Sparse and Greedy: Sparsifying Submodular Facility Location Problems

Erik M. Lindgren, Shanshan Wu, Alexandros G. Dimakis
The University of Texas at Austin
Department of Electrical and Computer Engineering
erikml@utexas.edu, shanshan@utexas.edu, dimakis@austin.utexas.edu

Abstract

Submodular facility location functions are widely used for summarizing large datasets and have found applications ranging from sensor placement, image retrieval, and clustering. A significant problem is that evaluating such functions typically requires the calculation of pairwise benefits for all items, which is computationally unmanageable for large problems. In this paper we propose a sparsification method that only computes nearest-neighbor benefits and can dramatically accelerate submodular greedy optimization. We show that the optimal solution of the sparsified problem will be at most a factor of a half from optimal, under minimal assumptions.

1 Introduction

Submodular optimization is an effective tool in machine learning theory and practice. Submodular functions satisfy a diminishing returns property that naturally applies to machine learning tasks such as data summarization, network monitoring, and clustering. One function that has been featured prominently is the submodular facility location (SFL) function. Given two sets, V and I, and a benefit matrix, choose a small set A from V such that the average benefit each element of I has to its most beneficial element in A is maximized. When $I = V$, the problem becomes a version of the k-medoids clustering problem.

This function has been used as an objective function to choose a set of documents or images that summarize a larger corpus [1, 2], to pick a set of locations to monitor in order to quickly identify important events in sensor or blog networks [3, 4], as well as clustering [5].

The SFL problem is NP-Hard and so approximation algorithms must be used to optimize it. The greedy algorithm gives a $(1 - 1/e)$-approximation to the optimal solution by the classic result of Nemhauser, Wolsey, and Fisher [6]. This approximation factor is tight: by appropriate choice of the benefit matrix, we can reduce an instance of the maximum coverage problem to an instance of SFL, which was shown by Feige to be NP-Hard to get a $(1 - 1/e + \epsilon)$-approximation for all $\epsilon > 0$ [7].

One issue is that the greedy algorithm has super-quadratic complexity $O(|V||I||A|)$. Because of this, there has been a large amount of work on improving the runtime of the greedy algorithm at the expense of approximation error. Specifically for the SFL problem, Wei, Iyer, and Bilmes considered sparsifying the benefit matrix first by using nearest neighbor methods and then running greedy using the sparsified matrix [8]. This work reported impressive 20x-80x speedups compared to the greedy algorithm with negligible empirical difference in the objective value.

Inspired by [8] we improve the theoretical analysis of the approximation error introduced by sparsification. Specifically, Wei et al. [8] require a strong assumption on the benefit matrix coming from a probability distribution, as well as $|A| = O(|V|)$. In this paper we obtain a constant factor approximation for nearest neighbor sparsification even after removing all assumptions besides those needed

for the greedy algorithm. We can also get results for other values of $|A|$, and if $|A| = O(|V|)$, our analysis shows that sparser matrices are possible compared to what was previously known.

Recently, Mirzasoleiman et al. [9] introduced a random sampling greedy algorithm that surprisingly has complexity independent of the size of A while maintaining a constant factor approximation. We show that nearest neighbor methods can be successfully combined with their method to obtain significant speedups. To the best of our knowledge, the proposed algorithm is significantly faster for optimizing SFL functions compared to the previous state of the art while maintaining empirical performance and provable guarantees similar to greedy.

2 Problem Formulation and Background

We define the submodular facility location problem as follows:

Submodular Facility Location (SFL) Problem. Given sets V and I, a benefit matrix $C \in \mathbb{R}_{\geq 0}^{I \times V}$, and a cardinality constraint k, return the set

$$\arg\max_{|A| \leq k} \sum_{i \in I} \max_{j \in A} C_{ij}. \tag{1}$$

To simplify notation, we assume that $|V| = |I| = n$. It is simple to generalize to the case where V and I are not the same size.

