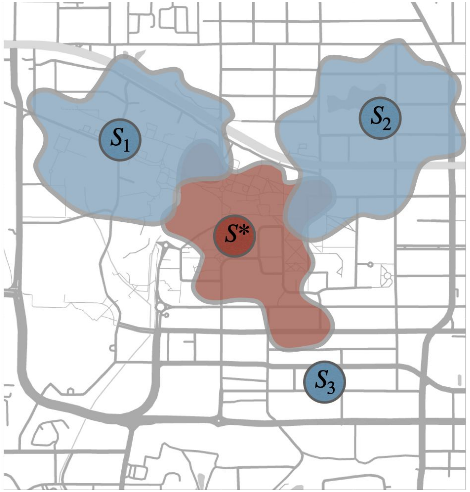
Best of Both Worlds: Practical and Theoretically Optimal Submodular Maximization in Parallel

Yixin Chen, Tonmoy Dey, Alan Kuhnle
Department of Computer Science
Florida State University

Submodularity

Can be defined as a property of a function where:

- Given an objective function $f(S) = \left| \bigcup_{s \in S} area(s) \right|$



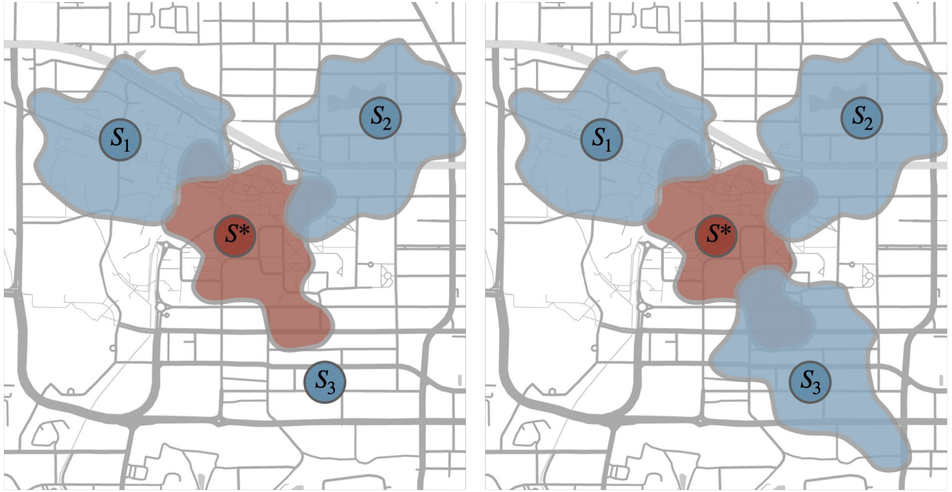
$$A = [S1, S2]$$

$$f(A \cup S^*) - f(A)$$

Submodularity

Can be defined as a property of a function where:

- Given an objective function $f(S) = \left| \bigcup_{s \in S} area(s) \right|$
- The marginal gain of adding an element to a set **diminishes with increase in size of the set**



$A = [S1, S2]$ $\quad$ $B = [S1, S2, S3]$

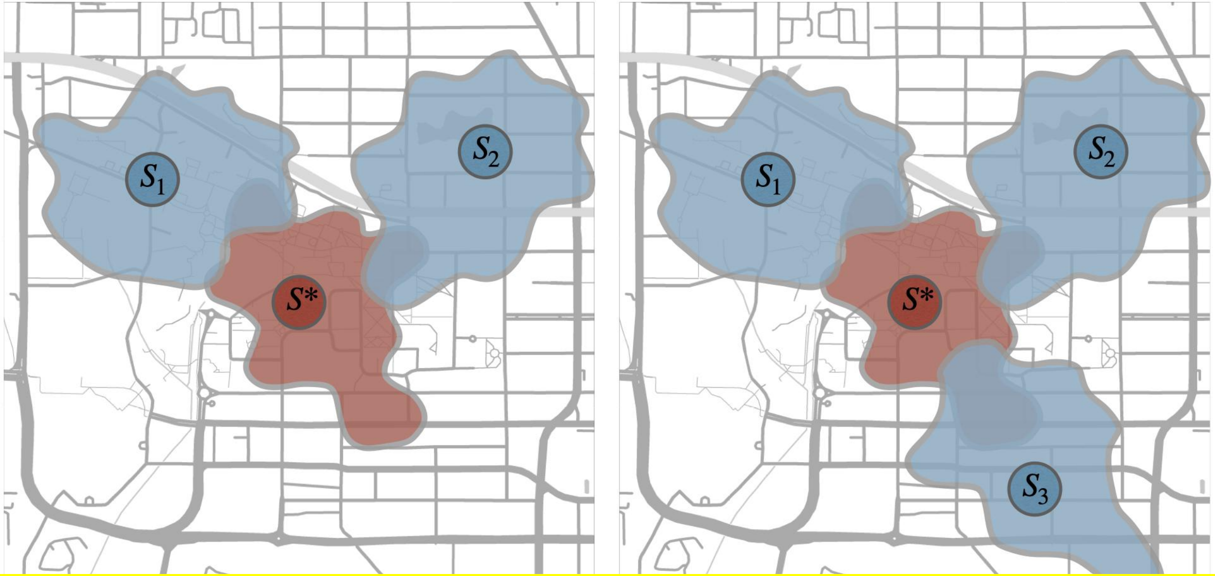
$$f(A \cup S^*) - f(A) \geq f(B \cup S^*) - f(B)$$

