Min-Max Submodular Ranking for Multiple Agents[*]

Qingyun Chen[1], Sungjin Im[1], Benjamin Moseley[2], Chenyang Xu[3,4†], Ruilong Zhang[5†]

[1] Electrical Engineering and Computer Science, University of California at Merced
[2] Tepper School of Business, Carnegie Mellon University
[3] Software Engineering Institute, East China Normal University
[4] College of Computer Science, Zhejiang University
[5] Department of Computer Science, City University of Hong Kong
qchen41@ucmerced.edu, sim3@ucmerced.edu, moseleyb@andrew.cmu.edu, xcy1995@zju.edu.cn, ruilzhang4-c@my.cityu.edu.hk

Abstract

In the submodular ranking (SR) problem, the input consists of a set of submodular functions defined on a ground set of elements. The goal is to order elements for all the functions to have value above a certain threshold as soon on average as possible, assuming we choose one element per time. The problem is flexible enough to capture various applications in machine learning, including decision trees.

This paper considers the min-max version of SR where multiple instances share the ground set. With the view of each instance being associated with an agent, the min-max problem is to order the common elements to minimize the maximum objective of all agents—thus, finding a fair solution for all agents. We give approximation algorithms for this problem and demonstrate their effectiveness in the application of finding a decision tree for multiple agents.

1 Introduction

The submodular ranking (SR) problem was proposed by (Azar and Gamzu 2011). The problem includes a ground set of elements $[n]$, and a collection of m monotone submodular[1] set functions $f_1, \dots, f_m$ defined over $[n]$, i.e., $f_j : 2^{[n]} \to \mathbb{R}_{\geq 0}$ for all $j \in [m]$. Each function f_j is additionally associated with a positive weight $w_j \in \mathbb{R}$. Given a permutation π of the ground set of elements, the cover time of a function f_j is defined to be the minimal number of elements in the prefix of π which forms a set such that the corresponding function value is larger than a unit threshold value[2]. The goal is to find a permutation of the elements such that the weighted average cover time is minimized.

The SR problem has a natural interpretation. Suppose there are m types of clients who are characterized by their utility function $\{f_j\}_{j\in[m]}$, and we have to satisfy a client sampled from a known probability distribution $\{w_j\}_{j\in[m]}$. Without knowing her type, we need to sequentially present items to satisfy her, corresponding to her utility f_j having a value above a threshold. Thus, the goal becomes finding the best ordering to minimize the average number of items shown until satisfying the randomly chosen client.

In the *min-max* submodular ranking for multiple agents, we are in the scenario where there are k SR instances sharing the common ground set of elements, and the goal is to find a common ordering that minimizes the maximum objective of all instances. Using the above analogy, suppose there are k different groups of clients. We have k clients to satisfy, one sample from each group. Then, we want to satisfy *all* the k clients as early as possible. In other words, we want to commit to an ordering that satisfies all groups fairly—formalized in a min-max objective.

1.1 Problem Definition

The problem of min-max submodular ranking for multiple agents (SRMA) is formally defined as follows. An instance of this problem consists of a ground set $U := [n]$, and a set of k agents $A := \{ a_1, \dots, a_k \}$. Every agent a_i, where $i \in [k]$, is associated with a collection of m monotone submodular set functions $f_1^{(i)}, \dots, f_m^{(i)}$. It is the case that $f_j^{(i)} : 2^{[n]} \to [0,1]$ with $f_j^{(i)}(U) = 1$ for all $i \in [k], j \in [m]$. In addition, every function $f_j^{(i)}$ is associated with a weight $w_j^{(i)} > 0$ for all $i \in [k], j \in [m]$.

Given a permutation π of the ground elements, the cover time $\texttt{cov}(f_j^{(i)}, \pi)$ of $f_j^{(i)}$ in π is defined as the smallest index t such that the function $f_j^{(i)}$ has value 1 for the first t elements in the given ordering. The goal is to find a permutation of the ground elements that minimizes the maximum total weighted cover time among all agents, i.e., finding a permutation π such that $\max_{i\in[k]} \left\{ \sum_{j=1}^{m} w_j^{(i)} \cdot \texttt{cov}(f_j^{(i)}, \pi) \right\}$ is

[*]All authors (ordered alphabetically) have equal contributions and are corresponding authors.

[†]The work was done when Chenyang Xu was a student at Zhejiang University and Ruilong Zhang visited CMU.

[1]A function $f : 2^{[n]} \to \mathbb{R}$ is submodular if for all $A, B \subseteq [n]$ we have $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$. The function is monotone if $f(A) \leq f(B)$ for all $A \subseteq B$.

[2]If the threshold is not unit-valued, one can easily obtain an equivalent instance by normalizing the corresponding function.

minimized. We assume that the minimum non-zero marginal value of all $m \cdot k$ functions is $\epsilon > 0$, i.e., for any $S \subseteq S'$, $f_j^{(i)}(S') > f_j^{(i)}(S)$ implies that $f_j^{(i)}(S') - f_j^{(i)}(S) \geq \epsilon > 0$ for all $i \in [k], j \in [m]$. Without loss of generality, we assume that any $w_j^{(i)} \geq 1$ and let $W := \max_{i \in [k]} \sum_{j=1}^{m} w_j^{(i)}$ be the maximum total weight among all agents.

1.2 Applications

In addition to the aforementioned natural interpretation, we discuss two concrete applications below in detail.

Optimal Decision Tree (ODT) with Multiple Probability Distributions. In (the non-adaptive) optimal decision tree problem with multiple probability distributions, we are given k probability distributions $\{p_j^{(i)}\}_{j \in [m]}$, $i \in [k]$ over m hypotheses and a set $U := [n]$ of binary tests. There is exactly one unknown hypothesis $\widetilde{j}^{(i)}$ drawn from $\{p_j^{(i)}\}_{j \in [m]}$ for each $i \in [k]$. The outcome of each test $e \in U$ is a partition of m hypotheses. Our goal is to find a permutation of U that minimizes the expected number of tests to identify *all* the k sampled hypotheses $\widetilde{j}^{(i)}, i \in [k]$.