We define an objective function $F \colon 2^V \mapsto \mathbb{R}_{\geq 0}$ as $F(\emptyset) = 0$ and for all other sets,

$$F(A) = \sum_{i \in I} \max_{j \in A} C_{ij}. \tag{2}$$

The function F is submodular, since for all $j \in V$ and sets $A \subseteq B \subseteq V \setminus \{j\}$, we have $F(A \cup \{j\}) - F(A) \geq F(B \cup \{j\}) - F(B)$, that is, the gain of an element is diminishes as we add elements. Since we assumed the elements of C are nonnegative, we know that F is monotone, since for all $A \subseteq B \subseteq V$, we have $F(A) \leq F(B)$. We also have F normalized, since $F(\emptyset) = 0$. Thus by [6] the greedy algorithm gives a $(1 - 1/e)$-approximation to the optimal solution.

The greedy algorithm works by starting with the empty set, then for k iterations adding to A the element which increases the value of F the most based on what is currently in A. For each element of I, if we cache the current most beneficial element in A, we can calculate the gain of an element of V in time $O(n)$, which means the greedy algorithm runs in time $O(n^2 k)$.

The STOCHASTICGREEDY algorithm of Mirzasoleiman et al. works similarly, except instead of calculating the gain of every element in $V \setminus A$, we first take a random sample of size $\frac{n}{k} \ln \frac{1}{\epsilon}$. This returns a set with expected error a factor of $(1 - 1/e - \epsilon)$ from the optimal solution and has runtime $O(n^2 \log \frac{1}{\epsilon})$ [9].

3 Using Sparsified Benefit Matrices

3.1 Algorithm and Approximation Guarantee

If C is a sparse matrix, we can associate a bipartite graph with the matrix by having and edge between $j \in V$ and $i \in I$ whenever $C_{ij} > 0$. We can use the graph to calculate the gain of an element much quicker, since we only need to look at the neighbors of the element versus the entire set I. Algorithm 1 demonstrates using a sparse matrix in the STOCHASTICGREEDY algorithm for SFL.

However, our problem instance may not have a sparse matrix. One way to transform the original matrix into a sparse one is by only keeping the values in the benefit matrix C that are t-nearest neighbors, that is, for every element i in I, we only keep the top t elements of $C_{i1}, C_{i2}, \ldots, C_{in}$ and set the rest equal to zero. This leads to a matrix with nt nonzero elements. We then want to get a good solution to the original problem by solving SFL using the sparse matrix. Our main theorem is that we can set the sparsity parameter t to be $O(\frac{n}{k} \log \frac{n}{k})$—which is a significant improvement for large enough k—while still having the optimal solution to the sparsified problem be at most half the value of the solution to the original problem.

Algorithm 1 StochasticGraphGreedy for SFL

Input: benefit matrix C, sparsity graph $G = (V, I, E)$
define function $N(j)$: return the neighbors of j in G
for all $i \in I$:
    $\beta_i \leftarrow 0$ ▷ Caches for the value of the current best element in A
$A \leftarrow \emptyset$
for k iterations:
    $V' \leftarrow$ random sample from $V \setminus A$ of size $\frac{n}{k} \ln \frac{1}{\epsilon}$
    for all $j \in V'$:
        $g_j \leftarrow 0$
        for all $i \in N(j)$:
            $g_j \leftarrow g_j + \max(C_{ij} - \beta_i, 0)$ ▷ Add the gain element j gives to i
    $j^* \leftarrow \arg\max_{V'} g_j$
    $A \leftarrow A \cup \{j^*\}$
    for all $i \in N(j^*)$:
        $\beta_i \leftarrow \max(\beta_i, C_{ij^*})$ ▷ Update the caches for the neighbors of j^*
return A

Theorem 1. *Let O_t be the optimal solution to SFL using the benefit matrix sparsified using t-nearest neighbors. Let $t^*(n, k)$ be the smallest value of the sparsity parameter t such that $\frac{n}{t}(1 + \ln t) \leq k$. For any $t \geq t^*(n, k)$, we have $F(O_t) \geq \frac{1}{2}\mathrm{OPT}$.*

If we optimize the sparse instance of the problem with an α-approximation algorithm, then we are at most a factor of $\alpha/2$ from the optimal solution.

