algorithm for a maximization problem has an approximation ratio \( \alpha > 1 \) if it always outputs a set \( S \) such that \( \alpha \) times of the approximation solution, \( \alpha f(S) \), is no less than the optimal value.

We further define the following maximum **rooted** submodular set function problem, which is a subroutine used to solve our maximum **connected** submodular set function problem.

**Problem 2.** Given a nondecreasing submodular function \( f \) on the set of subsets of \( V \) with \( f(\emptyset) = 0 \), a positive integer \( k \), and an element \( r \in V \), the **Maximum Rooted Submodular set function (MRS)** problem is to find a subset \( S \subseteq V \) such that

1) \( |S| \leq k \),

2) \( r \in S \),

3) \( f(S) \) is maximized.

Let \( MRS(f, V, k, r) \) denote an instance of the MRS problem.
Algorithm 1: Approximation algorithm for $MRS(f, V, k, r)$

**Input:** A set $V$, a nondecreasing submodular set function $f$ on the set of subsets of $V$, a positive integer $k$, a root $r \in V$, and the submodular set function $f'$ transformed from $f$ according to Eq. (1)

1. $Sol = \text{Apply Nemhauser's algorithm to } MS(f', V \setminus \{r\}, k - 1)$
2. $Sol = Sol \cup \{r\}$
3. return $Sol$

The main difference between the MRS problem and the MS problem is that the MRS problem requires a specific element $r$ be included in the solution. We can apply Algorithm 1 to approximate $MRS(f, V, k, r)$. The idea of Algorithm 1 is to always include $r$ in the solution, and transform the original set function $f$ to the following new set function $f'$, which takes the need to include $r$ into account.

$$f'(S) = \begin{cases} 
  f(S \cup \{r\}) & \text{if } S \subseteq V \setminus \{r\}, S \neq \emptyset \\
  0 & \text{if } S = \emptyset 
\end{cases} \quad (1)$$

The transformation in Eq. (1) forces that $f'(\emptyset) = 0$, and ensures that the value of choosing a non-empty set $S$ with respect to $f'$ is equal to that of choosing set $S \cup \{r\}$ with respect to $f$. We will prove in Lemma 2 that 1) $f'$ is still a nondecreasing submodular function, and 2) $MRS(f, V, k, r)$ is equivalent to $MS(f', V \setminus \{r\}, k - 1)$. As a result, we can simply approximate $MRS(f, V, k, r)$ by using Nemhauser’s algorithm to solve $MS(f', V \setminus \{r\}, k - 1)$.

**Lemma 2.** Algorithm 1 is an $e/(e - 1)$-approximation algorithm for the MRS problem.

**Proof:** When $k = 1$, the proof is trivial. For $k > 1$, the MRS problem, $MRS(f, V, k, r)$, can be formulated as $\max_S f(S \cup \{r\})$, for all $S \subseteq V \setminus \{r\}, |S| \leq k - 1$. Since $S$ can be either a non-empty set or an empty set, the optimum of the MRS problem is equivalent to the maximum between

$$\max_S f(S \cup \{r\}) = \max_S f'(S), \forall S \subseteq V \setminus \{r\}, 1 \leq |S| \leq k - 1 \quad (2)$$

and $f(\emptyset \cup \{r\}) = f(\{r\})$. However, since $f$ is nondecreasing, Eq. (2) is always the maximum between the above two equations. Therefore, $MRS(f, V, k, r)$ can be reformulated as

$$\max_S f'(S), \forall S \subseteq V \setminus \{r\}, |S| \leq k - 1. \quad (3)$$
Removing the constraint $1 \leq |S|$ does not change the optimal value because when $|S| = 0$, we have $f'(S) = f'() = 0$ and $f'$ is a non-negative function. As a result, if we can prove that $f'(S)$ is a nondecreasing submodular set function, the MRS problem in Eq. (3) is exactly equivalent to the MS problem $MS(f', V \setminus \{r\}, k - 1)$, which can thus be approximated by Nemhauser’s algorithm with an approximation ratio of $e/(e - 1)$.

Given that $f$ is nondecreasing submodular and $f(\emptyset) = f'() = 0$, we can prove $f'$ is submodular by considering the following two cases:

Case 1: $A \neq \emptyset$ and $B \neq \emptyset$. We get that
\[
f'(A) + f'(B) = f(A \cup \{r\}) + f(B \cup \{r\}) \\
\geq f((A \cup B) \cup \{r\}) + f((A \cap B) \cup \{r\}) \\
\geq f'(A \cup B) + f'(A \cap B).
\]

The two sides of the last inequality are the same when $A \cap B \neq \emptyset$.

Case 2: At least one of $A$ and $B$ is empty set. Without loss of generality, we assume that $A = \emptyset$. Then,
\[
f'(A) + f'(B) = f'(A \cap B) + f'(A \cup B).
\]

Finally, $f'$ is nondecreasing because $f'(A) \leq f(A \cup \{r\}) \leq f(B \cup \{r\}) = f'(B)$ for all $A \subset B \subseteq V \setminus \{r\}$ and $f'(A) = f'(B)$ when $A = B$. Hence, we obtain that $f'$ is a nondecreasing submodular set function, and the statement holds.

\[\blacksquare\]

B. Maximum Connected Submodular Set Problem

We study a problem that extends the MS problem to a graph scenario. Consider a graph $G = (V, E)$. The basic idea is to select a set $S$ from the vertex set $V$, while ensuring that the selected vertices in $S$ are connected in graph $G$. The problem can be formally defined as follows:

**Problem 3.** Given a graph $G = (V, E)$, a nondecreasing submodular function $f$ on the set of subsets of $V$ with $f(\emptyset) = 0$, and a positive integer $k$, the Maximum Connected Submodular set function (MCS) problem is to find a subset $S \subseteq V$ such that

1) the subgraph $G_S = (S, E_S)$ of $G$ is connected, where $E_S = \{(u, v) | (u, v) \in E, \forall u, v \in S\}$,
2) $|S| \leq k$,
3) $f(S)$ is maximized.
Algorithm 2: Approximation algorithm for $MCS(f, G, k)$

**Input:** A graph $G = (V, E)$, a nondecreasing submodular set function $f$ on the set of subsets of $V$, and a positive integer $k$

1. $Sol = \emptyset$
2. for $r \in V$ do
   3. $V_r = \{v | dist(v, r) \leq \lfloor \sqrt{k} \rfloor\}$
   4. $Sol_r = \text{Apply Algorithm 1 to } MRS(f, V_r, \lfloor \sqrt{k} \rfloor, r)$
   5. if $f(Sol_r) > f(Sol)$ then
      6. $r^* = r$
      7. $Sol = Sol_r$
8. for $v \in Sol_r$ do
   9. Find a shortest path from $v$ to $r^*$ on $G$, and add the nodes along the path to $Sol$
10. return $Sol$

Let $MCS(f, G, k)$ denote an instance of the MCS problem.

