A Local Search Algorithm for the Min-Sum Submodular Cover Problem

Lisa Hellerstein
Department of Computer Science and Engineering, New York University, Brooklyn, NY, USA

Thomas Lidbetter
Department of Engineering Systems and Environment, University of Virginia, Charlottesville, VA, USA
Department of Management Science and Information Systems, Rutgers Business School, Newark, NJ, USA

R. Teal Witter
Department of Computer Science and Engineering, New York University, Brooklyn, NY, USA

Abstract

We consider the problem of solving the Min-Sum Submodular Cover problem using local search. The Min-Sum Submodular Cover problem generalizes the NP-complete Min-Sum Set Cover problem, replacing the input set cover instance with a monotone submodular set function. A simple greedy algorithm achieves an approximation factor of 4, which is tight unless P=NP [Streeter and Golovin, NeurIPS, 2008]. We complement the greedy algorithm with analysis of a local search algorithm. Building on work of Munagala et al. [ICDT, 2005], we show that, using simple initialization, a straightforward local search algorithm achieves a $(4+\epsilon)$-approximate solution in time $O(n^3 \log(n/\epsilon))$, provided that the monotone submodular set function is also second-order supermodular. Second-order supermodularity has been shown to hold for a number of submodular functions of practical interest, including functions associated with set cover, matching, and facility location. We present experiments on two special cases of Min-Sum Submodular Cover and find that the local search algorithm can outperform the greedy algorithm on small data sets.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Local search, submodularity, second-order supermodularity, min-sum set cover

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.3

Related Version *Full Version*: `https://arxiv.org/abs/2209.03054`

Supplementary Material *Software*: `https://github.com/rtealwitter/Local-Search` archived at `swh:1:dir:a757879418c8fba172d787428be9b3480f008821`

Funding *Lisa Hellerstein*: NSF Award IIS-1909335
Thomas Lidbetter: NSF Award IIS-1909446

Acknowledgements We thank Christopher Musco for pointing out that the set function induced by entropy on continuous domains is not submodular.

1 Introduction

We consider the Min-Sum Submodular Cover problem, defined as follows. The input to the problem consists of an oracle for a monotone submodular function $u : 2^{[n]} \to \mathbb{R}_{\geq 0}$, and positive costs $c_1, \ldots, c_n \in \mathbb{R}_{>0}$, where $[n] = \{1, \ldots, n\}$. Let $c : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be such that for all $S \subseteq [n]$, $c(S) = \sum_{i \in S} c_i$. We refer to u as the "utility" and c as the "cost" function. The problem is to find the permutation of the elements of $[n]$ that minimizes

$$\sum_{i=1}^{n} c(S_i)\left(u(S_i) - u(S_{i-1})\right) \tag{1}$$

where S_i is the set containing the first i elements of the permutation. The Min-Sum Submodular Cover problem generalizes the NP-Complete Min-Sum Set Cover problem introduced by Feige et al. [5]. It has a simple greedy algorithm that achieves a 4-approximation [9, 18].[1] The 4-approximation is tight, assuming $P \neq NP$ [5].

In this work, we analyze a local search algorithm for Min-Sum Submodular Cover. Local search algorithms have been extensively applied to discrete optimization problems [3, 14, 15] and offer several benefits over other types of algorithms [2]. One advantage of local search algorithms compared to greedy methods in practice is their ability to explore a diverse set of solutions. In Section 4, we present the results of preliminary experiments which demonstrate that this ability can yield improved solutions to Min-Sum Submodular Cover instances.

The local search algorithm we consider works in iterations, starting with an initial solution. In an iteration, the algorithm updates the current solution to its best "neighbor". Once a solution is locally optimal (or after a fixed number of iterations), the algorithm returns the current solution.

Our analysis builds on previous work of Munagala et al. [16] for the Pipelined Set Cover problem (defined below). Munagala et al. exploit the observation that utility can be attributed to elements of the ground set for the set cover instance, and use these elements as variables in their linear program. The main challenge of generalizing their analysis is that the utility in general submodular set functions is more abstract and cannot be attributed to particular objects. In order to apply the linear program used in Munagala et al., our analysis relies on an additional property of the utility function called second-order supermodularity. We leave as an open question whether the algorithm gives a $(4+\epsilon)$-approximation even without second-order supermodularity.

Second-order supermodularity was first studied by Korula et al. [12]. It can be viewed as a natural extension of submodularity: If one considers the multilinear extension $F : [0,1]^n \to \mathbb{R}_{\geq 0}$ of a set function $f : \{0,1\}^n \to \mathbb{R}_{\geq 0}$ (which is a way of interpolating the values of f from the vertices of the Boolean hypercube to points in its interior), the submodularity of f is equivalent to the property that the second partial derivatives of F are non-positive. As mentioned by Korula et al. [12] and Iyer et al. [11], the second-order supermodularity of f is equivalent to the property that the third partial derivatives of F are non-negative.

The second-order supermodularity property is not overly restrictive; there are several classes of submodular functions that have this property including weighted coverage functions, weighted matching functions, and facility location [12, 10, 11]. Since this property was first defined, improved bounds have been obtained for optimization problems by assuming the property [12, 10]. The related properties of second-order modularity and second-order submodularity have also been used in analyzing local search algorithms for constrained submodular maximization problems [6, 7].