It can be checked that we have $t^*(n, k) \leq 5 \left\lceil \frac{n}{k} \max\left(\ln \frac{n}{k}, 1\right) \right\rceil$. Thus if $k = \Omega(n)$, we can take the sparsity parameter $t = O(1)$, which means that the benefit matrix C has only $O(n)$ nonzero entries. Also note that the only assumption we need is that the benefits between elements are nonnegative. When $k = \Omega(n)$, previous work was only able to take $t = O(\log n)$ and required the benefit matrix to come from a probability distribution [8].

The proof of Theorem 1 is graph-theoretic. The following is our key lemma.

Lemma 2. *Let $G = (V_1, V_2, E)$ be a bipartite graph such that each element of V_2 has at least t neighbors in V_1 and $|V_1| \leq t|V_2|$. Then there exists a set $\Gamma \subseteq V_1$ such that every element of V_2 has a neighbor in Γ and*

$$|\Gamma| \leq \frac{|V_1|}{t}\left(1 + \ln\left(t\frac{|V_2|}{|V_1|}\right)\right). \tag{3}$$

The lemma says that for any bipartite graph where every right vertex as degree at least t must have a small left dominating set. The value of $t^*(n, k)$ comes from needing a left dominating set of size at most k.

Since the sparsification depends on the number of elements being selected k, if k is small we may want to set t smaller than the value allowed by Theorem 1. The following corollary gives approximation bounds for setting t as if we were to choose ℓk elements, for all integer adjustment parameters ℓ.

Corollary 3. *Let t^*_ℓ be the smallest value of t such that $\frac{n}{t}(1 + \ln t) \leq \ell k$. For any $t \geq t^*_\ell$, we have $F(O_t) \geq \frac{1}{2\ell}\mathrm{OPT}$.*

3.2 Runtime

Ignoring the time to build the graph, the time to run StochasticGraphGreedy using the t-nearest neighbor graph for $t = t^*(n, k)$ is $O(\frac{n^2}{k} \log \frac{n}{k} \log \frac{1}{\epsilon})$. Thus for $k \gg \log n \log \frac{1}{\epsilon}$, we can run our algorithm in sub-quadratic time.

There has been a large amount of work on the problem of quickly building t-nearest neighbor graphs. If V and I are low dimensional vectors, then space partitioning methods and metric data structures

ℓ	Speedups (Greedy + t-NN)	Speedups (Greedy)	Relative Value
1	1.8	4.4	0.999
2	3.0	8.5	0.994
4	6.4	20.4	0.991
8	18.0	46.0	0.975

Figure 1: We show the speedups and relative value of STOCHASTICGRAPHGREEDY compared to STOCHASTICGREEDY [9]. We set the sparsity in STOCHASTICGRAPHGREEDY to be the smallest value of t such that $\frac{n}{t}(1+\ln t) \leq \ell k$ for various values of the adjustment parameter ℓ. We set the error ϵ of STOCHASTICGREEDY to be such that the two methods have the same theoretical approximation guarantee. We compare the runtime both including and not including the time to build the t-nearest neighbor graph.

such as kd-trees [10] work well and are exact. In high dimensional spaces locality-sensitive hashing gives good approximate results with theoretical guarantees [11]. Hashes exists for a variety of distances and similarity measures such as Euclidean [12], cosine [13], and dot product [21].

Nearest neighbor methods other than LSH have also been shown to work well for a variety of machine learning problems [14, 15, 16], see also [17] for GPU methods .

4 Experiments

We consider the problem of automatically classifying the genres of products such as music and movies and finding the quintessential element in each genre. First we need to associate each item with a vector. To do this we run nonnegative alternating least squares on rating data to fit a low rank matrix to the ratings data and get a feature vector for each element. We then compare vectors using the dot product. We use the dot product rather than the normalize dot product (cosine similarity) because we want the quintessential movie to be one that was well received.

We use the Yahoo! Music ratings database to get ratings for close to 137 thousand songs [18]. We then complete the ratings matrix with the Spark MLlib collaborative filtering library [19]. To create the nearest neighbor graph, we use the LSH for inner product introduced by Neyshabur and Srebro [20].

