# Report: Experimental Study for Submodular Maximization

Jiecao Chen
jiecchen@indiana.edu

March 17, 2016

k

## 1 Roadmap

In this part, we give an experimental study of submodular maximization. In Section 2 we cover the setup for our experiment; we then compare three greedy algorithms in Section 3; In Section 4 we compare three streaming algorithms with random sampling and the offline greedy algorithm; we discuss GREEDI-based distributed algorithms in Section 5 and conclude this report in Section 6.

## 2 The Setup

### 2.1 Submodular Functions

We mainly consider two type of submodular functions in our experiment.

**Set Cover Problem** In the *set cover problem*, we are given a collection of subsets of a set $E$, i.e. $V = \{C_1, C_2, \ldots, C_n\}$ where each $C_i \subseteq E$. We define a function $f : 2^V \to \mathbb{R}$ such that $f(S) = |\cup_{C \in S} C|$. We can interpret $f$ as follow: given $S$ as a subset of $V$, the value of $f(S)$ is the number of distinct elements covered by the sets in $S$.

One can easily verify that $f$ satisfies the diminishing return property thus is a submodular function. Furthermore, it is clear that $f$ is non-decreasing. Now given the cardinality constraint, we want to solve the following,

$$\arg\max_{S \subseteq V : |S| \le k} f(S).$$

**Weighted Vertex Function** Let $G = (V, E, W)$ be a graph where each $v \in V$ has a weight $w_v > 0$. For any $S \subseteq E$, $V(E)$ is the set of vertices of edges in $S$. Let $f : 2^E \to \mathbb{R}$ be a set function with $f(S) = \sum_{v \in V(S)} w_v$, then $f$ is submodular. We call $f$ a *weighted vertex function*.

### 2.2 Datasets

**Synthetic:** We create $1000$ subsets of $[1000] = \{1, 2, \ldots, 1000\}$ , each subset is randomly constructed as follows: 1) randomly sample an integer $n$ from $[1, 100)$ as the size; 2) randomly sample $n$ elements without replacement from $[1000]$. The SYNTHETIC dataset is then a collection of $1000$ sets.

**Facebook:** The FACEBOOK dataset is a friends list collected from Facebook users. This dataset is an undirected graph with $4039$ nodes, $88234$ edges. More information of this dataset can be found in [4] ego-Facebook. We assign each node its degree as its weight. The original edges are grouped by nodes.

**Accidents:** dataset is publicly available from `http://fimi.ua.ac.be/data/`. Each line of ACCIDENTS can be considered as a subset of $[n]$ for a fixed $n$. The original dataset has $340,183$ subsets. Due to lack of computation resources, we only take the first $10,000$ subsets in our experiment. More information of this dataset can be found in `http://fimi.ua.ac.be/data/accidents.pdf`.

### 2.3 Computation Environments

Due to lack of computation resources, all experiments were run in a personal laptop with 3.7GB Memory, 119GB SSD, 1.70GHzx2-Intel Core i5 CPU. The operating system is Linux Mint 7.2 Cinnamon 64-bit.

### 2.4 Code

Most code for this experiment is available at

- `https://github.com/jiecchen/SubmodularLib/tree/master/src.`

It contains about $3000$ lines of java and python code.

### 2.5 Comments on Randomized Algorithms

For randomized algorithms, we run experiments for about $10$ times; the results are then presented as error bar with $2\sigma$, which is $95\%$ confidence interval.

## 3 Simple Greedy Algorithms

In this section, we compare three different greedy algorithms in the problem of monotone submodular maximization subject to cardinality constraint. We compare their efficiency (measured by number of value queries) and the quality of solutions they returned.

### 3.1 Greedy Algorithms

Let $V$ be the ground set and $k$ be the cardinality constraint. $0 < \epsilon < 1$ is a parameter to control the error.

