Distributed Submodular Maximization in Massive Datasets

Huy L. Nguyen

Joint work with
Rafael Barbosa, Alina Ene, Justin Ward
Combinatorial Optimization

• Given
  – A set of objects V
  – A function f on subsets of V
  – A collection of feasible subsets I

• Find
  – A feasible subset of I that maximizes f

• Goal
  – Abstract/general f and I
  – Capture many interesting problems
  – Allow for efficient algorithms
Submodularity

We say that a function $f : 2^V \to \mathbb{R}_+$ is submodular if:

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

We say that $f$ is monotone if:

$$f(A) \leq f(B), \quad \forall A \subseteq B$$

Alternatively, $f$ is submodular if:

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

for all $A \subseteq B$ and $x \notin B$

Submodularity captures diminishing returns.
Submodularity

Examples of submodular functions:

- The number of elements covered by a collection of sets
- Entropy of a set of random variables
- The capacity of a cut in a directed or undirected graph
- Rank of a set of columns of a matrix
- Matroid rank functions
- Log determinant of a submatrix of a psd matrix
Example: Multimode Sensor Coverage

- We have distinct locations where we can place sensors
- Each sensor can operate in different modes, each with a distinct coverage profile
- Find sensor locations, each with a single mode to maximize coverage
Example: Identifying Representatives In Massive Data
Example: Identifying Representative Images

- We are given a huge set $X$ of images.
- Each image is stored multidimensional vector.
- We have a function $d$ giving the difference between two images.
- We want to pick a set $S$ of at most $k$ images to minimize the loss function:

$$L(S) = \frac{1}{|X|} \sum_{e \in X} \sum_{r \in S} \min d(e, r)$$

- Suppose we choose a distinguished vector $e_0$ (e.g. 0 vector), and set:

$$f(S) = L(\{e_0\}) - L(S \cup \{e_0\})$$

- The function $f$ is submodular. Our problem is then equivalent to maximizing $f$ under a single cardinality constraint.
Need for Parallelization

• Datasets grow very large
  – TinyImages has 80M images
  – Kosarak has 990K sets

• Need multiple machines to fit the dataset

• Use parallel frameworks such as MapReduce
Problem Definition

• Given set $V$ and submodular function $f$
• Hereditary constraint $I$ (cardinality at most $k$, matroid constraint of rank $k$, ... )
• Find a subset that satisfies $I$ and maximizes $f$
• Parameters
  - $n = |V|$
  - $k = \text{max size of feasible solutions}$
  - $m = \text{number of machines}$
Greedy Algorithm

Initialize $S = {}$

While there is some element $x$ that can be added to $S$:

Add to $S$ the element $x$ that maximizes the marginal gain $f(S \cup \{x\}) - f(S)$

Return $S$
Greedy Algorithm

- Approximation Guarantee
  - $1 - \frac{1}{e}$ for a cardinality constraint
  - $\frac{1}{2}$ for a matroid constraint
- Inherently sequential
- Not suitable for large datasets
Distributed Greedy
Performance of Distributed Greedy

• Only requires 2 rounds of communication
• Approximation ratio is:

\[ O\left(\frac{1}{\sqrt{\min(m, k)}}\right) \]

(where \( m \) is number of machines)
• Can construct bad examples
• Lower bounds for the distributed setting

(Indyk et al. ’14)
Power of Randomness
Power of Randomness

• Randomized distributed Greedy
  – Distribute the elements of V randomly in round 1
  – Select the best solution found in rounds 1 & 2

• Theorem: If Greedy achieves a $C$ approximation, randomized distributed Greedy achieves a $C/2$ approximation in expectation.

• Related results: [Mirrokni, Zadimoghaddam '15]
Intuition

• If elements in OPT are selected in round 1 with high probability
  – Most of OPT is present in round 2 so solution in round 2 is good

• If elements in OPT are selected in round 1 with low probability
  – OPT is not very different from typical solution so solution in round 1 is good
Power of Randomness

• Randomized distributed Greedy
  – Distribute the elements of V randomly in round 1
  – Select the best solution found in rounds 1 & 2

• Provable guarantees
  – Constant factor approx for several constraints

• Generality
  – Same approach to parallelize a class of algorithms
  – Only need a natural consistency property
  – Extends to non-monotone functions
Optimal Algorithms?

- Near-optimal algorithms?
- Framework to parallelize algorithms with almost no loss?

YES, using a few more rounds
Core Set
Core Set

Send Core Set
to every machine
Core Set
Core Set
Core Set

Grow Core Set over $1/\varepsilon$ rounds
Core Set

Grow Core Set over $\frac{1}{\varepsilon}$ rounds
Core Set

Grow Core Set over $1/\varepsilon$ rounds
Core Set

Grow Core Set over $1/\varepsilon$ rounds

Leads to only an $\varepsilon$ loss in the approximation

Intuition
Each round adds an $\varepsilon$ fraction of OPT to the Core Set
Matroid Coverage Experiments

Matroid Coverage (n=900, r=5)

Matroid Coverage (n=100, r=100)

It's better to distribute ellipses from each location across several machines!
Thank You!

Questions?