EFFICIENT MONOTONE SUBMODULAR MAXIMIZATION SUBJECT TO MATROID CONSTRAINTS IN SUBMODLIB

Eeshaan Jain & Ipsit Mantri
Department of Electrical Engineering
Indian Institute of Technology Bombay
{19D070022, 180070032}@iitb.ac.in

Sibasis Nayak
Department of Computer Science
Indian Institute of Technology Bombay
{190050115}@iitb.ac.in

ABSTRACT

Maximization of monotone submodular functions subject to cardinality constraints has been a topic of interest with various greedy algorithms, with many of these algorithms achieving a $\left(1-\frac{1}{e}\right)$ approximation on the optimal solution (OPT) at high computational speeds. However, these variants of classical greedy do not work well on matroid constraints, and only provide a $\frac{1}{2}$ approximation of OPT. To overcome this, `CONTINUOUS-GREEDY` was introduced in Calinescu et al. (2011) which gives a $\left(1-\frac{1}{e}-\epsilon\right)$ approximation of OPT but it has a $O(n^8)$ running time. In spite of that, the algorithm is useful and generalizable. An improvement to the algorithm is `ACCELERATED-CONTINUOUS-GREEDY` introduced in Badanidiyuru & Vondrák (2014) has $\widetilde{O}(n^2)$ running time and is much more efficient than the previous version. We plan to create a support for matroid-based optimization and implement these two optimization algorithms along with four matroid-constraint-based submodular functions in `SUBMODLIB` Kaushal et al. (2022), an efficient and scalable Python library for submodular optimization in Python with a C++ optimization engine. Since our project is contribution-based, the code is available on the following fork in the `matroid` branch: Submodlib-fork

1 INTRODUCTION

In the recent years, a lot of attention has been given to optimization of functions which give diminishing returns, termed as submodular functions. The reason mainly being - they arise naturally when dealing with certain kinds of problems. For example, the task of placing sensors in an area to maximize the amount of information gained from them has been shown to be submodular Guestrin et al. (2005). Similarly, influence maximization over social network graphs also has close relations to submodularity Kempe et al. (2003). Given a submodular function f and a constraint $\mathcal{F}$, we are interested in the optimization problem given as

$$\max_{S} f(S) \text{ over } S \in \mathcal{F} \tag{1}$$

The type of constraint usually depends on the problem we are tackling. In case of sensor placement or influence maximization, we have a cardinality constraint, i.e $|S| \leq k$. In case of welfare maximization, we have a matroid constraint, i.e $S \in \mathcal{M}$ where $\mathcal{M}$ is a matroid. It has been shown by Nemhauser et al. (1978) that for submodular functions which are monotone, a simple greedy algorithm works well under cardinality constraint and provides at least $\left(1-\frac{1}{e}\right)$ approximation of the optimal solution. However, it has been shown by Nemhauser et al. (1978) that for the matroid constraints, i.e given $\mathcal{M} = (E, \mathcal{I})$, the problem

$$\max_{S \in \mathcal{I}} f(S) \tag{2}$$

the greedy solution provides a $\frac{1}{2}$ approximation of the optimal solution. Hence for a complicated constraint, greedy algorithm doesn't work well anymore. The CONTINUOUS-GREEDY algorithm proposed by Calinescu et al. (2011) provides a $\left(1 - \frac{1}{e}\right)$ solution to the above problem by converting the discrete optimization problem to a continuous one using multilinear relaxation and converting the solution obtained there back into a discrete one. The main benefit of multilinear relaxation is the usage of convexity and concavity, and the above problem is transformed to

$$\max\{F(y) : y \in P(\mathcal{M})\} \tag{3}$$

where $P(\mathcal{M})$ is the matroid polytope of $\mathcal{M}$. Hence, it provides a two-step method to get the required approximation of OPT. One drawback of CONTINUOUS-GREEDY is that the running time of the algorithm is $\widetilde{O}(n^8)$ with $\widetilde{O}(n^7)$ oracle calls. An improvement to the above algorithm, called ACCELERATED-CONTINUOUS-GREEDY was proposed by Badanidiyuru & Vondrák (2014) which has a $\widetilde{O}(n^2)$ oracle calls and achieves a $\left(1 - \frac{1}{e} - \epsilon\right)$ approximation. This variant is faster than the previous proposed speedup solutions which have $\widetilde{O}(n^4)$ oracle calls. This speedup can be related to how STOCHASTIC-GREEDY speeds up CLASSICAL-GREEDY Mirzasoleiman et al. (2014).