**Theorem 1.** The MCS problem is $NP$-complete.

**Proof:** Note that $MS(f, V, k)$ is equivalent to $MCS(f, G=(V, E), k)$, if $G$ is a complete graph. Thus, since the MS problem is $NP$-complete [9], the MCS problem is also $NP$-complete.

We propose an $O(\sqrt{k})$-approximation algorithm, as shown in Algorithm 2, for the MCS problem. The high-level idea of Algorithm 2 is to select at most $k$ nodes in two steps. In the first step (Line 2-7), given any root $r \in V$, we first find a set $Sol_r$ of $\lfloor \sqrt{k} \rfloor$ nodes, where one is $r$. All the nodes in $Sol_r$ need not be connected, but are at most $\lfloor \sqrt{k} \rfloor$ hops away from $r$. We iteratively search for a root $r \in V$ to output a set $Sol = Sol_{r^*}$ that can generate the highest value, i.e., $r^* = \arg\max_{r \in V} f(Sol_r)$. Next, the second step (Line 8-9) adds at most $k - \lfloor \sqrt{k} \rfloor$ nodes to $Sol$ to make it connected. Specifically, we connect each $v \in Sol_{r^*}$ to $r^*$ by adding the nodes along the shortest path between $v$ and $r^*$ to $Sol$. Since the first step guarantees that the distance between $v$ and $r^*$ is at most $\lfloor \sqrt{k} \rfloor$, connecting $(\lfloor \sqrt{k} \rfloor - 1)$ non-root nodes in $Sol_{r^*}$ to $r^*$ requires at most $(\lfloor \sqrt{k} \rfloor - 1) \times (\lfloor \sqrt{k} \rfloor - 1)$ extra nodes, which is less than or equal to $k - \lfloor \sqrt{k} \rfloor$. We then get the following claim.
Claim 1. Algorithm 2 can output a feasible solution, i.e., a set of at most \( k \) connected nodes, of \( \text{MCS}(f, G, k) \).

To prove that Algorithm 2 can achieve an approximation ratio of \( O(\sqrt{k}) \), we divide our proof into two steps. We will first derive the ratio between \( f(Sol) \) and \( \text{OPT}_{\lceil \sqrt{k} \rceil} \), and then derive the ratio between \( \text{OPT}_{\lceil \sqrt{k} \rceil} \) and \( \text{OPT}_k \), where \( \text{OPT}_{\lceil \sqrt{k} \rceil} \) and \( \text{OPT}_k \) are the optimal solution of \( \text{MCS}(f, G, \lceil \sqrt{k} \rceil) \) and \( \text{MCS}(f, G, k) \), respectively. Given the above two proofs, we can naturally derive the ratio between our solution and \( \text{OPT}_k \).

Before starting our proof, we introduce the following claim, which plays an important role in the proof of Lemma 3.

Claim 2. There always exist \( n = O(\sqrt{k}) \) connected subgraphs \( G^i = (V^i, E^i) \) of graph \( G_{OPT_k} \) with vertex set \( \text{OPT}_k \), where \( |V^i| \leq \lceil \sqrt{k} \rceil \) for all \( 1 \leq i \leq n \), such that \( \bigcup_{i=1}^{n} V^i = \text{OPT}_k \).

Proof: Since every connected graph has a spanning tree, we will prove the following stronger statement: For any tree \( T \) with size \( k \), there always exist \( n = O(\frac{k}{m}) \) subtrees \( T^i = (V^i, E^i) \) of \( T = (V, E) \), where \( |V^i| \leq m \) for all \( 1 \leq i \leq n \), such that \( \bigcup_{i=1}^{n} V^i = V \). By substituting any spanning tree of \( G_{OPT_k} \) and \( \lceil \sqrt{k} \rceil \) for \( T \) and \( m \), respectively, the proof holds.

The case where \( m = 1 \) is trivial. For other cases, we will provide a constructive proof by Algorithm 3. Let \( T_v \) denote the subtree of \( T \) rooted at \( v \). The algorithm operates iteratively. In each iteration, the algorithm first finds a specific node \( v \) (Line 3), and includes several trees \( T_{tmp} \) in a set \( TreeSet \) to cover \( T_v \) (Line 5-14). Once \( T_v \) is covered, we remove \( T_v \) and the edge adjacent to \( v \) from \( T \) (Line 15). The idea of choosing \( v \) is to find a node such that each subtree \( T_{c_i} \) rooted at any of \( v \)'s children, \( c_i \), has a size less than \( m \), but \( |T_v| \geq m \) (Line 3). The iterative algorithm terminates when we cannot find any such \( v \), and leaves less than \( m \) uncovered nodes in \( T \). We can therefore use a single tree \( T_{last} = T \) to cover them (Line 17). Let \( v_1, v_2, \cdots, v_q \) be the vertices \( v \) selected by Line 3. Since we remove \( T_v \) from \( T \) once we cover it, we get that \( q \leq \frac{|T_v|}{m} = \frac{k}{m} \).

To ensure \( T \) is covered by no more than \( n = O(\frac{k}{m}) \) subtrees, we need to perform some optimizations. Specifically, we cannot simply cover each \( T_{c_i} \) by one subtree \( T_{tmp} = T_{c_i} \); otherwise, the number of subtrees used to cover \( T \) might exceed \( n = O(\frac{k}{m}) \). Therefore, to reduce the number of subtrees required to cover each \( T_v \), our algorithm tries to divide all \( T_{c_i} \) s into as
Algorithm 3: Tree covering