Monotonicity

A function $f : 2^V \to \mathbb{R}$ is monotone if :

- for every $A \subseteq B \subseteq V, f(A) \leq f(B)$
- Or, for every $A \subseteq V$ and $e \in V$
- it holds that $\Delta(e \mid A) \geq 0$

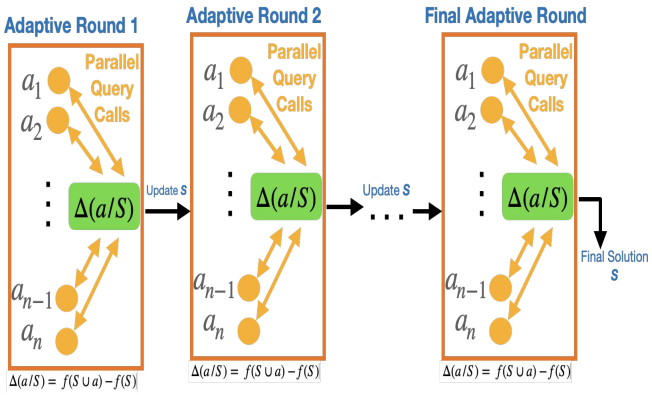
Submodular Maximization - Cardinality Constraint



Given a cardinality constraint ***k***
Objective: Maximize the coverage function with no more than k elements:
i.e **Maximize f(S), subject to |S| ≤ k**

Submodular Maximization - Performance Metrics

- **Metrics**
  - Approximation Ratio:
    - the minimal ratio of the solution to the optimal result
  - Query Complexity:
    - total number of query calls
  - Adaptivity:
    - Introduced by Balskanski and Singer[1] for submodular optimization
    - Defined as the minimal number of sequential rounds required to achieve a constant factor approximation when polynomially-many queries can be executed in parallel at each round.
    - It is the metric used to define how efficiently the algorithm can parallelize each iteration



[1] Eric Balkanski and Yaron Singer. **The adaptive complexity of maximizing a submodular function.** In ACM SIGACT Symposium on Theory of Computing (STOC), 2018.

Related Work

- Optimal ratio[1]: $1 - 1/e$
- Lower bound of query complexity[2]: $\Omega(n)$
- Lower bound of adaptivity[3]: $\Omega(\log(n)/\log\log(n))$

[1] G L Nemhauser and L A Wolsey. **Best Algorithms for Approximating the Maximum of a Submodular Set Function.** Mathematics of Operations Research, 3(3):177–188, 1978.
[2] Alan Kuhnle. **Quick Streaming Algorithms for Maximization of Monotone Submodular Functions in Linear Time.** In Artificial Intelligence and Statistics (AISTATS), 2021.
[3] Eric Balkanski and Yaron Singer. **The adaptive complexity of maximizing a submodular function.** In ACM SIGACT Symposium on Theory of Computing (STOC), 2018.

Related Work

- Several previous works get nearly theoretically optimal result
- Impractical with large constant factors

Reference	Ratio	Adaptivity	Queries
Ene and Nguyen [14]	$1 - 1/e - \varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(n\text{poly}(\log n, 1/\varepsilon)\right)$
Chekuri and Quanrud [11] (RPG)	$1 - 1/e - \varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$ †	$O\left(\frac{n}{\varepsilon^4}\log(n)\right)$ †
Fahrbach et al. [17] (BSM)	$1 - 1/e - \varepsilon$ †	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(\frac{n}{\varepsilon^3}\log\log k\right)$ †
Fahrbach et al. [17] (SM)	$1 - 1/e - \varepsilon$ †	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(\frac{n}{\varepsilon^3}\log(1/\varepsilon)\right)$ †
Breuer et al. [9] (FAST)	$1 - 1/e - \varepsilon$ † ‡	$O\left(\frac{1}{\varepsilon^2}\log(n)\log^2\left(\frac{\log(k)}{\varepsilon}\right)\right)$	$O\left(\frac{n}{\varepsilon^2}\log\left(\frac{\log(k)}{\varepsilon}\right)\right)$
LS+PGB [Theorem 3]	$1 - 1/e - \varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n/\varepsilon)\right)$	$O\left(\frac{n}{\varepsilon^2}\right)$ †