There are a variety of problems which are based around matroid-constraint-based optimization. One of them was stated in 2, and three related functions - Submodular Welfare Problem (SWP), Separable and Generalized Assignment Problem (SAP/GAP), have been described in Section 2.

SUBMODILIB Kaushal et al. (2022) is a submodular optimization library with a C++ optimization engine and has implemented many submodular functions, such as Facility Location, Graph Cut etc. along with constraint-based optimizers. However, currently there is no support for any matroid-based optimizers or matroid-constraint-based functions in the library. In this project, we introduce formally the problem at hand, and implement the functions and optimizers in the SUBMODLIB compatible format which can be merged with the main library.

2 PRELIMINARIES

2.1 SUBMODULAR FUNCTIONS

A set function $f : 2^V \to \mathbb{R}$ is said to be *submodular*, if for all $A, B \subseteq V$,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$$

This can equivalently be stated as, for $C \subseteq D \subseteq V$ and any $i \in V \setminus D$,

$$f(C \cup \{i\}) - f(C) \geq f(D \cup \{i\}) - f(D) \equiv \Delta(i|C) \geq \Delta(i|D)$$

A set function f is said to be *monotone* if $f(A) \geq f(B)$ whenever $A \supseteq B$. The set function f is said to be *non-negative* if the co-domain of f is $\mathbb{R}_+$. When considering the optimization problem at hand, we will consider non-negative monotone submodular functions.
The notion of submodular functions is extended with smoothness by Wolsey (1982). A function $F : [0, 1]^V \to \mathbb{R}_+$ is *smooth submodular* if

$$F(x) + F(y) \geq F(x \vee y) + F(x \wedge y)$$

where $(x \vee y) = \max\{x, y\}$ and $(x \wedge y) = \min\{x, y\}$, both in an coordinate-wise fashion. F is *monotone* if for $x \leq y$ coordinate-wise, $F(x) \leq F(y)$.
The above can be stated together as follows, $F : [0, 1]^V \to \mathbb{R}$ is smooth monotone submodular if

1. (Smoothness) $F \in \mathcal{C}_2([0, 1]^V)$, i.e the second-order partial derivatives exists everywhere
2. (Monotonicity) For each $j \in V$, $\frac{\partial F}{\partial y_j} \geq 0$ everywhere
3. (Submodularity) For any $i, j \in V$, $\frac{\partial^2 F}{\partial y_i \partial y_j} \leq 0$ everywhere

2.2 MATROIDS

A matroid $\mathcal{M} = (E, \mathcal{I})$ is a structure with a finite ground set E, the universe and a collection of subsets of 2^E (power set of E), $\mathcal{I}$ called independent sets, such that

1. $\emptyset \in \mathcal{I}$
2. $\forall\ I \in \mathcal{I}$ and $J \subseteq I$, $J \in \mathcal{I}$
3. (Exchange axiom) $\forall\ I, J \in \mathcal{I}$ and $|I| < |J|$, there exists $x \in J \setminus I$ such that $I \cup \{x\} \in \mathcal{I}$

A base B of a matroid $\mathcal{M}$ is a maximal independent set, and the exchange axiom guarantees that all bases of $\mathcal{M}$ have the same cardinality. Some examples of matroids are -

1. **k-Uniform Matroid:** A matroid $\mathcal{M} = (E, \mathcal{I})$ is *k-Uniform* if
$$\mathcal{I} = \{X \subseteq E : |X| \leq k\}$$
2. **Partition Matroid:** A matroid $\mathcal{M} = (E, \mathcal{I})$ is a partition matroid if E is partitioned into disjoint sets $E_1, \cdots, E_e$ and
$$\mathcal{I} = \{X \subseteq E : |E_i \cap X| \leq k_i \text{ for } i = 1, \cdots, e\}$$