We now define two special cases of Min-Sum Submodular Cover: the Pipelined Set Cover problem and the Min-Sum Facility Location problem.

Pipelined Set Cover

The inputs to the problem consist of (i) m "ground" elements $\{1, \ldots, m\} = [m]$, (ii) $D_1, \ldots, D_n$, a family of n subsets of the ground elements $[m]$ such that $\bigcup_{i \in [n]} D_i = [m]$, and (iii) positive costs $c_1, \ldots, c_n$ associated with each D_i. Let $u : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be such that for all $S \subseteq [n]$, $u(S)$ is the number of ground elements in $\bigcup_{i \in S} D_i$. We call u a "coverage" function.

[1] See the Appendix A.1 for comments on Streeter and Golovin [18].

Let $c : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be such that for all $S \subseteq [n]$, $c(S) = \sum_{i \in S} c_i$. The problem is to find the permutation of $[n]$ that minimizes the objective function in the Min-Sum Submodular Cover problem, $\sum_{i=1}^{n} c(S_i)\,(u(S_i) - u(S_{i-1}))$. Thus Pipelined Set Cover problem is equivalent to the special case of the Min-Sum Submodular Cover problem where the utility function u is a coverage function. The Min-Sum Set Cover problem is the special case of Pipelined Set Cover with unit costs. (We note that Munagala et al. also present results for Weighted Pipelined Set Cover, where the ground elements have weights.)

Min-Sum Facility Location

Consider the following problem facility location problem, studied by Krause and Golovin [13]. There is a set $[n]$ of possible locations where facilities could be opened, to serve a collection of m customers. Opening a facility at location a provides a service of value $M_{a,b}$ to customer b, where $M \in \mathbb{R}_{\geq 0}^{n \times m}$. The utility of opening facilities in a subset S of the locations is $u(S)$, where $u(S) = \sum_{b=1}^{m} \max_{a \in S} M_{a,b}$. This corresponds to the total value obtained by all the customers, assuming each customer chooses the open facility with highest service value. The problem of Krause and Golovin is to maximize the utility function u subject to a constraint on the number of facilities that can be opened.

We introduce a min-sum version of this facility location problem by considering the Min-Sum Submodular Cover problem with the utility function u just described, and with c_i representing the time to open a facility i. This problem corresponds to a situation where facilities will be opened in all n locations, but they can only be opened one at a time. $M_{a,b}$ represents the estimated value facility a will provide to customer b per unit of time, once facility a is opened. Minimizing the objective value $\sum_{i=1}^{n} c(S_i)\,(u(S_i) - u(S_{i-1}))$ corresponds to finding the order to build facilities so as to minimize lost value as facilities are built.

Our Contributions

We introduce the study of solving Min-Sum Submodular Cover using local search. Building on work of Munagala et al. [16], who presented a local-search algorithm for Pipelined Set Cover, we generalize their LP-based analysis by redefining a key quantity in their proof and using second-order supermodularity. We show that local search produces a $(4+\epsilon)$-approximate solution for Min-Sum Submodular Cover in time $O(n^3 \log(\frac{d}{\epsilon}))$, assuming second-order supermodularity of the utility function, when initialized with a d-approximate solution. We prove that a permutation listing the items in non-decreasing cost order is an n-approximate solution. Thus initializing local search with a non-decreasing cost permutation enables us to reach a $(4+\epsilon)$-approximate solution in time $O(n^3 \log(\frac{n}{\epsilon}))$. Applying this result to Pipelined Set Cover improves on the $O(n^3 \log(\frac{mn}{\epsilon}))$ time bound from Munagala et al., where m is the size of the ground set of the set cover instance, by eliminating the dependence on m. We also present results of experiments on two types of Min-Sum Submodular Cover problems: Pipelined Set Cover and Min-Sum Facility Location. Our empirical findings suggest that local search can reliably produce better solutions than the natural greedy algorithm on small data sets.

2 Preliminaries

Let $f(e|S) := f(S \cup \{e\}) - f(S)$ be the marginal utility of adding element e to set S. With this notation in hand, we define several useful properties of set functions.

▶ **Definition 1** (Set Function Properties). *Consider a positive integer n and set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$. We first define the following properties of f, which hold if the inequality given below for that property holds for all $S \subseteq [n]$ and all $i, j, k \in [n] \setminus S$,*

- *monotone:* $f(S \cup \{i\}) \geq f(S)$,
- *submodular (diminishing returns):* $f(i|S) \geq f(i|S \cup \{j\})$,
- *second-order supermodular:* $f(i|S) - f(i|S \cup \{j\}) \geq f(i|S \cup \{k\}) - f(i|S \cup \{k, j\})$.

Note that the way in which we have written the above properties illustrates the progression from monotonicity to submodularity and submodularity to second-order supermodularity: we arrive at the "next" property by subtracting the left-hand side from the right-hand side. Another related property is modularity: for all $S \subseteq [n]$, $f(S) = \sum_{i \in S} f(\{i\})$. In this paper, the functions we consider will be monotone set functions that are normalized, i.e., $f(\emptyset) = 0$ unless otherwise stated.