This problem generalizes the traditional optimal decision tree problem which assumes $k = 1$. The problem has the following motivation. Suppose there are m possible diseases and their occurrence rate varies depending on the demographics. A diagnosis process should typically be unified and the process should be fair to all demographic groups. If we adopt the min-max fairness notion, the goal then becomes to find a common diagnostic protocol that successfully diagnoses the disease to minimize the maximum diagnosis time for any of the k groups. The groups are referred to as agents in our problem.

As observed in (Jia et al. 2019), the optimal decision tree is a special case of submodular ranking. To see this, fix agent a_i. For notational simplicity we drop a_i. Let $T_j(e)$ be the set of hypotheses that have an outcome different from j for test e. For each hypothesis j, we define a monotone submodular function $f_j : 2^U \to [0, 1]$ with a weight p_j as follows:

$$f_j(S) = \left| \bigcup_{e \in S} T_j(e) \right| \cdot \frac{1}{m-1}$$

which is the fraction of hypotheses other than j that have been ruled out by a set of tests S. Then, hypothesis $\widetilde{j}$ can be identified if and only if $f_{\widetilde{j}}(S) = 1$.

Web Search Ranking with Multiple User Groups. Besides fair decision tree, our problem also captures several practical applications, as discussed in (Azar and Gamzu 2011). Here, we describe the application of web search ranking with multiple user groups. A user group is drawn from a given distribution defined over k user groups. We would like to display the search results sequentially from top to bottom. We assume that each user browses search results from top to bottom. Each user is satisfied when she has found sufficient web pages relevant to the search, and the satisfaction time corresponds to the number of search results she checked. The satisfaction time of a user group is defined as the total satisfied time of all users in this group. The min-max objective ensures fairness among different groups of users.

1.3 Our Contributions

To our knowledge, this is the first work that studies submodular ranking, or its related problems, such as optimal decision trees, in the presence of multiple agents (or groups). Finding an order to satisfy all agents equally is critical to be fair to them. This is because the optimum ordering can be highly different for each agent.

We first consider a natural adaptation of the normalized greedy algorithm (Azar and Gamzu 2011), which is the best algorithm for submodular ranking. We show that the adaptation is $O(k \ln(1/\epsilon))$-approximation and has an $\Omega(\sqrt{k})$ lower bound on the approximation ratio.

To get rid of the polynomial dependence on k, we then develop a new algorithm, which we term *balanced adaptive greedy*. Our new algorithm overcomes the polynomial dependence on k by carefully balancing agents in each phase. To illustrate the idea, for simplicity, assume all weights associated with the functions are 1 in this paragraph. We iteratively set checkpoints: in the p-th iteration, we ensure that we satisfy all except about $m/2^p$ functions for each agent. By balancing the progress among different agents, we obtain a $O(\ln(1/\epsilon) \log(\min\{n, m\}) \log k)$ approximation (Theorem 2), where n is the number of elements to be ordered. When $\min\{n, m\} = o(2^k)$, the balanced adaptive greedy algorithm has a better approximation ratio than the natural adaptation of the normalized greedy algorithm. For the case that $\min\{n, m\} = \Omega(2^k)$, we can simply run both of the two algorithms and pick the better solution which yields an algorithm that is always better than the natural normalized greedy algorithm.

We complement our result by showing that it is NP-hard to approximate the problem within a factor of $O(\ln(1/\epsilon) + \log k)$ (Theorem 3). While we show a tight $\Theta(\log k)$-approximation for generalized min-sum set cover over multiple agents which is a special case of our problem (see Theorem 4), reducing the gap in the general case is left as an open problem.

We demonstrate that our algorithm outperforms other baseline algorithms for real-world data sets. This shows that the theory is predictive of practice. The experiments are performed for optimal decision trees, which is perhaps the most important application of submodular ranking.

1.4 Related Works

Submodular Optimization. Submodularity commonly arises in various practical scenarios. It is best characterized by diminishing marginal gain, and it is a common phenomenon observed in all disciplines. Particularly in machine learning, submodularity is useful as a considerable number of problems can be cast into submodular optimizations, and (continuous extensions of) submodular functions can be used as regularization functions (Bach et al. 2013). Due to the extensive literature on submodularity, we only discuss the most relevant work here. Given a monotone submodular function $f : 2^U \to \mathbb{Z}_{\geq 0}$, choosing a $S \subseteq U$ of minimum cardinality s.t. $f(S) \geq T$ for some target $T \geq 0$ admits a $O(\log T)$-approximation (Williamson and Shmoys 2011), which can be achieved by iteratively choosing an element that increase the function value the most, i.e., an element with the maximum marginal gain. Or, if f has a range $[0, 1]$, $T = 1$ and the non-zero marginal increase is at least $\epsilon > 0$, we can obtain an $O(\ln(1/\epsilon))$-approximation.

Submodular Ranking. The submodular ranking problem was introduced by (Azar and Gamzu 2011). In (Azar and Gamzu 2011), they gave an elegant greedy algorithm that achieves $O(\ln \frac{1}{\epsilon})$-approximation and provided an asymptotically matching lower bound. The key idea was to renormalize the submodular functions over the course of the algorithm. That is, if S is the elements chosen so far, we choose an element e maximizing $\sum_{f \in \mathcal{F}} w_f \cdot \frac{f(S \cup \{e\}) - f(S)}{1 - f(S)}$, where $\mathcal{F}$ is the set of uncovered functions. Thus, if a function f_j is nearly covered, then the algorithm considers f_j equally by renormalizing it by the residual to full coverage, i.e., $1 - f_j(S)$. Later, (Im, Nagarajan, and van der Zwaan 2016) gave a simpler analysis of the same greedy algorithm and extended it to other settings involving metrics. Special cases of submodular ranking include min-sum set cover (Feige, Lovász, and Tetali 2004) and generalized min-sum set cover (Azar, Gamzu, and Yin 2009; Bansal et al. 2021). For stochastic extension of the problem, see (Im, Nagarajan, and van der Zwaan 2016; Agarwal, Assadi, and Khanna 2019; Jia et al. 2019).