The rank function of a matroid $\mathcal{M} = (E, \mathcal{I})$ is a set function $r_{\mathcal{M}} : 2^E \to \mathbb{N}$ such that for any $S \subseteq E$,
$$r_{\mathcal{M}}(S) = \max\{|X| : X \subseteq S, X \in \mathcal{I}\}$$
As an analogy, it can be thought to be similar to the rank function of matrices, or dimension in vector spaces. For example, for the *k-Uniform* matroid defined above, $r_{\mathcal{M}}(S) = \min\{|S|, k\}$ and the rank of the *partition* matroid is $\sum_i k_i$.
The matroid-rank theorem states that $r : 2^E \to \mathbb{N}$ is a rank function for a matroid *if and only if*

1. $r(\emptyset) = 0$ and $r(A \cup \{e\}) - r(A) \in \{0, 1\}\ \forall\ A \subseteq E, e \in E$
2. r is submodular, i.e for any $S, T \subseteq E$ we have $r(S) + r(T) \geq r(S \cup T) + r(S \cap T)$

Having defined the matroid rank function, we can finally state what a matroid polytope is. Given a a matroid $\mathcal{M} = (E, \mathcal{I})$, the matroid polytope $P(\mathcal{M})$ is defined as
$$P(\mathcal{M}) = \text{conv}(\{x_S \in \mathbb{R}^{|E|} : S \in \mathcal{I}\})$$
We can also write that
$$P(\mathcal{M}) = \{x \in \mathbb{R}^{|E|} : x(S) \leq r(S)\ \forall\ S \subseteq E \\ x_e \geq 0\ \forall\ e \in E\}$$
where $x(S) = \sum_{e \in S} x_e$. Now, the base of a matroid can be defined as the set $S \in \mathcal{I}$ such that $r_{\mathcal{M}}(S) = r_{\mathcal{M}}(E)$ and the base polytope is defined as
$$B(\mathcal{M}) = \left\{y \in P(\mathcal{M}) : \sum_{i \in E} y_i = r_{\mathcal{M}}(E)\right\}$$

2.3 Multilinear Extension

For a monotone submodular set function $f : 2^V \to \mathbb{R}_+$, a canonical extension to a smooth monotone submodular function can be obtained as follows. For $y \in [0, 1]^V$, let $\hat{y}$ denote a random vector in $\{0, 1\}^V$ where each coordinate is independently rounded to 1 with probability y_j. $\hat{y} \in \{0, 1\}^V$ can be identified with $R \subseteq V$ with the indicator given as $\hat{y} = \mathbf{1}_R$. Then the multilinear function F (multilinear $\implies \frac{\partial^2 F}{\partial y_j^2} = 0$) can be defined as
$$F(y) = \mathbb{E}[f(\hat{y})] = \sum_{R \subseteq X} f(R) \prod_{i \in R} y_i \prod_{j \in R} (1 - y_j)$$

2.4 SUBMODLIB

As described in Section 1, `SUBMODLIB` is a python library for submodular optimization with a C++ optimization engine. The current optimizers in the library include `CLASSICAL-GREEDY`, `LAZY-GREEDY` and `STOCHASTIC-GREEDY` all of which provide a $\left(1 - \frac{1}{e}\right)$ approximation of OPT in the case of cardinality constraints, with a running time of atmost $O(kn)$. A list of common

submodular, mutual information, conditonal gain and clustering functions have also been implemented which are based around cardinality constraints. The usage of the library can be seen on their GitHub page, but in short the functions and optimizers are exposed separately and for cardinality constraints are interchangeable. This is slightly in constrast with how matroid-constraint-based optimizers are formulated, and we describe that now.