**Input:** A tree $T = (V, E)$ and a positive integer $m$

```plaintext
1. $TreeSet = \emptyset$
2. while True do
3.   Choose an arbitrary non-leaf node $v$ such that each child $d$ of $v$ has $|T_d| < m$ and $|T_v| \geq m$
4.   if $v$ exists then
5.     Let $c_1, c_2, ..., c_l$ be the children of $v$
6.     $G = \emptyset$
7.     for $i = 1 \rightarrow l$ do
8.       $G = G \cup T_{c_i}$
9.       if $|G| \geq m$ then
10.      Add $T_{tmp1} = T_{c_i}$ to $TreeSet$
11.      Add $T_{tmp2} = SuperTree(G \setminus T_{c_i})$ to $TreeSet$
12.     $G = \emptyset$
13.   if $G \neq \emptyset$ then
14.     Add $T_{tmp} = SuperTree(G)$ into $TreeSet$
15.   else
16.     Remove $T_v$ and the edge adjacent to $v$ from $T$
17.   Add $T_{last} = T$ into $TreeSet$
18. return $TreeSet$
```

few disjoint groups as possible, while ensuring that each group can be covered by at most two subtrees $T_{tmp}$s, each of which is no larger than $m$. To find one such group (Line 7-12), we initialize an empty group $G = \emptyset$, keep merging arbitrary $T_{c_i}$ into the group $G$, and immediately stop merging once the number of vertices in $G$ exceeds or equals $m$. Let $T_{last}^G$ denote the last tree added to group $G$. Note that the order of $T_{c_i}$s to be merged can be arbitrary. We use Fig. 2 to explain why the resulting group $G$ can be covered by only two $T_{tmp}$s. Let $SuperTree(G)$ denote the tree that includes $G$, $v$, and all the edges connecting $v$ to $G$. Recall that, before $T_{last}^G$ is added to the group, the number of vertices in the group is still less than $m$. Hence, we can use the first tree $T_{tmp1} = T_{last}^G$ to cover $T_{last}^G$, and use the second tree $T_{tmp2} = SuperTree(G \setminus T_{last}^G)$ to cover the rest of $T_{c_i}$s in the group, as shown in Fig. 2.
Fig. 2: An example of Claim 2. The order of $T_c$s to be merged is from left to right. Each group is covered by two trees, $T_{tmp1}$ and $T_{tmp2}$, except for the last one, $Group_{last}$, which is covered by one tree $T_{tmp}$. Note that the total number of groups is at most $\lceil \frac{|T_v|-1}{m} \rceil$, because the size of each group is at least $m$ (except for the last one).

We can continuously find several such groups until each subtree $T_c$ is included in one group. The last group $G$ might have only fewer than $m$ vertices. If this is the case, we can simply cover it by a single tree $T_{tmp} = SuperTree(G)$ (Line 14), like the right-most $T_{tmp}$ in Fig. 2.

By our algorithm, we know that $T_v \setminus \{v\}$ will be divided into at most $\lceil \frac{|T_v|-1}{m} \rceil$ groups. Then, $T_v$ can be covered by at most $2\lceil \frac{|T_v|-1}{m} \rceil$ $T_{tmp}$s. Observe that in Line 3, if $v$ does not exist, then $|T| = |T_{last}| < m$. Otherwise, if $|T| \geq m \geq 2$ and $v$ does not exist, then a non-leaf node exists and for all non-leaf nodes $u$ with $|T_u| \geq m$, $u$ must have a child $d$ with $|T_d| \geq m$. Since $m \geq 2$, $d$ is, again, a non-leaf node. However, this contradicts the fact that $T$ has a finite number of vertices and has no loop. Hence, the number of $T_{tmp}$s and $T_{last}$ required to cover $T = (\bigcup_{i=1}^{q} T_v_i) \cup T_{last}$ is at most

$$\sum_{i=1}^{q} 2\lceil \frac{|T_v|-1}{m} \rceil + 2 \sum_{i=1}^{q} \left( \frac{|T_v|}{m} + 1 \right) + 1 \leq \frac{2k}{m} + 2q + 1 \leq \frac{2k}{m} + \frac{2k}{m} + 1 = O(\frac{k}{m}).$$

**Lemma 3.** $O(\sqrt{k})f(OPT_{\sqrt{k}}) \geq f(OPT_k)$.

**Proof:** Based on Claim 2 and Lemma 1, we get that there always exist some connected subsets $V^i$’s of $V$, where $|V^i| \leq \lfloor \sqrt{k} \rfloor$ for all $1 \leq i \leq n = O(\sqrt{k})$, satisfying the following inequality: $f(OPT_k) = f(\bigcup_{i=1}^{n} V^i) \leq \sum_{i=1}^{n} f(V^i) \leq O(\sqrt{k})f(OPT_{\sqrt{k}})$. The last inequality holds because each $V^i$ is a feasible solution to $MCS(f, G, \lfloor \sqrt{k} \rfloor)$ and hence, $f(V^i) \leq f(OPT_{\sqrt{k}})$.
Theorem 2. Algorithm 2 is an $O(\sqrt{k})$-approximation algorithm for the MCS problem.

Proof: Based on Claim 1, the solution $Sol$ outputted by Algorithm 2 is feasible. The proof then proceeds as follows. We will first prove the following claim: $\frac{e}{e-1} f(Sol) \geq f(OPT_{\lceil \sqrt{k} \rceil})$. Then, the proof follows because, by Lemma 3, $\frac{e}{e-1} O(\sqrt{k}) f(Sol) \geq O(\sqrt{k}) f(OPT_{\lceil \sqrt{k} \rceil}) \geq f(OPT_k)$.

To prove the claim, let $r$ be any vertex in $OPT_{\lceil \sqrt{k} \rceil}$; then, all the other vertices in $OPT_{\lceil \sqrt{k} \rceil}$ must also be included in $V_r$ (Line 3), i.e., $OPT_{\lceil \sqrt{k} \rceil} \subseteq V_r$, because the diameter of $OPT_{\lceil \sqrt{k} \rceil}$ is at most $\lfloor \sqrt{k} \rfloor - 1$. Hence, the optimal value of $MRS(f, V_r, \lceil \sqrt{k} \rceil, r)$, denoted by $OPT'$, must be greater than or equal to $f(OPT_{\lceil \sqrt{k} \rceil})$, because $OPT'$ need not be connected. According to Lemma 2, we then have $\frac{e}{e-1} f(Sol_r) \geq OPT' \geq f(OPT_{\lceil \sqrt{k} \rceil})$. Finally, since our algorithm searches for the root $r^*$ that can output the maximum $f(Sol_{r^*})$ and ensures $Sol_{r^*} \subseteq Sol$, we prove that $\frac{e}{e-1} f(Sol) \geq \frac{e}{e-1} f(Sol_{r^*}) \geq \frac{e}{e-1} f(Sol_r) \geq f(OPT_{\lceil \sqrt{k} \rceil})$. The first inequality is due to that $f$ is nondecreasing.

