# A Radius-Sensitive Approximation Algorithm for Connected Submodular Maximization

Philip Cervenjak
School of Computing and Information Systems, The University of Melbourne
Melbourne, Australia
pcervenjak@student.unimelb.edu.au

Junhao Gan
School of Computing and Information Systems, The University of Melbourne
Melbourne, Australia
junhao.gan@unimelb.edu.au

Naonori Kakimura
Department of Mathematics, Keio University
Yokohama, Japan
kakimura@math.keio.ac.jp

Seeun William Umboh
School of Computing and Information Systems, The University of Melbourne & ARC Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA)
Melbourne, Australia
william.umboh@unimelb.edu.au

Anthony Wirth
School of Computer Science, The University of Sydney & School of Computing and Information Systems, The University of Melbourne
Sydney, Australia
anthony.wirth@sydney.edu.au

## ABSTRACT

*Connected Submodular Maximization* (**CSM**) is a graph problem with important applications to wireless network deployment, path planning, epidemic outbreaks, and cancer genome studies. In **CSM**, we are given a graph $G$, a non-negative monotone submodular function $f$ on subsets of the vertex set of $G$, and an integer $k$. The goal is to select a tree in $G$, with $k$ edges, whose vertex set maximizes $f$. We also study the more general Directed and Directed Rooted variants of **CSM** (**DCSM** and **DRCSM** respectively). In both variants, $G$ is directed and the solution must be an out-tree in $G$, with $k$ edges, whose vertex set maximizes $f$; **DRCSM** further specifies a vertex to be the root of the selected out-tree. For **CSM**, several previous works have proposed polynomial time approximation algorithms; the state-of-the-art polynomial time algorithm achieves a $\Omega(\frac{1}{\sqrt{k}})$-approximation. We can also parameterize the approximation factor by the radius of the optimal solution, denoted by $r$; the state-of-the-art polynomial time algorithm achieves a $\Omega(\frac{1}{r})$-approximation.

In this paper, we improve on the state-of-the-art approximation factor for **CSM** with respect to $r$ as well as $k$, noting that $r \le k$. We propose a polynomial time framework that, for (Directed) **CSM**, achieves a $\Omega(\frac{\varepsilon^3}{r^\varepsilon})$-approximation for every constant $\varepsilon \in (0, 1]$. For **DRCSM**, our framework achieves a $\Omega(\frac{\delta\varepsilon^3}{r^\varepsilon})$-approximation that violates the size constraint by at most a factor of $1 + \delta$ for every $\delta \in [\frac{1}{k}, 1]$. A key component of our framework is GreedyRadius, an algorithm for **DRCSM** that outputs a bicriteria approximation, i.e., an approximate solution that violates the size constraint by at most some factor. GreedyRadius takes an algorithm with a bicriteria approximation factor in terms of $k$ and outputs a solution with the same bicriteria approximation factor (up to constants) in terms of $r$. Moreover, to use as a subroutine for **DRCSM**, we propose the algorithm RecApprox-$d$, which achieves a $\frac{1}{d+1}$-approximation that violates the size constraint by at most a factor of $(d+1)^2 k^{\frac{1}{d}}$. RecApprox-$d$ uses a recursive greedy strategy, with $d$ denoting the number of levels of recursion used. This enables the dependence on $\varepsilon$ in the approximation factors of our overall framework.

## KEYWORDS

Combinatorial Optimization; Submodular Maximization; Network Design; Approximation Algorithms; Graph Algorithms

**ACM Reference Format:**
Philip Cervenjak, Junhao Gan, Naonori Kakimura, Seeun William Umboh, and Anthony Wirth. 2026. A Radius-Sensitive Approximation Algorithm for Connected Submodular Maximization. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/IPZQ7320



*Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.* © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/IPZQ7320

## 1 INTRODUCTION

We study the problem of *Connected Submodular Maximization* (**CSM**). In this problem, we are given an undirected graph $G = (V, E)$ with $n$ vertices and $m$ edges, a non-negative monotone submodular set function $f$ whose ground set is $V$, and an integer $k \ge 1$; the goal is to select a tree $S \subseteq G$, with $k$ edges, whose vertex set maximizes $f$.[1] We also study *Directed Connected Submodular Maximization* (**DCSM**), which generalizes **CSM** by letting $G$ be a directed graph; the goal is to select an out-tree (i.e., arborescence) $S \subseteq G$, with $k$ edges, whose vertex set maximizes $f$. We further study *Directed Rooted Connected Submodular Maximization* (**DRCSM**), which generalizes **DCSM** by specifying a vertex $v$ to be the root of the selected out-tree.

[1] Our formulation of **CSM** is equivalent to the formulation where the goal is to select a set $S' \subseteq V$, of $k'$ vertices that are connected in $G$, that maximizes $f$. This is because $S'$ must have a spanning tree with $k' - 1 = k$ edges.

**CSM** has received significant attention owing to its important applications. These include deploying a connected network, of limited size, with maximum coverage or throughput; specifically positioning a connected network of wireless routers or relays [6, 17], unmanned aerial vehicles [18, 20, 28–30], or wireless power chargers [31–33]. Another application of **CSM** is identifying a limited number of geographically connected regions that, if undervaccinated, are most susceptible to an epidemic outbreak [1]. This is motivated by the need for public health interventions, which are most effective within a small number of localized undervaccinated regions. Further, **CSM** is related to maintaining a communication network between robots in multi-robot task planning [25].

An important special case of **CSM** is *Connected Maximum Coverage* (**CMC**). Here, we are given a universe of elements $X$, an undirected graph $G$ where each vertex is a subset of $X$, and an integer $k \geq 1$; the goal is to select a tree $S \subseteq G$, with $k$ edges, whose vertex set maximizes the number of its covered elements. **CMC** has application to deploying a wireless sensor network, of limited size, with maximum coverage [11, 12, 15]. **CMC** also has application to identifying genetic mutations associated with cancers [10, 27]. In this application, we are given a universe of cancer patients and a gene interaction network, wherein each genetic mutation is associated with the subset of patients that have the mutation. As cancer is widely assumed to be caused by a network of mutated genes (called a *pathway*), the goal is to select a limited set of connected gene mutations that 'explain' the most cases of cancer (i.e., cover the most patients). **CMC** is also related to the Watchman Route Problem, where we are given a 'map' of various locations, and the goal is to find a minimum-length path in the map such that every location is visible from somewhere along the path [19, 26]. Further, Directed Rooted Connected Maximum Coverage (**DRCMC**), i.e., the special case of **DRCSM** with a coverage objective, is related to problems where we are given a network that models interactions between agents, and the goal is to select a minimum-size rooted out-tree that reconstructs the propagation of some activity throughout the network, such as an epidemic outbreak [21, 24].

*Approximation Algorithms.* Most of the previous works on **CSM** have focused on *polynomial time approximation algorithms.* **CSM** was first studied by Kuo et al. [17], where they showed a polynomial time $\frac{1-1/e}{5(\sqrt{k+1}+1)}$-approximation algorithm. Xu et al. [30] improved the approximation factor to $\frac{1-1/e}{\lfloor\sqrt{k+1}\rfloor}$, and Li et al. [18] further improved it to $\Omega(\sqrt{\frac{s}{k}})$, where $s$ is a chosen integer parameter (their algorithm runs in polynomial time if $s$ is set constant). Note that these works have only found constant-factor improvements in approximation. There also exists, for any constant $\varepsilon > 0$, a polynomial time $\Omega(\frac{1}{\log^{2+\varepsilon} n})$-approximation algorithm for **CSM** as implied by Theorem 3.1 of Im et al. [13]. This, in turn, is implied by an algorithm for Polymatroid Steiner Tree due to Calinescu and Zelikovsky [2]. However, this algorithm crucially relies on taking a tree embedding of the input graph, which does not work for general directed graphs.

We also mention that a number of works have studied budgeted variants of (Directed Rooted) **CSM** [4, 7, 17] and (Directed Rooted) **CMC** [3, 4, 11, 12, 23], i.e., with edge or vertex costs. We outline these works in the related work section of the full paper, though we mention the state-of-the-art works on budgeted (Directed Rooted) **CSM** here. Ghuge and Nagarajan [7] studied **DRCSM** with edge costs, showing a quasi-polynomial time $\Omega(\frac{\log\log k'}{\log k'})$-approximation algorithm, where $k'$ is the number of vertices in an optimal tree. D'Angelo et al. [4] studied (Directed) **CSM** and **DRCSM** with vertex costs, showing polynomial time algorithms: letting $B$ denote the budget, they respectively gave a $\Omega(\frac{1}{\sqrt{B}})$-approximation algorithm and, for every $\delta \in (0, 1]$, a $\Omega(\frac{\delta^3}{\sqrt{B}})$-approximation algorithm that violates the budget by at most a factor of $1 + \delta$.