2.5 MATROID CONSTRAINTS

The constraint $\mathcal{F}$ can be either simple, such as the cardinality constraint, or more complex such as a matroid or knapsack constraint. In particular, 2 is of importance and has been studied previously. In fact, there is a randomized algorithm which gives a $\left(1-\frac{1}{e}\right)$ approximation to the problem $\max\{f(S) : S \in \mathcal{I}\}$ where $f : 2^E \to \mathbb{R}$ is a monotone submodular function given by a value oracle and $\mathcal{M} = (E, \mathcal{I})$ is a matroid given by the membership oracle Calinescu et al. (2011). Apart from 2, there are some other matroid-constraint-based submodular problems of great interest as follows.

2.5.1 SUBMODULAR WELFARE PROBLEM

Firstly, the general framework for the welfare problem is called the Social Welfare Maximization Problem and is shown to come up often in combinatorial auctions Lehmann et al. (2002). Given a set X of m items, and n players each of which as a utility function $w_i : 2^X \to \mathbb{R}_+$, the goal is to partition X into disjoint subsets $S_1, \cdots, S_n$ inorder to maximize the social welfare given as $\sum_{i=1}^{n} w_i(S_i)$. When the utility function w_i are submodular, the problem is of great interest, as it has been shown that `CLASSICAL-GREEDY` obtains a $\frac{1}{2}$ approximation of OPT Nemhauser et al. (1978) and further has been shown that a $\left(1-\frac{1}{e}+\epsilon\right)$ approximation implies $P = NP$ Khot et al. (2005). However, the problem can be reduced to a matroid-constraint-based problem with $\left(1-\frac{1}{e}\right)$ approximation of OPT. The reduction following Calinescu et al. (2011) is as follows -

(Step 1) Given the set of items A and n players, the ground set is defined as $X = [n] \times A$, i.e one clone of each item for each player. For each player, define the mapping $\pi_i : 2^X \to 2^A$ which takes all clones in S to corresponding player as

$$\pi_i(S) = \{j \in A : (i, j) \in S\}$$

(Step 2) Given the utility functions $w_1, \cdots, w_n : 2^A \to \mathbb{R}_+$, define $f : 2^X \to \mathbb{R}_+$ as

$$f(S) = \sum_{i=1}^{n} w_i(\pi_i(S))$$

which is submodular if w_i are submodular.

(Step 3) We need to partition $A = S_1 \cup \cdots \cup S_n$ such that $\sum_i w_i(S_i)$ is maximized. This problem is equivalent to finding $S = \bigcup_{i=1}^{n}(\{i\} \times S_i) \subseteq X$ containing at most one clone of each item, so that $f(S)$ is maximized. Hence, we have the independence set as (for $X_j = [n] \times \{j\}$)

$$\mathcal{I} = \{S \subseteq X : \forall\, j; |S \cup X_j| \leq 1\}$$

Thus, we have reduced the SWP to the form of 2.

2.5.2 GENERALIZED AND SEPARABLE ASSIGNMENT PROBLEMS

The Separable Assignment Problem consists of m items and n bins. Each bin has a collection of feasible sets F_j satisfying down-closure (if $A \in F_j$ and $B \subseteq A$, then $B \in F_j$). Each item has a value v_i^j depending on the bin it is placed in. In SAP, we need to choose disjoint feasible sets $S_j \in F_j$ to maximize $\sum_j \sum_i v_i^j$. The reduction to matroid constraint following Calinescu et al. (2011) is as follows -

(Step 1) Define the ground set $X = \{(j, S) : 1 \leq j \leq n, S \in F_j\}$, i.e for each bin, create a separate copy of the items which are packed only in j

(Step 2) Define the function $f : 2^X \to \mathbb{R}_+$ which is monotone submodular as

$$f(\mathcal{S}) = \sum_i \max_j \{v_i^j : \exists\, (j, S) \in \mathcal{S}, i \in S\}$$

(Step 3) The matroid constraint is $\mathcal{M} = (X, \mathcal{I})$ where $\mathcal{S} \in \mathcal{I}$ *if and only if* $\mathcal{S}$ contains at most one pair (j, S_j) for each j.