† indicates the result holds with constant probability or in expectation;
‡ indicates the result does not hold on all instances of SM;
while no symbol indicates the result holds with probability greater than 1-$O(1/n)$

Related Work - FAST[1]

- speed up the algorithms using the adaptive sequencing technique
- Sacrifice the theoretical guarantee
- Significantly, no ratio for k < 850

Reference	Ratio	Adaptivity	Queries
Ene and Nguyen [14]	$1-1/e-\varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(n\text{poly}(\log n, 1/\varepsilon)\right)$
Chekuri and Quanrud [11] (RPG)	$1-1/e-\varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$ †	$O\left(\frac{n}{\varepsilon^4}\log(n)\right)$ †
Fahrbach et al. [17] (BSM)	$1-1/e-\varepsilon$ †	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(\frac{n}{\varepsilon^3}\log\log k\right)$ †
Fahrbach et al. [17] (SM)	$1-1/e-\varepsilon$ †	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(\frac{n}{\varepsilon^3}\log(1/\varepsilon)\right)$ †
Breuer et al. [9] (FAST)	$1-1/e-\varepsilon$ † ‡	$O\left(\frac{1}{\varepsilon^2}\log(n)\log^2\left(\frac{\log(k)}{\varepsilon}\right)\right)$	$O\left(\frac{n}{\varepsilon^2}\log\left(\frac{\log(k)}{\varepsilon}\right)\right)$
LS+PGB [Theorem 3]	$1-1/e-\varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n/\varepsilon)\right)$	$O\left(\frac{n}{\varepsilon^2}\right)$ †

† indicates the result holds with constant probability or in expectation;
‡ indicates the result does not hold on all instances of SM;
while no symbol indicates the result holds with probability greater than $1-O(1/n)$

[1] Adam Breuer, Eric Balkanski, and Yaron Singer. The FAST Algorithm for Submodular Maximization. In International Conference on Machine Learning (ICML), 2019.

Our Main Algorithm - LS+PGB

- Provides theoretical guarantee for all *k* values
- Empirically outperforms, in terms of runtime, adaptivity, total queries, and objective values, the previous state-of-the-art algorithm FAST (Breuer et al.)

Reference	Ratio	Adaptivity	Queries
Ene and Nguyen [14]	$1-1/e-\varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(n\mathrm{poly}(\log n, 1/\varepsilon)\right)$
Chekuri and Quanrud [11] (RPG)	$1-1/e-\varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$ †	$O\left(\frac{n}{\varepsilon^4}\log(n)\right)$ †
Fahrbach et al. [17] (BSM)	$1-1/e-\varepsilon$ †	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(\frac{n}{\varepsilon^3}\log\log k\right)$ †
Fahrbach et al. [17] (SM)	$1-1/e-\varepsilon$ †	$O\left(\frac{1}{\varepsilon^2}\log(n)\right)$	$O\left(\frac{n}{\varepsilon^3}\log(1/\varepsilon)\right)$ †
Breuer et al. [9] (FAST)	$1-1/e-\varepsilon$ † ‡	$O\left(\frac{1}{\varepsilon^2}\log(n)\log^2\left(\frac{\log(k)}{\varepsilon}\right)\right)$	$O\left(\frac{n}{\varepsilon^2}\log\left(\frac{\log(k)}{\varepsilon}\right)\right)$
LS+PGB [Theorem 3]	$1-1/e-\varepsilon$	$O\left(\frac{1}{\varepsilon^2}\log(n/\varepsilon)\right)$	$O\left(\frac{n}{\varepsilon^2}\right)$ †

† indicates the result holds with constant probability or in expectation;
‡ indicates the result does not hold on all instances of SM;
while no symbol indicates the result holds with probability greater than 1-$O(1/n)$

Contributions

- **LinearSeq**: obtains a constant ratio $(4+O(\varepsilon))^{-1}$ in expected $O(n)$ query complexity and $O(\log(n))$ adaptivity with probability $1-1/n$
  - Modified: achieves $O(\log(n/k))$ adaptivity with sacrificing the ratio to be $(5+O(\varepsilon))^{-1}$

Contributions

- **LinearSeq**: obtains a constant facto $(4+O(\varepsilon))^{-1}$ in expected $O(n)$ query complexity and $O(\log(n))$ adaptivity with probability $1-1/n$
    - Modified: achieves $O(\log(n/k))$ adaptivity with sacrificing the ratio to be $(5+O(\varepsilon))^{-1}$
- **ThresholdSeq**: the average marginal gain is larger than a specified threshold with probability $1-1/n$ in expected $O(n)$ query complexity and adaptive rounds $O(\log(n))$