The Min-Sum Submodular Cover problem is a special case of the Min-Sum Permutation Problem, defined by Happach et al. [8]. That problem has the same objective function as Min-Sum Submodular Cover, and minimization may be over all permutations, or only over a subset of them. The only assumptions on u and c in [8] are that they are monotone and normalized.

3 A Local Search Algorithm for Min-Sum Submodular Cover

Munagala et al. [16] gave a local search algorithm for the special case of the Min-Sum Submodular Cover problem where u is a coverage function. Applying the same approach to the general Min-Sum Submodular Cover problem, we have the following local search algorithm: initialize the algorithm with a permutation π of $[n]$. Define a neighbor π' of π to be a permutation that can be be produced from π by removing the element in some position i of π and reinserting it in position j. Find the neighbor π' of π with lowest objective value (given by Equation 1). If that value is less than the objective value of π, then replace π by π' and repeat. Otherwise, output π. Pseudocode for this algorithm is given in Algorithm 1.

The analysis of Munagala et al. [16] shows that in the special case where u is a coverage function, Algorithm 1 achieves a $(4 + \epsilon)$-approximation to the optimal permutation. We generalize their analysis to all utility functions u that are submodular and second-order supermodular (in addition to being monotone and normalized, which we assume is the case for all utility functions in this paper).

Let π_c be a non-decreasing cost permutation, i.e., $c(\{\pi_c(i)\}) \leq c(\{\pi_c(i+1)\})$ for $i \in [n-1]$. We prove the following results.

▶ **Theorem 2.** *Fix a positive integer n. Let $u : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be a submodular and second-order supermodular set function and let $c : 2^{[n]} \to \mathbb{R}_{\geq 0}$ be a modular set function. If Algorithm 1 converges before terminating, then the solution it returns is a 4-approximation to Min-Sum Submodular Cover on u and c.*

Unfortunately, we cannot guarantee that Algorithm 1 will converge before terminating. The next result guarantees a $(4 + \epsilon)$-approximation when the algorithm terminates.

▶ **Theorem 3.** *Consider the positive integer n, utility function u, and cost function c considered in Theorem 2. Fix $\epsilon > 0$. Let π be a d-approximate permutation. If Algorithm 1 does not converge before terminating, then the solution it returns (after $2n^3 \log(d/\epsilon)$ iterations), is a $(4 + \epsilon)$-approximation to Min-Sum Submodular Cover on u and c.*

Algorithm 1 Local search algorithm to produce a $(4+\epsilon)$-approximation.

```
Input: ε > 0, n > 0, utility function u : 2^[n] → R_≥0,
cost function c : 2^[n] → R_≥0, d-approximate permutation π
Output: permutation π
for iteration in {1, ..., 2n^3 log(d/ε)} do
    π* ← π
    for i, j ∈ [n] do
        # π' is π with π(i) moved to position j
        π' ← move(π, i, j)
        # objective(u, c, π) is Equation (1)
        if objective(u, c, π') < objective(u, c, π*) do
            π* ← π'
    if π* = π
        # Algorithm converged:
        # π is locally optimal with respect to moves
        return π # 4-approximation
    π ← π*
return π # (4+ε)-approximation
```

Assuming constant access query access to u and c, Algorithm 1 returns a $(4+\epsilon)$-approximation in $2n^3 \log(d/\epsilon)$ time.

As in Munagala et al. [16], in our analysis we consider a modified version of local search based on "insertions" rather than "moves." We find it easier to analyze local search with insertions and the approximation result immediately applies to local search with moves since a permutation that is locally optimal with respect to moves is also locally optimal with respect to insertions. In each iteration of local search with insertions, rather than considering the set of neighbors π' of π, the modified algorithm considers a set of what we will call pseudo-neighbors. Each is derived from π by taking an element appearing in some position i of π, and inserting a *second copy* of the element into some position $j < i$. Each pseudo-neighbor of π corresponds to a unique neighbor of π, produced from the pseudo-neighbor by removing the original copy of the repeated element (which appears closer to the end of the permutation).

Define the objective value of a pseudo-neighbor π' (which has length $n+1$) to be $\sum_{i=1}^{n+1} c'(S_i)[u(S_i) - u(S_{i-1})]$, where here S_i is the prefix of π' containing its first i elements, $u(S_i)$ is the value of u for the set of *distinct* items in S_i, and $c'(S_i) = \sum_{j=1}^{i} c(\{s_j\})$ where s_j is the element in position j of π'. That is, if both copies of the repeated element appear within the first i positions of π', then $c'(S_i)$ charges for both copies.

If the objective value of π is no greater than the value of its pseudo-neighbors, then the modified algorithm outputs π. Otherwise, the algorithm takes the pseudo-neighbor with lowest objective value, deletes the original copy of its repeated element, and uses the resulting permutation as the new value of π in the next iteration.

The objective value of a pseudo-neighbor of π is clearly greater than or equal to the objective value of the corresponding neighbor. Therefore, if π has no neighbor with lower objective value, then it has no pseudo-neighbor with lower objective value. It follows that the bounds we prove on the modified local search algorithm (with insertions) also apply to the original local search algorithm (with moves).

We prove Theorems 2 and 3 in the remainder of this section.

