A Generalization of Submodular Cover via the Diminishing Return Property on the Integer Lattice

Authored by:

Yuichi Yoshida
Tasuku Soma

Abstract

We consider a generalization of the submodular cover problem based on the concept of diminishing return property on the integer lattice. We are motivated by real scenarios in machine learning that cannot be captured by (traditional) submodular set functions. We show that the generalized submodular cover problem can be applied to various problems and devise a bicriteria approximation algorithm. Our algorithm is guaranteed to output a log-factor approximate solution that satisfies the constraints with the desired accuracy. The running time of our algorithm is roughly $O(nlog (nr) log{r})$, where n is the size of the ground set and r is the maximum value of a coordinate. The dependency on r is exponentially better than the naive reduction algorithms. Several experiments on real and artificial datasets demonstrate that the solution quality of our algorithm is comparable to naive algorithms, while the running time is several orders of magnitude faster.

1 Paper Body

A function f : 2S ? R+ is called submodular if f (X) + f (Y) ? f (X ? Y) + f (X ? Y) for all X, Y ? S, where S is a finite ground set. An equivalent and more intuitive definition is by the diminishing return property: f (X ? {s}) ? f (X) ? f (Y ? {s}) ? f (Y) for all X ? Y and s ? S Y . In the last decade, the optimization of a submodular function has attracted particular interest in the machine learning community. One reason of this is that many real-world models naturally admit the diminishing return property. For example, document summarization [12, 13], influence maximization in viral marketing [7], and sensor placement [10] can be described with the concept of submodularity, and efficient algorithms have been devised by exploiting submodularity (for further details, refer to [8]). A variety of proposed models in machine learning [4, 13, 18] boil down to the submodular cover problem [21]; for given monotone and nonnegative

submodular functions f, c : 2S ? R+ , and ? ¿ 0, we are to minimize c(X) subject to f (X) ? ?. (1) Intuitively, c(X) and f (X) represent the cost and the quality of a solution, respectively. The objective of this problem is to find X of minimum cost with the worst quality guarantee ?. Although this problem is NP-hard since it generalizes the set cover problem, a simple greedy algorithm achieves tight log-factor approximation and it practically performs very well. The aforementioned submodular models are based on the submodularity of a set function, a function defined on 2S . However, we often encounter problems that cannot be captured by a set function. Let us give two examples: Sensor Placement: Let us consider the following sensor placement scenario. Suppose that we have several types of sensors with various energy levels. We assume a simple trade-off between 1

information gain and cost. Sensors of a high energy level can collect a considerable amount of information, but we have to pay a high cost for placing them. Sensors of a low energy level can be placed at a low cost, but they can only gather limited information. In this scenario, we want to decide which type of sensor should be placed at each spot, rather than just deciding whether to place a sensor or not. Such a scenario is beyond the existing models based on submodular set functions. Optimal Budget Allocation: A similar situation also arises in the optimal budget allocation problem [2]. In this problem, we want to allocate budget among ad sources so that (at least) a certain number of customers is influenced while minimizing the total budget. Again, we have to decide how much budget should be set aside for each ad source, and hence set functions cannot capture the problem. We note that a function f : 2S ? R+ can be seen as a function defined on a Boolean hypercube {0, 1}S . Then, the above real scenarios prompt us to generalize the submodularity and the diminishing return property to functions defined on the integer lattice ZS+ . The most natural generalization of the diminishing return property to a function f : ZS+ ? R+ is the following inequality: f (x + ?s) ? f (x) ? f (y + ?s) ? f (y) (2) for x ? y and s ? S, where ?s is the s-th unit vector. If f satisfies (2), then f also satisfies the following lattice submodular inequality: f (x) + f (y) ? f (x ? y) + f (x ? y) (3) for all x, y ? ZS+ , where ? and ? are the coordinate-wise max and min operations, respectively. While the submodularity and the diminishing return property are equivalent for set functions, this is not the case for functions over the integer lattice; the diminishing return property (2) is stronger than the lattice submodular inequality (3). We say that f is lattice submodular if f satisfies (3), and if f further satisfies (2) we say that f is diminishing return submodular (DR-submodular for short). One might feel that the DR-submodularity (2) is too restrictive. However, considering the fact that the diminishing return is more crucial in applications, we may regard the DR-submodularity (2) as the most natural generalization of the submodularity, at least for applications mentioned so far [17, 6]. For example, under a natural condition, the objective function in the optimal budget allocation satisfies (2) [17]. The DR-submodularity was also considered in the context of submodular welfare [6]. In this paper, we consider the following generalization of the submodular cover problem for set functions: Given a monotone DR-submodular