The Generalized Assignment Problem is a special case of SAP, where the feasible set is a knapsack constraint given as $F_j = \{S : \sum_i s_i^j \leq 1\}$. The rest of the analysis stands same as SAP.

3 Methods and Analysis

3.1 Continuous-Greedy

As seen in Section 2, multilinear relaxation brings the discrete problem to a continuous space, and the optimization is performed there. There are two questions to answer now - how to approximate the function in the continuous space and how to bring it back to the discrete domain. To do this, we have two steps

- The *continuous greedy* process approximates $\max\{F(y) : y \in P(\mathcal{M})\}$ within a $\left(1 - \frac{1}{e}\right)$ factor
- The *pipage-rounding* algorithm converts a fractional solution to a discrete solution satisfying $f(S) \geq F(y) \geq \left(1 - \frac{1}{e}\right)OPT$

The continuous greedy process functions with an overtime flow given as

$$\frac{dy}{dt} = v_{max}(y) \quad \text{where } v_{max} = \arg\max_{v \in P}(v \cdot \nabla F(y)) \tag{4}$$

It has been proved in Calinescu et al. (2011) that $y(1) \in P$ and $F(y(1)) \geq \left(1 - \frac{1}{e}\right)OPT$. In the earlier version of algorithms following the *continuous greedy* process, $v_{max} = \mathbf{e}_j$ and that produces a $\frac{1}{2}$ approximation.
The *pipage rounding* technique was introduced in Ageev & Sviridenko (2004) and the goal is go from a solution $y^* \in P(\mathcal{M})$ to the discrete solution $S \in \mathcal{I}$.

Algorithm 1: Continuous Greedy

Input : $f : 2^E \to \mathbb{R}_+$, $\mathcal{M} = (E, \mathcal{I})$
Output: Fractional solution y

$d \leftarrow \frac{1}{9 \cdot (r_{\mathcal{M}}(E))^2}$;
$n \leftarrow |E|$;
$y \leftarrow \mathbf{0}$;
for $t \leftarrow 0; t \leq 1; t \leftarrow t + \delta$ **do**
  $R(t) \leftarrow$contains each j independently w.p $y_j(t)$;
  $\omega_j(t) \leftarrow$estimate($\mathbb{E}[f_{R(t)}(j)]$) using $\frac{10}{\delta^2}(1 + \ln n)$ samples;
  $I(t) \leftarrow$ maximal independent set in $\mathcal{M}$ with weights $\omega_j(t)$;
  $y(t + \delta) \leftarrow y(t) + \mathbf{1}_{I(t)}$;
end
return $y(1)$

3.2 Accelerated-Continuous-Greedy

Although `CONTINUOUS-GREEDY` works and provides the required approximation, the running time is $\widetilde{O}(n^8)$ due the repeated sampling and small time steps in the algorithm. Hence, we need a faster algorithm for any practical usage. The `ACCELERATED-CONTINUOUS-GREEDY` presented in Badanidiyuru & Vondrák (2014) interpolates the fast `CLASSICAL-GREEDY` with the slow `CONTINUOUS-GREEDY` with a δ parameter such that $\delta = 1$ corresponds to the former and $\delta \in (0, 1)$ corresponds to discretized version of the latter. The main reason why the acceleration is provided is because of updating partial derivatives after each increment, which gives a cleaner analysis and mimics the discrete greedy algorithm. Along with that, due to the discrete greedy nature,

we take larger steps and thus need fewer samples per iteration, and fewer iterations. Moreover, any $\delta > 0$ gives a $\left(1 - \frac{1}{e} - \mathcal{O}(\delta)\right)$ approximation.