Despite the aforementioned approximation algorithms, there appear to be no interesting approximation hardness results for **CSM**. The most we can say is that it is NP-hard to find an approximation better than $1 - \frac{1}{e}$ [5], since the cardinality constrained problem is a special case (by taking a complete graph as input). However, for budgeted variants of **CSM**, there are near-logarithmic approximation hardness results [8, 9, 16]. We outline these hardness results in the related work section of the full paper.

*Radius-Sensitive Approximation Algorithms.* Previous works have also considered polynomial time algorithms for **CSM** whose approximation factors depend on one or more parameters that restrict the input graph, $G$, or the input function, $f$. These algorithms are useful as, for certain parameter values, they can outperform algorithms whose approximation factors depend on $n$ or $k$. In this paper, we are interested in the *radius* of the optimal (out-)tree, denoted by $r$. Other parameters that have been considered in the approximation factor include the doubling dimension of a metric graph [1], the curvature of $f$ [20], and the $h$-hop independence of $f$ [20, 29].

For an instance of **CSM**, we define the radius, $r$, of an optimal tree. First, let $v$ be the *center* of the optimal tree, i.e., $v$ is the vertex that minimizes its maximum shortest distance to any other vertex in the optimal tree. Then $r$ is this maximum distance. For an instance of **DCSM** or **DRCSM**, we define the radius, $r$, of an optimal out-tree to be equivalent to its height, i.e., the number of edges in the longest directed path from its root. Note that $r \leq \lceil\frac{k}{2}\rceil$ holds for an undirected tree and $r \leq k$ for a directed out-tree.

The optimal solution radius $r$ is a natural parameter to consider as a smaller $r$ makes the vertices of an optimal (out-)tree more reachable from its center $v$, thus enabling better approximations. To see this intuitively, suppose $r = 1$; in this case, we can guess $v$, initialize our solution with it, and then run the standard greedy algorithm on the (out-)neighbors of $v$. This achieves a $(1 - \frac{1}{e})$-approximation, the same as for the cardinality constrained variant [22]. Moreover, the optimal solution radius is expected to be small in 'small-world' graphs, which are graphs where the average shortest distance between a pair of vertices is $O(\log n)$.

Although the optimal solution radius $r$ is a natural parameter, only two previous works have studied it in the approximation factor for **CSM** or the special case Connected Maximum Coverage (**CMC**), both achieving $\Omega(\frac{1}{r})$-approximations. Vandin et al. [27] first showed a polynomial time $\frac{e-1}{(2e-1)r}$-approximation algorithm for **CMC**. Hochbaum and Rao [10] improved this approximation by a constant factor to $\max\{(1-\frac{1}{e})(\frac{1}{r}-\frac{1}{k+1}), \frac{1}{k+1}\}$ and extended their result to **CSM**. However, observe that when $r = \Theta(k)$, these algorithms only achieve a $\Omega(\frac{1}{k})$-approximation, the same as the trivial approximation achieved by selecting the most valuable vertex in $G$. Thus, it would be ideal to achieve a unified approximation with the same dependence on $r$ and $k$, asymptotically speaking.

## 1.1 Our Contributions

Our main result is Theorem 1.1 below.

Theorem 1.1. *Let $(G, f, k)$ be an instance of (Directed) **CSM**. Then, for every $\varepsilon \in (0, 1]$, there exists an $\frac{\varepsilon^3}{16(1+2\varepsilon)^3 r^\varepsilon}$-approximation algorithm for (Directed) **CSM** that runs in time $O(kn^{\lceil \frac{1}{\varepsilon} \rceil + 2} r^{2\lceil \frac{1}{\varepsilon} \rceil + 1})$.*

Theorem 1.1 shows that, for every constant $\varepsilon < \frac{1}{2}$, we improve on the previous polynomial time $\Omega(\frac{1}{\sqrt{k}})$-approximations [17, 18, 30] (recalling that $r \le k$). Moreover, for every constant $\varepsilon < 1$, we improve on the previous polynomial time $\Omega(\frac{1}{r})$-approximations [10, 27].

We also obtain Theorem 1.2 for the more general problem of Directed Rooted Connected Submodular Maximization (**DRCSM**). This theorem gives a bicriteria $(\alpha, \beta)$-approximation algorithm, which is an algorithm that outputs an $\alpha$-approximate solution that may violate the size constraint by at most a factor of $\beta \ge 1$.

Theorem 1.2. *Let $(G, f, k, v)$ be an instance of **DRCSM**. Then, for every $\varepsilon \in (0, 1]$ and every $\delta \in [\frac{1}{k}, 1]$, there exists a bicriteria $(\frac{\delta\varepsilon^3}{16(1+2\varepsilon)^3 r^\varepsilon}, 1 + \delta)$-approximation algorithm for **DRCSM** that runs in time $O(kn^{\lceil \frac{1}{\varepsilon} \rceil + 2} r^{2\lceil \frac{1}{\varepsilon} \rceil + 1})$.*

*Our Algorithmic Framework.* Our framework for achieving Theorems 1.1 and 1.2 involves combining two novel algorithms for **DRCSM**, namely GreedyRadius and RecApprox-$d$, whose guarantees we outline below.

GreedyRadius takes a bicriteria $(\alpha(k), \beta(k))$-approximation algorithm for **DRCSM** that runs in time $\Gamma(n, k)$ and outputs a bicriteria $(\frac{1}{2}\alpha(r), 4\beta(r))$-approximation in time $O(\frac{kn}{r}\Gamma(n, r))$; we formally state these guarantees in Theorem 3.1. By selecting a valuable size-$k$ out-subtree of this solution, we obtain a feasible $\frac{\alpha(r)}{16\beta(r)}$-approximation for (Directed) **CSM**; we formally state this result in Corollary 3.6. Alternatively, by applying a simple trimming process, we obtain a bicriteria $(\frac{\delta\alpha(r)}{16\beta(r)}, 1 + \delta)$-approximation for **DRCSM**; we formally state this result in Corollary 3.8.

We further propose RecApprox-$d$ as an algorithm that can be used as a subroutine in our framework. For every integer $d \ge 1$, RecApprox-$d$ is a bicriteria $(\frac{1}{d+1}, (d+1)^2 k^{\frac{1}{d}})$-approximation algorithm for **DRCSM** that runs in time $O(n^{d+1}k^{2d+2})$. We formally state the performance guarantees of RecApprox-$d$ in Theorem 4.4.

By running GreedyRadius with subroutine RecApprox-$d$, we obtain a bicriteria $(\frac{1}{2(d+1)}, 4(d+1)^2 r^{\frac{1}{d}})$-approximation algorithm for **DRCSM** that runs in time $O(kn^{d+2}r^{2d+1})$. Then, by selecting a size-$k$ out-subtree (Corollary 3.6), we obtain a feasible $\frac{1}{16(d+1)^3 r^{1/d}}$-approximation for (Directed) **CSM**. Otherwise, by the trimming process (Corollary 3.8), we obtain a bicriteria $(\frac{\delta}{16(d+1)^3 r^{1/d}}, 1 + \delta)$-approximation for **DRCSM**. We assign $d = \lceil \frac{1}{\varepsilon} \rceil$ in the above approximation factors and use the bounds $\frac{1}{\varepsilon} \le \lceil \frac{1}{\varepsilon} \rceil \le \frac{1}{\varepsilon} + 1$ to derive Theorems 1.1 and 1.2 respectively.

In general, we can plug any algorithm for **DRCSM** into our framework to convert its (bicriteria) approximation factor's dependence on $k$ to $r$. For example, we can plug in the quasi-polynomial time $\Omega(\frac{\log\log k}{\log k})$-approximation algorithm by Ghuge and Nagarajan [7] to achieve a bicriteria $(\Omega(\frac{\log\log r}{\log r}), 4)$-approximation.

## 1.2 Technical Overview

Here we outline the main ideas used in GreedyRadius and RecApprox-$d$, while comparing with previous approaches. Both of our algorithms aim to construct an out-tree for **DRCSM** by greedily combining valuable out-subtrees, rather than valuable vertices, along with connecting paths from the root vertex.