Optimal Decision Tree. The optimal decision tree problem (with one distribution) has been extensively studied. The best known approximation ratio for the problem is $O(\log m)$ (Gupta, Nagarajan, and Ravi 2017) where m is the number of hypotheses and it is asymptotically optimal unless P = NP (Chakaravarthy et al. 2007). As discussed above, it is shown in (Jia et al. 2019) how the optimal decision tree problem is captured by the submodular ranking. For applications of optimal decision trees, see (Dasgupta 2004). Submodular ranking only captures non-adaptive optimal decision trees. For adaptive and noisy decision trees, see (Golovin, Krause, and Ray 2010; Jia et al. 2019).

Fair Algorithms. Fairness is an increasingly important criterion in various machine learning applications (Barocas, Hardt, and Narayanan 2017; Chouldechova and Roth 2020; Mehrabi et al. 2021; Halabi et al. 2020; Abernethy et al. 2022), yet certain fairness conditions cannot be satisfied simultaneously (Kleinberg, Mullainathan, and Raghavan 2016; Corbett-Davies et al. 2017). In this paper, we take the min-max fairness notion, which is simple yet widely accepted. For min-max, or max-min fairness, see (Radunovic and Le Boudec 2007).

2 Warm-up Algorithm: A Natural Adaptation of Normalized Greedy

The normalized greedy algorithm (Azar and Gamzu 2011) obtains the best possible approximation ratio of $O(\ln \frac{1}{\epsilon})$ on the traditional (single-agent) submodular ranking problem, where ϵ is the minimum non-zero marginal value of functions. The algorithm picks the elements sequentially and the final element sequence gives a permutation of all elements. Each time, the algorithm chooses an element e maximizing $\sum_{f \in \mathcal{F}} w_f \cdot \frac{f(S \cup \{e\}) - f(S)}{1 - f(S)}$, where S is the elements chosen so far and $\mathcal{F}$ is the set of uncovered functions.

For the multi-agent setting, there exists a natural adaptation of this algorithm: simply view all $m \cdot k$ functions as a large single-agent submodular ranking instance and run normalized greedy. For simplicity, refer to this adaption as algorithm NG. The following shows that this algorithm has to lose a polynomial factor of k on the approximation ratio.

Theorem 1. *The approximation ratio of algorithm NG is $\Omega(\sqrt{k})$, even when the sum $\sum_j w_j^{(i)}$ of function weights for each agent i is the same.*

Proof. Consider the following SRMA instance. There are k agents and each agent has at most $\sqrt{k}$ functions; $\sqrt{k}$ is assumed to be an integer. We first describe a weighted set cover instance and explain how the instance is mapped to an instance for our problem. We are given a ground set of $k + \sqrt{k}$ elements $E = \{ e_1, e_2, \ldots, e_{k+\sqrt{k}} \}$ and a collection of singleton sets corresponding to the elements. Each agent $i \in [k-1]$ has two sets, $\{ e_i \}$ with weight $1 + \delta$ and $\{ e_k \}$ with weight $\sqrt{k} - 1 - \delta$, where $\delta > 0$ is a tiny value used for tie-breaking. The last agent k is special and she has $\sqrt{k}$ sets, $\{ e_{k+1} \}, \ldots, \{ e_{k+\sqrt{k}} \}$, each with weight 1. Note that every agent has exactly the same total weight $\sqrt{k}$.

Each set is "covered" when we choose the unique element in the singleton set. This results in an instance for our problem: for each set $\{e\}$ with weight w, we create a distinct 0-1 valued function f with an equal weight w such that $f(S) = 1$ if and only if $e \in S$. It is obvious to see that all created functions are submodular and have function values in $\{0, 1\}$. It is worth noting that e_k is special and all functions of the largest weight $(\sqrt{k} - 1)$ get covered simultaneously when e_k is selected.

Algorithm NG selects e_k in the first step. After that, the contribution of each element in $E \setminus \{ e_k \}$ in each following step is the same, which is $\sqrt{k} - 1$. Thus, the permutation π returned by algorithm NG is $(e_k, e_1, \ldots, e_{k-1}, e_{k+1}, \ldots, e_{k+\sqrt{k}})$. The objective for this ordering is at least $\Omega(k^{1.5})$ since until we choose $e_{k+\sqrt{k}/2}$, the last agent k has at least $\sqrt{k}/2$ functions unsatisfied. However, another permutation $\pi' = (e_k, e_{k+1}, \ldots, e_{k+\sqrt{k}}, e_1, \ldots, e_{k-1})$ obtains an objective value of $O(k)$. This implies that the approximation ratio of the algorithm is $\Theta(\sqrt{k})$ and completes the proof. □

One can easily show that the approximation ratio of algorithm NG is $O(k \ln(\frac{1}{\epsilon}))$ by observing that the total cover time among all agents is at most k times the maximum cover time. Due to space limit, the details are deferred to the full version of this paper.