Contributions

- **LinearSeq**: obtains a constant facto $(4+O(\varepsilon))^{-1}$ in expected $O(n)$ query complexity and $O(\log(n))$ adaptivity with probability $1-1/n$
  - Modified: achieves $O(\log(n/k))$ adaptivity with sacrificing the ratio to be $(5+O(\varepsilon))^{-1}$
- **ThresholdSeq**: the average marginal gain is larger than a specified threshold with probability $1-1/n$ in expected $O(n)$ query complexity and adaptive rounds $O(\log(n))$
- **LS+PGB**: uses LinearSeq as preprocessing algorithm and combines ThresholdSeq with boost mechanism; obtains nearly the optimal result

Highly adaptive linear-time algorithm - ¼ ratio

Algorithm 2 Highly Adaptive Linear-Time Algorithm

Input: evaluation oracle $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}^{+}$, constraint k
Initialize $A \leftarrow \emptyset$
for $u \in \mathcal{N}$ **do**
  if $\Delta(u \mid A) \geq f(A)/k$ **then**
    $A \leftarrow A \cup \{u\}$
return $A' \leftarrow \{\text{last } k \text{ elements added to } A\}$

- Use $f(A)/k$ as threshold to ensure that:
  - The last k elements in A contain a constant fraction of the value $f(A)$
  - $f(A)$ is within a constant fraction of OPT
- Totally $2n$ query calls and n adaptive rounds

Highly adaptive linear-time algorithm - ¼ ratio

Algorithm 2 Highly Adaptive Linear-Time Algorithm

Input: evaluation oracle $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}^{+}$, constraint k
Initialize $A \leftarrow \emptyset$
for $u \in \mathcal{N}$ **do**
  if $\Delta(u \mid A) \geq f(A)/k$ **then**
    $A \leftarrow A \cup \{u\}$
return $A' \leftarrow \{\text{last } k \text{ elements added to } A\}$

Problem: How to parallelize this algorithm to a lowly adaptive version without loss much of approximation ratio and query complexity

- Use *f*(*A*)/*k* as threshold to ensure that:
  - The last *k* elements in *A* contain a constant fraction of the value *f*(*A*)
  - *f*(*A*) is within a constant fraction of OPT
- Totally 2*n* query calls and *n* adaptive rounds

LinearSeq

- ¼ ratio, *O(n)* query and *O(log n)* adaptive complexity

Algorithm 1 The algorithm that obtains ratio $(4 + O(\varepsilon))^{-1}$ in $O\left(\log(n)/\varepsilon^3\right)$ adaptive rounds and expected $O\left(n/\varepsilon^3\right)$ queries.

procedure LinearSeq($f, \mathcal{N}, k, \varepsilon$)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, error ε
  $a = \arg\max_{u \in \mathcal{N}} f(\{u\})$
  Initialize $A \leftarrow \{a\}$, $V \leftarrow \mathcal{N}$, $\ell = \lceil 4(1 + 1/(\beta\varepsilon)) \log(n) \rceil$, $\beta = \varepsilon/(16 \log(8/(1 - e^{-\varepsilon/2})))$
  for $j \leftarrow 1$ to ℓ **do**
    Update $V \leftarrow \{x \in V : \Delta(x \mid A) \geq f(A)/k\}$ and filter out the rest
    if $|V| = 0$ **then break**
    $V = \{v_1, v_2, \ldots, v_{|V|}\} \leftarrow$ **random-permutation**(V)
    $\Lambda \leftarrow \{\lfloor (1+\varepsilon)^u \rfloor : 1 \leq \lfloor (1+\varepsilon)^u \rfloor \leq k, u \in \mathbb{N}\}$
      $\cup \{\lfloor k + u\varepsilon k \rfloor : \lfloor k + u\varepsilon k \rfloor \leq |V|, u \in \mathbb{N}\} \cup \{|V|\}$
    $B[\lambda_i] =$ **false**, for $\lambda_i \in \Lambda$
    for $\lambda_i \in \Lambda$ in parallel **do**
      $T_{\lambda_{i-1}} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_{i-1}}\}$; $T_{\lambda_i} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_i}\}$; $T'_{\lambda_i} \leftarrow T_{\lambda_i} \backslash T_{\lambda_{i-1}}$
      if $\Delta\left(T'_{\lambda_i} \mid A \cup T_{\lambda_{i-1}}\right) / |T'_{\lambda_i}| \geq (1-\varepsilon) f(A \cup T_{\lambda_{i-1}})/k$ **then** $B[\lambda_i] \leftarrow$ **true**
    $\lambda^* \leftarrow \max\{\lambda_i \in \Lambda : B[\lambda_i] =$ **false** and (($\lambda_i \leq k$ and $B[1]$ to $B[\lambda_{i-1}]$ are all **true**) or ($\lambda_i > k$ and $\exists m \geq 1$ s.t. $|\bigcup_{u=m}^{i-1} T'_{\lambda_u}| \geq k$ and $B[\lambda_m]$ to $B[\lambda_{i-1}]$ are all **true**))$\}$
    $A \leftarrow A \cup T_{\lambda^*}$
  if $|V| > 0$ **then return** *failure*
  return $A' \leftarrow$ last k elements added to A