*GreedyRadius.* We first explain why the previous-best algorithm [10] only achieves a $\Omega(\frac{1}{r})$-approximation for **CSM**. This algorithm first initializes a solution, $S$, with the root $v$, and then uses the following greedy approach. While $S$ can be feasibly updated: (1) find a vertex, $\hat{w}$, of distance at most $r$ from $v$, whose shortest path from $v$ has maximum marginal gain to $S$, and (2) add this shortest $v$–$\hat{w}$ path to $S$. The issue here is that, for a given $\hat{w}$, there may be many shortest $v$–$\hat{w}$ paths (which cannot all be considered in polynomial time). This means it is possible for the algorithm to just add $v$–$\hat{w}$ paths in which the only valuable vertex is $\hat{w}$. Thus, it may incur up to $r$ edges to add each valuable vertex to the solution.

To improve on the above approach, GreedyRadius constructs a solution $S$ by greedily adding valuable out-subtrees. GreedyRadius constructs each added out-subtree by guessing the root, $\hat{w}$, of a valuable size-$r$ out-subtree, $\hat{T}$, of the optimal out-tree; then it calls the given $(\alpha(k), \beta(k))$-approximation subroutine to solve a sub-instance of **DRCSM** with size constraint $r$ and the guess of root $\hat{w}$. The subroutine must output an $(\alpha(r), \beta(r))$-approximation of $\hat{T}$. By combining sufficiently many of these out-subtrees, along with connecting paths of length at most $r - 1$, the solution $S$ is a $(\frac{1}{2}\alpha(r), 4\beta(r))$-approximate out-tree.

*RecApprox-$d$.* RecApprox-$d$ is a procedure that initializes our main recursive algorithm, RecApprox, so that the maximum depth of its recursion tree (counted by 'edges') is $d$; we call $d$ the *recursion depth* of RecApprox-$d$. Note that we use $b$ to denote the size constraint of a sub-instance of **DRCSM** to distinguish it from the size constraint, $k$, of a main instance of **DRCSM**.

RecApprox uses a recursive greedy strategy that generalizes the strategy by Kuo et al. [17] for implicitly achieving a bicriteria $(1 - \frac{1}{e}, b)$-approximation for **DRCSM** (which they used to achieve an overall $\Omega(\frac{1}{\sqrt{k}})$-approximation for **CSM**). We explain the basic approach of Kuo et al. [17] first: given an instance of **DRCSM** with size constraint $b$ and root $v$, initialize a solution, $S$, with the root $v$. Then greedily add $b$ vertices, of distance at most $b$ from $v$, to $S$, along with connecting paths from $v$.

To implement a recursive greedy strategy, RecApprox takes in its input a value $q > 1$; this is to reduce the size constraint to at most $\frac{b}{q}$ in the recursive calls. Then RecApprox constructs a solution $S$ by greedily adding valuable out-subtrees. RecApprox constructs each added out-subtree by guessing the size, $\hat{c} \le \frac{b}{q}$, and root, $\hat{w}$, of a valuable out-subtree, $\hat{T}$, of the optimal out-tree; then it makes a recursive call to solve a sub-instance of **DRCSM** with the guesses of size $\hat{c}$ and root $\hat{w}$. RecApprox continues adding valuable out-subtrees until the sum of their corresponding size-guesses is $b$. RecApprox also adds connecting paths of length at most $b - 1$.

Lastly, to achieve the bicriteria $(\frac{1}{d+1}, (d+1)^2 k^{\frac{1}{d}})$-approximation for **DRCSM**, RecApprox-$d$ calls RecApprox with size constraint $k$, root $v$, and value $q = k^{\frac{1}{d}}$, and returns the solution output by RecApprox. We can see that the assignment of $q = k^{\frac{1}{d}}$ ensures a recursion depth of $d$ since it makes the size constraints in a chain of recursive calls at most $k^{\frac{d}{d}}, k^{\frac{d-1}{d}}, \dots, k^{\frac{1}{d}}, k^{\frac{0}{d}}$.

We point out that RecApprox uses a similar recursion to an existing algorithm by Calinescu and Zelikovsky [2] for the problem of Polymatroid Directed Steiner Tree. Using additional ideas, it is possible to adapt their algorithm to achieve a bicriteria approximation for **DRCSM** similar to that of RecApprox, but the resulting running time is worse than that of RecApprox by a polynomial factor. We compare our algorithm with theirs and sketch how to adapt it for **DRCSM** in the related work section of the full paper.

### 1.3 Paper Structure

We present preliminaries in Section 2, GreedyRadius in Section 3, RecApprox and RecApprox-$d$ in Section 4, and conclusions in Section 5. We present related work in the full paper.

## 2 PRELIMINARIES

In this paper, $G$ denotes an undirected or directed graph, $V(G)$ denotes the vertex set of $G$, and $E(G)$ denotes the edge set of $G$. Define functions $n(G) \coloneqq |V(G)|$, which is the *cardinality* of $G$, and $m(G) \coloneqq |E(G)|$, which is the *size* of $G$. If $G$ refers to the input graph of our problem, we simply use $V$, $E$, $n$, and $m$ to denote the vertex set, edge set, cardinality, and size of $G$ respectively.

Given an undirected or directed graph $G$, the *distance* from vertex $v$ to vertex $w$, $\text{dist}_G(v, w)$, is the number of edges in a shortest (directed) path from $v$ to $w$. If $G$ refers to the graph in the problem input, we may simply use $\text{dist}(v, w)$ to denote the distance from $v$ to $w$.

*Undirected Graph Terms.* Given an undirected graph $T$, the *eccentricity* of a vertex $v \in V(T)$, $\epsilon_T(v)$, is the maximum distance from $v$ to $w$ over all $w \in V(T)$; that is, $\epsilon_T(v) \coloneqq \max_{w \in V(T)} \text{dist}_T(v, w)$.

The *radius*, $r$, of $T$ is the minimum eccentricity over all $v \in V(T)$; that is, $r = \min_{v \in V(T)} \epsilon_T(v)$. A *center* of $T$ is a vertex $v \in V(T)$ whose eccentricity is equal to the radius (there can be more than one center).

*Directed Graph Terms.* A directed graph $T$ is an *out-tree* (i.e., *arborescence*) if there exists a vertex $v \in V(T)$ such that for every $w \in V(T)$, there is exactly one directed path in $T$ from $v$ to $w$. The vertex $v$ is called the *root* of $T$.

The *radius* (i.e., *height*), $r$, of an out-tree $T$ is the maximum distance from its root $v$ to $w$ over all $w \in V(T)$; that is, $r = \max_{w \in V(T)} \text{dist}_T(v, w)$.

*Submodular Functions.* Let $f$ be a set function whose ground set is $V$. Given subsets $X, Y \subseteq V$, let $f(X \mid Y) \coloneqq f(X \cup Y) - f(Y)$, which is called the *marginal gain* of $X$ to $Y$. As an abuse of notation, we may apply $f$ to a subgraph $S \subseteq G$, which means applying it to the vertex set of $S$. Also, we may apply $f$ to a single vertex $v \in V$, which means applying it to the set $\{v\}$.

We consider functions $f$ that are non-negative monotone submodular, as defined below.

*Definition 2.1 (Submodular function).* A set function $f : 2^V \to \mathbb{R}$ is *submodular* iff for all $X, Y \subseteq V$ where $X \subseteq Y$, and for all $v \in V \setminus Y$, $f(v \mid X) \geq f(v \mid Y)$.

A set function $f$ is *monotone* iff for all $X, Y \subseteq V$ where $X \subseteq Y$, $f(X) \leq f(Y)$; and $f$ is *non-negative* iff for all $X \subseteq V$, $f(X) \geq 0$.

We make the standard assumption that $f$ (or the marginal gain function) is queried via a *value oracle*; we specifically assume a *strong* value oracle, which allows both feasible and infeasible sets to be queried. For convenience, we analyze the running time of each algorithm by only counting the number of value oracle queries it makes.

*Problem Definitions.* The problem of *Connected Submodular Maximization* (**CSM**) is defined as follows. We are given an undirected graph $G = (V, E)$ with $n$ vertices and $m$ edges, a non-negative monotone submodular function $f : 2^V \to \mathbb{R}_{\geq 0}$, and an integer $k \geq 1$. The goal is to select a tree $S \subseteq G$, with $m(S) \leq k$, that maximizes $f$.

The problem of *Directed Connected Submodular Maximization* (**DCSM**) generalizes **CSM** by letting $G$ be a directed graph and additionally requiring that the selected $S \subseteq G$ is an out-tree. The problem of *Directed Rooted Connected Submodular Maximization* (**DRCSM**) in turn generalizes **DCSM** by specifying a vertex $v$ to be the root of the selected out-tree $S \subseteq G$.