3 Balanced Adaptive Greedy for SRMA

In this section, we build on the simple algorithm NG to give a new combinatorial algorithm that obtains a logarithmic ap-

Algorithm 1: Balanced Adaptive Greedy for SRMA

```
Input: Ground elements $[n]$; Agent set $A$; A collection of $k$ weight vectors $\{ \mathbf{w}^{(i)} \in \mathbb{R}_+^m \mid i \in [k] \}$; A collection of $m \cdot k$ submodular functions $\{ f_j^{(i)} : 2^{[n]} \to [0,1] \mid i \in [k], j \in [m] \}$.
Output: A permutation $\pi : [n] \to [n]$ of the elements.
 1: $p \leftarrow 1; q \leftarrow 1; t \leftarrow 1; B_p \leftarrow (\frac{2}{3})^p \cdot W$
 2: For each agent $a_i$, let $R_t^{(i)}$ be the set of functions of agent $a_i$ that are not satisfied earlier than time $t$.
 3: while there exists an agent $a_i$ with $w(R_t^{(i)}) > B_p$ do
 4:     $q \leftarrow 1; A_{p,q} \leftarrow \{ a_i \in A \mid w(R_t^{(i)}) > B_p \}$.
 5:     while $|A_{p,q}| \geq 0$ do
 6:         $A'_{p,q} \leftarrow A_{p,q}$.
 7:         while $|A_{p,q}| \geq \frac{3}{4} \cdot |A'_{p,q}|$ do
 8:             $F_{\pi_{t-1}}(e) \leftarrow \sum_{a_i \in A'_{p,q}} \sum_{j \in [m]} w_j^{(i)} \cdot \frac{f_j^{(i)}(\pi_{t-1} \cup \{ e \}) - f_j^{(i)}(\pi_{t-1})}{1 - f_j^{(i)}(\pi_{t-1})}$, $\forall e \in U \setminus \pi_{t-1}$.
 9:             $\pi(t) \leftarrow \arg\max_{e \in U \setminus \pi_{t-1}} F_{\pi_{t-1}}(e)$.
10:             $t \leftarrow t+1; A_{p,q} \leftarrow \{ a_i \in A \mid w(R_t^{(i)}) > B_p \}$.
11:         end while
12:         $q \leftarrow q+1; A_{p,q} \leftarrow A_{p,q-1}$.
13:     end while
14:     $p \leftarrow p+1$.
15: end while
16: return Permutation $\pi$.
```

proximation. As the proof of Theorem 1 demonstrates, the shortfall of the previous algorithm is that it does not take into account the progress of agents in the process of covering functions. After the first step, the last agent k has many more uncovered functions than other agents, but the algorithm does not prioritize the elements that can cover the functions of this agent. In other words, algorithm NG performs poorly in balancing agents' progress, which prevents the algorithm from getting rid of the polynomial dependence on k. Based on this crucial observation, we propose the balanced adaptive greedy algorithm.

We introduce some notation first. Let $R_t^{(i)}$ be the set of functions of agent a_i that are not satisfied earlier than time t. Note that $R_t^{(i)}$ includes the functions that are satisfied exactly at time t. Let $w(R_t^{(i)})$ be the total weight of functions in $R_t^{(i)}$. Recall that W is the maximum total function weight among all agents. Let $\mathcal{B} = \{ (\frac{2}{3})^1 \cdot W, (\frac{2}{3})^2 \cdot W, \ldots \}$ be a sequence, where each entry is referred to as the baseline in each iteration. Let B_p be the p-th term of $\mathcal{B}$, i.e., $B_p = (\frac{2}{3})^p \cdot W$. Roughly speaking, in the p-th iteration, we want the uncovered functions of any agent to have a total weight at most B_p. Note that the size of $\mathcal{B}$ is $\Theta(\log W)$. Given a permutation π, let $\pi(t)$ be the element selected in the t-th time slot. Let π_t be the set of elements that are scheduled before or at time t, i.e., $\pi_t = \{ \pi(1), \ldots, \pi(t) \}$.

Algorithm 1 has two loops: outer iteration (line 3-15) and inner iteration (line 7-11). Let p, q be the index of the outer and inner iterations, respectively. At the beginning of the p-th outer iteration, we remove all the satisfied functions and elements selected in the previous iterations, and we obtain a subinstance, denoted by I_p. At the end of the p-th outer iteration, Algorithm 1 ensures that the total weight of unsatisfied functions of each agent is at most $B_p = (\frac{2}{3})^p \cdot W$. Let A_p be the set of agents whose total weight of unsatisfied functions is more than B_p. At the end of (p,q)-th inner iteration, Algorithm 1 guarantees that the number of agents with the total weight of unsatisfied function larger than B_p is at most $(\frac{3}{4})^q \cdot |A_p|$. Together, this implies there are at most $\Theta(\log W)$ outer iterations because B_p decreases geometrically. Similarly, there are at most $\Theta(\log k)$ inner iterations for each outer iteration.

Naturally, the algorithm chooses the element that gives the highest total weighted functional increase over all agents in $A_{p,q}$ and their corresponding submodular functions. When computing the marginal increase, each function is divided by how much is left to be fully covered.

For technical reasons, during one iteration of line 7-11, some agents may be satisfied, i.e., their total weight of unsatisfied functions is at most B_p. Instead of dropping them immediately, we wait until $1/4$-the proportion of agents is satisfied, and then drop them together. This is the role of line 6 of Algorithm 1.

4 Analysis

Given an arbitrary instance I of SRMA, let $\mathrm{OPT}(I)$ be an optimal weighted cover time of instance I. Our goal is to show Theorem 2. Recall that $W = \max_{i \in [k]} \sum_{j=1}^{m} w_j^{(i)}$ is the maximum total weight among all agents; we assume that all weights are integers. We will first show the upper bound depending on $\log W$. We will later show how to obtain another bound depending on $\log n$ at the end of the analysis, where n is the number of elements.

Theorem 2. *Algorithm 1 obtains an approximation ratio of $O(\ln(1/\epsilon) \log(\min\{n, W\}) \log k)$, where ϵ is the minimum non-zero marginal value among all $m \cdot k$ monotone submodular functions, k is the number of agents and W is the maximum total integer weight of all functions among all agents.*

We first show a lower bound of $\mathrm{OPT}(I)$. Let π^* be an optimal permutation of the ground elements. Let $\widetilde{R}_t^{(i)}$ be the set of functions of agent a_i that are not satisfied earlier than time t in π^*. Note that $\widetilde{R}_t^{(i)}$ includes the functions that are satisfied exactly at time t. Let $w(\widetilde{R}_t^{(i)})$ be the total weight of functions in $\widetilde{R}_t^{(i)}$. Let T^* be the set of times from $t = 1$ to the first time that all agents in the optimal solution satisfy $w(\widetilde{R}_t^{(i)}) \leq \alpha \cdot W$, where $\alpha \in (0,1)$ is a constant. Let $E(T^*)$ be the corresponding elements in the optimal solution π^*, i.e., $E(T^*) = \{ \pi^*(1), \ldots, \pi^*(|T^*|) \}$.

