

Unconstrained Submodular Maximization in MapReduce

Sikander Randhawa, Ben Chugg, Angad Kalra

THE UNIVERSITY OF BRITISH COLUMBIA
VANCOUVER, CANADA

Background

Formally, a set function $f : 2^X \rightarrow \mathbb{R}$ is *submodular* iff

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

for all $A \subset B$.

Intuition: Think of decreasing marginal gains. The more you have, the less you appreciate gaining something new.

These functions find applications in graph theory, game theory and machine learning.

Background

Problem of interest:

$$\max_{S \subset X} f(S).$$

Has been well studied in many settings! In general this problem is NP-Hard [?].

So we settle for constraints and approximations:

- Greedy algorithm gives a $(1 - \frac{1}{e})$ -approx. for Cardinality Constraints (monotone) [?]
- A continuous variant also gives a $(1 - \frac{1}{e})$ -approx. for Matroid Constraints (monotone) [?].

Unconstrained, Non-Monotone Submodular Maximization (USM)

Less constraints \Rightarrow harder to maximize. Our problem of interest:

- Unconstrained, non-monotone submodular function maximization,
- But, in a parallel setting (i.e. MapReduce)
 - Most applications of submodular maximization arise when dataset is too large to fit on one machine. Understanding how to parallelize computation is important.

MapReduce

What is MapReduce?

- Distributed framework for computation.
- Computation proceeds in rounds.
 - **At most polylog number of rounds permitted.**
- In each round, many machines perform computation.
 - Limited to polytime computation.
- After each round, machines communicate.
- Each machine limited to $o(n)$ storage.

This will be our distributed framework. Developed by Karloff et al. [?].

Our Approach

Explore canonical parallelizations of existing well-known algorithms.

One algorithm seemed better than others:

Local Search (Feige et al. [?]):

- Achieves ($\frac{1}{3}$) approximation factor.
 - Comparable to state of the art.
- There was an intuitive parallelization.
 - Split the search across many machines.
- Approximation guarantee would not deteriorate after parallelizing.
 - In fact, get approximation factor for free!

Local Search

Good algorithm for USM exists in the centralized setting!

Local Search (Idea)

- 1 Start with best singleton, $\{v\}$.
- 2 While we can improve the solution, repeat the following:
- 3 Add or remove an element as long as it increases our solution value.
- 4 Return the solution.

Number of total swaps is bounded by $\tilde{O}(n^2) \Rightarrow$ halting guaranteed after $\tilde{O}(n^2)$ rounds.

- Really need $O(\log^k(n))$ rounds, so must widdle this down!

A Parallelization Approach

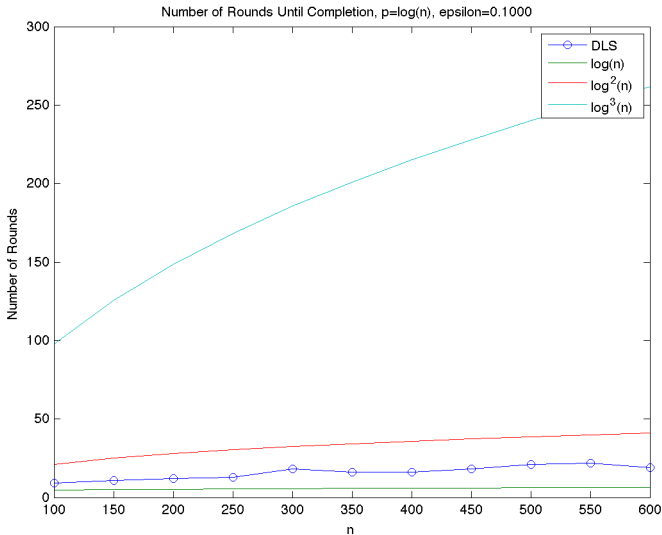
We have a promising algorithm, how do we parallelize?

- Start with arbitrary solution, S .
- Randomly split the universe, U into $\{U_i, \dots, U_p\}$.
 - Machine i gets U_i .
- Each machine treats U_i as the full universe and performs local search starting from S .
- When all machines stop:
 - Update S to be the best solution across the machines.
- Repeat until S doesn't change.