Note that we use $k$ to constrain the *size*, i.e., number of edges, of $S$. This is convenient for us since our analyses heavily rely on partitioning trees into edge-disjoint (out-)subtrees and constructing solutions out of edge-disjoint (out-)subtrees.

We use $T^*$ to denote an optimal (out-)tree to any one of our problems, and $v$ and $r$ to denote its root and radius (height) respectively.

*Bicriteria Approximation.* Let $\alpha \in (0, 1]$ and $\beta \geq 1$ be values, which we call the *approximation factor* and the *violation factor* respectively. Then, for any given instance of one of our problems, an $(\alpha, \beta)$-*approximation* is a solution $S$ with value $f(S) \geq \alpha f(T^*)$ and size $m(S) \leq \beta k$. Further, for any one of our problems, an $(\alpha, \beta)$-*approximation algorithm* returns an $(\alpha, \beta)$-approximation for every instance of the problem.

*Tree Partitioning.* Throughout our analyses, we use Lemma 2.2 below, which is a simplified version of Lemma 2 of Khani and Salavatipour [14].

Lemma 2.2 (Tree partitioning). *Let $s > 0$ be a real value, and $T$ be an out-tree satisfying $m(T) \geq s$. Then $T$ can be partitioned into $\Delta$ edge-disjoint out-subtrees $T_1, \dots, T_\Delta$ such that for each $j \in [1, \Delta]$: $1 \leq m(T_j) \leq \lfloor 2s \rfloor$, and $1 \leq \Delta \leq \lfloor \frac{m(T)}{s} \rfloor$.*

## 3 GREEDYRADIUS: RADIUS-SENSITIVE APPROXIMATION ALGORITHM

We present the algorithm GreedyRadius for **DRCSM**, with pseudocode in Algorithm 1 and performance guarantees in Theorem 3.1. We explain how to use GreedyRadius to achieve a feasible approximation for (Directed) **CSM** in Section 3.3 and a bicriteria approximation with $(1 + \delta)$-violation factor for **DRCSM** in Section 3.4.

### 3.1 Overview of GreedyRadius

GreedyRadius takes as input a directed graph $G$, a non-negative monotone submodular function $f$, a size constraint $k \geq 1$, a root

**Algorithm 1:** GreedyRadius

**Input:** $G = (V, E)$: directed graph, $f : 2^V \to \mathbb{R}_{\geq 0}$: value oracle, $k \geq 1$: size constraint, $v \in V$: root, $r \in [1, k]$: radius of $T^*$, Alg: $(\alpha, \beta)$-approximation subroutine.
**Output:** $S \subseteq G$: an out-tree, with root $v$, that $(\frac{1}{2}\alpha(r), 4\beta(r))$-approximates $T^*$.

1 $S_{\leq 0} \leftarrow (\{v\}, \varnothing)$ // Initialize the output solution
2 $W \leftarrow$ set of vertices $w \in V$ where $\mathrm{dist}(v, w) \leq r - 1$
3 $t \leftarrow \lfloor \frac{2k}{r} \rfloor$
4 **for** $i = 1, \ldots, t$ **do** // Add approximate out-subtrees over $t$ iterations
5     **for** $w \in W$ **do**
6         $S_i^w \leftarrow \text{Alg}(G, f(\cdot \mid S_{\leq i-1}), r, w)$ // $m(S_i^w) \leq \beta(r)r$
7     $w_i \leftarrow \arg\max_{w \in W} f(S_i^w \mid S_{\leq i-1})$ // Select an out-subtree greedily
8     $P_i \leftarrow$ shortest path from $v$ to $w_i$ // $m(P_i) \leq r \leq \beta(r)r$
9     $S_{\leq i} \leftarrow S_{\leq i-1} \cup S_i^{w_i} \cup P_i$ // $m(S_i^{w_i} \cup P_i) \leq 2\beta(r)r$
10 $S \leftarrow S_{\leq i}$
11 **return** $S$

vertex $v$, the optimal solution's radius $r$, and an $(\alpha(k), \beta(k))$-approximation subroutine Alg; we assume that Alg takes an instance $(G, f, k, v)$ of **DRCSM** as input. GreedyRadius outputs an out-tree $S \subseteq G$, with root $v$, that $(\frac{1}{2}\alpha(r), 4\beta(r))$-approximates $T^*$.

To give a high-level overview of how GreedyRadius works, we first assume $T^*$ is partitioned into $\Delta \leq \lfloor \frac{2k}{r} \rfloor$ edge-disjoint out-subtrees, $T_1^*, \ldots, T_\Delta^*$, each of size at most $r$ (Corollary 3.2). The roots of these out-subtrees are all of distance $r-1$ from $v$. GreedyRadius initializes the output solution, $S$, with the root, $v$, and updates it over $t = \lfloor \frac{2k}{r} \rfloor$ iterations. Let $S_{\leq 0} = (\{v\}, \varnothing)$ and, for each $i \in [1, t]$, let $S_{\leq i}$ be the $i$th partial solution, $\hat{T}_i$ be the out-subtree from $T_1^*, \ldots, T_\Delta^*$ with maximum marginal gain to $S_{\leq i-1}$, and $\hat{w}_i$ be the root of $\hat{T}_i$. Then, in the $i$th iteration, GreedyRadius performs these steps: (1) guess $w_i = \hat{w}_i$, (2) call Alg to construct an out-subtree $S_i^{w_i}$, with root $w_i$, that $(\alpha(r), \beta(r))$-approximates $\hat{T}_i$, and (3) add $S_i^{w_i}$ and a $v$–$w_i$ connecting path to $S_{\leq i-1}$ to get $S_{\leq i}$.

## 3.2 Analysis of GreedyRadius

Our main result is Theorem 3.1 below. The approximation factor follows from Lemma 3.3 and the violation factor follows from Lemma 3.4. The running time easily follows from GreedyRadius making at most $\lfloor \frac{2k}{r} \rfloor n$ calls to both Alg and $f$ (in the statement of Theorem 3.1, $\Gamma(n, r)$ is the running time of Alg with an input graph $G$ of $n$ vertices, and size constraint $r$).

Theorem 3.1. *Let $(G, f, k, v)$ be an instance of **DRCSM** and Alg be an $(\alpha(k), \beta(k))$-approximation algorithm for **DRCSM** that runs in time $\Gamma(n, k)$. Then GreedyRadius, with subroutine Alg, is a $(\frac{1}{2}\alpha(r), 4\beta(r))$-approximation algorithm for **DRCSM** that runs in time $O(\frac{kn}{r}\Gamma(n, r))$.*

*Analysis Notation for GreedyRadius.* We use the following notation based on the pseudocode of GreedyRadius (Algorithm 1). Recall that $r$ denotes the radius of the optimal out-tree $T^*$. Let $t = \lfloor \frac{2k}{r} \rfloor$ denote the total number of updates GreedyRadius makes in Line 9 to construct the output solution. Let $S_{\leq 0} = (\{v\}, \varnothing)$ denote the initial state of the output solution as in Line 1, and $S_{\leq i}$ denote the $i$th partial solution as in Line 9 of the $i$th iteration of the Line 4 loop. Further, let $S_i^{w_i}$ denote the $i$th approximate out-subtree appended to the output solution, and $P_i$ denote the path added to connect $v$ to $w_i$, the root of $S_i^{w_i}$.

*Partitioning the Optimal Solution for GreedyRadius.* First, an optimal out-tree $T^*$ can be partitioned into edge-disjoint out-subtrees each of size at most $r$ as in Corollary 3.2 below. This corollary directly follows from Lemma 2.2 by setting $s = \frac{r}{2}$ and $T = T^*$, where $m(T^*) = k$.

Corollary 3.2. *Let $r \geq 1$ be an integer. Then $T^*$ can be partitioned into $\Delta$ edge-disjoint out-subtrees $T_1^*, \ldots, T_\Delta^*$ such that for each $j \in [1, \Delta]$: $1 \leq m(T_j^*) \leq r$, and $1 \leq \Delta \leq \lfloor \frac{2k}{r} \rfloor$.*

*Approximation Factor of GreedyRadius.*

Lemma 3.3. *Let $(G, f, k, v)$ be an instance of **DRCSM**. Then GreedyRadius outputs an out-tree $S$ satisfying $f(S) \geq \frac{1}{2}\alpha(r)f(T^*)$.*

Proof. As shown by Corollary 3.2, $T^*$ can be partitioned into $\Delta$ edge-disjoint out-subtrees $T_1^*, \ldots, T_\Delta^*$ each of size at most $r$, where $1 \leq \Delta \leq \lfloor \frac{2k}{r} \rfloor$. Assume without loss of generality that $\Delta = t = \lfloor \frac{2k}{r} \rfloor$, recalling that $t$ denotes the total number of updates in Line 9 to construct $S$.