- filter out the elements with small marginal gains

LinearSeq

- ¼ ratio, *O(n)* query and *O(log n)* adaptive complexity

Algorithm 1 The algorithm that obtains ratio $(4 + O(\varepsilon))^{-1}$ in $O\left(\log(n)/\varepsilon^3\right)$ adaptive rounds and expected $O\left(n/\varepsilon^3\right)$ queries.

procedure LINEARSEQ($f, \mathcal{N}, k, \varepsilon$)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, error ε
  $a = \arg\max_{u \in \mathcal{N}} f(\{u\})$
  Initialize $A \leftarrow \{a\}$, $V \leftarrow \mathcal{N}$, $\ell = \lceil 4(1 + 1/(\beta\varepsilon)) \log(n) \rceil$, $\beta = \varepsilon/(16 \log(8/(1 - e^{-\varepsilon/2})))$
  for $j \leftarrow 1$ to ℓ **do**
    Update $V \leftarrow \{x \in V : \Delta(x \mid A) \geq f(A)/k\}$ and filter out the rest
    if $|V| = 0$ **then break**
    $V = \{v_1, v_2, \ldots, v_{|V|}\} \leftarrow$ **random-permutation**(V)
    $\Lambda \leftarrow \{\lfloor (1+\varepsilon)^u \rfloor : 1 \leq \lfloor (1+\varepsilon)^u \rfloor \leq k, u \in \mathbb{N}\}$
      $\cup \{\lfloor k + u\varepsilon k \rfloor : \lfloor k + u\varepsilon k \rfloor \leq |V|, u \in \mathbb{N}\} \cup \{|V|\}$
    $B[\lambda_i] =$ **false**, for $\lambda_i \in \Lambda$
    for $\lambda_i \in \Lambda$ in parallel **do**
      $T_{\lambda_{i-1}} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_{i-1}}\}$; $T_{\lambda_i} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_i}\}$; $T'_{\lambda_i} \leftarrow T_{\lambda_i} \setminus T_{\lambda_{i-1}}$
      if $\Delta\left(T'_{\lambda_i} \mid A \cup T_{\lambda_{i-1}}\right)/|T'_{\lambda_i}| \geq (1-\varepsilon) f(A \cup T_{\lambda_{i-1}})/k$ **then** $B[\lambda_i] \leftarrow$ **true**
    $\lambda^* \leftarrow \max\{\lambda_i \in \Lambda : B[\lambda_i] =$ **false** and $((\lambda_i \leq k$ and $B[1]$ to $B[\lambda_{i-1}]$ are all **true**$)$ or $(\lambda_i > k$ and $\exists m \geq 1$ s.t. $|\bigcup_{u=m}^{i-1} T'_{\lambda_u}| \geq k$ and $B[\lambda_m]$ to $B[\lambda_{i-1}]$ are all **true**$))\}$
    $A \leftarrow A \cup T_{\lambda^*}$
  if $|V| > 0$ **then return** *failure*
  return $A' \leftarrow$ last k elements added to A

- get a randomly selected sequence

LinearSeq

- ¼ ratio, *O(n)* query and *O(log n)* adaptive complexity

Algorithm 1 The algorithm that obtains ratio $(4 + O(\varepsilon))^{-1}$ in $O\left(\log(n)/\varepsilon^3\right)$ adaptive rounds and expected $O\left(n/\varepsilon^3\right)$ queries.