We have the following lower bound of the optimal solution. An example of Fact 1 can be found in Fig. 1.

Fact 1. *(Lower Bound of* OPT*)* $\alpha \cdot W \cdot |T^*| \leq \mathrm{OPT}(I)$.

Recall that I_p is the subinstance of the initial instance at the beginning of the p-th iteration of Algorithm 1. Therefore, we have $\mathrm{OPT}(I_p) \leq \mathrm{OPT}(I)$ for all $p \leq \lceil \log W \rceil$. Let

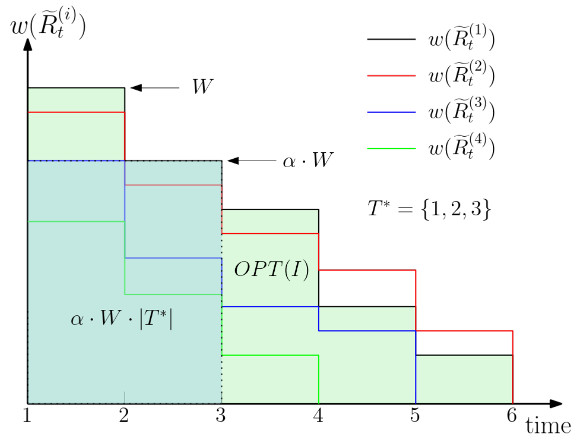


Figure 1: An illustration of the optimal value's lower bound. There are four agents in the figure. The area enclosed by the x-axis, y-axis, and curve $w(\widetilde{R}_t^{(i)})$ is the total weighted cover time of agent a_i. The optimal value will be the largest one. Then, the total area of $\alpha \cdot W \cdot |T^*|$ will be completely included in the area that is formed by the optimal solution.

$\pi^*(I_p)$ be the optimal permutation of instance I_p. Recall that $B_p = (\frac{2}{3})^p \cdot W$. Let T_p^* be the set of times from $t = 1$ to the first time such that all agents satisfy $w(\widetilde{R}_t^{(i)}) \le \frac{1}{12} \cdot B_p$ in $\pi^*(I_p)$. By Fact 1, we know that $\frac{1}{12} \cdot B_p \cdot |T_p^*| \le \mathrm{OPT}(I_p) \le \mathrm{OPT}(I)$. Let T_p be the set of times that are used by Algorithm 1 in the p-th outer iteration, i.e., $T_p = t_p, ..., t_{p+1} - 1$ where t_p is the time at the beginning of the p-th outer iteration. Let $E(T_p)$ be the set of elements that are selected by Algorithm 1 in the p-th outer iteration. Note that $|T_p|$ is the length of the p-th outer iteration. Recall that every outer iteration contains $\Theta(\log k)$ inner iterations. In the remainder of this paper, we denote the q-th inner iteration of the p-th outer iteration as (p, q)-th iteration.

We now give a simple upper bound of the solution returned by the algorithm.

Fact 2. *(Upper Bound of* ALG*)* $\mathrm{ALG}(I) \le \sum_p |T_p| \cdot B_{p-1} = \sum_p \frac{2}{3} \cdot |T_p| \cdot B_p$.

Our analysis relies on the following key lemma. Roughly speaking, Lemma 1 measures how far the algorithm is behind the optimal solution after each outer iteration.

Lemma 1. *For any* $p \le \lceil \log W \rceil$, $|T_p| \le O\left((1 + \ln(\frac{1}{\epsilon})) \cdot \log k\right) \cdot |T_p^*|$, *where* ϵ *is the minimum non-zero marginal value and* k *is the number of agents.*

It now remains to show Lemma 1. The correctness of Lemma 1 relies on the following lemma (Lemma 2). Let $T_{p,q}$ be the times that are used by Algorithm 1 in the (p, q)-th iteration. Let $E(T_{p,q})$ be the set of elements that are selected by Algorithm 1 in the (p, q)-th iteration. Note that $|T_{p,q}|$ is the length of the q-th inner iteration in p-th outer inner iteration. Let $T_{p,q}^*$ be the set of times from $t = 1$ to the first time such that all agents in $A_{p,q}$ satisfy $w(\widetilde{R}_t^{(i)}) \le \frac{1}{12} \cdot B_p$ in $\pi^*(I_p)$.

Lemma 2. *For any* $p \le \lceil \log W \rceil$ *and* $q \le \lceil \log k \rceil$, *we have* $|T_{p,q}| \le \left(1 + \ln(\frac{1}{\epsilon})\right) \cdot |T_{p,q}^*|$.

Based on Lemma 2, the proof of Lemma 1 is straightforward since $T_{p,q}^* \subseteq T_p^*$ for any $p \le \lceil \log W \rceil$ and the number of inner iterations is $\Theta(\log k)$. Thus, the major technical difficulty of our algorithm is to prove Lemma 2. Let π be the permutation returned by Algorithm 1. Note that the time set $[n]$ can be partitioned into $O(\log W \cdot \log k)$ sets. Recall that $T_{p,q}$ is the time set that are used in the (p, q)-th iteration and $A_{p,q}$ is the set of unsatisfied agents at the beginning of the (p, q)-th iteration, i.e., $A_{p,q} = \{ a_i \in A \mid w(R_t^{(i)}) > B_p \}$. For notation convenience, we define $F_{\pi_{t-1}}(e)$ as follows:

$$F_{\pi_{t-1}}(e)$$
$$:= \sum_{a_i \in A_{p,q}} \sum_{j \in [m]} w_j^{(i)} \cdot \frac{f_j^{(i)}(\pi_{t-1} \cup \{ e \}) - f_j^{(i)}(\pi_{t-1})}{1 - f_j^{(i)}(\pi_{t-1})}$$

Now we present two technical lemmas sufficient to prove Lemma 2.