function f : ZS+ ? R+ , a subadditive function c : ZS+ ? R+ , ? ¿ 0, and r ? Z+ , we are to minimize c(x)

subject to f (x) ? ?,

0 ? x ? r1,

(4)

ZS+ .

where we say that c is subadditive if c(x + y) ? c(x) + c(y) for all x, y ? We call problem (4) the DR-submodular cover problem. This problem encompasses problems that boil down to the submodular cover problem for set functions and their generalizations to the integer lattice. Furthermore, the cost function c is generalized to a subadditive function. In particular, we note that two examples given above can be rephrased using this problem (see Section 4 for details). If c is also monotone DR-submodular, one can reduce the problem (4) to the set version (1) (for technical details, see Section 3.1). The problem of this naive reduction is that it only yields a pseudo-polynomial time algorithm; the running time depends on r rather than log r. Since r can be huge in many practical settings (e.g., the maximum energy level of a sensor), even linear dependence on r could make an algorithm impractical. Furthermore, for a general subadditive function c, this naive reduction does not work. 1.1

Our Contribution

For the problem (4), we devise a bicriteria approximation algorithm based on the decreasing threshold technique of [3]. More precisely, our algorithm takes the additional parameters 0 ¡ , ? ¡ 1. The

S output x ? Z+ of our algorithm is guaranteed to satisfy that c(x) is at most (1 + 3)? 1 + log ?d times the optimum and f (x) ? (1 ? ?)?, where ? is the curvature of c (see Section 3 for the definition), d = maxs f (?s) is the maximum value of f over all standard unit vectors, and ? is the minimum value of the positive increments of f in the feasible region. 2

Running Time (dependency on r): An important feature of our algorithm is that the running time depends on the bit length of r only polynomially whereas the naive reduction algorithms depend on it exponentially as mentioned above. More precisely, the running time of our algorithm is max O(n log nrc ?cmin log r), which is polynomial in the input size, whereas the naive algorithm is only psuedo-polynomial time algorithm. In fact, our experiments using real and synthetic datasets show that our algorithm is considerably faster than naive algorithms. Furthermore, in terms of the objective value (that is, the cost of the output), our algorithm also exhibits comparable performance. Approximation Guarantee: Our approximation guarantee on the cost is almost tight. Note that the DR submodular cover problem (4) includes the set cover problem, in which we are given a collection of sets, and we want to find a minimum number of sets that covers all the elements. In our context, S corresponds to the collection of sets, the cost c is the number of chosen sets, and f is the number of covered elements. It is known that we cannot obtain an o(log m)-approximation unless P 6= NP, where m is the number of elements [16]. However, since for the set cover problem we have ? = 1, d = O(m), and ? = 1, our approximation guarantee is O(log m). 1.2