3.1 Proof of Theorem 2: 4-approximation

Say a permutation π is locally optimal if no pseudo-neighbor has lower objective value. We begin by proving that a locally optimal solution satisfies a certain inequality, expressed in terms of variables b_{ij}. This inequality is taken from the analysis in Munagala et al. [16], but we define the variables b_{ij} differently here. We will use the following technical Observation 4 to prove Lemma 5.

▶ **Observation 4.** *Consider three sequences of non-negative real numbers, $X_0, \ldots, X_n$, $Y_0, \ldots, Y_n$, and $C_0, \ldots, C_n$. Let $j \in [n]$. Suppose that the following hold: (i) $X_0 \geq Y_0$, (ii) for all $r \in \{j, \ldots, n\}$, $C_0 \leq C_r$ and $X_r \leq Y_r$, and (iii) $X_0 + \sum_{r=j}^{n} X_r = Y_0 + \sum_{r=j}^{n} Y_r$. Then*

$$C_0 X_0 + \sum_{r=j}^{n} C_r X_r \leq C_0 Y_0 + \sum_{r=j}^{n} C_r Y_r.$$

Proof. Rewriting the final assumption yields

$$\begin{aligned} X_0 - Y_0 = \sum_{r=j}^{n} (Y_r - X_r) \iff C_0(X_0 - Y_0) &= \sum_{r=j}^{n} C_0(Y_r - X_r) \\ \Rightarrow C_0(X_0 - Y_0) &\leq \sum_{r=j}^{n} C_r(Y_r - X_r) \end{aligned}$$

where the second implication follows from the non-negativity of $Y_r - X_r$ and the assumption that $C_0 \leq C_r$. Observation 4 follows immediately. ◀

▶ **Lemma 5.** *Suppose $u : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is a submodular and second-order supermodular set function and $c : 2^{[n]} \to \mathbb{R}_{\geq 0}$ is a modular set function. Let L_j denote the first j elements of the locally optimal permutation and O_i denote the first i elements of the optimal permutation. Similarly, we use l_j to represent the jth element of the local permutation and o_i to represent the ith element of the optimal permutation. Then*

$$\sum_{r=j}^{n} c(L_r) \sum_{s=1}^{n} b_{sr} \leq [c(o_i) + c(L_{j-1})] \sum_{r=j}^{n} b_{ir} + \sum_{r=j}^{n} [c(o_i) + c(L_r)], \sum_{\substack{s=1 \\ s \neq i}}^{n} b_{sr} \tag{2}$$

where

$$\begin{aligned} b_{ij} &= u(o_i | O_{i-1} \cup L_{j-1}) - u(o_i | O_{i-1} \cup L_{j-1} \cup \{l_j\}) \\ &= u(l_j | O_{i-1} \cup L_{j-1}) - u(l_j | O_{i-1} \cup L_{j-1} \cup \{o_i\}). \end{aligned}$$

Proof. When u is a coverage function, as in the analysis in Munagala et al. [16], b_{ij} represents the number of ground elements covered in the optimal permutation by subset o_i (and not by $o_1, \ldots, o_{i-1}$) and in the local permutation by subset l_j (and not by $l_1, \ldots, l_{j-1}$). Our definitions of b_{ij} generalize this intuition to functions where the utility is more abstract. In particular, we can use telescoping sums to derive the following identities:

$$\sum_{i=1}^{n} b_{ij} = u(l_j | L_{j-1}) \qquad \sum_{r=j}^{n} b_{ir} = u(o_i | O_{i-1} \cup L_{j-1}) \qquad \sum_{\substack{s=1 \\ s \neq i}}^{n} b_{sr} = u(l_r | L_{r-1}) - b_{ir} \tag{3}$$

Then Equation (2) is equivalent to

$$\sum_{r=j}^{n} c(L_r)u(l_r|L_{r-1}) \leq [c(o_i) + c(L_{j-1})]u(o_i|O_{i-1} \cup L_{j-1}) + \sum_{r=j}^{n}[c(o_i) + c(L_r)][u(l_r|L_{r-1}) - b_{ir}]. \tag{4}$$

Notice that Equation (4) is not *quite* equivalent to the property of local optimality with respect to insertions. Instead, we know the following (very similar) inequality holds by the property that L is locally optimal:

$$\sum_{r=j}^{n} c(L_r)u(l_r|L_{r-1}) \leq [c(o_i) + c(L_{j-1})]u(o_i|L_{j-1}) + \sum_{r=j}^{n}[c(o_i) + c(L_r)]u(l_r|L_{r-1} \cup \{o_i\}). \tag{5}$$

We will now show that the right-hand side of Equation (5) lower bounds the right-hand side of Equation (4). Then Equation (4) follows from Equation (5). We do this through the following four conditions combined with Observation 4:

$$c(L_{j-1}) \leq c(L_{r-1}) \tag{6}$$

$$u(o_i|L_{j-1}) \geq u(o_i|O_{i-1} \cup L_{j-1}) \tag{7}$$

$$u(l_r|L_{r-1} \cup \{o_i\}) \leq u(l_r|L_{r-1}) - b_{ir} \tag{8}$$

$$u(o_i|L_{j-1}) + \sum_{r=j}^{n} u(l_r|L_{r-1} \cup \{o_i\}) = u(o_i|O_{i-1} \cup L_{j-1}) + \sum_{r=j}^{n}[u(l_r|L_{r-1}) - b_{ir}] \tag{9}$$

for all $i, j, r \in [n]$ with $j \leq r \leq n$. Equation (6) holds by the monotonicity of c and Equation (7) holds by the submodularity of u. Equation (8) holds because u is second-order supermodular. (This is where we use second-order supermodularity.)