procedure LINEARSEQ($f, \mathcal{N}, k, \varepsilon$)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}^+$, constraint k, error ε
  $a = \arg\max_{u \in \mathcal{N}} f(\{u\})$
  Initialize $A \leftarrow \{a\}$, $V \leftarrow \mathcal{N}$, $\ell = \lceil 4(1 + 1/(\beta\varepsilon)) \log(n) \rceil$, $\beta = \varepsilon/(16 \log(8/(1 - e^{-\varepsilon/2})))$
  for $j \leftarrow 1$ to ℓ **do**
    Update $V \leftarrow \{x \in V : \Delta(x \mid A) \geq f(A)/k\}$ and filter out the rest
    if $|V| = 0$ **then break**
    $V = \{v_1, v_2, \ldots, v_{|V|}\} \leftarrow$ **random-permutation**(V)
    $\Lambda \leftarrow \{\lfloor (1+\varepsilon)^u \rfloor : 1 \leq \lfloor (1+\varepsilon)^u \rfloor \leq k, u \in \mathbb{N}\}$
      $\cup \{\lfloor k + u\varepsilon k \rfloor : \lfloor k + u\varepsilon k \rfloor \leq |V|, u \in \mathbb{N}\} \cup \{|V|\}$
    $B[\lambda_i] =$ **false**, for $\lambda_i \in \Lambda$
    for $\lambda_i \in \Lambda$ in parallel **do**
      $T_{\lambda_{i-1}} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_{i-1}}\}$; $T_{\lambda_i} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_i}\}$; $T'_{\lambda_i} \leftarrow T_{\lambda_i} \backslash T_{\lambda_{i-1}}$
      if $\Delta\left(T'_{\lambda_i} \mid A \cup T_{\lambda_{i-1}}\right) / |T'_{\lambda_i}| \geq (1-\varepsilon) f(A \cup T_{\lambda_{i-1}})/k$ **then** $B[\lambda_i] \leftarrow$ **true**
    $\lambda^* \leftarrow \max\{\lambda_i \in \Lambda : B[\lambda_i] =$ **false** and $((\lambda_i \leq k$ and $B[1]$ to $B[\lambda_{i-1}]$ are all **true**$)$ or $(\lambda_i > k$ and $\exists m \geq 1$ s.t. $|\bigcup_{u=m}^{i-1} T'_{\lambda_u}| \geq k$ and $B[\lambda_m]$ to $B[\lambda_{i-1}]$ are all **true**$))\}$
    $A \leftarrow A \cup T_{\lambda^*}$
  if $|V| > 0$ **then return** *failure*
  return $A' \leftarrow$ last k elements added to A

- split *V* into blocks to reduce the query calls

LinearSeq

- ¼ ratio, *O(n)* query and *O(log n)* adaptive complexity

Algorithm 1 The algorithm that obtains ratio $(4+O(\varepsilon))^{-1}$ in $O\left(\log(n)/\varepsilon^3\right)$ adaptive rounds and expected $O\left(n/\varepsilon^3\right)$ queries.

---

procedure LINEARSEQ($f, \mathcal{N}, k, \varepsilon$)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, error ε
  $a = \arg\max_{u \in \mathcal{N}} f(\{u\})$
  Initialize $A \leftarrow \{a\}$, $V \leftarrow \mathcal{N}$, $\ell = \lceil 4(1+1/(\beta\varepsilon))\log(n) \rceil$, $\beta = \varepsilon/(16\log(8/(1-e^{-\varepsilon/2})))$
  for $j \leftarrow 1$ to ℓ **do**
    Update $V \leftarrow \{x \in V : \Delta(x \mid A) \geq f(A)/k\}$ and filter out the rest
    if $|V| = 0$ **then break**
    $V = \{v_1, v_2, \ldots, v_{|V|}\} \leftarrow$**random-permutation**$(V)$
    $\Lambda \leftarrow \{\lfloor(1+\varepsilon)^u\rfloor : 1 \leq \lfloor(1+\varepsilon)^u\rfloor \leq k, u \in \mathbb{N}\}$
      $\cup\{\lfloor k + u\varepsilon k\rfloor : \lfloor k + u\varepsilon k\rfloor \leq |V|, u \in \mathbb{N}\} \cup \{|V|\}$
    $B[\lambda_i] =$ **false**, for $\lambda_i \in \Lambda$
    for $\lambda_i \in \Lambda$ in parallel **do**
      $T_{\lambda_{i-1}} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_{i-1}}\}$; $T_{\lambda_i} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_i}\}$; $T'_{\lambda_i} \leftarrow T_{\lambda_i} \backslash T_{\lambda_{i-1}}$
      if $\Delta\left(T'_{\lambda_i} \mid A \cup T_{\lambda_{i-1}}\right) / |T'_{\lambda_i}| \geq (1-\varepsilon) f(A \cup T_{\lambda_{i-1}})/k$ **then** $B[\lambda_i] \leftarrow$ **true**
    $\lambda^* \leftarrow \max\{\lambda_i \in \Lambda : B[\lambda_i] =$ **false** and $((\lambda_i \leq k$ and $B[1]$ to $B[\lambda_{i-1}]$ are all **true**$)$ or
  $(\lambda_i > k$ and $\exists m \geq 1$ s.t. $|\bigcup_{u=m}^{i-1} T'_{\lambda_u}| \geq k$ and $B[\lambda_m]$ to $B[\lambda_{i-1}]$ are all **true**$))\}$
    $A \leftarrow A \cup T_{\lambda^*}$
  if $|V| > 0$ **then return** *failure*
  return $A' \leftarrow$ last k elements added to A

- check the marginal gain of each block in parallel

LinearSeq

- ¼ ratio, *O(n)* query and *O(log n)* adaptive complexity

Algorithm 1 The algorithm that obtains ratio $(4 + O(\varepsilon))^{-1}$ in $O\left(\log(n)/\varepsilon^3\right)$ adaptive rounds and expected $O\left(n/\varepsilon^3\right)$ queries.