Now consider the $i$th iteration of the Line 4 update loop. Let $\hat{T}_i$ be the subtree amongst $T_1^*, \ldots, T_t^*$ that maximizes $f(\hat{T}_i \mid S_{\leq i-1})$. By a standard analysis from the submodularity of $f$ and the maximality of $\hat{T}_i$, Ineq. (1) below holds.

$$f(\hat{T}_i \mid S_{\leq i-1}) \geq \frac{1}{t}(f(T^*) - f(S_{\leq i-1})). \tag{1}$$

Let $\hat{w}_i$ be the root of $\hat{T}_i$. It holds that $\hat{w}_i$ is within distance $r-1$ of $v$, so GreedyRadius must guess $w = \hat{w}_i$ in the inner Line 5 loop. Also, $\hat{T}_i$ has size at most $r$ by definition. Thus, $\hat{T}_i$ is a feasible solution to the instance $(G, f(\cdot \mid S_{\leq i-1}), r, w)$ of **DRCSM**. This means that the corresponding call to Alg in Line 6 outputs an out-tree $S_i^{\hat{w}_i}$ satisfying

$$f(S_i^{\hat{w}_i} \mid S_{\leq i-1}) \geq \alpha(r)f(\hat{T}_i \mid S_{\leq i-1}). \tag{2}$$

In the $i$th Line 9 update, RecApprox actually adds the out-subtree with maximum marginal gain along with its connecting path, namely $S_i^{w_i} \cup P_i$. Hence, we prove Ineq. (3) for all iterations $i \in [1, t]$ below; the 1st inequality holds by the monotonicity of $f$, the 2nd by the maximality of $S_i^{w_i}$, the 3rd by Ineq. (2), and the 4th by Ineq. (1).

$$\begin{aligned} f(S_i^{w_i} \cup P_i \mid S_{\leq i-1}) &\geq f(S_i^{w_i} \mid S_{\leq i-1}) \geq f(S_i^{\hat{w}_i} \mid S_{\leq i-1}) \\ &\geq \alpha(r)f(\hat{T}_i \mid S_{\leq i-1}) \\ &\geq \frac{\alpha(r)}{t}(f(T^*) - f(S_{\leq i-1})), \\ f(T^*) - f(S_{\leq i}) &\leq \left(1 - \frac{\alpha(r)}{t}\right)(f(T^*) - f(S_{\leq i-1})). \end{aligned} \tag{3}$$

Finally, GreedyRadius makes $t$ updates to construct $S$, so we can chain Ineq. (3) $t$ times. Thus, we prove the lemma below; the

2nd inequality holds by the non-negativity of $f$.

$$f(T^*) - f(S) \le \left(1 - \frac{\alpha(r)}{t}\right)^t (f(T^*) - f(S_{\le 0}))$$
$$\le \left(1 - \frac{\alpha(r)}{t}\right)^t f(T^*)$$
$$\le e^{-\alpha(r)} f(T^*) \le \left(1 - \frac{\alpha(r)}{2}\right) f(T^*),$$
$$f(S) \ge \frac{\alpha(r)}{2} f(T^*).$$
□

*Violation Factor of* GreedyRadius.

Lemma 3.4. *Let $(G, f, k, v)$ be an instance of **DRCSM**. Then* GreedyRadius *outputs an out-tree $S$ satisfying $m(S) \le 4\beta(r)k$.*

Proof. For each $i \in [1, t]$, the out-subtree $S_i^{w_i}$ is constructed by the subroutine Alg with size constraint $r$ (Line 6 of GreedyRadius). Thus, it holds that $m(S_i^{w_i}) \le \beta(r)r$. Further, the connecting path $P_i$ has size $m(P_i) \le r \le \beta(r)r$. Therefore, $m(S_i^{w_i} \cup P_i) \le 2\beta(r)r$. From this, we bound the size of $S$ below, recalling that $t = \lfloor \frac{2k}{r} \rfloor$.

$$m(S) \le \sum_{i=1}^{t} m(S_i^{w_i} \cup P_i) \le 2t\beta(r)r \le \frac{4k}{r}\beta(r)r = 4\beta(r)k.$$
□

## 3.3 Feasible Approximation for (Directed) CSM

Here we explain how GreedyRadius can be used to achieve a feasible $\frac{\alpha(r)}{16\beta(r)}$-approximation for the problem of (Directed) **CSM**.

First, an out-tree $S$ can be partitioned into out-subtrees as in Corollary 3.5 below. This corollary directly follows from Lemma 2.2 by setting $s = \frac{k}{2}$ and $T = S$, where $m(S) \le 4\beta(r)k$.

Corollary 3.5. *Let $r \ge 1$ be an integer. Then $S$ can be partitioned into $\Delta$ edge-disjoint out-subtrees $S_1, \dots, S_\Delta$ such that for each $j \in [1, \Delta]$: $1 \le m(S_j) \le k$, and $1 \le \Delta \le \lfloor 8\beta(r) \rfloor$.*

Now we prove the required approximation result in Corollary 3.6, which follows from Theorem 3.1 and Corollary 3.5.

Corollary 3.6. *Let $(G, f, k)$ be an instance of (Directed) **CSM** and* Alg *be an $(\alpha(k), \beta(k))$-approximation algorithm for **DRCSM** that runs in time $\Gamma(n, k)$. Then there exists an $\frac{\alpha(r)}{16\beta(r)}$-approximation algorithm for (Directed) **CSM** that runs in time $O(\frac{kn}{r}\Gamma(n, r))$.*

Proof. Given an instance $(G, f, k)$ of **CSM** (or **DCSM**) and its optimal out-tree $T^*$, guess $v$ as the center (or root) vertex of $T^*$.

First run GreedyRadius with input $(G, f, k, v, r, \text{Alg})$. By Theorem 3.1, this gives an out-tree $S$ with value $f(S) \ge \frac{1}{2}\alpha(r) f(T^*)$ and size $m(S) \le 4\beta(r)k$ in time $O(\frac{kn}{r}\Gamma(n, r))$.

Now partition $S$ into $\Delta \le \lfloor 8\beta(r) \rfloor$ edge-disjoint out-subtrees $S_1, \dots, S_\Delta$ as in Corollary 3.5. Let $\hat{S}$ be the out-subtree with maximum value $f(\hat{S})$, which is the final out-tree as required. We have that $\hat{S}$ is a feasible out-tree as $m(\hat{S}) \le k$. Further, by the submodularity and non-negativity of $f$, and the bound of $\Delta \le \lfloor 8\beta(r) \rfloor$, $\hat{S}$ has the required approximation factor as shown below.

$$f(\hat{S}) \ge \frac{1}{\Delta} f(S) \ge \frac{1}{8\beta(r)} f(S) \ge \frac{\alpha(r)}{16\beta(r)} f(T^*).$$
□

## 3.4 Bicriteria Approximation for DRCSM with $(1 + \delta)$-Violation Factor

Here we explain how GreedyRadius can be used to achieve, for every $\delta \in [\frac{1}{k}, 1]$, a $(\frac{\delta\alpha(r)}{16\beta(r)}, 1 + \delta)$-approximation for **DRCSM**.

First, an out-tree $S$ can be partitioned into out-subtrees as in Corollary 3.7 below. This corollary directly follows from Lemma 2.2 by setting $s = \frac{\delta k}{2}$ and $T = S$, where $m(S) \le 4\beta(r)k$.

Corollary 3.7. *Let $r \ge 1$ be an integer and $\delta \in [\frac{1}{k}, 1]$ be a value. Then $S$ can be partitioned into $\Delta$ edge-disjoint out-subtrees $S_1, \dots, S_\Delta$ such that for each $j \in [1, \Delta]$: $1 \le m(S_j) \le \delta k$, and $1 \le \Delta \le \lfloor \frac{8\beta(r)}{\delta} \rfloor$.*

Now we prove the required approximation result in Corollary 3.8, which follows from Theorem 3.1 and Corollary 3.7.

Corollary 3.8. *Let $(G, f, k, v)$ be an instance of **DRCSM** and* Alg *be an $(\alpha(k), \beta(k))$-approximation algorithm for **DRCSM** that runs in time $\Gamma(n, k)$. Then, for every $\delta \in [\frac{1}{k}, 1]$, there exists a $(\frac{\delta\alpha(r)}{16\beta(r)}, 1 + \delta)$-approximation algorithm for **DRCSM** that runs in time $O(\frac{kn}{r}\Gamma(n, r))$.*

Proof. Given an instance $(G, f, k, v)$ of **DRCSM**, assume that we prune the input graph $G$ so that it only contains those vertices $w$ with $\text{dist}(v, w) \le k$ (which preserves the optimal solution).