Finally, Equation (9) holds by the following argument: Notice that the left-hand side is equal to $u([n]) - u(L_{j-1})$ since we sequentially sum the marginal utility of adding the next element to our current chain. Similarly, the right-hand side simplifies to $u([n]) - u(L_{j-1})$ after cancellation using Equation (3).

Using the above, we apply Observation 4 with $C_0 = c(o_i) + c(L_{j-1})$, $C_{r-j+1} = c(o_i) + c(L_{r-1})$, $X_0 = u(o_i|L_{j-1})$, $X_{r-j+1} = u(l_r|L_{r-1} \cup \{o_i\})$, $Y_0 = u(o_i|O_{i-1} \cup L_{j-1})$, and $Y_{r-j+1} = u(l_r|L_{r-1}) - b_{ir}$. This yields

$$[c(o_i) + c(L_{j-1})]u(o_i|L_{j-1}) + \sum_{r=j}^{n}[c(o_i) + c(L_{r-1})][u(l_r|L_{r-1} \cup \{o_i\})]$$

$$\leq [c(o_i) + c(L_{j-1})]u(o_i|O_{i-1} \cup L_{j-1}) + \sum_{r=j}^{n}[c(o_i) + c(L_{r-1})][u(l_r|L_{r-1}) - b_{ir}].$$

Then the right-hand side of Equation (4) must be greater than or equal to the right-hand side of Equation (5). Therefore Lemma 5 follows by Equation (5). ◀

Proof of Theorem 2. We will now prove that a locally optimal permutation with respect to insertions is (at worst) a 4-approximation of the optimal permutation. To do this, we will show that the same linear program used by Munagala et al. [16] to prove their 4-approximation for coverage functions also applies to all utility functions that are monotone, submodular and second-order supermodular. The linear program in Munagala et al. [16] is:

$$\text{maximize} \sum_{j=1}^{n} c(L_j) \sum_{i=1}^{n} b_{ij} \tag{10}$$

$$\text{subject to} \sum_{i=1}^{n} c(O_i) \sum_{j=1}^{n} b_{ij} \leq 1 \quad \text{and}$$

$$\sum_{r=j}^{n} c(L_r) \sum_{s=1}^{n} b_{sr} \leq (c(o_i) + c(L_{j-1})) \sum_{r=j}^{n} b_{ir} + \sum_{r=j}^{n} (c(o_i) + c(L_r)) \sum_{\substack{s=1 \\ s \neq i}}^{n} b_{sr} \quad \forall i, j \in [n] \tag{11}$$

where the variables b_{ij} are non-negative for all $i, j \in [n]$.

The first constraint scales the variables in the optimal permutation so that the objective is the ratio of the local permutation to the optimal permutation. By Lemma 5, the second constraint must hold for locally optimal L, with the given values for b_{ij}. As shown in Munagala et al. [16], there is a feasible solution to the dual of the LP with objective value 4 so, by strong duality, the locally optimal permutation achieves a 4-approximation to the optimal permutation. ◀

3.2 Proof of Theorem 3: $(4 + \epsilon)$-approximation

In this section, we show that local search reaches a $(4 + \epsilon)$-approximation in reasonable time. The proof is based on the following fact, which bounds the progress made in each "round" of the local search (i.e., in each iteration of the while loop in Algorithm 1).

▶ **Fact 6.** *Let π be a permutation of $[n]$ that is an M-approximation to Min-Sum Submodular Cover. Then applying one round of local search to this permutation will either establish that it is a local optimum or yield a new permutation that is an $(M - \frac{M-4}{2n})$-approximation.*

Proof. In Munagala et al. [16], they prove that Fact 6 holds for a Pipelined Filter Ordering instance, i.e., when u is a coverage function. Consider an M-approximate solution. Let A be a variable representing the reduction in approximation factor after making the local search step. As in Equation (11), the variables b_{ij} are non-negative real numbers and correspond to a concept of shared utility formalized in Lemma 5. Then the following linear program is what Munagala et al. [16] use to lower-bound the improvement in approximation ratio:

$$\text{minimize } A$$

$$\text{subject to} \sum_{i=1}^{n} c(O_i) \sum_{j=1}^{n} b_{ij} \leq 1 \text{ and}$$

$$\sum_{r=j}^{n} c(L_r) \sum_{s=1}^{n} b_{sr} \leq A + (c(o_i) + c(L_{j-1})) \sum_{r=j}^{n} b_{ir} + \sum_{r=j}^{n} (c(o_i) + c(L_r)) \sum_{\substack{s=1 \\ s \neq i}}^{n} b_{sr} \text{ and}$$

$$\sum_{j=1}^{n} c(L_j) \sum_{i=1}^{n} b_{ij} \geq M \quad \forall i, j \in [n].$$

For the more general case where u is an abstract submodular and second-order supermodular utility function, the first and third inequalities trivially hold for our generalized definition of b_{ij}. The second inequality follows from Lemma 5. By taking the dual, Munagala et al. [16] show that the objective of the primal and therefore the reduction in approximation

ratio is at least $\frac{M-4}{2n}$ which gives Fact 6. When u is an arbitrary submodular and second-order supermodular function, it follows from Lemma 5 that the same linear program still lower-bounds the improvement. ◀

To achieve a good bound on the time required for the local search for Min-Sum Submodular Cover, we want to begin the local search from a permutation that is not too far from optimal. The following theorem gives such a permutation.