procedure LINEARSEQ$(f, \mathcal{N}, k, \varepsilon)$

  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, error ε

  $a = \arg\max_{u \in \mathcal{N}} f(\{u\})$

  Initialize $A \leftarrow \{a\}$, $V \leftarrow \mathcal{N}$, $\ell = \lceil 4(1 + 1/(\beta\varepsilon)) \log(n) \rceil$, $\beta = \varepsilon/(16 \log(8/(1 - e^{-\varepsilon/2})))$

  for $j \leftarrow 1$ to ℓ **do**

    Update $V \leftarrow \{x \in V : \Delta(x \mid A) \geq f(A)/k\}$ and filter out the rest

    if $|V| = 0$ **then break**

    $V = \{v_1, v_2, \ldots, v_{|V|}\} \leftarrow$ **random-permutation**(V)

    $\Lambda \leftarrow \{\lfloor (1 + \varepsilon)^u \rfloor : 1 \leq \lfloor (1 + \varepsilon)^u \rfloor \leq k, u \in \mathbb{N}\}$
      $\cup \{\lfloor k + u\varepsilon k \rfloor : \lfloor k + u\varepsilon k \rfloor \leq |V|, u \in \mathbb{N}\} \cup \{|V|\}$

    $B[\lambda_i] =$ **false**, for $\lambda_i \in \Lambda$

    for $\lambda_i \in \Lambda$ in parallel **do**

      $T_{\lambda_{i-1}} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_{i-1}}\}$; $T_{\lambda_i} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_i}\}$; $T'_{\lambda_i} \leftarrow T_{\lambda_i} \backslash T_{\lambda_{i-1}}$

      if $\Delta\left(T'_{\lambda_i} \mid A \cup T_{\lambda_{i-1}}\right) / |T'_{\lambda_i}| \geq (1 - \varepsilon) f(A \cup T_{\lambda_{i-1}})/k$ **then** $B[\lambda_i] \leftarrow$ **true**

    $\lambda^* \leftarrow \max\{\lambda_i \in \Lambda : B[\lambda_i] =$ **false** and $((\lambda_i \leq k$ and $B[1]$ to $B[\lambda_{i-1}]$ are all **true**$)$ or $(\lambda_i > k$ and $\exists m \geq 1$ s.t. $|\bigcup_{u=m}^{i-1} T'_{\lambda_u}| \geq k$ and $B[\lambda_m]$ to $B[\lambda_{i-1}]$ are all **true**$))\}$

    $A \leftarrow A \cup T_{\lambda^*}$

  if $|V| > 0$ **then return** *failure*

  return $A' \leftarrow$ last k elements added to A

- prefix selection
  - ensure that the selected subset obtains large marginal gain
  - ensure that an ε/2-fraction of *V* can be filtered out at next iteration with high probability

ThresholdSeq

- *O(n)* query and *O(log n)* adaptive complexity

Algorithm 3 A Parallelizable Greedy Algorithm for Fixed Threshold τ

procedure ThresholdSeq($f, \mathcal{N}, k, \delta, \varepsilon, \tau$)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, revision δ, error ε, threshold τ
  Initialize $A \leftarrow \emptyset$, $V \leftarrow \mathcal{N}$, $\ell = \lceil 4(1+2/\varepsilon)\log(n/\delta)\rceil$
  for $j \leftarrow 1$ to ℓ **do**
    Update $V \leftarrow \{x \in V : \Delta(x \mid A) \geq \tau\}$ and filter out the rest
    if $|V| = 0$ **then**
      return A
    $V \leftarrow$ **random-permutation**(V).
    $s \leftarrow \min\{k - |A|, |V|\}$
    $\Lambda \leftarrow \{\lfloor (1+\varepsilon)^u \rfloor : 1 \leq \lfloor (1+\varepsilon)^u \rfloor \leq s, u \in \mathbb{N}\} \cup \{s\}$
    $B \leftarrow \emptyset$
    for $\lambda_i \in \Lambda$ in parallel **do**
      $T_{\lambda_i} \leftarrow \{v_1, v_2, \ldots, v_{\lambda_i}\}$
      if $\Delta(T_{\lambda_i} \mid A) / |T_{\lambda_i}| \geq (1-\varepsilon)\tau$ **then**
        $B \leftarrow B \cup \{\lambda_i\}$
    $\lambda^* \leftarrow \min\{\lambda_i \in \Lambda : \lambda_i > b, \forall b \in B\}$
    $A \leftarrow A \cup T_{\lambda^*}$
    if $|A| = k$ **then**
      return A
  return *failure*

- analogous to LinearSeq
- $f(A)/|A| \geq (1-\varepsilon)\tau/(1+\varepsilon)$
- constant threshold τ
- much simpler while only consider the first min{*k*-|*A*|,|*V*|} elements
- stop when |*A*|=*k*