The implementation of `ACCELERATED-CONTINUOUS-GREEDY` currently utilizes the Swap-Rounding function, which we found to be inefficient. Along with that, the Pipage-Rounding function in `CONTINUOUS-GREEDY` can also have improvements. Hence, we enhance the working of the algorithm, and switch the Swap-Rounding procedure with our optimized Pipage-Rounding procedure. We found that the first two steps of the **HitConstraint** subroutine called by Pipage-Rounding in Calinescu et al. (2011) were inefficient, and moreover the subroutine was called twice. We found that there was no need to do so and thus we wrote a separate Pipage-Rounding algorithm as shown below:

Algorithm 2: Efficient Pipage-Rounding

function Pipage-Rounding $(f, \mathbf{x}, \mathcal{I})$;
Input : $f : 2^E \to \mathbb{R}_+$, $\mathbf{x} \in [0,1]^E$, $\mathcal{I} \subseteq 2^E$
Output: A set $S \subseteq E$ satisfying $S \in \mathcal{I}$
$T \leftarrow []$;
for $i \leftarrow 0$; $i <$ *length*$(\mathbf{x})$; $i \leftarrow i + 1$ **do**
  if $0 < \mathbf{x}[i] < 1$ **then**
    T.push(i);
  end
end
try:
  while *length*$(T) > 0$ **do**
    $i, j \leftarrow T[0], T[1]$;
    if $\mathbf{x}[i] + \mathbf{x}[j] < 1$ **then**
      $p \leftarrow \frac{\mathbf{x}[j]}{\mathbf{x}[i]+\mathbf{x}[j]}$;
      if *rand()* $< p$ **then**
        $\mathbf{x}[j] \leftarrow \mathbf{x}[i] + \mathbf{x}[j]$;
        $\mathbf{x}[i] \leftarrow 0$;
        delete$(T, 0)$;
      end
      else
        $\mathbf{x}[i] \leftarrow \mathbf{x}[i] + \mathbf{x}[j]$;
        $\mathbf{x}[j] \leftarrow 0$;
        delete$(T, 1)$;
      end
    end
    else
      $p \leftarrow \frac{1-\mathbf{x}[i]}{2-\mathbf{x}[i]-\mathbf{x}[j]}$;
      if *rand()* $< p$ **then**
        $\mathbf{x}[i] \leftarrow \mathbf{x}[i] + \mathbf{x}[j] - 1$;
        $\mathbf{x}[j] \leftarrow 1$;
        delete$(T, 1)$;
      end
      else
        $\mathbf{x}[j] \leftarrow \mathbf{x}[i] + \mathbf{x}[j] - 1$;
        $\mathbf{x}[i] \leftarrow 1$;
        delete$(T, 0)$;
      end
    end
  end
catch:
  $S \leftarrow E[\mathbf{x}]$;
  return S
end
$S \leftarrow E[\mathbf{x}]$;
return S

Algorithm 3: Decreasing Threshold

function Decreasing-Threshold $(f, \mathbf{x}, \epsilon, \mathcal{I})$;
Input : $f : 2^E \to \mathbb{R}_+$, $\mathbf{x} \in [0,1]^E$, $\epsilon \in [0,1]$, $\mathcal{I} \subseteq 2^E$
Output: A set $S \subseteq E$ satisfying $S \in \mathcal{I}$
$B \leftarrow \emptyset$;
$d \leftarrow \max_{j \in E} f(j)$;
for $w = d; w \geq \frac{\epsilon}{r} d; w \leftarrow w(1-\epsilon)$ **do**
  for $e \in E$ **do**
    $w_e(B, \mathbf{x}) \leftarrow$estimate($\mathbb{E}[f_{R(\mathbf{x}+\epsilon \cdot \mathbf{1}_B)}(e)]$) using $\frac{r \log n}{\epsilon^2}$ samples;
    if $B \cup \{e\} \in \mathcal{I}$ *and* $w_e(B, \mathbf{x}) \geq w$ **then**
      $B \leftarrow B \cup \{e\}$
    end
  end
end
return B

Algorithm 4: Accelerated Continuous Greedy

Input : $f : 2^E \to \mathbb{R}_+$, $\mathcal{I} \subseteq 2^E$
Output: A set $S \subseteq E$ satisfying $S \in \mathcal{I}$
$\mathbf{x} \leftarrow 0$;
for $t \leftarrow \epsilon; t \leq 1; t \leftarrow t + \epsilon$ **do**
  $B \leftarrow$Decreasing-Threshold$(f, \mathbf{x}, \epsilon, \mathcal{I})$;
  $\mathbf{x} \leftarrow \mathbf{x} + \epsilon \cdot \mathbf{1}_B$;
end
$S \leftarrow$Pipage-Rounding$(\mathbf{x}, \mathcal{I})$;
return S

4 Implementation and Changes

4.1 General

In general, since we are working with matroids, we defined hash functions for nesting sets for collections pairwise and group-wise. Along with that functions for calculating matroid rank and random sample were created.