C. Applications

As illustrated in the introduction, given the number of routers and gateway, $k$, we can model each possible location for placing a router or gateway as a vertex, and an edge connecting two vertices if, and only if, the distance between the two vertices is less than or equal to the transmission range, $r$, i.e., they can communicate with each other. The vertices and edges then form a graph $G$, and we need to find a connected subgraph of $G$ with size $k$. For the optimization goals, we first consider the number of users that can be covered by the deployment. We assume that a user is covered by a router (or gateway) if the distance between them is less than or equal to $r$. We will prove with Theorem 3 that the number of covered users is indeed a submodular set function. Below is the formal problem definition. Note that we do not differentiate between the router and the gateway, since interchanging the location of the router and the gateway does not affect the connectivity of the network.

Problem 4. Consider a scenario where there is a user set $U$ and a location set $V$ on the plane and the communication range of each node is $r$. Let $C(v)$ be the set of users in $U$ covered by the router location $v \in V$, i.e., $C(v) = \{ u | dist(u, v) \leq r, \forall u \in U \}$. For a location subset $S$ of $V$, we define $C(S)$ as $\bigcup_{s \in S} C(s)$. Then, given the number of routers $k$, the Maximum Connected
Coverage (MCC) problem searches for a subset $S \subseteq V$, such that

1) $S$ is connected with respect to graph $G_{V,r}$, where $G_{V,r} = (V, \{(u, v)|\text{dist}(u, v) \leq r, \forall u, v \in V\})$.
2) $|S| \leq k$.
3) $|C(S)|$ is maximized.

**Theorem 3.** The MCC problem can be transformed to the MCS problem by setting $f(S) = |C(S)|$. Thus, Algorithm 2 is an $O(\sqrt{k})$-approximation algorithm for the MCC problem.

**Proof:** This transformation is correct because it has been shown in [11] that $f(S) = |C(S)|$ is a nondecreasing submodular set function with $f(\emptyset) = 0$.

In IEEE 802.16j, one of the major concerns is the network throughput. The following problem considers maximizing the sum of data rates of all covered users. Note that a user may be covered by multiple routers, and we let each user associate with the router that can provide it the highest data rate. Again, by Theorem 4, we will prove the network throughput is a submodular set function.

**Problem 5.** Given a user set $U$, a location set $V$, and the communication range of each node, $r$. Let $T(u, v)$ denote a data rate function, indicating the data rate of user $u$ as connecting to a router in location $v \in V$. For example, the data rate can be computed by the path loss models in [12]. We define the sum rate $T(S)$ as $\sum_{u \in C(S)} \max_{s \in S} T(u, s)$ for any subset $S$ of $V$. Then, given the number of routers $k$, the Maximum Throughput Connected Coverage (MTCC) problem searches for a subset $S \subseteq V$, such that

1) $S$ is connected with respect to graph $G_{V,r}$.
2) $|S| \leq k$.
3) $T(S)$ is maximized.

Let $MTCC(V, U, r, k, T)$ denote an instance of the MTCC problem.

**Theorem 4.** Any instance $MTCC(V, U, r, k, T)$ can be transformed to the MCS problem $MCS(f, G_{V,r}, k)$ by setting $f(S) = T(S)$. Thus, Algorithm 2 is an $O(\sqrt{k})$-approximation algorithm for the MTCC problem.

**Proof:** Again, it is sufficient to show that $f(S) = T(S)$ is a nondecreasing submodular
set function with \( f(\emptyset) = 0 \). Obviously, \( T \) is nondecreasing and \( T(\emptyset) = 0 \). To prove that \( T \) is submodular, we show that \( T \) satisfies condition 2 in Definition 1, i.e.,

\[
T(A \cup \{v\}) - T(A) \geq T(B \cup \{v\}) - T(B),
\]

for all \( A \subseteq B \subseteq V \) and \( v \in V \setminus B \).

Consider three subsets of \( C(v) \), \( OA \), \( OB \), and \( New \), where \( OA = C(A) \cap C(v) \), \( OB = (C(B) \setminus C(A)) \cap C(v) \), and \( New = C(v) \setminus C(B) \). Clearly, \( C(v) = OA \cup OB \cup New \), and these three sets are mutually disjoint. We can interpret Eq. (4) as the comparison of throughput gains by adding \( v \) to \( A \) and \( B \). We discuss the gain that any node in \( U \) can get in the following four cases.

Case 1: For any node \( u \in New \), since it is covered by neither \( A \) nor \( B \), it can get the same throughput gain \( T(u, v) \) whether \( v \) is added to \( A \) or \( B \).

Case 2: For any node \( u \in OA \), the gains by adding \( v \) to \( A \) and \( B \) are \( g_A = \max(0, T(u, v) - \max_{s \in A} T(u, s)) \) and \( g_B = \max(0, T(u, v) - \max_{s \in B} T(u, s)) \), respectively. Since \( A \subseteq B \), we obtain \( g_A \geq g_B \) for all nodes in \( OA \).

Case 3: For any node \( u \in OB \), the gains by adding \( v \) to \( A \) and \( B \) are \( g_A = T(u, v) \) and \( g_B = \max(0, T(u, v) - \max_{s \in B} T(u, s)) \), respectively. Obviously, \( g_A \geq g_B \).

Case 4: For any node \( u \notin C(v) \), its gain equals 0 whether \( v \) is added to \( A \) or \( B \).

All the above cases ensure that every node in \( U \) can get a higher or equal gain when \( v \) is added to \( A \), and hence Eq. (4) holds.

We can apply our algorithm to the above problems with other kinds of \( C(v) \) functions. The communication graph of \( V \) can also be an arbitrary graph. Theorem 3 and 4 will still hold. In addition, sometimes it would be desirable if each user has a different priority or weight. Naturally, the optimization goal would become to maximize the total weight of users covered by the routers, which can still be written as a submodular set function. Based on this observation, we get that the coverage area can also be represented as a submodular set function.

III. CONNECTED SUBMODULAR FUNCTION PROBLEM WITH BUDGET CONSTRAINT

A. Problem Definition

**Problem 6.** Given a vertex-weighted graph \( G = (V, E) \) with a weight function \( w : V \rightarrow \mathbb{Z}^+ \), a nondecreasing submodular function \( f \) on the set of subsets of \( V \) with \( f(\emptyset) = 0 \), and a positive
Algorithm 4: Approximation algorithm for $MRSB(f, V, B, r)$

**Input:** A set $V$, a nondecreasing submodular set function $f$ on the set of subsets of $V$, a positive integer $B$, a root $r \in V$, and the submodular set function $f'$ transformed from $f$ according to Eq. (1)

1. $Sol = \text{Apply Sviridenko’s algorithm to } MSB(f', V \setminus \{r\}, B - w(r))$
2. $Sol = Sol \cup \{r\}$
3. return $Sol$

integer $B$, the Maximum Connected Submodular set function with Budget constraint (MCSB) problem asks for a subset $S \subseteq V$ such that

1) $S$ is connected with respect to $G$,
2) The weight of $S$ is less than $B$, i.e., $w(S) = \sum_{s \in S} w(s) \leq B$,
3) $f(S)$ is maximized.