Related Work

Our result can be compared with several results in the literature for the submodular cover problem for set functions. It is shown by Wolsey [21] that if c(X) = —X—, a simple greedy algorithm yields (1 + log ?d)-approximation, which coincides with our approximation ratio except for the (1 + 3) factor. Note that ? = 1 when c(X) = —X—, or more generally, when c is modular. Recently, Wan et al. [20] discussed a slightly different setting, in which c is also submodular and both f and c are integer valued. They proved that the greedy algorithm achieves ?H(d)-approximation, where H(d) = 1+1/2+? ? ?+1/d is the d-th harmonic number. Again, their ratio asymptotically coincides with our approximation ratio (Note that ? ? 1 when f is integer valued). Another common submodular-based model in machine learning is in the form of the submodular maximization problem: Given a monotone submodular set function f : {0, 1}S ? R+ and a feasible S set P ? [0, 1] (e.g., a matroid polytope or a knapsack polytope), we want to maximize f (x) subject to x ? P ? {0, 1}S . Such models can be widely found in various tasks as already described. We note that the submodular cover problem and the submodular maximization problem are somewhat dual to each other. Indeed, Iyer and Bilmes [5] showed that a bicriteria algorithm of one of these problems yields a bicriteria algorithm for the other. Being parallel to our setting, generalizing the submodular maximization problem to the integer lattice ZS+ is a natural question. In this direction, Soma et al. [17] considered the maximization of lattice submodular functions (not necessarily being DR-submodular) and devised a constant-factor approximation pseudo-polynomial time algorithm. We note that our result is not implied by [17] via the duality of [5]. In fact, such reduction only yields a pseudo-polynomial time algorithm. 1.3

Organization of This Paper

The rest of this paper is organized as follows: Section 2 sets the mathematical basics of submodular functions over the integer lattice. Section 3 describes our algorithm and the statement of our main theorem. In Section 4, we show various experimental results using real and artificial datasets. Section 5 sketches the proof of the main theorem. Finally, we conclude the paper in Section 6.

2

Preliminaries

Let S be a finite set. For each s ? S, we denote the s-th unit vector by ?s ; that is, ?s (t) = 1 if t = s, otherwise ?s (t) = 0. A function f : ZS ? R is said to be lattice submodular if f (x) + f (y) ? f (x ? y) + f (x ? y) for all x, y ? ZS . A function f is monotone if f (x) ? f (y) for all x, y ? ZS with x ? y. For x, y ? ZS and a function f : ZS ? R, we denote f (y — x) := f (y + x) ? f (x). A function f is diminishing return submodular (or DR-submodular) if f (x + ?s) ? f (x) ? f (y + ?s) ? f (y) for each x ? y ? ZS and s ? S. For a DR-submodular function f , one can immediately check that f (k?s — x) ? f (k?s — y) for arbitrary x ? y, s ? S, and k ? Z+ . A function f is subadditive if f (x + y) ? f (x) + f (y) for x, y ? ZS . For each x ? ZS+ , we define {x} to be the multiset in which each s ? S is contained x(s) times. 3

In [17], a lattice submodular function f : ZS ? R is said to have the dimin-

ishing return property if f is coordinate-wise concave: f (x + 2?s) ? f (x + ?s) ? f (x + ?s) ? f (x) for each x ? ZS and s ? S. We note that our definition is consistent with [17]. Formally, we have the following lemma, whose proof can be found in Appendix. Lemma 2.1. A function f : ZS ? R is DR-submodular if and only if f is lattice submodular and coordinate-wise concave. The following is fundamental for a monotone DR-submodular function. A proof is placed in Appendix due to the limitation of space. P Lemma 2.2. For a monotone DR-submodular function f , f (x) ? f (y) ? s?{x} f (?s — y) for arbitrary x, y ? ZS .