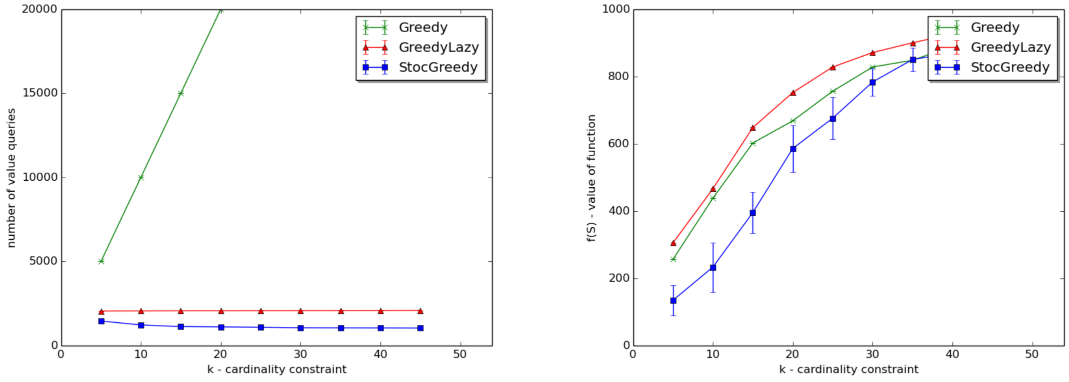
- GREEDY is the standard greedy algorithm that gives $(1-e^{-1})$-approximation. See Algorithm 1 for details.
- GREEDYLAZY is the greedy algorithm with the trick of lazy evaluation [5]. The idea goes as follows: instead of computing $\Delta_f(e|S)$ for each $e \in V\backslash S$ in Line 3 of Algorithm 1, GREEDYLAZY keeps an upper bound $\rho(e)$ (initially $+\infty$) on the marginal gain sorted in decreasing order (or kept in a heap). In each iteration, the GREEDYLAZY algorithm evaluates the element on top of the heap and updates its upper bound via $\rho(e) \leftarrow \Delta(e|S)$. If the updated $\rho(e) \geq \rho(e')$ for all other $e'$, submodularity guarantees that $e$ is the element with the largest marginal gain. The GREEDYLAZY algorithm again gives $(1-e^{-1})$-approximation.
- STOCGREEDY is a sampling-based greedy algorithm. In each step where we add one new element to the solution, we consider only a small fraction of the remaining elements ($O(\frac{|V|}{k}\log\frac{1}{\epsilon})$). In the selected portion of elements, we also use lazy evaluation to obtain further speedup. This algorithm gives $(1-e^{-1}-\epsilon)$-approximation in expectation. We omit its detailed description.

**Algorithm 1:** GREEDY for submodular maximization subject to cardinality constraint

```
Input: $V$ the ground set, $f$ the submodular function, $k$ the cardinality constraint
Output: a set $S \subseteq V$
$S \leftarrow \emptyset$
while $|S| < k$ do
    $e \leftarrow \arg\max_{e \in V \setminus S} \Delta_f(e|S)$
    $S \leftarrow S \cup \{e\}$
return $S$
```

### 3.2 Results for SYNTHETIC

Figure 1a shows that both GREEDYLAZY and STOCGREEDY significantly outperform GREEDY in terms of number of value queries being called. STOCGREEDY is even faster than GREEDYLAZY . Figure 1b shows that GREEDY , GREEDYLAZY and STOCGREEDY return solutions with similar quality; STOCGREEDY produces slightly worse solution compared with other two.



(a) Efficiency

(b) Quality

Figure 1: Experiment on SYNTHETIC dataset

## 4 Streaming Submodular Maximization

We compare three different streaming algorithms in the problem of monotone submodular maximization subject to cardinality constraint. In generally it is hard to compare those algorithms fairly because they are configured by different type of parameters. To address this issue, we fix parameters so that all three algorithms consume roughly the same amount of time. We also compare them with the standard greedy algorithm (here we call it OFFLINE ) and a streaming algorithm (called RANDOMSAMPLING ) that generates the solution by using random sampling (the notable reservoir sampling is being used internally).

### 4.1 Streaming Algorithms

- SIEVESTREAM [1] gives $(1/2 - \epsilon)$-approximation. See 2 for details.
- RANDOMSTREAM [2] gives $\frac{1-\epsilon}{2+\epsilon}$-approximation. See 3 for details.
- CIRCUITSTREAM [2,7] gives $1/4$-approximation. See 4 for details.