Let $MCSB(f, G, B)$ denote an instance of the MCSB problem.

The only difference between the MCSB problem and the MCS problem is that, in the MCS problem, the number of elements in the solution set, or the size of the set, is at most $k$ (cardinality constraint); in the MCSB problem, we force that the total weight of elements in the solution set, or the weight of the set, is at most $B$ (budget constraint). Using the same idea, we can transform the MS and MRS problems into the MSB and MRSB problems, respectively. That is, we change the first constraint in Problem 1 and 2 to $w(S) \leq B$. Sviridenko [13] proposed an $e/(e - 1)$-approximation algorithm for the MSB problem. With the same technique we use in Algorithm 1, we can obtain an $e/(e - 1)$-approximation algorithm for the MRSB problem, Algorithm 4.

**Lemma 4.** Algorithm 4 is an $e/(e - 1)$-approximation algorithm for the MRSB problem.

**Proof:** When $B = w(r)$, the proof is trivial. For $B > w(r)$, the MRSB problem, $MRSB(f, V, B, r)$, can be formulated as $\max_S f(S \cup \{r\})$, for all $S \subseteq V \setminus \{r\}, w(S) \leq B - w(r)$. Since $S$ can be either a non-empty set or an empty set, the optimum of the MRSB problem is equivalent to the maximum between

$$\max_S f(S \cup \{r\}) = \max_S f'(S), \forall S \subseteq V \setminus \{r\}, 0 < w(S) \leq B - w(r)$$

(5)
and \( f(\emptyset \cup \{r\}) = f(\{r\}) \). However, since \( f \) is nondecreasing, Eq. (5) is always the maximum between the above two equations. Therefore, \( MRSB(f, V, B, r) \) can be reformulated as

\[
\max_S f'(S), \forall S \subseteq V \setminus \{r\}, |S| \leq B - w(r).
\]

Removing the constraint \( 0 < w(S) \) does not change the optimal value because when \( w(S) = 0, f'(S) = f'(\emptyset) = 0 \) and \( f' \) is a non-negative function. Since we have shown that \( f' \) is a nondecreasing submodular set function in the proof of Lemma 2, the proof is then completed.

\[\square\]

**B. Approximation Algorithm**

Before we introduce the high-level idea of this algorithm, we first explain why Algorithm 2 cannot work here. The main difficulty is that Claim 2 cannot be applied to this problem. Specifically, the budget version analog of Claim 2 is the following (the stronger one): For any tree \( T \) with weight \( B \), there always exist \( n = O\left(\frac{B}{m}\right) \) subtrees \( T^i = (V^i, E^i) \) of \( T = (V, E) \), where \( w(V^i) \leq m \) for all \( 1 \leq i \leq n \), such that \( \bigcup_{i=1}^{n} V^i = V \). To show the above statement is wrong, consider a tree \( T \) that contains just one vertex \( v \) with \( w(v) = B \). Obviously, if \( m < B \), then it is impossible to cover \( T \) by trees with a weight no larger than \( m \). Thus, we allow the weight of each root of \( T^i \), \( w(r_i) \), to exceed \( m \); meanwhile, we also make sure that the remaining weight of \( T^i \), \( w(T^i) - w(r_i) \), is still less than \( m \). Hence, the claim becomes the following:

**Claim 3.** For any tree \( T \) with weight \( B \), there always exist \( n = O\left(\frac{B}{m}\right) \) subtrees \( T^i = (V^i, E^i) \) of \( T = (V, E) \), where \( w(V^i) \leq m + w(r_i) \), and \( r_i \) is the root of \( T^i \), for all \( 1 \leq i \leq n \), such that \( \bigcup_{i=1}^{n} V^i = V \).

**Proof:** We prove this claim by Algorithm 5. We will first show that all the trees \( T^i \) in \( TreeSet \) have sizes less than or equal to \( m + w(r_i) \). Then we will show that the number of trees in \( TreeSet \) is \( O\left(\frac{B}{m}\right) \). For those trees added in Line 4, the size is exactly \( w(r_i) \). For \( T_{tmp1} \), since \( w(T_{e_i}) < m \) by Line 7, \( w(T_{tmp1}) = w(T_{e_i}) < m < m + w(c_i) \). For \( T_{tmp2} \), since \( w(G \setminus T_{c_i}) \leq m \), \( w(T_{tmp2}) = w(v) + w(G \setminus T_{c_i}) \leq m + w(v) \). By the same reason, \( w(T_{tmp}) \leq m + w(v) \). We prove that \( w(T_{last}) \leq m + w(r) \), where \( r \) is the root of \( T \), by contradiction. If \( w(T_{last}) > m + w(r) \), then \( r \) is a non-leaf node. And, for all non-leaf nodes \( u \) with \( w(T_u) \geq m \), one child \( d \) of \( u \) must
Algorithm 5: Tree covering with budget

**Input**: A tree $T = (V, E)$ and a positive integer $m$

1. $\text{TreeSet} = \emptyset$

2. while True do

3.     if There exists a leaf node $v$ with $w(v) \geq m$ then

4.         Add $T_v$ to $\text{TreeSet}$

5.         Remove $T_v$ and the edge adjacent to $v$ from $T$

6.         Continue

7.     Choose an arbitrary non-leaf node $v$ such that each child $d$ of $v$ has $w(T_d) < m$ and $w(T_v) \geq m$

8.     if $v$ exists then

9.         Let $c_1, c_2, \ldots, c_l$ be the children of $v$

10.        $G = \emptyset$

11.       for $i = 1 \rightarrow l$ do

12.               $G = G \cup T_{c_i}$

13.               if $w(G) \geq m + 1$ then

14.                   Add $T_{\text{tmp1}} = T_{c_i}$ to $\text{TreeSet}$

15.                   Add $T_{\text{tmp2}} = \text{SuperTree}(G \setminus T_{c_i})$ to $\text{TreeSet}$

16.               $G = \emptyset$

17.       if $G \neq \emptyset$ then

18.           Add $T_{\text{tmp}} = \text{SuperTree}(G)$ into $\text{TreeSet}$

19.           Remove $T_v$ and the edge adjacent to $v$ from $T$

20.     else

21.         Add $T_{\text{last}} = T$ into $\text{TreeSet}$

22. return $\text{TreeSet}$

have $w(T_d) \geq m$. However, since all the leaf nodes weigh less than $m$ by Line 3, $d$ is, again, a non-leaf node, which contradicts the fact that $T$ has finite weight and $T$ contains no loop.