3

Algorithm for the DR-submodular Cover

Recall the DR-submodular cover problem (4). Let f : ZS+ ? R+ be a monotone DR-submodular function and let c : ZS+ ? R+ be a subadditive cost function. The objective is to minimize c(x) subject to f (x) ? ? and 0 ? x ? r1, where ? ¿ 0 and r ? Z+ are the given constants. Without loss of generality, we can assume that max{f (x) : 0 ? x ? r1} = ? (otherwise, we can consider fb(x) := min{f (x), ?} instead of f). Furthermore, we can assume c(x) ¿ 0 for any x ? ZS+ . A pseudocode description of our algorithm is presented in Algorithm 1. The algorithm can be viewed as a modified version of the greedy algorithm and works as follows: We start with the initial solution x = 0 and increase each coordinate of x gradually. To determine the amount of increments, the algorithm maintains a threshold ? that is initialized to be sufficiently large enough. For each s ? S, the algorithm finds the largest integer step size 0 ¡ k ? r ? x(s) such that the marginal cost-gain (k?s —x) ratio f kc(? is above the threshold ?. If such k exists, the algorithm updates x to x + k?s . After s) repeating this for each s ? S, the algorithm decreases the threshold ? by a factor of (1 ?). If x becomes feasible, the algorithm returns the current x. Even if x does not become feasible, the final x satisfies f (x) ? (1 ? ?)? if we iterate until ? gets sufficiently small. Algorithm 1 Decreasing Threshold for the DR-Submodular Cover Problem Input: f : ZS+ ? R+ , c : ZS+ ? R+ , r ? N, ? ¿ 0, ¿ 0, ? ¿ 0. Output: 0 ? x ? r1 such that f (x) ? ?. 1: x ? 0, d ? max f (?s), cmin ? min c(?s), cmax ? max c(?s) s?S

s?S

s?S

d ? 2: for (? = cmin ; ? ? ncmax r d; ? ? ?(1 ?)) do 3: for all s ? S do 4: Find maximum integer 0 ¡ k ? r ? x(s) such that 5: If such k exists then x ? x + k?s . 6: If f (x) ? ? then break the outer for loop. 7: return x

f (k?s —x) kc(?s)

? ? with binary search.

Before we claim the theorem, we need to define several parameters on f and c. Let ? := min{f (?s — x) : s ? S, x ? ZS+ , f (?s — x) ¿ 0} and d := maxs f (?s). Let cmax := maxs c(?s) and cmin := mins c(?s). Define the curvature of c to be P s?{x? } c(?s) ? := ? min . (5) x :optimal solution c(x?) Definition 3.1. For ? ? 1 and 0 ¡ ? ¡ 1, a vector x ? ZS+ is a (?, ?)-bicriteria approximate solution if c(x) ? ? ? c(x?), f (x) ? (1 ? ?)?, and 0 ? x ? r1. Our main theorem is described below. We sketch the proof in Section 5.

Theorem 3.2. Algorithm 1 outputs a (1 + 3)? 1 + log ?d , ? -bicriteria approximate solution

max in O n log nrc ?cmin log r time. 4

3.1

Discussion

Integer-valued Case. Let us make a simple remark on the case that f is integer valued. Without loss of generality, we can assume ? ? Z+ . Then, Algorithm 1 always returns a feasible solution for any 0 ¡ ? ¡ 1/?. Therefore, our algorithm can be easily modified to an approximation algorithm if f is integer valued. Definition of Curvature. Several authors [5, 19] use a different notion of curvature called the total curvature, whose natural extension for a function over the integer lattice is as follows: The —r1??s) total curvature ? of c : ZS+ ? R+ is defined as ? := 1 ? mins?S c(?sc(? . Note that ? = 0 s) if c is modular, while ? = 1 if c is modular. For example, Iyer and Bilmes [5] devised a bicriteria approximation algorithm whose approximation guarantee is roughly O((1 ? ?)?1 log ?d). Let us investigate the relation between ? and ? for DR-submodular functions. One can show that 1 ? ? ? ? ? (1 ? ?)?1 (see Lemma E.1 in Appendix), which means that our bound in terms of ? is tighter than one in terms of (1 ? ?)?1 . Comparison to Naive Reduction Algorithm. If c is also a monotone DR-submodular function, one can reduce (4) to the set version (1) as follows. For each s ? S, create r copies of s and let ? ? S, ? define x ? ? ZS+ be the integral vector such that x ? (s) S? be the set of these copies. For X X X ? Then, f?(X) ? := f (x ?) is submodular. Similarly, is the number of copies of s contained in X. X ? := c(x ?) is also submodular if c is a DR-submodular function. Therefore we may apply a c?(X) X standard greedy algorithm of [20, 21] to the reduced problem and this is exactly what Greedy does in our experiment (see Section 4). However, this straightforward reduction only yields a pseudo? = nr; even if the original algorithm was linear, the resulting polynomial time algorithm since —S— algorithm would require O(nr) time. Indeed this difference is not negligible since r can be quite large in practical applications, as illustrated by our experimental evaluation. Lazy Evaluation. We finally note that we can combine the lazy evaluation technique [11, 14], which significantly reduces runtime in practice, with our algorithm. Specifically, we first push all (?s) the elements in S to a max-based priority queue. Here, the key of an element s ? S is fc(? . Then s) the inner loop of Algorithm 1 is modified as follows: Instead of checking all the elements in S, we pop elements whose keys are at least ?. For each popped element s ? S, we find k such that (k?s —x) 0 ¡ k ? r ? x(s) with f kc(? ? ? with binary search. If there is such k, we update x with s) x + k?s . Finally, we push s again with the key