We also compare above three algorithms with OFFLINE and RANDOMSAMPLING as mentioned.

**Algorithm 2:** SieveStream for submodular maximization

```
Input: $V$ as data stream, $f$ a monotone submodular function, $k$ the size
        constraint, $\epsilon$ a parameter
Output: a set $S \subseteq V$
$O = \{(1+\epsilon)^i \mid i \in \mathbb{Z}\}$
/* maintain the sets only for the necessary $v$'s lazily */
For each $v \in O,\ S_v \leftarrow \emptyset$
$m \leftarrow 0$
for each $e$ in the data stream do
    $m \leftarrow \max\{m, f(\{e\})\}$
    $O \leftarrow \{(1+\epsilon)^i \mid m \leq (1+\epsilon)^i \leq 2 \cdot k \cdot m\}$
    Delete all $S_v$ such that $v \in O$
    for $v \in O$ do
        if $\Delta(e|S_v) \geq \frac{v/2 - f(S_v)}{k - |S_v|}$ and $|S_v| < k$ then
            $S_v \leftarrow S_v \cup \{e\}$
return $\arg\max_{S_v : v \in O} f(S_v)$
```

**Algorithm 3:** RandomStream for submodular maximization

```
Input: $V$ as data stream, $f$ a non-negative submodular function, $k$ the cardinality
        constraint, $\epsilon$ a parameter
Output: a set $S \subseteq V$
$B \leftarrow \emptyset, S \leftarrow \emptyset$
for each $e$ in the data stream do
    if $|S| < k$ and $\Delta(e|S) > \alpha$ then
        $B \leftarrow B + e$
    if $|B| > \frac{k}{\epsilon}$ then
        $e \leftarrow$ uniformly random from $B$
        $B \leftarrow B - e, S \leftarrow S + e$
        for all $e' \in B$ s.t. $\Delta(e'|S) \leq \alpha$ do
            $B \leftarrow B - e'$
$S' \leftarrow$ offline algorithm on $B$
return $\arg\max_{A \in \{S, S'\}} f(A)$
```

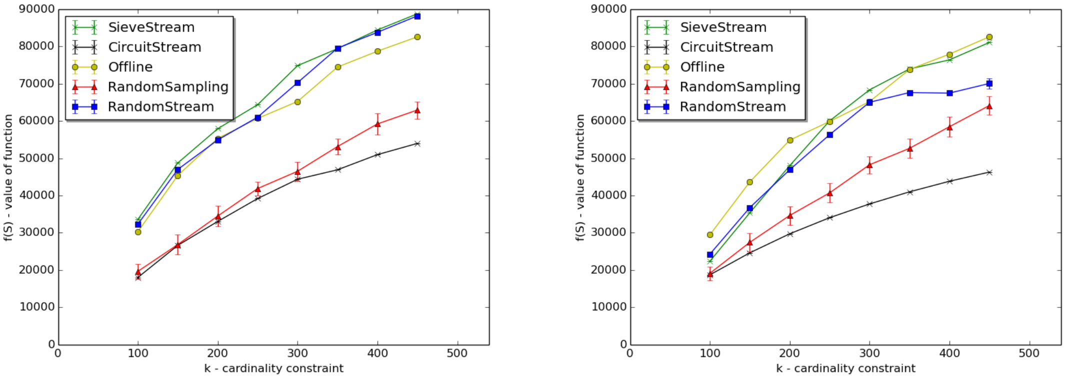
**Algorithm 4:** CircuitStream for submodular maximization

```
Input: $V$ as data stream, $f$ a monotone submodular function, $k$ is the cardinality
        constraint, $\gamma > 0$ is a parameter
Output: a set $S \subseteq V$
for each $e$ in the data stream do
    if $|S| < k$ then
        $S \leftarrow S + e$
    else
        $e^* \leftarrow \arg\min_{e' \in S} f(\{e'\})$
        if $f(\{e\}) \geq (1+\gamma) \cdot f(\{e^*\})$ then
            $S \leftarrow S \cup \{e\} \backslash \{e^*\}$
return $S$
```