Lemma 3. *For any* $p \le \lceil \log W \rceil$ *and* $q \le \lceil \log k \rceil$, *we have*

$$\sum_{t \in T_{p,q}} F_{\pi_{t-1}}(e_t) \le \left(1 + \ln(\frac{1}{\epsilon})\right) \cdot |A_{p,q}| \cdot B_{p-1}$$

Lemma 4. *For any* $p \le \lceil \log W \rceil$ *and* $q \le \lceil \log k \rceil$, *we have*

$$\sum_{t \in T_{p,q}} F_{\pi_{t-1}}(e_t) \ge \frac{2}{3} \cdot \frac{|T_{p,q}|}{|T_{p,q}^*|} \cdot |A_{p,q}| \cdot B_p$$

The proof of Lemma 3 uses a property of a monotone function, while the proof of Lemma 4 is more algorithm-specific. Recall that in each step t of an iteration (p, q), Algorithm 1 will greedily choose the element e_t among all unselected elements such that $F_{\pi_{t-1}}(e_t)$ is maximized. Then we have that the inequality $F_{\pi_{t-1}}(e_t) \ge F_{\pi_{t-1}}(e)$ holds for any element $e \in U$, and hence, for any element selected by the optimal solution. By an average argument, we can build the relationship between $F_{\pi_{t-1}}(e)$ and $|T_{p,q}^*|$, and prove Lemma 4. Lemma 2 follows from Lemma 3 and Lemma 4 simply by concatenating the two inequalities. Due to space limit, the detailed proofs are omitted in this version.

Proof of Theorem 2. Combining Fact 1, Fact 2 and Lemma 1, we have $\mathrm{ALG}(I) \le O((1 + \ln(\frac{1}{\epsilon})) \log k \log W) \cdot \mathrm{OPT}(I)$. To obtain the other bound of $\log n$ claimed in Theorem 2, we make a simple observation. Once the total weight of the uncovered functions drops below W/n for any agent, they incur at most $(W/n)n = W$ cost in total afterward as there are at most n elements to be ordered. Until this moment, $O(\log n)$ different values of p were considered. Thus, in the analysis, we only need to consider $O(\log n)$ different values of p, not $O(\log W)$. This gives the desired approximation guarantee. □

5 Inapproximability Results for SRMA and Tight Approximation for GMSC

In this section, we consider a special case of SRMA, which is called Generalized Min-sum Set Cover for Multiple Agents (GMSC), and leverage it to give a lower bound on the approximation ratio of SRMA. Due to space limit, we only present key lemmas in the following and defer the detailed problem formulation, proofs, and algorithms to the full version of this paper.

Lemma 5. *For the generalized min-sum set cover for multiple agents problem, given any constant $c < 1$, there is no $c \cdot \ln k$-approximation algorithm unless P=NP, where k is the number of agents.*

Note that our problem admits the submodular ranking problem as a special case which is $c \ln(1/\epsilon)$-hard to approximate for some constant $c > 0$ (Azar and Gamzu 2011). Thus, our problem has a natural lower bound $\Omega(\ln(1/\epsilon))$. Hence, by combining Lemma 5 and the lower bound of classical submodular ranking, one can easily get a $\Omega(\ln(1/\epsilon) + \log k)$ lower bound for SRMA .

Theorem 3. *The problem of min-max submodular ranking for multiple agents cannot be approximated within a factor of $c \cdot (\ln(1/\epsilon) + \log k)$ for some constant $c > 0$ unless P=NP, where ϵ is the minimum non-zero marginal value and k is the number of agents.*

Now we show a tight approximation for GMSC. The algorithm is an LP-based randomized algorithm. Let $x_{e,t}$ be the variable that indicates whether the element e is scheduled at time t. To reduce the integrality gap, we introduce an exponential number of constraints. We show that the ellipsoid algorithm can solve such an LP. We partition the whole timeline into multiple phases based on the optimal fractional solution. In each phase, we round variable $x_{e,t}$ to obtain an integer solution. The main difference from (Bansal, Gupta, and Krishnaswamy 2010), we repeat the rounding algorithm $\Theta(\log k)$ time and interleave the resulting solutions over time to avoid the bad events for all k agents simultaneously.

Theorem 4. *There is a randomized algorithm that achieves $\Theta(\log k)$-approximation for generalized min-sum set cover for multiple agents, where k is the number of agents.*

6 Experiments

This section investigates the empirical performance of our algorithms. We seek to show that the theory is predictive of practice on real data. We give experimental results for the min-max optimal decision tree over multiple agents. At the high level, there is a set of objects where each object is associated with an attribute vector. We can view the data as a table with the objects as rows and attributes as columns. Each agent i has a target object o_i sampled from a different distribution over the objects. Our goal is to order the columns to find all agents' targets as soon as possible. When column c is chosen, for each agent i, rows that have a different value from o_i in column c are discarded. The target object o_i is identified when all the other objects are inconsistent with o_i in one of the columns probed.

Data Preparation. In the experiments, three public data sets are considered: MFCC data set[3], PPPTS data set[4], and CTG data set[5]. These are the data sets in the field of life science. Each one is in table format and the entries are real-valued. The sizes of the three data sets are 7195×22, 45730×9, and 2126×23 respectively, where $h \times n$ implies h rows and n columns.