f (?s —x) c(?s)

if x(s) ¡ r.

The correctness of this technique is obvious because of the DR-submodularity of f . In particular, (?s —x) , where x is the current vector. the key of each element s ? S in the queue is always at least f c(? s) Hence, we never miss s ? S with

4 4.1
f (k?s —x) kc(?s)
? ?.

Experiments Experimental Setting

We conducted experiments on a Linux server with an Intel Xeon E5-2690 (2.90 GHz) processor and 256 GB of main memory. The experiments required, at most, 4 GB of memory. All the algorithms were implemented in C++ and compiled with g++ 4.6.3. In our experiments, the cost function c : ZS+ ? R+ is always chosen as c(x) = kxk1 := P S s?S x(s). Let f : Z+ ? R+ be a submodular function and ? be the worst quality guarantee. We implemented the following four methods: ? Decreasing-threshold is our method with the lazy evaluation technique. We chose ? = 0.01 as stated otherwise. ? Greedy is a method in which, starting from x = 0, we iteratively increment x(s) for s ? S that maximizes f (x + ?s) ? f (x) until we get f (x) ? ?. We also implemented the lazy evaluation technique [11]. 5

? Degree is a method in which we assign x(s) a value proportional to the marginal f (?s) ? f (0), where kxk1 is determined by binary search so that f (x) ? ?. Precisely speaking, x(s) is approximately proportional to the marginal since x(s) must be an integer. ? Uniform is a method that returns k1 for minimum k ? Z+ such that f (k1) ? ?. We use the following real-world and synthetic datasets to confirm the accuracy and efficiency of our method against other methods. We set r = 100, 000 for both problems. Sensor placement. We used a dataset acquired by running simulations on a 129-vertex sensor network used in Battle of the Water Sensor Networks (BWSN) [15]. We used the ?bwsn-utilities? [1] program to simulate 3000 random injection events to this network for a duration of 96 hours. Let S and E be the set of the 129 sensors in the network and the set of the 3000 events, respectively. For each sensor s ? S and event e ? E, a value z(s, e) is provided, which denotes the time, in minutes, the pollution has reached s after the injection time.1 We define a function f : ZS+ ? R+ as follows: Let x ? ZS+ be a vector, where we regard x(s) as the energy level of the sensor s. Suppose that when the pollution reaches a sensor s, the probability x(s) that we can detect it is 1 ? (1 ? p) , where p = 0.0001. In other words, by spending unit energy, we obtain an extra chance of detecting the pollution with probability p. For each event e ? E, let se be the first sensor where the pollution is detected in that injection event. Note that se is a random variable. Let z? = max z(s, e). Then, we define f as follows: e?E,s?S

f (x) = E E[z? ? z(se , e)], e?E se

where z(se , e) is defined as z? when there is no sensor that managed to detect the pollution. Intuitively speaking, E[z? ? z(se , e)] expresses how much time we managed to save in the event e se

on average. Then, we take the average over all the events. A similar function was also used in [11] to measure the performance of a sensor allocation although they only considered the case p = 1. This corresponds to the case that by spending unit energy at a sensor s, we can always detect the pollution that has reached s. We note that f (x) is DR-submodular (see Lemma F.1 for the proof). Budget allocation problem. In order to observe the behavior of our algorithm