▶ **Theorem 7** (Non-Decreasing Cost). *A permutation that orders the elements $i \in [n]$ in non-decreasing order of $c(\{i\})$ is an n-approximate solution to Min-Sum Submodular Cover.*

Proof. Suppose, without loss of generality, that $c(\{1\}) \leq \cdots \leq c(\{n\})$.

Suppose an optimal solution is given by the sets $\emptyset = S_0, S_1 \ldots, S_n$, and for each $j \in [n]$, let $S'_j = [\max S_j]$, where $\max S_j$ is the maximum integer in S_j. For instance, for $n = 4$ and $S_1 = \{2\}$, $S_2 = \{12\}$, $S_3 = \{124\}$, $S_4 = \{1234\}$, we have $S'_1 = \{12\}$, $S'_2 = \{12\}$, $S'_3 = \{1234\}$, $S'_4 = \{1234\}$. Setting $[0]$ and S'_0 equal to $\emptyset$, observe that

$$\sum_{j=1}^{n} c([j])(u([j]) - u([j-1])) \leq \sum_{j=1}^{n} c(S'_j)(u([j]) - u([j-1])) = \sum_{j=1}^{n} c(S'_j)(u(S'_j) - u(S'_{j-1})),$$

where the inequality follows by the monotonicity of c and the equality follows (not term-wise but in total) by considering when utility is accrued by each side.

Now, for each $j \geq 0$, we have $c(S_j) \geq c(\{\max S_j\})$, by the monotonicity of c. Also, since $c(\{\max S_j\}) \geq c(\{i\})$ for every $i \in S_j$ by the indexing assumption,

$$n \cdot c(S_j) \geq n \cdot c(\{\max S_j\}) = n \cdot c(\{\max S'_j\}) \geq c(S'_j)$$

where the equality follows from the definition of S'_j and the second inequality follows since there are at most n elements in S'_j. Then the objective value of the optimal permutation is at least $\frac{1}{n}\sum_{j=1}^{n} c(S'_j)(u(S_j) - u(S_{j-1}))$, or equivalently by charging utility to each increase in cost,

$$\frac{1}{n}\sum_{j=1}^{n}(u([n]) - u(S_{j-1}))(c(S'_j) - c(S'_{j-1})).$$

By the monotonicity of u, this sum is at least

$$\frac{1}{n}\sum_{j=1}^{n}(u([n]) - u(S'_{j-1}))(c(S'_j) - c(S'_{j-1})) \Leftrightarrow \frac{1}{n}\sum_{j=1}^{n} c(S'_j)(u(S'_j) - u(S'_{j-1})).$$

This is at least $1/n$ times the objective value of the increasing cost permutation, by our earlier observation. ◀

We note that Theorem 7 also holds for Min-Sum Permutation Problems minimizing over all permutations where u only satisfies monotonicity and c only satisfies monotonicity and subadditivity (both are still normalized).

We can now prove that the output of Algorithm 1 is a $(4+\epsilon)$-approximation to Min-Sum Submodular Cover. The time bound assumes constant time oracle queries.

Proof of Theorem 3. The improvement in the quality of the solution in each round of local search, guaranteed by Fact 6, implies that a $(4+\epsilon)$-approximation is achieved within $O(n\log(\frac{d}{\epsilon}))$ rounds of Algorithm 1, when it is initialized with a d-approximate permutation. This implication was stated without proof by Munagala et al. [16], in proving the same

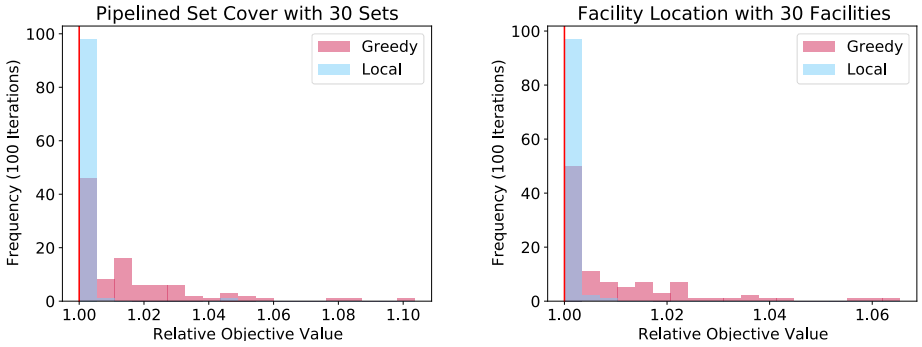

Figure 1 Histograms of greedy and local search performance with $n = 30$ in Pipelined Set Cover and Min-Sum Facility Location. The relative objective values are normalized with respect to the best of the 6 generated solutions (1 greedy and 5 local search). Frequency is reported from the 100 random instances generated for each dataset.