The instances are constructed as follows. We first discretize real-valued entries, so entries in each column can have at most ten different values. This is because the objects' vectors could have small variations, and we seek to make them more standardized. Let the ground set of elements U be all the columns for a table, and view each row j as an object. Create a submodular function f_j for each object j: For a subset $S \subseteq U$, $f_j(S)$ represents the number of rows that have different attributes from row j after checking the columns in S. If $f_j(S) = f(U)$, the object in row j can be completely distinguished from other objects by checking columns in S. We then normalize functions to have a range of $[0, 1]$. Note that the functions are created essentially in the same way they are in the reduction of the optimal decision tree to submodular ranking, as discussed in Section 1. All agents have the same table, that is, the same objects, functions, and columns. We construct a different weight vector for each agent, where each entry has a value sampled uniformly at random from $[1, 100]$. In the following, we use K and M to denote the number of agents and the number of functions, respectively.

Baseline Algorithms and Parameter Setting. We refer to our main Algorithm 1 as balanced adaptive greedy (BAG). We also refer to the naive adaptation of the algorithm proposed by (Azar and Gamzu 2011) as normalized greedy (NG). For full description of NG, see Section 2. The algorithms are compared to two natural baseline algorithms. One is called the random (R) algorithm, which directly outputs a random permutation of elements. The other is the greedy (G) algorithm, which selects the element that maximizes the total increment of all $K \cdot M$ functions each time. Notice that Algorithm 1, the decreasing ratio of the sequence in algorithm BAG is set to be $2/3$, but in practice, this decreasing ratio is flexible. In the experiments, we test the performance of algorithm BAG with decreasing ratios in $[0, 0.05, 0.1, \ldots, 0.95, 1]$ and pick the best decreasing ratio.

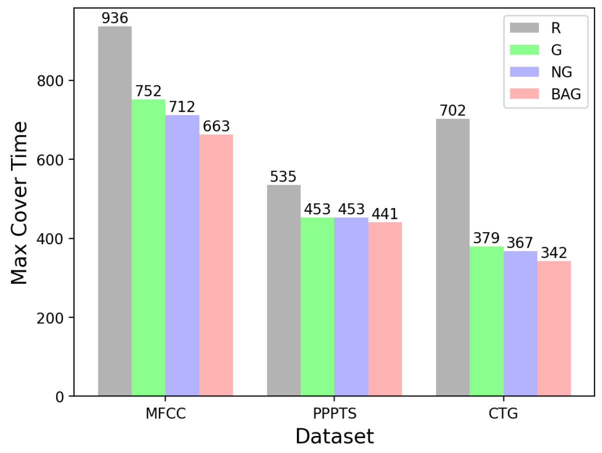
We conduct the experiments on a machine running Ubuntu 18.04 with an i7-7800X CPU and 48 GB memory. We investigate the performance of algorithms on different data sets under different values of K and M. The results are averaged over four runs. The data sets give the same trend. Thus, we first show the algorithms' performance with $K = M = 10$ on the three data sets and only present the results on the MFCC data set when K and M vary in Fig. 2. The results on the other two data sets appear in the full version of this paper.

Empirical Discussion. From the figures, we see that the proposed balanced adaptive greedy algorithm always obtains the best performance for all datasets and all values of K and M. Moreover, Fig. 2(b) shows that as K increases, the objective value of each algorithm generally increases, im-

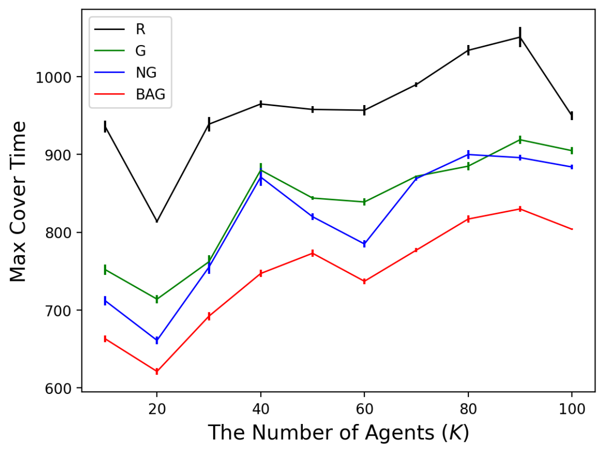
[3]https://archive.ics.uci.edu/ml/datasets/Anuran+Calls+\%28MFCCs\%29

[4]https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure\#

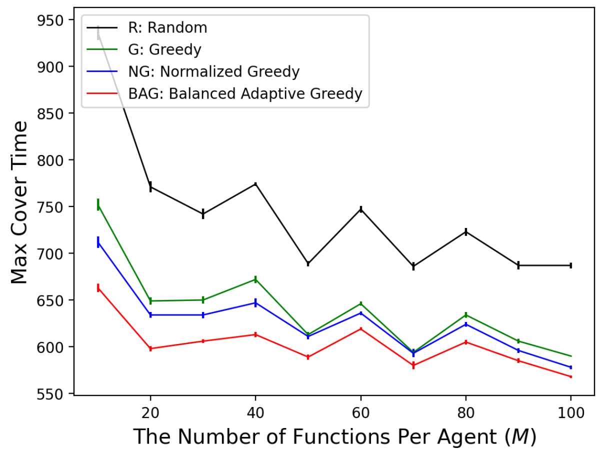
[5]https://archive.ics.uci.edu/ml/datasets/Cardiotocography



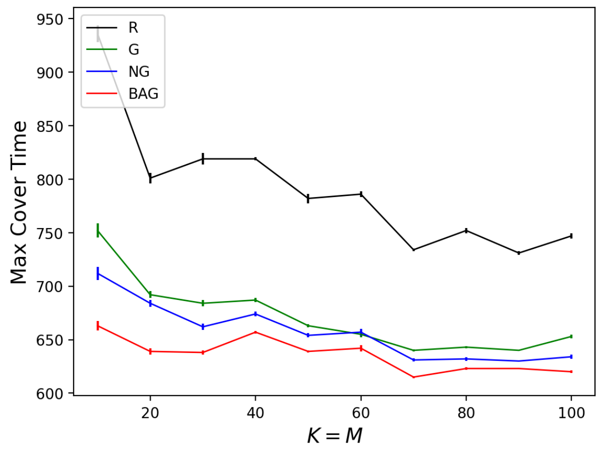
(a)