for large-scale instances, we created a synthetic instance of the budget allocation problem [2, 17] as follows: The instance can be represented as a bipartite graph (S, T ; E), where S is a set of 5,000 vertices and T is a set of 50,000 vertices. We regard a vertex in S as an ad source, and a vertex in T as a person. Then, we fix the degrees of vertices in S so that their distribution obeys the power law of ? := 2.5; that is, the fraction of ad sources with out-degree d is proportional to d?? . For a vertex s ? S of the supposed degree d, we choose d vertices in T uniformly at random and connect them to s with edges. We define a function f : ZS+ ? R+ as

X Y x(s) f (x) = 1? (1 ? p) , (6) t?T

s??(t)

where ?(t) is the set of vertices connected to t and p = 0.0001. Here, we suppose that, by investing a unit cost to an ad source s ? S, we have an extra chance of influencing a person t ? T with s ? ?(t) with probability p. Then, f (x) can be seen as the expected number of people influenced by ad sources. We note that f is known to be a monotone DR-submodular function [17]. 4.2

Experimental Results

Figure 1 illustrates the obtained objective value kxk1 for various choices of the worst quality guarantee ? on each dataset. We chose = 0.01 in Decreasing threshold. We can observe that Decreasing threshold attains almost the same objective value as Greedy, and it outperforms Degree and Uniform. Figure 2 illustrates the runtime for various choices of the worst quality guarantee ? on each dataset. We chose = 0.01 in Decreasing threshold. We can observe that the runtime growth of Decreasing threshold is significantly slower than that of Greedy. 1

Although three other values are provided, they showed similar empirical results and we omit them.

6
4
30000
2
10
15000
1
10
0
10000
10
5000
10
-1
0 0
500
1000
1500 ?
2000

2500
3000
1
10
0
10
-1
10
-2
10
-3
10
500
1000
1500 ?
2000
2500
0
3000
(a) Sensor placement (BWSN)
2.5 1e8
0
500
4
2
10
2500
3000
1.0 0.1 0.01 0.001 0.0001 Greedy
3
10
2
time (s)
2000
4
3
1.0
1500 ?
10
Greedy Decreasing threshold Degree Uniform
10
1.5
1000
(a) Relative cost increase
10
Greedy Decreasing threshold Degree Uniform

2.0
0
1.0 0.1 0.01 0.001 0.0001
2
10
-2
10
(a) Sensor placement (BWSN)
Objective value
Relative increase of the objective value
10
time (s)
Objective value
20000
10
Uniform Decreasing threshold Degree Greedy
3
time (s)
25000
3
10
Uniform Decreasing threshold Degree Greedy
1
10
10
1
10
0
10 0.5
0
10
-1
10
0.0 0
-2
5000
10000 ?
15000
10
20000
-1
0
5000
10000 ?
15000
20000

10
0
500
1000
1500 ?
2000
2500
(b) Budget allocation (synthetic)
(b) Budget allocation (synthetic)
(b) Runtime
Figure 1: Objective values
Figure 2: Runtime
Figure 3: Effect of
3000

Figures 3(a) and 3(b) show the relative increase of the objective value and the runtime, respectively, of our method against Greedy on the BWSN dataset. We can observe that the relative increase of the objective value gets smaller as ? increases. This phenomenon can be well explained by considering the extreme case that ? = max f (r1). In this case, we need to choose x = r1 anyway in order to achieve the worst quality guarantee, and the order of increasing coordinates of x does not matter. Also, we can see that the empirical runtime grows as a function of 1 , which matches our theoretical bound.