## 4.2 Results for Facebook

Figure 2a shows the results of streaming algorithms that run on shuffled Facebook dataset. The results are interesting because SieveStream and RandomStream can produce better solutions (measured by function value) than Offline . Part of the reason is that both SieveStream and RandomStream run many copies of fixed-threshold streaming algorithms in parallel and latter pick the best among the solutions produced by those algorithms; it might be more likely to generate "good" solutions. Perhaps somewhat surprising, the solution produced by CIRCUITSTREAM is even worse than RANDOMSAMPLING . The result of CIRCUITSTREAM , however, does not violate the theoretical result (which only guarantees 1/4-approximation); more experiments are required before we can explain this phenomenon.

Figure 2b shows the results on non-shuffled FACEBOOK dataset where edges are grouped by vertices. As we can see from this figure, OFFLINE is independent of the order of input data due to the way it works (see Algorithm 1); but the order does matter to streaming algorithms. In the non-shuffled dataset, OFFLINE generates the best solution and the solution returned by SIEVESTREAM is comparably good; RANDOMSTREAM generates slightly worse result while CIRCUITSTREAM again has the worse performance.



(a) Shuffled edges

(b) Edges grouped by vertices

Figure 2: Streaming Algorithms on FACEBOOK; $\epsilon$ is set to be $0.2$ for both SIEVESTREAM and RANDOMSTREAM ; $\gamma$ is set to be $1.0$ for CIRCUITSTREAM .

## 5 Distributed Submodular Maximization

In this section we mainly focus on testing GREEDI-based but omit the algorithms in Kumar et al. [3]. Here are several reasons 1). Kumar et al. [3] has already been experimentally studied and it requires more rounds than GREEDI-based algorithms; 2). the implementation of algorithms in [3] is quite involved and one has to tune many parameters to make it work well. Comparing this algorithm with others can therefore be difficult; 3) on the other hand, GREEDI-based algorithms has less experimental studies but more theoretical results. It would be interesting to conduct experiments to verify those theories.

### 5.1 GREEDI-based Algorithms

We described here the general framework of GREEDI-based Algorithms. Let $V$ be the ground set we consider; $m$ is the number of machines and $C \in \mathbb{Z}^+$ is an parameter; also let $k$ be the cardinality constraint. The algorithm goes as follows:

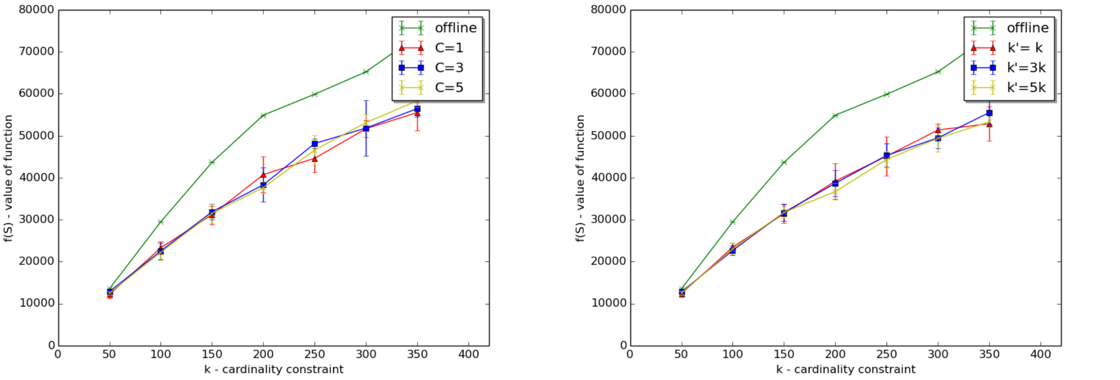
- Randomly assign each $v$ to $C$ out of $m$ machines, we obtain subsets $V_1, \ldots, V_m$; each is $\subseteq V$.
- Let ALG be an offline algorithm for submodular maximization, $k'$ be a cardinality constraint (it can be different from $k$). Run ALG on each $V_i$ with constraint $k'$, we obtains $U_1, U_2, \ldots, U_m$ as results.
- Let $U = \cup_i S_i$, run ALG on $U$ with parameter $k$, we obtain $S$ as the result. Also run ALG on $U_1, \ldots, U_m$ with parameter $k$ to obtain $S_1, S_2, \ldots, S_m$.

