# Minimization of Submodular Functions

## - from Chapter 45 of Schrijver -

Azad Tasan

July 11, 2020

# Outline

# The goal

Let $f$ be a submodular function on the set $S = \{1, \ldots, n\}$ with rational values.

(i) Find

$$\min_{T \subset S} f(T)$$

(ii) Find

$$\arg\min_{T \subset S} f(T)$$

The algorithm we present will solve these problems, given an oracle for the value of $f$.

**Throughout we assume $f(\emptyset) = 0$.**

# First some definitions

**Definition**

Let $S = \{1, \ldots, n\}$, $f$ submodular on $P(S)$ and $\prec$ an ordering on $S$. Then we define $b^{\prec} \in EP_f$ component-wise as

$$b_v^{\prec} = f(\{s \prec v\} \cup \{v\}) - f(\{s \prec v\})$$

**Definition**

For a set $U$, we write

$$b^{\prec}(U) := \sum_{i \in U} b_i^{\prec}$$

- Orderings = Permutations.
- Compare: These are the vertices of $EP_f$ as shown by Sandro.

# A useful lemma

## Lemma

*Let $S = \{1, \dots, n\}$, $f$ submodular on $P(S)$ and $\prec$ an ordering on $S$. Let $U \subset S$ be downward closed with respect to $\prec$. Then*

$$b^{\prec}(U) = f(U)$$

## Proof.

We get a telescoping sum and use $f(\emptyset) = 0$.

$$\begin{aligned} b^{\prec}(U) = \sum_{s \in U} b_s^{\prec} &= \sum_{s \in U} f(\{v \prec s\} \cup \{s\}) - f(\{v \prec s\}) \\ &= f(U) - f(\emptyset) = f(U) \end{aligned}$$

$\square$

# A useful corollary

## Corollary

*For $l$ orderings $\prec_1, \ldots, \prec_l$, and $U$ downward closed with respect to all $\prec_i$, we have*

$$\lambda_1 b^{\prec_1}(U) + \cdots + \lambda_m b^{\prec_m}(U) = (\sum_i \lambda_i) f(U)$$

*In particular, if the combination is convex, we obtain $f(U)$.*

## Proof.

Immediate from the lemma. □

- The algorithm will use the corollary.
- We construct a vector $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$.
- If the set $U$ is downward closed with respect to all $\prec_i$ and contains all negative and no positive entries of $x$
  $\rightarrow$ By the corollary we have a minimizer.

# On the algorithm

- We will update a vector as $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$.
- By means of adding orderings.
- Work on creating a $U$ which is downward closed with respect to all $\prec_i$ and contains all negative and no positive entries of $x$.
- $\rightarrow$ in the end we have a minimizer.

# The associated graph

### Definition (Associated graph)

Let $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$. Define a graph on $S$ by $D = (S, A)$ and

$$(v, w) \in A \iff \exists i : v \prec_i w$$

### Lemma

*Let $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$. A set $U$ is downward closed wrt. all $\prec_i$ if and only if there are no edges pointing into $U$ considered as vertices in $D$.*

### Proof.

By definition of $A$. □

### Definition (P and N)

Let $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$. Then $P$ contains all indices where $x$ is positive and $N$ all where $x$ is negative.

# Another lemma

## Lemma

*Let* $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$*. A set* $U$ *downward closed with respect to all* $\prec_i$ *containing* $N$ *is a minimizer of* $f$ *if* $U \cap P = \emptyset$*.*

## Proof.

By a previous lemma

$$x(U) = f(U)$$

$x \in EP_f$ and thus $x(W) \leq f(W)$ for all $W \subset S$. Also $x(U) \leq x(W)$ for all $W \subset S$ as $U$ contains all negative and no positive indices of $x$. In total

$$f(U) = x(U) \leq x(W) \leq f(W)$$

thus $U$ minimizes $f$. □

# The stopping condition

## Lemma

*Let* $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$ *and* $D = (S, A)$ *the associated graph. If there is no path from* $P$ *to* $N$ *in* $D$*, then we have a minimizer by*

$$U = \{s \in S : \textit{there is a path from } s \textit{ to } N\}$$

## Proof.

$U$ is downward closed with respect to all $\prec_i$ by a previous lemma.
$U$ contains $N$ and is disjoint with $P$.
By the previous lemma, $U$ is a minimizer. $\square$

# More definitions

**Definition**

Let $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$ and $D = (S, A)$ the associated graph. We define $d$ as the function giving distance from $P$ in $D$. That is $d(s)$ is the minimal length of a path from $P$ to $s$.

- Throughout the loop of the algorithm, $d$ will not decrease for any $v \in S$ and will eventually increase for some.
- This works towards cutting connectivity from $P$ to $N$.
- When this connectivity is cut, we have reached the stopping condition.