(b)



(c)



(d)

Figure 2: Fig. 2(a) shows the results on different datasets when both the number of agents K and the number of functions per agent M are 10. Others show the performance of algorithms on the MFCC dataset when K and M vary. In Fig. 2(b), we fix $M = 10$ and increase K, while in Fig. 2(c), we fix $K = 10$ and increase M. Finally, Fig. 2(d) shows the algorithms' performance when K and M are set to be the same value and increase together.

plying that the instance becomes harder. In these harder instances, algorithm BAG has a more significant improvement over other methods. Conversely, Fig. 2(c) indicates that we get easier instances as M increases because all the curves generally give downward trends. In this case, although the benefit of our balancing strategy becomes smaller, algorithm BAG still obtains the best performance.

7 Conclusion

The paper is the first to study the submodular ranking problem in the presence of multiple agents. The objective is to minimize the maximum cover time of all agents, i.e., optimizing the worst-case fairness over the agents. This problem generalizes to designing optimal decision trees over multiple agents and also captures other practical applications. By observing the shortfall of the existing techniques, we introduce a new algorithm, balanced adaptive greedy. Theoretically, the algorithm is shown to have strong approximation guarantees. The paper shows empirically that the theory is predictive of experimental performances. Balanced adaptive greedy is shown to outperform strong baselines in the experiments, including the most natural greedy strategies.

The paper gives a tight approximation algorithm for generalized min-sum set cover on multiple agents, which is a special case of our model. The upper bound shown in this paper matches the lower bound introduced. The tight approximation for the general case is left as an interesting open problem. Beyond the generalized min-sum set cover problem, another special case of our problem is also interesting in which the monotone submodular functions of each agent are the same. Observing the special case above also captures the problem of Optimal Decision Tree with Multiple Probability Distribution. Thus, improving the approximation for this particular case will be interesting.

Acknowledgments

Chenyang Xu was supported in part by Science and Technology Innovation 2030 –"The Next Generation of Artificial Intelligence" Major Project No.2018AAA0100900. Qingyun Chen and Sungjin Im were supported in part by NSF CCF-1844939 and CCF-2121745. Benjamin Moseley was supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909. We thank the anonymous reviewers for their insightful comments and suggestions.

References

Abernethy, J. D.; Awasthi, P.; Kleindessner, M.; Morgenstern, J.; Russell, C.; and Zhang, J. 2022. Active Sampling for Min-Max Fairness. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, 53–65. PMLR.

Agarwal, A.; Assadi, S.; and Khanna, S. 2019. Stochastic submodular cover with limited adaptivity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 323–342. SIAM.

Azar, Y.; and Gamzu, I. 2011. Ranking with Submodular Valuations. In *SODA*, 1070–1079. SIAM.

Azar, Y.; Gamzu, I.; and Yin, X. 2009. Multiple intents reranking. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 669–678.

Bach, F.; et al. 2013. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning*, 6(2-3): 145–373.

Bansal, N.; Batra, J.; Farhadi, M.; and Tetali, P. 2021. Improved approximations for min sum vertex cover and generalized min sum set cover. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 998–1005. SIAM.

Bansal, N.; Gupta, A.; and Krishnaswamy, R. 2010. A Constant Factor Approximation Algorithm for Generalized Min-Sum Set Cover. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, 1539–1545. SIAM.

Barocas, S.; Hardt, M.; and Narayanan, A. 2017. Fairness in machine learning. *Nips tutorial*, 1: 2.

Chakaravarthy, V. T.; Pandit, V.; Roy, S.; Awasthi, P.; and Mohania, M. 2007. Decision trees for entity identification: Approximation algorithms and hardness results. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 53–62.

Chouldechova, A.; and Roth, A. 2020. A snapshot of the frontiers of fairness in machine learning. *Communications of the ACM*, 63(5): 82–89.

Corbett-Davies, S.; Pierson, E.; Feller, A.; Goel, S.; and Huq, A. 2017. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd acm sigkdd international conference on knowledge discovery and data mining*, 797–806.

Dasgupta, S. 2004. Analysis of a greedy active learning strategy. *Advances in neural information processing systems*, 17.

Feige, U.; Lovász, L.; and Tetali, P. 2004. Approximating min sum set cover. *Algorithmica*, 40(4): 219–234.

Golovin, D.; Krause, A.; and Ray, D. 2010. Near-optimal bayesian active learning with noisy observations. *Advances in Neural Information Processing Systems*, 23.

Gupta, A.; Nagarajan, V.; and Ravi, R. 2017. Approximation algorithms for optimal decision trees and adaptive TSP problems. *Mathematics of Operations Research*, 42(3): 876–896.

Halabi, M. E.; Mitrovic, S.; Norouzi-Fard, A.; Tardos, J.; and Tarnawski, J. 2020. Fairness in Streaming Submodular Maximization: Algorithms and Hardness. In *NeurIPS*.

Im, S.; Nagarajan, V.; and van der Zwaan, R. 2016. Minimum Latency Submodular Cover. *ACM Trans. Algorithms*, 13(1): 13:1–13:28.

Jia, S.; Navidi, F.; Ravi, R.; et al. 2019. Optimal decision tree with noisy outcomes. *Advances in neural information processing systems*, 32.

Kleinberg, J.; Mullainathan, S.; and Raghavan, M. 2016. Inherent trade-offs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*.

Mehrabi, N.; Morstatter, F.; Saxena, N.; Lerman, K.; and Galstyan, A. 2021. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6): 1–35.

Radunovic, B.; and Le Boudec, J.-Y. 2007. A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Transactions on networking*, 15(5): 1073–1083.

Williamson, D. P.; and Shmoys, D. B. 2011. *The design of approximation algorithms*. Cambridge university press.