- Return the best (i.e. with largest function value) solution among $S, S_1, \ldots, S_m$.

In our experiment, we use GreedyLazy as ALG. It would be interesting to see how $k', C$ will affect the quality of the returned results.

### 5.2 Experimental Results

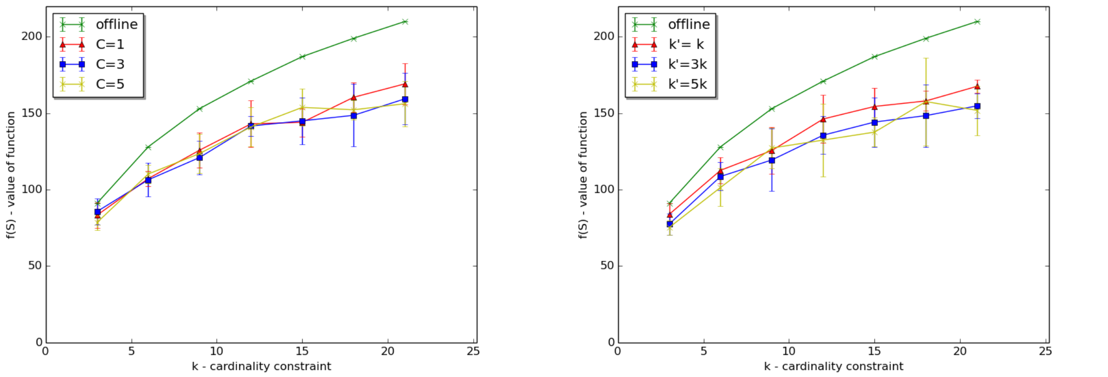
Mirrokni et al. [6] argued that when $k'$ and $C$ are increased, the approximation ratio of the GreeDi-Based algorithms may slightly increase (however, only slightly above $1/2$). From Figure 3 and Figure 4, we conclude that those minor theoretical improvement may be less important in practice. In fact, positive correlation between $k'$, $C$ and the approximation is not observed.



(a) Different multiplicity $C$; set $k' = k$; number of machines is 50.

(b) Different $k'$; $C$ is set to be 1; number of machines is set to be 50.

Figure 3: GreeDi-based Algorithms on Facebook dataset.



(a) Different multiplicity $C$; set $k' = k$; number of machines is 20.

(b) Different $k'$; $C$ is set to be 1; number of machines is set to be 20.

Figure 4: GreeDi-based Algorithms on Accidents dataset.

# 6 Conclusion

We implemented and tested several algorithms for submodular maximization. In particular, we focus on monotone submodular maximization subject to cardinality constraint, which is the case in most applications we have seen in our survey. Many of the algorithms included in this report are the first time being implemented, and some interesting phenomenons have been observed: e.g. when input data is shuffled, streaming algorithms may give solutions with better quality than the offline greedy algorithm, or increasing $k$ and $C$ actually does not help too much in increasing the approximation ratio. Due to lack of computation resources, our experiments were simulated in a laptop instead of a cluster. More datasets are necessary to obtain convincing conclusions. We leave this to future work, possibly a research paper that experimentally evaluates algorithms for streaming/distributed submodular maximization.

## References

[1] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680. ACM, 2014.

[2] C. Chekuri, S. Gupta, and K. Quanrud. Streaming algorithms for submodular function maximization. *arXiv preprint arXiv:1504.08024*, 2015.

[3] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing*, 2(3):14, 2015.

[4] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[5] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pages 234–243. Springer, 1978.

[6] V. Mirrokni and M. Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. *arXiv preprint arXiv:1506.06715*, 2015.

[7] A. B. Varadaraja. Buyback problem-approximate matroid intersection with cancellation costs. In *Automata, Languages and Programming*, pages 379–390. Springer, 2011.