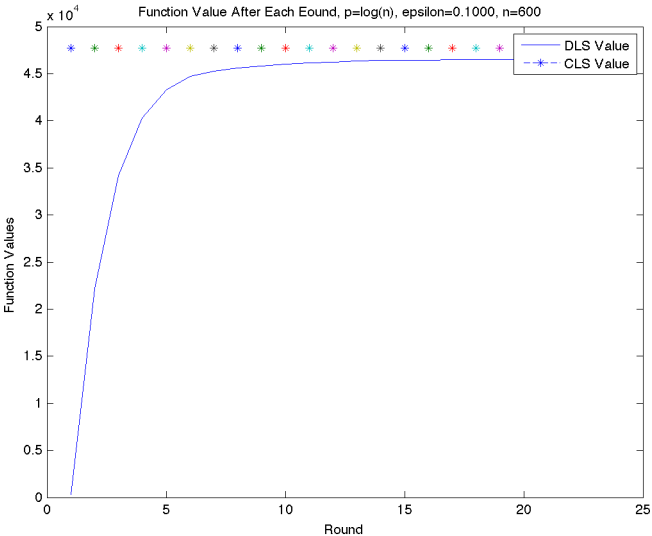
Main Problem: How do we guarantee that we only require a polylog number of rounds?

Turn to empirical evidence for building motivation, ideas, and intuition.

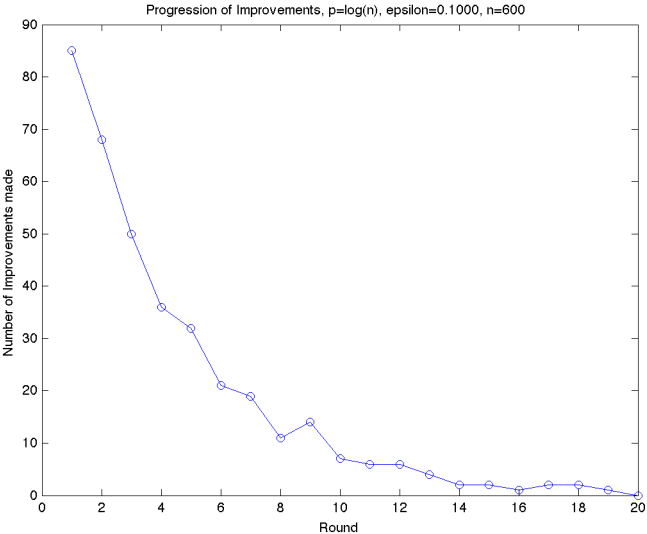
Number of Rounds



Function Value



Number of Improvements



Unconstrained Submodular Maximization in MapReduce

Sikander
Randhawa,
Ben Chugg,
Angad
Kalra

Background

Local Search

Results

Number of Rounds	Function Value
1	0.000000
2	0.000000
3	0.000000
4	0.000000
5	0.000000
6	0.000000
7	0.000000
8	0.000000
9	0.000000
10	0.000000
11	0.000000
12	0.000000
13	0.000000
14	0.000000
15	0.000000
16	0.000000
17	0.000000
18	0.000000
19	0.000000
20	0.000000
21	0.000000
22	0.000000
23	0.000000
24	0.000000
25	0.000000
26	0.000000
27	0.000000
28	0.000000
29	0.000000
30	0.000000
31	0.000000
32	0.000000
33	0.000000
34	0.000000
35	0.000000
36	0.000000
37	0.000000
38	0.000000
39	0.000000
40	0.000000
41	0.000000
42	0.000000
43	0.000000
44	0.000000
45	0.000000
46	0.000000
47	0.000000
48	0.000000
49	0.000000
50	0.000000
51	0.000000
52	0.000000
53	0.000000
54	0.000000
55	0.000000
56	0.000000
57	0.000000
58	0.000000
59	0.000000
60	0.000000
61	0.000000
62	0.000000
63	0.000000
64	0.000000
65	0.000000
66	0.000000
67	0.000000
68	0.000000
69	0.000000
70	0.000000
71	0.000000
72	0.000000
73	0.000000
74	0.000000
75	0.000000
76	0.000000
77	0.000000
78	0.000000
79	0.000000
80	0.000000
81	0.000000
82	0.000000
83	0.000000
84	0.000000
85	0.000000
86	0.000000
87	0.000000
88	0.000000
89	0.000000
90	0.000000
91	0.000000
92	0.000000
93	0.000000
94	0.000000
95	0.000000
96	0.000000
97	0.000000
98	0.000000
99	0.000000
100	0.000000

Improvements

Results

References