5

Proof of Theorem 3.2

In this section, we outline the proof of the main theorem. Proofs of some minor claims can be found in Appendix. First, we introduce a notation. Let us assume that x is updated L times in the algorithm. Let xi be the variable x after the i-th update (i = 0, . . . , L). Note that x0 = 0 and xL is the final output of the algorithm. Let si ? S and ki ? Z+ be the pair used in the i-th update for i = 1, . . . , L; that is, k c(?si) for i = 1, . . . , L. Let xi = xi?1 + ki ?si for i = 1, . . . , L. Let ?0 := 0 and ?i := f (kii?s —x i?1) i

? ?0 := 0 and ? ?i := ?i?1 for i = 1, . . . , L, where ?i is the threshold value on the i-thP update. Note that ? ?i?1 ? ? ?i for i = 1, . . . , L. Let x? be an optimal solution such that ? ? c(x?) = s?{x? } c(?s). We regard that in the i-th update, the elements of {x? } are charged by the value of ?i (f (?s — xi?1) ? f (?s — xi)). Then, the total charge on {x? } is defined as T (x, f) :=

L X X

?i (f (?s — xi?1) ? f (?s — xi)).

s?{x? } i=1

Claim 5.1. Let us fix 1 ? i ? L arbitrary and let ? be the threshold value on the i-th update. Then, f (?s — xi?1) ? f (ki ?si — xi?1) ? ? and ? (s ? S). ki c(?si) c(?s) 1? Eliminating ? from the inequalities in Claim 5.1, we obtain ki c(?si) 1 c(?s) ? (i = 1, . . . , L, f (ki ?si — xi?1) 1 ? f (?s — xi?1) 7

s ? S)

(7)

Furthermore, we have ?i ? ? ?i ? Claim 5.2. c(x) ?

1 1? ?i

for i = 1, . . . , L.

1 1? T (x, f). ?

Claim 5.3. For each s ? {x }, the total charge on s is at most

1 1? (1

+ log(d/?))c(?s).

Proof. Let us fix s ? {x? } and let l be the minimum i such that f (?s — xi) = 0. By (7), we have ki c(?si) 1 c(?s) ?i = ? ? . (i = 1, . . . , l) f (ki ?si — xi?1) 1 ? f (?s — xi?1) Then, we have L l?1 X X ?i (f (?s — xi?1) ? f (?s — xi)) = ?i (f (?s — xi?1) ? f (?s — xi)) + ?l f (?s — xl?1) i=1

i=1

l?1 X (f (?s — xi?1) ? f (?s — xi)) f (?s — xl?1) 1 c(?s) + ? 1? f (?s — xi?1) f (?s — xl?1) i=1

?

l?1

X 1 f (?s — xi) c(?s) 1 + 1? 1? f (?s — xi?1) i=1

l?1

X 1 f (?s — xi?1) c(?s) 1 + (since 1 ? 1/x ? log x for x ? 1) log 1? f (?s — xi) i=1

f (?s — x0) 1 1 d c(?s) 1 + log = ? 1 + log c(?s) 1? f (?s — xl?1) 1? ?

?

Proof of Theorem 3.2. Combining these claims, we have

X

1 1 d d c(x) ? ? T (x, f) ? ? 1 + log ? c(?) ? (1 + 3) ? 1 + log ? ?c(x?). s 1? (1 ?)2 ? ? ? s?{x }

Thus, x is an approximate solution with the desired ratio. Let us see that x approximately satisfies the constraint; that is, f (x) ? (1 ? ?)?. We will now consider a slightly modified version of the algorithm; in the modified algorithm, the threshold is updated until f (x) = ?. Let x0 be the output of the modified algorithm. Then, we have X X ?c(?s) f (x0) ? f (x) ? f (?s — x) ? d ? ?d ? ?? cmax nr 0 0 s?{x }

s?{x }

The third inequality holds since c(?s) ? cmax and —{x0 }— ? nr. Thus f (x) ? (1 ? ?)?.