We then compare the runtime and value of STOCHASTICGRAPHGREEDY and STOCHASTICGREEDY [9] for various values of the sparsity parameter t. We choose the approximation error ϵ in STOCHASTICGREEDY so that the two methods have the same theoretical approximation error. We set the number of cluster centers k to be 500. We chose the LSH parameters so that 99% of elements have t neighbors. For runtime, we compare the speed ups including and not including the time to build the graph (see Figure 1). We see speedups of 18x, and not including the time to build the graph we see speedups of 46x. STOCHASTICGRAPHGREEDY as always within 97.5% of STOCHASTICGREEDY in objective value.

The Yahoo! Music database was anonymized, so we could not see the actual clusters made, but we did the same process with the MovieLens ratings database [22]. We show five cluster centers and the five movies with highest dot product to the cluster center in Figure 2. As can be seen, each cluster seems quite interpretable and the representative movie seems to indeed capture the nature of each group.

Happy Gilmore	A Nightmare on Elm Street	The Notebook	Shawshank Redemption	Pulp Fiction
Tommy Boy (3.55)	Friday the 13th (3.34)	August Rush (3.67)	Schindler's List (4.77)	Black Mirror (4.67)
Billy Madison (3.52)	Halloween II (3.21)	P.S. I Love You (3.67)	The Usual Suspects (4.73)	Reservoir Dogs (4.67)
Dumb & Dumber (3.46)	Hellbound: Hellraiser II (3.16)	The Holiday (3.66)	Life Is Beautiful (4.63)	American Beauty (4.65)
Ace Ventura: Pet Detective (3.38)	A Nightmare on Elm Street 3 (3.13)	Remember Me (3.60)	Intouchables (4.55)	A Clockwork Orange (4.64)
Road Trip (3.32)	Child's Play (3.10)	A Walk to Remember (3.54)	Saving Private Ryan (4.55)	Trainspotting (4.58)

Figure 2: Five cluster centers and a list of the five movies with the feature vector that has the highest dot product with the cluster center (dot product contained in parenthesis). As can be seen, each cluster seems quite interpretable and the representative movie seems to indeed capture the nature of each group.

References

[1] H. Lin and J. A. Bilmes. Learning mixtures of submodular shells with application to document summarization. *UAI*, 2012.

[2] S. Tschiatschek, R. K. Iyer, Haochen Wei, and J. A. Bilmes. Learning mixtures of submodular functions for image collection summarization. In *NIPS*, 2014.

[3] A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, 2008.

[4] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, 2007.

[5] A. Krause and R. G. Gomes. Budgeted nonparametric learning from data streams. In *ICML*, 2010.

[6] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsI. *Mathematical Programming*, 14(1):265–294, 1978.

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

[8] K. Wei, R. Iyer, and J. Bilmes. Fast multi-stage submodular maximization. In *ICML*, 2014.

[9] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrak, and A. Krause. Lazier than lazy greedy. In *AAAI*, 2015.

[10] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[11] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, 2004.

[13] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002.

[14] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, 2006.

[15] J. Chen, H.-R Fang, and Y. Saad. Fast approximate k–NN graph construction for high dimensional data via recursive lanczos bisection. *JMLR*, 10:1989–2012, 2009.

[16] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, 2011.

[17] V. Garcia, E. Debreuve, F. Nielsen, and M. Barlaud. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. In *International Conference on Image Processing*, 2010.

[18] Yahoo! Labs. R2 - Yahoo! Music user ratings of songs with artist, album, and genre meta information, v. 1.0. `webscope.sandbox.yahoo.com`.

[19] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine learning in Apache Spark. *arXiv:1505.06807*, 2015.

[20] B. Neyshabur and N. Srebro. On symmetric and asymmetric LSHs for inner product search. In *ICML*, 2015.

[21] A. Shrivastava and P. Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *NIPS*, 2014.

[22] GroupLens. MovieLens 20M dataset, 2015. `http://grouplens.org/datasets/movielens/20m/`.

[23] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, 3rd edition, 2008.