Let $l$ be the number of trees added in Line 4 and $v_1, v_2, \ldots, v_q$ be the vertices $v$ selected by Line 7, then $l$ and $q \leq \frac{B}{m}$ and the number of trees in $\text{TreeSet}$ is at most $\sum_{i=1}^{q} 2\left\lceil \frac{w(T_{v_i}) - w(v_i)}{m+1} \right\rceil + 1 < 2 \sum_{i=1}^{q} \left( \frac{w(T_{v_i})}{m+1} + 1 \right) + 1 + 1 \leq \frac{2B}{m+1} + 2q + l + 1 \leq \frac{2B}{m+1} + \frac{2B}{m} + \frac{B}{m} + 1 = O(\frac{B}{m})$

Now we are ready to introduce the notion behind our approximation algorithm, Algorithm 6. The algorithm can be divided into four steps. First, in Line 2, we construct a directed graph
Algorithm 6: Approximation algorithm for \( \text{MCSB}(f, G, B) \)

**Input:** A vertex-weighted graph \( G = (V, E) \), a nondecreasing submodular set function \( f \) on the set of subsets of \( V \), and a positive integer \( B \)

1. \( \text{Sol} = \emptyset \)
2. Construct a directed edge-weighted graph \( G' = (V', E') \), where \( V' = V \), \( E' = \{< u, v >, < v, u > | (v, u) \in E \} \) with \( w'( < u, v >) = w(u) \), and \( < u, v > \) indicates an edge directed from \( u \) to \( v \)
3. for \( r \in V \) do
   4. if \( w(r) > B \) then
      5. Continue
   6. \( V_r = \{ v | \text{dist}_{G'}(v, r) \leq \lfloor \sqrt{B} \rfloor \} \), where \( \text{dist}_{G'} \) is the shortest distance from \( v \) to \( r \) on \( G' \)
   7. \( \text{Sol}_r = \text{Apply Algorithm 4 to } \text{MRSB}(f, V_r, w(r) + \lfloor \sqrt{B} \rfloor, r) \)
   8. if \( f(\text{Sol}_r) > f(\text{Sol}) \) then
      9. \( r^* = r \)
     10. \( \text{Sol} = \text{Sol}_{r^*} \)
4. for \( v \in \text{Sol}_{r^*} \) do
   5. Find a shortest path from \( v \) to \( r^* \) on \( G' \), and add the nodes along the path to \( \text{Sol} \)
6. Compute a tree \( T \) spanning all the nodes in \( \text{Sol} \), such that the degree \( d \) of \( r^* \) in \( T \) is minimized
7. Let \( c_1, c_2, \cdots, c_d \) be the children of \( r^* \)
8. Assign the set with maximum \( f \)-value among \( \{ T_{c_1}, T_{c_2}, \cdots, T_{c_d}, r^* \} \) to \( \text{Sol} \)
9. return \( \text{Sol} \)

\( G' \) that has the following property: the length of path \( p \) from \( u \) to \( r \) on \( G' \) is equal to the total weight of the nodes on \( p \) besides \( r \). The second and third steps are similar to Algorithm 2. In the second step (Line 3-10), given any root \( r \in V \), we first find a set \( \text{Sol}_r \) with weight at most \( w(r) + \lfloor \sqrt{B} \rfloor \). All the nodes in \( \text{Sol}_r \) need not be connected, but the distances to \( r \) are at most \( \lfloor \sqrt{B} \rfloor \) in \( G' \). We iteratively search for a root \( r \in V \) to output a set \( \text{Sol} = \text{Sol}_{r^*} \) that can generate the highest value, i.e., \( r^* = \arg \max_{r \in V} f(\text{Sol}_r) \). Note that because the weight of each vertex is a positive integer, the number of vertices in \( \text{Sol} \) is at most \( \lfloor \sqrt{B} \rfloor + 1 \). Next, for each \( v \in \text{Sol} \), the third step (Line 11-12) adds the nodes on the shortest path from \( v \) to \( r^* \) on \( G' \) to \( \text{Sol} \) to make it connected. By the property of \( G' \) and the definition of \( V_{r^*} \), the total weight of nodes on the shortest path from \( v \) to \( r^* \) on \( G' \) besides \( r^* \) is at most \( \lfloor \sqrt{B} \rfloor \). Hence, the total
weight of Sol is at most \( w(r^\ast) + [\sqrt{B}]\sqrt{B} \leq w(r^\ast) + B \) and might be larger than \( B \). We use the fourth step (Line 13-15) to make Sol become a feasible solution, i.e., \( w(Sol) \leq B \). In this step, we decompose Sol into as few connected components as possible, but also ensure that the weight of each component is less than or equal to \( B \). The idea is simple: observe that the weight of \( Sol \setminus \{r^\ast\} \) is at most \( B \), so we can decompose Sol into \( d(r^\ast) + 1 \) connected components, where \( d(r^\ast) \) is the degree of \( r^\ast \) in the tree spanning Sol. We can therefore choose the component with maximum \( f \)-value as the final solution. Hence, the smaller \( d(r^\ast) \) we have, the better approximation ratio we get. The proof proceeds as follows: we first prove the ratio between OPT and the Sol_{r^\ast} outputted in Line 10 is \( O(\sqrt{B}) \) by Lemma 5. Then we show that there exists a spanning tree with \( d(r^\ast) < 6 \) for the unit disk graph by Lemma 6. Theorem 5 and Corollary 1 are the final results.

**Lemma 5.** \( O(\sqrt{B})f(Sol_{r^\ast}) \geq f(OPT) \).

**Proof:** Denote \( \text{MRCRB}(f, G, r, B) \) as the problem instance of the rooted version of the MCSB problem, which requires \( r \) be included in the solution. In addition, let \( OPT_{r,B} \) be the optimum of the instance. We first claim that for all \( Sol \) in Line 7, \( \frac{e}{e-1}f(Sol_r) \geq f(OPT_{r,w(r)+[\sqrt{B}]}) \).