bounds for Pipelined Set Cover. For completeness, we present a proof of the implication in Appendix A.2. The $O(n^3 \log(\frac{d}{\epsilon}))$ time bound is achieved by spending $O(n^2)$ per round. To accomplish this, we do not recompute all n terms of the objective function for each of the $\theta(n^2)$ neighbors of the current solution. Instead, by considering "adjacent" neighbors sequentially, we can compute the objective function value for the next neighbor from the value obtained for the previous neighbor in constant time, by recomputing only two terms of the objective function. The time bound for the non-decreasing cost permutation follows from Theorem 7. ◀

4 Experiments

Assuming constant-time oracle access to the utility functions, the greedy algorithm runs in $O(n^2)$ total time, while our local search algorithm spends time $O(n^2)$ in each round. In our experiments, with no oracle, we had to compute the utility function.

The greedy algorithm is certainly faster than the local search, but it only explores one type of solution, where cost effective elements appear earlier in the permutation. Local search initialized with random permutations can sample from the entire solution space. Our experiments compare the greedy solution to local search solutions from four random initial permutations, and from a non-decreasing cost permutation. We run each local search for n steps (rather than running it to convergence, or until it is guaranteed to find a $(4+\epsilon)$-approximate solution). We see empirically that the best of the 5 local search solutions tends to be better than the worst, and also better than the greedy solution. In applications where computing is cheap, n is not large, and the quality of the solution is crucial, using local search may be preferable to using greedy.

Our experiments compare greedy and local search on 100 random instances of two problems: Pipelined Set Cover and Min-Sum Facility Location. For our random instances, we set $n = 30$ and each cost c_i to be a uniform random value between 0 and 1. Figure 1 shows the results of our experiments, with objective values given relative to the best of the 6 solutions (1 greedy and 5 local search). Local search finds the best solution in almost 100% of the 100 instances whereas greedy finds the best in roughly 50%.

Pipelined Set Cover

We perform experiments on synthetic, randomly generated instances of the (unweighted) Pipelined Set Cover problem with correlated subsets, following an approach of Babu et al. [4]. Recall that an instance of (unweighted) Pipelined Set Cover consists of a finite ground set $[m]$ and a collection of subsets $D_i \subseteq V$ for $i \in [n]$. The utility of a set $S \subseteq [n]$ is $u(S) = |\bigcup_{i \in S} D_i|$. The n subsets D_i in our random instance are divided into $\lceil n/\Gamma \rceil$ groups, where Γ is a "correlation factor." The instance has the following properties, for each element $j \in [m]$. For all $i \in [n]$, $\Pr[j \in D_i]$ is a fixed value p. For two subsets D_i and $D_{i'}$ in different groups, membership of j in D_i is independent of its membership in $D_{i'}$. For two subsets D_i and $D_{i'}$ in the same group, the probability that j has the same membership status in D_i and $D_{i'}$ (i.e., is either in both subsets, or in neither), is a fixed value ρ. In our experiments, $m = 2n$, $\Gamma = 4$, $p = 0.3$, and $\rho = 0.7858$. In Appendix A.3, we describe in detail how we generated the instance.

Min-Sum Facility Location

We use the locations of n Citi Bike stations [1] in New York City as the facilities for our facility location data set. For calculating the utility to customers, we uniformly generate customer locations within the range of latitude and longitude of the stations. The value $M_{a,b}$ for customer b and station a is the inverse of the Euclidean distance between them.

References

1 Citi bike system data. `https://ride.citibikenyc.com/system-data`, 2021. Accessed: 2021-12-1.

2 Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

3 Daniel Antunes, Claire Mathieu, and Nabil H. Mustafa. Combinatorics of Local Search: An Optimal 4-Local Hall's Theorem for Planar Graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ESA.2017.8`.

4 Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive ordering of pipelined stream filters. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 407–418, 2004.

5 Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

6 Mehrdad Ghadiri, Richard Santiago, and Bruce Shepherd. Beyond submodular maximization. *arXiv preprint*, 2019. `arXiv:1904.09216`.

7 Mehrdad Ghadiri, Richard Santiago, and Bruce Shepherd. A parameterized family of meta-submodular functions. *arXiv preprint*, 2020. `arXiv:2006.13754`.

8 Felix Happach, Lisa Hellerstein, and Thomas Lidbetter. A general framework for approximating min sum ordering problems. *INFORMS Journal on Computing*, 34(3):1437–1452, 2022.

9 Satoru Iwata, Prasad Tetali, and Pushkar Tripathi. Approximating minimum linear ordering problems. In *Proceedings of Approx-Random*, 2012.

10 Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory*, pages 722–754. PMLR, 2021.

11 Rishabh Iyer, Ninad Khargonkar, Jeff Bilmes, and Himanshu Asnani. Generalized submodular information measures: Theoretical properties, examples, optimization algorithms, and applications. *IEEE Transactions on Information Theory*, 68(2):752–781, 2021.

12 Nitish Korula, Vahab Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM Journal on Computing*, 47(3):1056–1086, 2018.

13 Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2014.

14 Gilad Kutiel and Dror Rawitz. Local Search Algorithms for Maximum Carpool Matching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55:1–55:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2017.55.