# The update rule

In every iteration we choose $t$ and $s$ as follows

**Definition ($t$ and $s$)**

Let $x$ and $D$ as previously. We choose $t \in N$ with $d(t)$ maximal among elements in $N$. To break the tie we choose $t$ maximal among those maximizing $d(t)$.
We take $s$ to be the predecessor to $t$ on a shortest path from $P$ to $t$. To break the tie we choose $s$ maximal. This means $d(t) = d(s) + 1$.

**Definition**

Let $\prec$ be an ordering on $S$. For any $s, u \in S$, we have the ordering $\prec^{s,u}$ where we moved $u$ before $s$.

- Example: $S = \{1, 2, 3, 4, 5, 6\}$ with ordering $\prec = 123456$. We choose $s = 2$, $u = 5$ and get $\prec^{s,u} = 152346$.

# The update rule

## Definition

Let $x$ and $D$ as previously. For choice of $s, t$, and ordering $\prec_i$, we have the orderings $\prec_i^{s,u}$ for all $s \prec_i u \preceq_i t$.

## Lemma

*Let $x$ and $D$ as previously. For some $\delta \geq 0$, the vector*

$$x + \lambda_1 \delta(\chi_t - \chi_s)$$

*can be written as a convex combination of the $b^{\prec_i^{s,u}}$ with $\prec_i^{s,u}$ as above.*

## Proof.

Without, see Schrijver Chapter 45. □

# The update rule

## Lemma

*Let $x$ and $D$ as previously. Also any point on the line between $x$ and $x + \lambda_1 \delta(\chi_t - \chi_s)$ can be written as a convex combination of the $b^{\prec_l}$ and $b^{\prec_i^{s,u}}$, for a fixed $i$. If*

## Proof.

The line between two points is their convex hull. □

## Definition ($x'$)

We choose the next $x$, or $x'$, to be the point closest to $x + \lambda_1 \delta(\chi_t - \chi_s)$ such that the value at $t$ stays nonnegative.

We have to update $P$ and $N$ to reflect the new $x'$, as well as the edges in $D$ to reflect the new orderings.

It might be an idea to reduce the representation to less terms using linear algebra.

# Observations

- By construction, $P' \subset P$.
- We increase the connectivity in $D$.
- We can do this as long as there is a path from $P$ to $N$.
- If there is no longer such a path we can terminate as shown previously.
- It remains to show this algorithm terminates.
- We will first show that $d$ does not decrease.

# $d$ does not decrease

## Lemma

*Throughout the loop, we have that $d'(v) \geq d(v)$ for the distance from $P$ function $d$. That is, distances from $P$ never decrease.*

## Proof.

Each new edge added comes from an added ordering. Thus it comes from some $\prec_i^{s,u}$, $s \prec_i u \preceq_i$, where we moved $u$ before $s$. The only change in ordering, i.e. added edge $(v, w)$ is from when $v = u$. Thus

$$s \preceq_i w \prec_i v \preceq_i t$$

$\square$

# $d$ does not decrease

## Lemma

*Throughout the loop, we have that $d'(v) \geq d(v)$ for the distance from $P$ function $d$. That is, distances from $P$ never decrease.*

## Proof.

Assume towards a contradiction $d(w)$ decreased for some $w$. Then there is a new edge $(v, w)$ with $d(w) \geq d(v) + 2$. For this we use that

$$d(w) - 1 \geq d'(w) = \min_{k \in (w,t]_{\prec_i}} d(k) + 1 = d(\tilde{k}) + 1$$

We now rearrange to see

$$d(w) \geq d(\tilde{k}) + 2$$

$\square$

# $d$ does not decrease

## Lemma

*Throughout the loop, we have that $d'(v) \geq d(v)$ for the distance from $P$ function $d$. That is, distances from $P$ never decrease.*

## Proof.

We now have a new edge $(v, w)$ with $d(w) \geq d(v) + 2$ and $s \preceq_i w \prec_i v \preceq_i t$. $s \preceq_i w$ thus $d(w) \leq d(s) + 1$, the same argument for $d(t) \leq d(v) + 1$. By choice of $s, t$, we have $d(s) + 1 = d(t)$. Altogether

$$d(w) \leq d(s) + 1 \leq d(t) \leq d(v) + 1$$

and thus a contradiction

$$d(v) + 2 \leq d(w) \leq d(v) + 1$$

□

# $\alpha$ and $\beta$

**Definition ($\alpha$)**

Let $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$ and $s$ and $t$ chosen. We define

$$\alpha = \max_i |(s,t]_{\prec_i}|$$

as the maximal length of an interval *stot* with respect to our orderings.

**Definition ($\beta$)**

Let $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$ and $s$ and $t$ chosen. We define

$$\beta = \# i : (|(s,t]_{\prec_i}| = \alpha)$$

as the number of orderings achieving the maximum $\alpha$.