6

Conclusions

In this paper, motivated by real scenarios in machine learning, we generalized the submodular cover problem via the diminishing return property over the integer lattice. We proposed a bicriteria approximation algorithm with the following properties: (i) The approximation ratio to the cost almost matches the one guaranteed by the greedy algorithm [21] and is almost tight in general. (ii) We can satisfy the worst solution quality with the desired accuracy. (iii) The running time of our algorithm is roughly O(n log n log r). The dependency on r is exponentially better than that of the greedy algorithm. We confirmed by experiment that compared with the greedy algorithm, the solution quality

of our algorithm is almost the same and the runtime is several orders of magnitude faster. Acknowledgments The first author is supported by JSPS Grant-in-Aid for JSPS Fellows. The second author is supported by JSPS Grant-in-Aid for Young Scientists (B) (No. 26730009), MEXT Grant-in-Aid for Scientific Research on Innovative Areas (24106003), and JST, ERATO, Kawarabayashi Large Graph Project. The authors thank Satoru Iwata and Yuji Nakatsukasa for reading a draft of this paper. 8

2 References

[1] http://www.water-simulation.com/wsp/about/bwsn/. [2] N. Alon, I. Gamzu, and M. Tennenholtz. Optimizing budget allocation among channels and influencers. In Proc. of WWW, pages 381?388, 2012. [3] A. Badanidiyuru and J. Vondr?ak. Fast algorithms for maximizing submodular functions. In Proc. of SODA, pages 1497?1514, 2014. [4] Y. Chen, H. Shioi, C. A. F. Montesinos, L. P. Koh, S. Wich, and A. Krause. Active detection via adaptive submodularity. In Proc. of ICML, pages 55?63, 2014. [5] R. Iyer and J. Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In Proc. of NIPS, pages 2436?2444, 2013. [6] M. Kapralov, I. Post, and J. Vondrak. Online submodular welfare maximization: Greedy is optimal. In Proc. of SODA, pages 1216?1225, 2012. [7] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In Proc. of KDD, pages 137?146, 2003. [8] A. Krause and D. Golovin. Submodular function maximization. In Tractability: Practical Approaches to Hard Problems, pages 71?104. Cambridge University Press, 2014. [9] A. Krause and J. Leskovec. Efficient sensor placement optimization for securing large water distribution networks. Journal of Water Resources Planning and Management, 134(6):516? 526, 2008. [10] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. The Journal of Machine Learning Research, 9:235?284, 2008. [11] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In Proc. of KDD, pages 420?429, 2007. [12] H. Lin and J. Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 912?920, 2010. [13] H. Lin and J. Bilmes. A class of submodular functions for document summarization. In Proc. of NAACL, pages 510?520, 2011. [14] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. Optimization Techniques, Lecture Notes in Control and Information Sciences, 7:234?243, 1978. [15] A. Ostfeld, J. G. Uber, E. Salomons, J. W. Berry, W. E. Hart, C. A. Phillips, J.-P. Watson, G. Dorini, P. Jonkergouw, Z. Kapelan, F. di Pierro, S.-T. Khu, D. Savic, D. Eliades, M. Polycarpou, S. R. Ghimire, B. D. Barkdoll, R. Gueli, J. J. Huang, E. A. McBean, W. James, A. Krause, J. Leskovec, S. Isovitsch, J. Xu, C. Guestrin, J. VanBriesen, M. Small, P. Fischbeck, A. Preis, M. Propato, O. Piller, G. B. Trachtman, Z. Y.

Wu, and T. Walski. The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. Journal of Water Resources Planning and Management, 134(6):556?568, 2008. [16] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In Proc. of STOC, pages 475?484, 1997. [17] T. Soma, N. Kakimura, K. Inaba, and K. Kawarabayashi. Optimal budget allocation: Theoretical guarantee and efficient algorithm. In Proc. of ICML, 2014. [18] H. O. Song, R. Girshick, S. Jegelka, J. Mairal, Z. Harchaoui, and T. Darrell. On learning to localize objects with minimal supervision. In Proc. of ICML, 2014. [19] M. Sviridenko, J. Vondr?ak, and J. Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. In Proc. of SODA, pages 1134?1148, 2015. [20] P.-J. Wan, D.-Z. Du, P. Pardalos, and W. Wu. Greedy approximations for minimum submodular cover with submodular cost. Computational Optimization and Applications, 45(2):463?474, 2009. [21] L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica, 2(4):385?393, 1982. 9