15 Wenjun Li, Jianer Chen, and Jianxin Wang. Deeper local search for better approximation on maximum internal spanning trees. In *European Symposium on Algorithms*, pages 642–653. Springer, 2014.

16 Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The pipelined set cover problem. In *International Conference on Database Theory*, pages 83–98. Springer, 2005.

17 Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. Technical report, Carnegie-Mellon University School of Computer Science, 2007.

18 Matthew J. Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *Neural Information Processing Systems (NeurIPS)*, pages 1577–1584, 2008.

A Appendix

A.1 Prior Use of the Term "Min-Sum Submodular Cover"

The first proof that a greedy algorithm for Min-Sum Submodular Cover yields a 4-approximation was given by Streeter and Golovin. The proof appears both in a Technical Report [17] and an associated conference paper [18]. They actually gave their proof for a more general problem than the one we considered in this paper, in which $u : 2^{[n]\times\mathbb{R}} \to \mathbb{R}$, and the output is a sequence of pairs of the form $(i, \tau) \in [n] \times \mathbb{R}$. In their Technical Report, Streeter and Golovin used the name Min-Sum Submodular Cover to refer to the more general problem, but they did not use this name (nor any other) to refer to the problem in their conference paper. We opted to use the name Min-Sum Submodular Cover to refer to the problem we defined in this paper, as we believe this usage of the name is natural given the connection to Min-Sum Submodular Cover.

We note that the definition Streeter and Golovin gave for the more general problem is problematic as written. The greedy algorithm may not be well-defined for functions u that are non-zero for subsets of $[n] \times \mathbb{R}$ that include pairs (i, τ), where τ is infinitesimally small. However, the results in the paper are not dependent on allowing such u, and the problem with the definition can be fixed by restricting the domain of u.

A.2 Bounding the number of rounds in the proof of Theorem 3

We show using Fact 6 that Algorithm 1 yields a $(4+\epsilon)$-approximation in at most $O(n \log(\frac{d}{\epsilon}))$ rounds from a d-approximate permutation.

We introduce a recurrence relation $T(\ell) = aT(\ell-1) - b$ where $a = 2n/(2n-1)$ and $b = 4/(2n-1)$. We can derive this recurrence by setting $T(\ell) = M$ and $T(\ell-1) = M-(M-4)/2n$. Intuitively, ℓ is the number of iterations until we reach $(4+\epsilon)$ and so we set $T(0) = 4+\epsilon$. By repeatedly expanding $T(\ell)$, we get

$$T(\ell) = -b\sum_{i=0}^{\ell-1} a^i + a^\ell(4+\epsilon) = -b\frac{a^\ell - 1}{a-1} + a^\ell(4+\epsilon) = \epsilon\left(\frac{2n}{2n-1}\right)^\ell + 4$$

where the last equality follows by plugging in the values of a and b. We claim that a d-approximate permutation is at most $2n\log(\frac{d}{\epsilon})$ rounds from a $(4+\epsilon)$-approximation. We can verify this by evaluating

$$T\left(2n\log\left(\frac{d}{\epsilon}\right)\right) = \epsilon\left(\frac{2n}{2n-1}\right)^{2n\log(\frac{d}{\epsilon})} + 4 = d^{2n\log\left(\frac{2n}{2n-1}\right)} + 4.$$

Notice that

$$x\log\left(\frac{x}{x-1}\right) \geq 1 \iff e^x\left(\frac{x}{x-1}\right) \geq e$$

which is certainly true for $x > 2$. It follows that $T(2n\log(\frac{d}{\epsilon})) \geq d$ so local search converges in at most $2n\log(\frac{d}{\epsilon})$ rounds.

A.3 Generation of the Pipelined Set Cover Data Set

We generate the n subsets $D_1, \ldots, D_n$ of $[m]$ randomly as follows. Initially, for each group G of subsets, we generate an advice bit $a_{G,j}$ for each element $j \in [m]$, which is True with probability p, and False with probability $1-p$. Then for each element j, and for each D_i in group G, we do the following: with probability p' we use advice bit $a_{G,j}$ to determine whether or not to include j in D_i (if $a_{G,j}$ is True, we include j in D_i, else we do not). With probability $(1-p')$, we ignore the advice bit, and instead, we include j in D_i with probability p, and exclude it with probability $(1-p)$. The probability that j is in a given set D_j is clearly p.

For $i \neq i'$, if subsets D_i and $D_{i'}$ are in different groups, then membership of an element j in D_i is clearly independent of its membership of $D_{i'}$. If D_i and $D_{i'}$ are subsets in the same group G, then the probability that j has the same membership status in both subsets can be calculated by noting that this can happen in two ways: either the advice bit $a_{G,j}$ was used to determine membership in both subsets, or $a_{G,j}$ was ignored for one or both of the two subsets and membership ended up being the same in both subsets. The first event happens with probability $p' \cdot p'$. The second happens with probability $(1-p' \cdot p')(p \cdot p + (1-p) \cdot (1-p))$. Because $p = 0.3$ and $p' = 0.7$, the probability that the membership status of j is the same for both subsets is .7858.

We note that it is possible that this process results in subsets D_i and $D_{i'}$ where $i \neq i'$ and $D_i = D_{i'}$. We do not eliminate such duplicates.