# $(d(t), t, s, \alpha, \beta)$ decreases lexicographically

## Lemma

*For each step of the algorithm we have* $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$*, as well as choices of* $t$*,* $s$*. This yields* $\alpha$*,* $\beta$*. With* $'$ *denoting the values at the next iteration we have*

$$(d'(t'), t', s', \alpha', \beta') \prec_{lex} (d(t), t, s, \alpha, \beta)$$

*if* $d'(v) = d(v)$ *for all* $v \in S$*.*

## Proof.

Proof by case distinction: If the first entry does not decrease, the second must, etc.
By assumption, $d'(t') = d(t')$. By construction, $t' = s$ or $t' \in N$, as $t' \in N' \subset N \cup \{s\}$. If $t' = s$, $d'(t') = d(s) = d(t) - 1$, by choice of $s$. Otherwise, $t' \in N$, and thus $(d'(t'), t') = (d(t'), t') \prec_{\text{lex}} (d(t), t)$, by choice of $t$. This gives us the first two cases. □

# $(d(t), t, s, \alpha, \beta)$ decreases lexicographically

**Lemma**

$$(d'(t'), t', s', \alpha', \beta') \prec_{lex} (d(t), t, s, \alpha, \beta)$$

*if* $d'(v) = d(v)$ *for all* $v \in S$.

**Proof.**

We have seen that $(d'(t'), t') \prec_{\text{lex}} (d(t), t)$. Thus we now assume $d(t') = d(t)$, $t' = t$, and look at $s'$. $(s', t') = (s', t) \in A'$, by choice of $s'$. There are no new edges into $t$. Thus $(s', t) \in A$ and $s'$ would have been a valid choice for $s$. The choice $s$ is maximal among valid choices, thus $s' \leq s$. □

# $(d(t), t, s, \alpha, \beta)$ decreases lexicographically

## Lemma

$$(d'(t'), t', s', \alpha', \beta') \prec_{lex} (d(t), t, s, \alpha, \beta)$$

*if $d'(v) = d(v)$ for all $v \in S$.*

## Proof.

We are in the case $(d'(t'), t', s') = (d(t), t, s)$. $\alpha$ is next. As $t' = t$, $s' = s$, we are looking at

$$\max_{i'} |(s, t]_{\prec_i}|$$

But all added orderings have

$$|(s, t]_{\prec_i^{s,u}}| < \alpha$$

as we move $u$ out of the interval. Thus $\alpha' \leq \alpha$. □

# $(d(t), t, s, \alpha, \beta)$ decreases lexicographically

**Lemma**

$$(d'(t'), t', s', \alpha', \beta') \prec_{lex} (d(t), t, s, \alpha, \beta)$$

*if $d'(v) = d(v)$ for all $v \in S$.*

**Proof.**

Lastly, if $\alpha' = \alpha$, $\beta' < \beta$, as we do not need $\prec_i$ to express $x'$, as shown previously, and $\prec_i$ was chosen with $|(s, t]_{\prec_i}| = \alpha$. $\beta$ counts the amount of those maximal ones. Thus $\beta' = \beta - 1$. $\square$

# The Algorithm

**procedure** ($S = \{1, \dots, n\}, f$, Oracle for values of $f$)

  **initialize** $x = b^{\prec} \in \mathbb{R}^S$ $\triangleright$ Arbitrary choice of $\prec$

  **while** Path $P \to N$ in $D$ **do**

    Choose new $t \in N$, $s \in S$

    Update A, P, N $\leftarrow$ Update $x$

  **end while**

  **return** $U = \{s \in S : \exists s \to N\}$

**end procedure**

# The algorithm terminates

**Theorem**

*The algorithm terminates.*

**Proof.**

$d(v)$ has to increase or stay equal for any iteration. Moreover, as $(d(t), t, s, \alpha, \beta)$ decreases for any iteration, we have that at some point $d(v)$ has to increase for some $v$. But $d(v) \leq |S|$. Thus the loop terminates after finite time. $\square$

# The algorithm is correct

We had the lemma

**Lemma**

*Let* $x = \lambda_1 b^{\prec_1} + \cdots + \lambda_m b^{\prec_m}$ *and* $D = (S, A)$ *the associated graph. If there is no path from* $P$ *to* $N$ *in* $D$*, then we have a minimizer by*

$$U = \{s \in S : \textit{there is a path from } s \textit{ to } N\}$$

As we are now in said situation, the algorithm yields a minimizer.

# Outlook

- We did not discuss the running time or space complexity.
- Iwata showed that the running time of a modified algorithm can be reduced to $|S|^9 \log^2(|S|)$.
- There exist algorithms for minimizing submodular functions using a $P_f$ polytope membership oracle instead of a value oracle.
- There exist purely combinatorial algorithms, i.e. only using comparison, addition $\rightarrow$ extension to abelian group-valued functions.
- Adaptations can find non-trivial solutions, i.e. not empty and not everything. Important for e.g. minimum cut in a graph.

# References


- Alexander Schrijver. (2003). Combinatorial Optimization. Submodular Functions and Polymatroids.
- Alexander Schrijver. (2003). Combinatorial Optimization. Submodular Function Minimization.
- Satoru Iwata. (2002). A Fully Combinatorial Algorithm for Submodular Function Minimization.