Since all the elements in \( OPT_{r,w(r)+[\sqrt{B}]} \) must be in \( V_r \), the optimum of \( MRSB(f, V_r, w(r) + [\sqrt{B}], r) \) is larger than or equal to \( f(OPT_{r,w(r)+[\sqrt{B}]} \)). And because Sol_{r} is an \( \frac{e}{e-1} \)-approximation solution to \( MRSB(f, V_r, w(r) + [\sqrt{B}], r) \), the claim then follows.

Let \( r^\prime = \arg \max_{r \in V} f(OPT_{r,w(r)+[\sqrt{B}]}). \) Based on Line 8-10, we actually have \( \frac{e}{e-1}f(Sol_{r^\prime}) \geq f(OPT_{r^\prime,w(r^\prime)+[\sqrt{B}]} \)). By substituting any spanning tree of \( G_{OPT} \) and \( [\sqrt{B}] \) for \( T \) and \( m \) in Claim 3, respectively, we get the following inequality:

\[
f(OPT) = f(\bigcup_{i=1}^{n} V_i) \leq \sum_{i=1}^{n} f(V_i) \\
\leq O(\sqrt{B}) \cdot f(OPT_{r_1,w(r_1)+[\sqrt{B}]}) \\
\leq O(\sqrt{B}) \cdot f(OPT_{r^\prime,w(r^\prime)+[\sqrt{B}]} \)) \\
\leq O(\sqrt{B}) \cdot e/(e-1) \cdot f(Sol_{r^\ast}) = O(\sqrt{B}) \cdot f(Sol_{r^\ast}).
\]

The second inequality holds since each \( V_i \) has \( w(V_i) \leq w(r_i) + [\sqrt{B}] \), and hence, \( V_i \) is a feasible solution of \( \text{MRCRB}(f, G, r_i, w(r_i) + [\sqrt{B}]) \).
Lemma 6. There exists a tree spanning $Sol$ such that the degree $d(r^*)$ of $r^*$ is less than 6 when $G$ is a unit disk graph.

Proof: It is similar to the one for Lemma 3.1 in [14].

By the above Lemmas, we obtain the following results.

Theorem 5. Algorithm 6 is a $(D_{\text{max}} + 1)O(\sqrt{B})$-approximation algorithm for the general MCSB problem, where $D_{\text{max}}$ is the maximum degree in $G$.

Corollary 1. When the underlying graph is an unit disk graph (UDG), Algorithm 6 is an $O(\sqrt{B})$-approximation algorithm for the MCSB problem.

IV. Numerical Results

In this section, we evaluate the performance of our algorithm under two metrics: the number of covered users and the total throughput of the covered users, namely, the MCC problem and the MTCC problem. For the MCC problem, we say a user is covered by a router if the distance between them is less than or equal to the transmission range of the router. All routers have the same transmission range. For the MTCC problem, we assume that each client is equipped with an 802.11b interface, which supports rates 1, 2, 5.5, and 11 Mb/s. We use the 802.11b PHY Simulink Model [15] as the channel error model, and set the transmission power of each node and noise level to 18 dBm and -95 dBm, respectively. For both objective functions, we also evaluate the performance when different locations have different costs, i.e., the performance of Algorithm 6.

To generate users’ locations, we use two methods. In the first method, we apply Zipf’s law to randomly assign each user on a 1200 m × 1200 m field. In the second method, we assign each user’s position according to the locations of families near Yangmingshan in Taipei city [16]. We will explain the detailed setting and show the corresponding simulation results for these two methods in Sections IV-A and IV-B, respectively.

A. User Location by Zipf’s law

In this simulation, we will generate 200 users on a 1200 m × 1200 m field. To locate these users, we first select 10 out of the 200 users as cluster heads, and uniformly assign each one to
the plane at random. For the remaining users, we first assign each one uniformly randomly to one of the clusters. Then we locate the user according to Zipf’s law and the location of its cluster head. More specifically, we let each cluster be a disk of radius of 50 m × 4. The probability of a user being assigned to a location where the distance to the cluster head is between 50 m × k and 50 m × (k − 1), k = 1, 2, 3, 4, is \( \frac{1/k}{\sum_{i=1}^{4} 1/i} \). After deciding the distance to the cluster head according to the probability distribution, we then assign the user to one of the locations satisfying this distance requirement, with each location equally likely to be chosen.

Regarding the candidate locations of the routers, we assume the candidate locations are arranged in a grid topology with grid size 100 m × 100 m. We set the communication range of a router to 150 m. For the cost of each candidate location in the MCSB problem, we randomly assign a cost between 1 and 10 to each candidate location. The results are obtained by averaging the data of 20 different instances.

Fig. 3(a) shows the results of the MCC problem and compares our algorithm with the algorithm proposed by Vandin et al. [17] and an upper bound of the optimum. The idea behind this bound is that if a candidate location \( r \) is included in the optimal solution, then all the other nodes in the optimal solution must be at most \( k \) hops away from \( r \). For each vertex \( r \in V \), we then apply Algorithm 1, which does not consider the connectivity constraint, on the vertices within \( k \) hops to \( r \). The highest value among all the above outputs, i.e., all possible \( r \), is thus an upper bound. We also compare to a random solution, which adds a random vertex into the solution, and keeps adding one vertex adjacent to the current solution to the solution at a time until the size is \( k \).

Furthermore, in some applications, we may need to include a specific location in our solution. Since our algorithm and Vandin’s algorithm try all the possible starting nodes (i.e., the \( r \) in line 2 in Algorithm 2), we extend both algorithms to the following scenario: Instead of searching over all the vertices and selecting the best root, we randomly choose a specific location \( r \) as the root and examine its performance. Note that after our algorithm is executed, the solution might select fewer than \( k \) router locations. If this is the case, we apply the greedy strategy in [17] to make the resulting solution have size \( k \).

We can see that the performance of our algorithm and Vandin’s algorithm is very close. However, for the rooted problem, Vandin’s algorithm performs slightly better than ours. The reason is that Vandin’s algorithm takes nodes far away from the root into consideration. As to our algorithm, no matter how few users are around the root, our algorithm still chooses
Fig. 3: Two applications of the MCS problem. (a) Impact of the number of routers on the number of covered users. (b) Impact of the number of routers on the total throughput.

at least $\lfloor \sqrt{k} \rfloor$ vertices around it, which may result in a poor performance. However, Vandin’s algorithm and our algorithm along with the specific root selection outperform the arbitrary solution. In addition, the ratio between our algorithm and the upper bound matches the analysis of the approximation ratio. Even though Vandin’s algorithm achieves a similar performance for the MCC problem, the algorithm cannot be generalized to solve other optimization goals, e.g., maximizing the throughput, or the MCSB problem. In contrast, our algorithm is a more general solution to all the connected submodular set function problems.