4.2 Continuous Greedy and Related Functions

For implementing Continuous Greedy, there was no inbuilt structure we could use firstly, as `SetFunction` has been developed for cardinality constraints. As we are dealing with more complex constraints, we implemented the `matroidSetFunction` which has different data types for each operations (since the constraint is no more integer-based but set-based). The functions implemented are

1. `maximize()`: Performs the maximization
2. `matroidGain()`: Calculates the weights
3. `getMaxIndependenceSet()`: Generates the maximal independent sets for reduction
4. `evaluateFinalSet()`: Evaluates the final set

These are in consistence with the `Continuous-Greedy` algorithm defined above. For the submodular functions, we chose SWP, SAP and GAP. As GAP is just a special instance of SAP, only the first two submodular functions were defined. For SWP, we assume for each player and each subset to be given as an oracle. For SAP we give the value function and the Feasible sets as an oracle.

4.3 ACCELERATED CONTINUOUS GREEDY

The accelerated continuous greedy algorithm uses the monotone submodular function as a value oracle and the matroid constraint as a membership oracle. The implementation of various submodular functions as value oracle is already present in submodlib. Hence, if the user provides the independence set as an input, then any submodular function can be maximized over a matroid constraint. This can be easily built into the current system by overloading the `maximize()` operation in `SetFunction`. We implemented the following functions in the `AcceleratedContinuousGreedyOptimizer`:

1. `decreasing_threshold()`: Implements the Decreasing-Threshold algorithm.
2. `maximize_fractional()`: Maximizes the submodular function using the multilinear extension and gives an optimal fractional solution. Uses the `decreasing_threshold()`
3. `pipage_rounding()`: Implements the Efficient Pipage-Rounding
4. `maximize()`: A wrapper function which calls the above functions in order to get the optimal set.

5 FUTURE WORK

We have currently implemented the optimizers and functions in our fork of `SUBMODLIB`. The next step is allowing memoization to happen and merging with the main branch, after discussions with the `SUBMODLIB` team.

REFERENCES

Alexander A. Ageev and Maxim Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8:307–328, 2004.

Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pp. 1497–1514, USA, 2014. Society for Industrial and Applied Mathematics. ISBN 9781611973389.

Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011. doi: 10.1137/080733991. URL `https://doi.org/10.1137/080733991`.

Carlos Guestrin, Andreas Krause, and Ajit Paul Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pp. 265–272, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931805. doi: 10.1145/1102351.1102385. URL `https://doi.org/10.1145/1102351.1102385`.

Vishal Kaushal, Ganesh Ramakrishnan, and Rishabh Iyer. Submodlib: A submodular optimization library, 2022. URL `https://arxiv.org/abs/2202.10680`.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pp. 137–146, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137370. doi: 10.1145/956750.956769. URL `https://doi.org/10.1145/956750.956769`.

Subhash Khot, Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Inapproximability results for combinatorial auctions with submodular utility functions. In *Proceedings of the First International Conference on Internet and Network Economics*, WINE'05, pp. 92–101, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540309004. doi: 10.1007/11600930_10. URL `https://doi.org/10.1007/11600930_10`.

Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *CoRR*, cs.GT/0202015, 2002. URL https://arxiv.org/abs/cs/0202015.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. *CoRR*, abs/1409.7938, 2014. URL http://arxiv.org/abs/1409.7938.

George Nemhauser, Laurence Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14:265–294, 12 1978. doi: 10.1007/BF01588971.

Laurence A. Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research*, 7(3):410–425, 1982. ISSN 0364765X, 15265471. URL http://www.jstor.org/stable/3689607.