First run GreedyRadius with input $(G, f, k, v, r, \text{Alg})$. By Theorem 3.1, this gives an out-tree $S$ with value $f(S) \ge \frac{1}{2}\alpha(r) f(T^*)$ and size $m(S) \le 4\beta(r)k$ in time $O(\frac{kn}{r}\Gamma(n, r))$.

Now partition $S$ into $\Delta$ edge-disjoint out-subtrees $S_1, \dots, S_\Delta$ as in Corollary 3.7. Let $S_{j^*}$ be the out-subtree with maximum value, and $w_{j^*}$ be its root. Let $\hat{S}$ be the new out-subtree formed by combining a shortest $v$–$w_{j^*}$ path with $S_{j^*}$, giving the final out-tree as required. By the monotonicity, submodularity, and non-negativity of $f$, and the bound of $\Delta \le \lfloor \frac{8\beta(r)}{\delta} \rfloor$, $\hat{S}$ has the required approximation factor as shown below.

$$f(\hat{S}) \ge f(S_{j^*}) \ge \frac{1}{\Delta} f(S) \ge \frac{\delta}{8\beta(r)} f(S) \ge \frac{\delta\alpha(r)}{16\beta(r)} f(T^*).$$

Further, $\hat{S}$ has the required violation factor since the shortest $v$–$w_{j^*}$ path has at most $k$ edges (by the pruning of $G$), and $m(S_{j^*}) \le \delta k$, giving

$$m(\hat{S}) = \text{dist}(v, w_{j^*}) + m(S_{j^*}) \le k + \delta k = (1 + \delta)k.$$
□

# 4 RECAPPROX-$d$: RECURSIVE GREEDY APPROXIMATION ALGORITHM

We present the algorithm RecApprox for **DRCSM**, with pseudocode in Algorithm 2 and performance guarantees in Theorem 4.1. We also define RecApprox-$d$ to take an instance $(G, f, k, v)$ of **DRCSM** and output the solution from RecApprox$(G, f, k, v, k^{\frac{1}{d}})$. We give the performance guarantees of RecApprox-$d$ in Theorem 4.4.

## 4.1 Overview of RecApprox

RecApprox takes as input a directed graph $G$, a non-negative monotone submodular function $f$, a size constraint $b \ge 1$, a root vertex $v$, and a size divisor $q > 1$. We define the *recursion level* of a call to RecApprox to be the integer $\ell$ satisfying $q^{\ell-1} < b \le q^\ell$. RecApprox outputs an out-tree $S \subseteq G$, with root $v$, that $(\frac{1}{\ell+1}, 1 + 3\ell q + \ell \log_{3/2}(\frac{b}{q}))$-approximates $T^*$.

**Algorithm 2:** RecApprox

**Input:** $G = (V, E)$: directed graph, $f : 2^V \to \mathbb{R}_{\geq 0}$: value oracle, $b \geq 1$: size constraint, $v \in V$: root, $q > 1$: size divisor.

**Output:** $S \subseteq G$: an out-tree, with root $v$, that $(\frac{1}{\ell+1}, 1 + 3\ell q + \ell \log_{3/2}(\frac{b}{q}))$-approximates $T^*$, where $\ell$ is the integer satisfying $q^{\ell-1} < b \leq q^{\ell}$.

1 **if** $b = 1$ **then** // Base case: greedily add an out-neighbor of the root
2 $\quad W \leftarrow$ set of out-neighbors of $v$
3 $\quad \hat{w} \leftarrow \arg\max_{w \in W} f(\{v, w\})$
4 $\quad S \leftarrow (\{v, \hat{w}\}, \{(v, \hat{w})\})$
5 **else** // Recursive case: add approximate out-subtrees until all $b$ edges of $T^*$ have been approximated
6 $\quad S_{\leq 0} \leftarrow (\{v\}, \varnothing)$ // Initialize the output out-tree
7 $\quad W \leftarrow$ set of vertices $w \in V$ where $\mathrm{dist}(v, w) \leq b - 1$
8 $\quad b_0 \leftarrow b$
9 $\quad i \leftarrow 0$
10 $\quad$ **while** $b_i > 0$ **do**
11 $\quad\quad i \leftarrow i + 1$
12 $\quad\quad c_{\min(i)} \leftarrow \lceil \frac{1}{3} \max\{\min\{\frac{b}{q}, b_{i-1}\}, 1\} \rceil$
13 $\quad\quad c_{\max(i)} \leftarrow \lfloor \max\{\min\{\frac{b}{q}, b_{i-1}\}, 1\} \rfloor$
14 $\quad\quad$ **for** $w \in W$ and $c = c_{\min(i)}, \ldots, c_{\max(i)}$ **do**
15 $\quad\quad\quad S_i^{w,c} \leftarrow$ RecApprox$(G, f(\cdot \mid S_{\leq i-1}), c, w, q)$
16 $\quad\quad w_i, c_i \leftarrow \arg\max_{w \in W, c \in [c_{\min(i)}, c_{\max(i)}]} \frac{1}{c} f(S_i^{w,c} \mid S_{\leq i-1})$ // Select an out-subtree greedily by marginal density
17 $\quad\quad P_i \leftarrow$ shortest path from $v$ to $w_i$ // $m(P_i) \leq b$
18 $\quad\quad S_{\leq i} \leftarrow S_{\leq i-1} \cup S_i^{w_i,c_i} \cup P_i$
19 $\quad\quad b_i \leftarrow b_{i-1} - c_i$
20 $\quad S \leftarrow S_{\leq i}$
21 **return** $S$

In the base case where $b = 1$, RecApprox simply returns an out-tree $S$ consisting of the root, $v$, and an out-neighbor, $\hat{w}$, with maximum value.

To give a high-level overview of how RecApprox works in the recursive case where $b \geq 2$, we first assume $T^*$ can be partitioned into $\Delta(i)$ out-subtrees whose sizes depend on the $i$th iteration of the Line 10 loop. That is, $T^*$ can be partitioned into edge-disjoint out-subtrees, $T^*_{i,1}, \ldots, T^*_{i,\Delta(i)}$, each of size in the range $[c_{\min(i)}, c_{\max(i)}]$, where the values of $c_{\min(i)}$ and $c_{\max(i)}$ depend on the $i$th iteration. Note that the roots of the out-subtrees $T^*_{i,1}, \ldots, T^*_{i,\Delta(i)}$ are all of distance $b - 1$ from $v$.

RecApprox initializes the output solution, $S$, with the root, $v$, and updates it while the remaining number of edges to approximate, $b_i$, is non-zero. Let $S_{\leq 0} = (\{v\}, \varnothing)$ and, for each iteration $i$, let $S_{\leq i}$ be the $i$th partial solution, $\hat{T}_i$ be the out-subtree from $T^*_{i,1}, \ldots, T^*_{i,\Delta(i)}$ with maximum marginal density to $S_{\leq i-1}$, $\hat{w}_i$ be the root of $\hat{T}_i$, and $\hat{c}_i$ be the size of $\hat{T}_i$. Then, in the $i$th iteration, RecApprox performs these steps: (1) guess $w_i = \hat{w}$ and $c_i = \hat{c}_i$, (2) make a recursive call to construct an out-subtree $S_i^{w_i,c_i}$, with root $w_i$, that is a bicriteria approximation of $\hat{T}_i$, and (3) add $S_i^{w_i,c_i}$ and a $v$–$w_i$ connecting path to $S_{\leq i-1}$ to get $S_{\leq i}$.

## 4.2 Analysis of RecApprox

We state the performance guarantees of RecApprox in Theorem 4.1 below. The violation factor follows from Lemma 4.3. We prove the approximation factor and the running time in the full paper.

Theorem 4.1. *Let $(G, f, b, v)$ be an instance of **DRCSM**. For a given call to RecApprox, let $q > 1$ be the size divisor and $\ell$ be the recursion level, i.e., the integer satisfying $q^{\ell-1} < b \leq q^{\ell}$. Then RecApprox is a $(\frac{1}{\ell+1}, 1 + 3\ell q + \ell \log_{3/2}(\frac{b}{q}))$-approximation algorithm for **DRCSM** that runs in time $O(n^{\ell+1} b^{2\ell+2})$.*

*Analysis Notation for RecApprox.* Let $t$ denote the total number of updates RecApprox makes in Line 18 to construct the output solution. Where appropriate, we will recall notation from the pseudocode of RecApprox (Algorithm 2).