The simulation results of the MTCC problem are shown in Fig. 3(b). Because Vandin’s algorithm cannot be applied to this problem, we consider a heuristic here. Like our algorithm,
this heuristic loops over all the possible starting nodes and calculates a solution for each one. The heuristic then returns the best solution. For each starting node, we first add it into the solution and then apply a greedy strategy. The greedy strategy iteratively adds one vertex, which has the maximum increased throughput among the neighbors of the solution, to the current solution, until the size of the solution is equal to \( k \). We also use this greedy strategy to add more nodes to the output of our algorithm when the size of our solution can be further increased. From Fig. 3(b), we can see that our algorithms have better performances than the heuristic, whether for the specific root selection or the best root selection. Again, the gap between our algorithm and the arbitrary solution is large, and our algorithm approaches the upper bound faster.

Fig. 4 shows the simulation results when different candidate locations have different costs, where Vandin’s algorithm cannot be applied. Thus, to compare with our algorithm, we design two heuristics. The idea is similar to the one introduced in the previous simulation. The first heuristic adopts a different greedy strategy. Instead of choosing the vertex with the maximum increased throughput, the strategy now chooses the vertex with the smallest cost. The second heuristic is application-dependent. If the application is to maximize the number of covered users, we use the strategy of choosing the vertex with the maximum increased number of covered users; if the application is to maximize the total throughput, we just use the heuristic introduced in the previous simulation. We also use the second greedy strategy to further improve our algorithm when the output does not fully use the budget.

Our algorithm performs better than these two heuristics in both applications. Three things are noteworthy here. First, the second heuristic outperforms the first heuristic. Since the users are located according to Zipf’s law, a small fraction of routers have covered most of the users; the remaining routers cover only a few users or even none. The first heuristic, which considers only the cost of the location, may add many routers, each of which covers a very small number of users. Second, the specific root version of the second heuristic performs worse than the specific root version of the first heuristic in the first application when the budget is over 200. This is because when the budget is high enough, the greedy strategy used by the second heuristic might find no vertex to improve the coverage. Hence, an arbitrary neighbor is added to the solution. On the contrary, in the second application, even if the greedy strategy cannot find a vertex to improve the coverage, the strategy might still find a vertex that is closer to some covered users to improve the throughput. Third, all the algorithms approach the upper bound
more quickly in the case of maximizing the number of covered users. This is because, in the application of maximizing the throughput, the algorithms can still include a candidate location to further improve the throughput even if the neighboring users are already covered by some other candidate locations included earlier. Thus, the throughput is improved even without covering new users. See the first heuristic, which outputs the same solution on both applications, as an example. We know that when the budget is 400, the first heuristic has covered almost all the users, as shown in Fig. 4(a). However, in Fig. 4(b), this heuristic does not hit the upper bound until the budget is around 700.
B. User Location According to the Yangmingshan Census

Fig. 5: (a) An example of the problem instance. This area is approximately 12 km × 8 km. There are 7126 users and 1795 candidate locations in this area. The total cost of all the candidate locations is 60053. (b) The solution output by our algorithm for the maximum number of covered users when the budget is 15000. A user is represented by a blue dot. Circles on the grid are candidate locations for placing routers. A black edge exists between a router and a user if the user is covered by the router. The red edge is the connection between routers.

![Image](image.png)

Fig. 6: Impact of the budget amount on the number of covered users, using the Yangmingshan census.

Consider a scenario in which a government wants to deploy a wireless network to support temporary connectivity for disaster rescue in a certain area. Based on the geographic distribution of families, we can provide a network that can be served as a deployment guide for later use. Take Yangmingshan, a region in Taipei city, for an example. We estimate the location of each user by the census from [16]. To calculate the best positions to place the routers, we let the
candidate locations be arranged in a grid with grid size 150 m × 150 m. For the cost of setting up the routers, higher costs are assigned to the locations with a higher population density. More specifically, we divide the region into 106 areas according to the administrative division. Let $d_{\text{min}}$ denote the minimum population density among the 106 areas. For all candidate locations in an area with population density $d$, we set their costs to $\lceil d/d_{\text{min}} \rceil$. We set the communication range of a router to 250 m. These parameters can be further fine-tuned to reflect the local environment. See Fig. 5(a) for an example.

From Fig. 6, we can see that our algorithm has the best performance when the budget is greater than 3000. When the budget is small, e.g., less than 3000, the area covered by the output of the best root selection is small and has high population density. In such a small area, the user distribution does not follow Zipf’s law anymore. Hence, all the best root selection algorithms have similar performances.

V. RELATED WORK

When connectivity is not taken into account, there has been a substantial volume of research on maximizing submodular set functions. Among them, maximizing submodular set function with the cardinality constraint [9] and maximizing submodular function with the budget constraint [13] [18] are two problems closely related to ours. In [9], an $e/(e - 1)$-approximation algorithm is proposed for the problem with the cardinality constraint. For the problem with the budget constraint, [13] and [18] each gives an $e/(e - 1)$-approximation algorithm. The algorithm and its analysis are based on [19], which studies the problem of maximizing coverage under the budget constraint.

However, studies have considered optimization and connectivity constraints at the same time. For example, in [20] [21] [22], the authors study the problem of finding the smallest possible set of sensors that is connected and covers a given area. In [17], the authors study the problem of maximum coverage with the cardinality and connectivity constraints, i.e., the MCC problem, a special case of our problem. They propose a greedy-based algorithm and show that it can achieve an approximation ratio of $O(r)$, where $r$ is the radius of the optimal solution. In [23], the authors study the same problem as ours. However, the cost of their solution may violate the budget constraint. That is, their algorithm might generate an infeasible solution.
VI. CONCLUSION

In this paper, we study the problem of finding a connected set that maximizes some submodular set functions under the cardinality constraint or the budget constraint. We propose two \( O(\sqrt{k}) \)-approximation algorithms for them, where \( k \) is the available cardinality or budget. We investigate several potential applications in wireless networks and show that these problems are special cases of the maximum submodular set function problem discussed in this paper. We apply our algorithms to solve these applications and conduct simulations, using synthetic data and real traces, to compare our algorithm with some greedy-based heuristics. The simulation results show that our algorithms outperform these heuristics.

REFERENCES