*Violation Factor of RecApprox.* Before bounding the size of the output out-tree in Lemma 4.3, we bound the number of updates $t$ to the output out-tree.

Claim 4.2. $t \leq 3q + \log_{3/2}(\frac{b}{q})$.

Proof. Let $t_1$ be the number of iterations of the Line 10 loop where $b \geq b_{i-1} > \frac{b}{q}$ holds, and $t_2$ be the number of remaining iterations where $\frac{b}{q} \geq b_{i-1} > 0$ holds.

In each of the first $t_1$ iterations, RecApprox decrements $b_{i-1}$ by at least $c_{\min(i)} \geq \frac{b}{3q}$ to give $b_i$. Thus, it must hold that $b - (t_1 - 1)\frac{b}{3q} > \frac{b}{q}$. Solving for $t_1$ gives $t_1 < 3q - 2$.

In each of the remaining $t_2$ iterations, RecApprox decrements $b_{i-1}$ by at least $c_{\min(i)} \geq \frac{b_{i-1}}{3}$ to give $b_i$, with the last iteration possibly having $b_{i-1} = 1$. Thus, it must hold that $\left(\frac{2}{3}\right)^{(t_2-2)} \frac{b}{q} \geq 1$. Solving for $t_2$ gives $t_2 \leq \log_{3/2}(\frac{b}{q}) + 2$.

We now have $t = t_1 + t_2 \leq 3q + \log_{3/2}(\frac{b}{q})$, proving the claim. □

Lemma 4.3. *Let $(G, f, b, v)$ be an instance of **DRCSM**. Let $q > 1$ be the size divisor given to RecApprox, and $\ell$ be the integer satisfying $q^{\ell-1} < b \leq q^{\ell}$. Then RecApprox outputs an out-tree $S$ satisfying*

$$m(S) \leq \left(1 + 3\ell q + \ell \log_{\frac{3}{2}}\left(\frac{b}{q}\right)\right) b.$$

Proof. Let $M(b)$ be the function that gives the worst-case number of edges in an out-tree $S$ output by RecApprox, given a size constraint of $b$.

By the construction of $S$, the recurrence relation below holds for $M(b)$. We explain this recurrence below. Recall that $c_i$ denotes the size constraint used to recursively construct the $i$th out-subtree added to $S$.

- The $b = 1$ case of Eq. (4) holds since, for $b = 1$, RecApprox finds a solution of size 1 (Line 4).
- The $b \geq 2$ case of Eq. (4) holds since, for $b \geq 2$, RecApprox finds a solution by adding $t$ subtrees and connecting paths (Line 18). The $i$th added subtree has size at most $M(c_i)$ (Line 15). Moreover, the path $P_i$ connecting $v$ to the $i$th out-subtree has length at most $b$ (Line 7).
- Constraint (5) holds by Claim 4.2 and the fact that $t \leq b$.

- Constraint (6) holds by the assignments of $c_{\min(i)}$ and $c_{\max(i)}$ in Lines 12 and 13, and since, for each $i$, $c_{\min(i)} \le c_i \le c_{\max(i)}$.
- Constraint (7) holds since the Line 10 loop only terminates once $c_1 + \cdots + c_t = b$.

$$M(b) = \begin{cases} 1, & b = 1 \\ \sum_{i=1}^{t} (M(c_i) + b), & b \ge 2 \end{cases}, \tag{4}$$

where

$$1 \le t \le \min\left\{3q + \log_{\frac{3}{2}}\left(\frac{b}{q}\right), b\right\}, \tag{5}$$

$$\forall i \in [1,t]\colon 1 \le c_i \le \max\left\{1, \frac{b}{q}\right\}, \tag{6}$$

$$\sum_{i=1}^{t} c_i = b. \tag{7}$$

Now we prove the lemma by upper bounding $M(b)$ and, therefore, $m(S)$. We prove Ineq. (8) below by induction on the recursion level, which is the integer $\ell$ satisfying $q^{\ell-1} < b \le q^{\ell}$. Note that Ineq. (8) clearly holds when $\ell = 0$ since $b \le q^{\ell} = 1$ and this means that $M(b) = 1$ according to Eq. (4).

$$M(b) \le (1 + \ell t)b. \tag{8}$$

*Base Case ($\ell = 1$).* In this case, $1 < b \le q$, so $\frac{b}{q} \le 1$. Then, by Constraint (6), it holds that for all iterations $i \in [1,t]\colon c_i = 1$. Therefore, by the $b = 1$ case of Eq. (4), it holds that $M(c_i) = 1$. Thus, we simplify the $b \ge 2$ case of Eq. (4) below; the inequality holds by the bound $t \le b$ as implied by Constraint (5).

$$M(b) = \sum_{i=1}^{t} M(c_i) + tb = t + tb \le b + tb = (1 + t)b.$$

*Inductive Case ($\ell \ge 2$).* Assume for induction that Ineq. (8) holds for every recursion level below $\ell$. By Constraint (6) and $b \le q^{\ell}$, we have that for all iterations $i \in [1,t]\colon 1 \le c_i \le \frac{b}{q} \le q^{\ell-1}$. Thus, we can bound each $M(c_i)$ by induction. Using this, we simplify the $b \ge 2$ case of Eq. (4); the 3rd equality holds by Constraint (7) since it requires that $\sum_{i=1}^{t} c_i = b$.

$$\begin{aligned} M(b) &= \sum_{i=1}^{t} M(c_i) + tb \le \sum_{i=1}^{t} (1 + (\ell - 1)t)c_i + tb \\ &= (1 + (\ell - 1)t) \sum_{i=1}^{t} c_i + tb = (1 + (\ell - 1)t)b + tb \\ &= (1 + \ell t)b. \end{aligned}$$

We have proved Ineq. (8) in both the base case and inductive case. Finally, substituting $t \le 3q + \log_{3/2}\left(\frac{b}{q}\right)$ from Constraint (5) into Ineq. (8) proves the lemma. □

### 4.3 Analysis of RecApprox-$d$

Recall that RecApprox-$d$ simply takes an instance $(G, f, k, v)$ of **DRCSM** and outputs the solution from RecApprox$(G, f, k, v, k^{\frac{1}{d}})$. We state the performance guarantees of RecApprox-$d$ in Theorem 4.4. We prove this theorem in the full paper, though we summarize its proof below.

Theorem 4.4. *Let $(G, f, k, v)$ be an instance of* ***DRCSM*** *and $d \ge 1$ be an integer. Then RecApprox-$d$ is a $(\frac{1}{d+1}, (d+1)^2 k^{\frac{1}{d}})$-approximation algorithm for* ***DRCSM*** *that runs in time $O(n^{d+1} k^{2d+2})$.*

Essentially, we prove Theorem 4.4 from Theorem 4.1 by setting $b = k$ and $q = k^{\frac{1}{d}}$ since RecApprox-$d$ passes these values to RecApprox. Recalling from Theorem 4.1 that $\ell$ is the integer satisfying $q^{\ell-1} < b \le q^{\ell}$, we have that $k^{\frac{\ell-1}{d}} < k \le k^{\frac{\ell}{d}}$ and, thus, we also set $\ell = d$. These settings give the required approximation factor and running time. Further, they give a violation factor of $1 + 3dk^{\frac{1}{d}} + d\log_{3/2}(k^{\frac{d-1}{d}})$, which we upper bound by using the inequality $\ln(k) \le dk^{\frac{1}{d}}/e$ and simplifying, giving the required violation factor of $(d+1)^2 k^{\frac{1}{d}}$.

## 5 CONCLUSIONS

We presented a novel polynomial time framework that, for (Directed) **CSM**, achieves a $\Omega(\frac{\varepsilon^3}{r^{\varepsilon}})$-approximation for every constant $\varepsilon \in (0,1]$; and, for **DRCSM**, achieves a bicriteria $(\Omega(\frac{\delta\varepsilon^3}{r^{\varepsilon}}), 1 + \delta)$-approximation for every constant $\varepsilon \in (0,1]$ and every $\delta \in [\frac{1}{k}, 1]$. This outperforms the state-of-the-art with respect to the size constraint, $k$, and the optimal solution radius, $r$. As part of our framework, we proposed the algorithms GreedyRadius and RecApprox-$d$ for **DRCSM**. GreedyRadius takes a bicriteria $(\alpha(k), \beta(k))$-approximation subroutine and uses it to construct a $(\frac{1}{2}\alpha(r), 4\beta(r))$-approximate solution. RecApprox-$d$ can be used as this subroutine, giving a bicriteria $(\frac{1}{d+1}, (d+1)^2 k^{\frac{1}{d}})$-approximation in time $O(n^{d+1} k^{2d+2})$.

A potential future direction is to extend our framework to generalizations of **DRCSM** with edge or vertex costs, such as *Submodular Tree Orienteering* (edge costs) or *Directed Rooted Submodular Tree* (vertex costs). Further, for rooted network design problems such as **DRCSM** and the aforementioned problems, it is an open problem to show a polynomial time $\Omega(\frac{1}{k^{\varepsilon}})$ or $\Omega(\frac{1}{r^{\varepsilon}})$-approximation algorithm that does not violate the size or budget constraint. Lastly, it would be interesting to prove a stronger approximation hardness result for **CSM** than the one already implied by the cardinality constrained problem, as this could inform us on whether our algorithms achieve tight approximation factors with respect to $k$. If so, this would further motivate algorithms with beyond-worst-case guarantees.

## ACKNOWLEDGMENTS

Philip Cervenjak was supported by the Elizabeth and Vernon Puzey Scholarship and the Australian Government through the Research Training Program Scholarship. Junhao Gan is in part supported by the ARC Discovery Project DP230102908. Naonori Kakimura was supported by JSPS KAKENHI Grant Numbers JP23K21646, JP22H05001, and JST ERATO Grant Number JPMJER2301, Japan. Seeun William Umboh was supported by the Australian Government through the Australian Research Council (ARC) DP240101353.

The authors thank the anonymous reviewers for their feedback.

## REFERENCES

[1] Jose Cadena, Achla Marathe, and Anil Vullikanti. 2020. Finding Spatial Clusters Susceptible to Epidemic Outbreaks due to Undervaccination. In *Proceedings of*

*the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 1786–1788.

[2] G. Calinescu and A. Zelikovsky. 2005. The Polymatroid Steiner Problems. *Journal of Combinatorial Optimization* 9, 3 (2005), 281–294.

[3] Gianlorenzo D'Angelo and Esmaeil Delfaraz. 2025. Approximation Algorithms for Connected Maximum Coverage. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*. 538–546.

[4] Gianlorenzo D'Angelo, Esmaeil Delfaraz, and Hugo Gilbert. 2022. Budgeted Out-Tree Maximization with Submodular Prizes. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 9–1.

[5] Uriel Feige. 1998. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)* 45, 4 (1998), 634–652.

[6] Xiaofeng Gao, Junwei Lu, Haotian Wang, Fan Wu, and Guihai Chen. 2018. Algorithm design and analysis for wireless relay network deployment problem. *IEEE Transactions on Mobile Computing* 18, 10 (2018), 2257–2269.

[7] Rohan Ghuge and Viswanath Nagarajan. 2022. Quasi-polynomial algorithms for submodular tree orienteering and directed network design problems. *Mathematics of Operations Research* 47, 2 (2022), 1612–1630.

[8] Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. 2019. O (log2 k/log log k)-approximation algorithm for directed steiner tree: a tight quasi-polynomial-time algorithm. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 253–264.

[9] Eran Halperin and Robert Krauthgamer. 2003. Polylogarithmic inapproximability. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. 585–594.

[10] Dorit S. Hochbaum and Xu Rao. 2020. Approximation algorithms for connected maximum coverage problem for the discovery of mutated driver pathways in cancer. *Inform. Process. Lett.* 158 (2020), 105940.

[11] Chien-Chung Huang, Mathieu Mari, Claire Mathieu, Joseph S. B. Mitchell, and Nabil H Mustafa. 2019. Maximizing covered area in the euclidean plane with connectivity constraint. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 32–1.

[12] Lingxiao Huang, Jian Li, and Qicai Shi. 2015. Approximation algorithms for the connected sensor cover problem. In *International computing and combinatorics conference*. Springer, 183–196.

[13] Sungjin Im, Viswanath Nagarajan, and Ruben Van Der Zwaan. 2016. Minimum latency submodular cover. *ACM Transactions on Algorithms (TALG)* 13, 1 (2016), 1–28.

[14] M. Reza Khani and Mohammad R. Salavatipour. 2014. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. *Algorithmica* 69, 2 (2014), 443–460.

[15] Samir Khuller, Manish Purohit, and Kanthi K. Sarpatwar. 2014. Analyzing the optimal neighborhood: Algorithms for budgeted and partial connected dominating set problems. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 1702–1713.

[16] Guy Kortsarz and Zeev Nutov. 2011. Approximating some network design problems with node costs. *Theoretical Computer Science* 412, 35 (2011), 4482–4492.

[17] Tung-Wei Kuo, Kate Ching-Ju Lin, and Ming-Jer Tsai. 2014. Maximizing submodular set function with connectivity constraint: Theory and application to networks. *IEEE/ACM Transactions on Networking* 23, 2 (2014), 533–546.

[18] Shuyue Li, Chaocan Xiang, Wenzheng Xu, Jian Peng, Zichuan Xu, Jing Li, Weifa Liang, and Xiaohua Jia. 2023. Coverage Maximization of Heterogeneous UAV Networks. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 120–130.

[19] Yaakov Livne, Dor Atzmon, Shawn Skyler, Eli Boyarski, Amir Shapiro, and Ariel Felner. 2023. Optimally Solving the Multiple Watchman Route Problem with Heuristic Search. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. 905–913.

[20] Yang Lv, Chenchen Wu, Dachuan Xu, and Ruiqi Yang. 2024. H-hop independently submodular maximization problem with curvature. *High-Confidence Computing* (2024), 100208.

[21] Ritwick Mishra, Jack Heavey, Gursharn Kaur, Abhijin Adiga, and Anil Vullikanti. 2023. Reconstructing an epidemic outbreak using steiner connectivity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 11613–11620.

[22] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14, 1 (1978), 265–294.

[23] Yingli Ran, Zhao Zhang, Ker-I Ko, and Jun Liang. 2016. An approximation algorithm for maximum weight budgeted connected set cover. *Journal of Combinatorial Optimization* 31, 4 (2016), 1505–1517.

[24] Polina Rozenshtein, Aristides Gionis, B. Aditya Prakash, and Jilles Vreeken. 2016. Reconstructing an epidemic over time. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1835–1844.

[25] Guangyao Shi, Ishat E. Rabban, Lifeng Zhou, and Pratap Tokekar. 2021. Communication-aware multi-robot coordination with submodular maximization. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 8955–8961.

[26] Shawn Skyler, Dor Atzmon, Tamir Yaffe, and Ariel Felner. 2022. Solving the Watchman Route Problem with Heuristic Search. *Journal of Artificial Intelligence Research* 75 (2022), 747–791.

[27] Fabio Vandin, Eli Upfal, and Benjamin J. Raphael. 2011. Algorithms for detecting significantly mutated pathways in cancer. *Journal of Computational Biology* 18, 3 (2011), 507–522.

[28] Ziming Wang, Jing Li, He Xue, Wenzheng Xu, Weifa Liang, Zichuan Xu, Jian Peng, Pan Zhou, Xiaohua Jia, and Sajal K. Das. 2024. Approximation algorithm and applications for connected submodular function maximization problems. *IEEE Transactions on Networking* 33, 1 (2024), 241–254.

[29] Wenzheng Xu, Dezhong Peng, Weifa Liang, Xiaohua Jia, Zichuan Xu, Pan Zhou, Weigang Wu, and Xiang Chen. 2022. Maximizing h-hop independently submodular functions under connectivity constraint. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1099–1108.

[30] Wenzheng Xu, Yueying Sun, Rui Zou, Weifa Liang, Qiufen Xia, Feng Shan, Tian Wang, Xiaohua Jia, and Zheng Li. 2021. Throughput maximization of UAV networks. *IEEE/ACM Transactions on Networking* 30, 2 (2021), 881–895.

[31] Wei You, Meixuan Ren, Yuzhuo Ma, Die Wu, Jilin Yang, Xuxun Liu, and Tang Liu. 2023. Practical charger placement scheme for wireless rechargeable sensor networks with obstacles. *ACM Transactions on Sensor Networks* 20, 1 (2023), 1–23.

[32] Nan Yu, Haipeng Dai, Guihai Chen, Alex X. Liu, Bingchuan Tian, and Tian He. 2019. Connectivity-constrained placement of wireless chargers. *IEEE Transactions on Mobile Computing* 20, 3 (2019), 909–927.

[33] Pengzhan Zhou, Cong Wang, and Yuanyuan Yang. 2021. Design of self-sustainable wireless sensor networks with energy harvesting and wireless charging. *ACM Transactions on Sensor Networks (TOSN)* 17, 4 (2021), 1–38.