LS+PGB
- 1-1/e ratio, *O(n)* query and *O(log n)* adaptive complexity

Algorithm 4 A Parallelizable Greedy Algorithm to Boost to the Optimal Ratio.

procedure $\textsc{ParallelGreedyBoost}(f, \mathcal{N}, k, \alpha, \Gamma, \varepsilon)$
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, constant α, value Γ such that $\Gamma \leq f(O) \leq \Gamma/\alpha$, error ε
  Initialize $\tau \leftarrow \Gamma/(\alpha k)$, $\delta \leftarrow 1/(\log_{1-\varepsilon}(\alpha/3) + 1)$, $A \leftarrow \emptyset$
  while $\tau \geq \Gamma/(3k)$ **do**
    $\tau \leftarrow \tau(1-\varepsilon)$
    $S \leftarrow \textsc{ThresholdSeq}(f_A, \mathcal{N}, k - |A|, \delta, \varepsilon/3, \tau)$
    $A \leftarrow A \cup S$
    if $|A| = k$ **then**
      return A
  return A

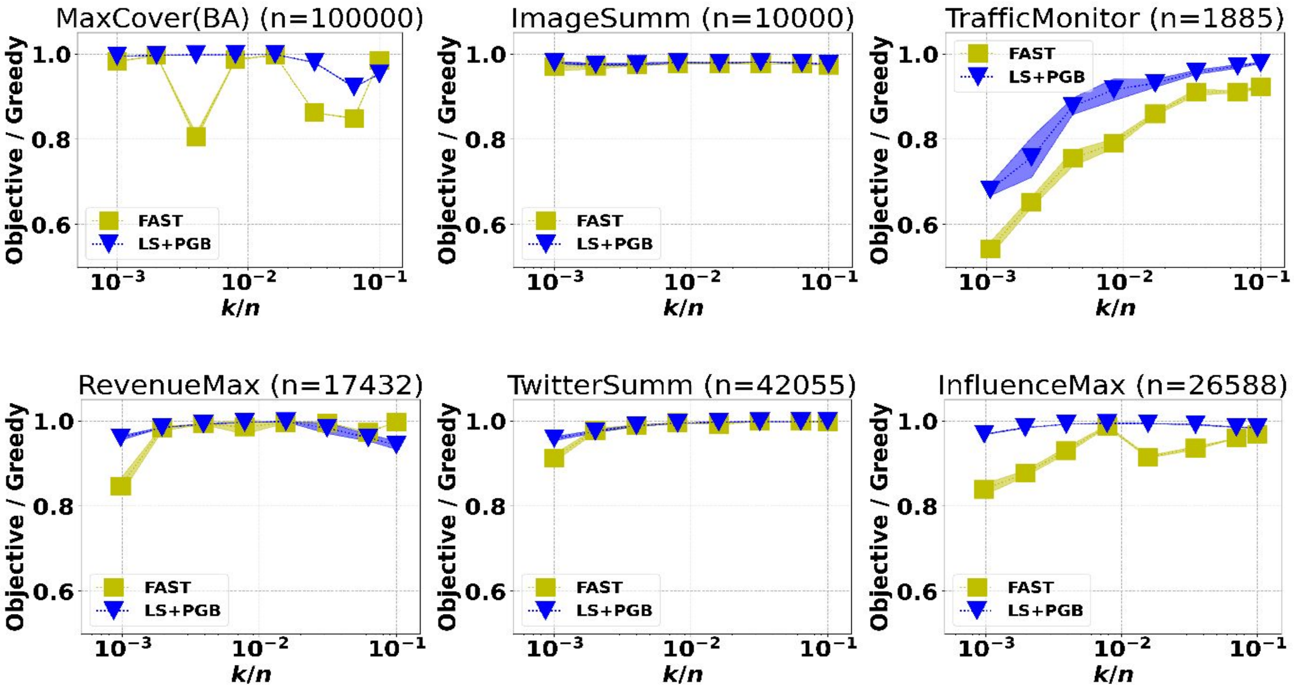
- Use LinearSeq as preprocessing algorithm to get an α-approximation solution Γ
- Γ and α are used to produce an initial threshold value τ
- The threshold value is iteratively decreased by the factor (1-ε)

Empirical Results - Environment Setup

- The experiments are conducted on a server running **Ubuntu 20.04.2** with kernel **5.8.0**
- The hardware of the system consists of **40 Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz** cores with **75 threads** made available to the algorithms for the experiments.
- All algorithms were implemented using **Open-MPI** and **mpi4py** library
- The experiments were performed across **six applications** with **groundset size ranging from 1,885 - 100,000** and **K values ranging** from **0.1% - 10% of groundset.**
- For evaluation, the metrics of **total time, total queries, adaptive rounds** and **objective value** were used for comparison.
- Our algorithm (**LS+PGB**) is compared to the previous state-of-art algorithm **FAST**[1]**.**

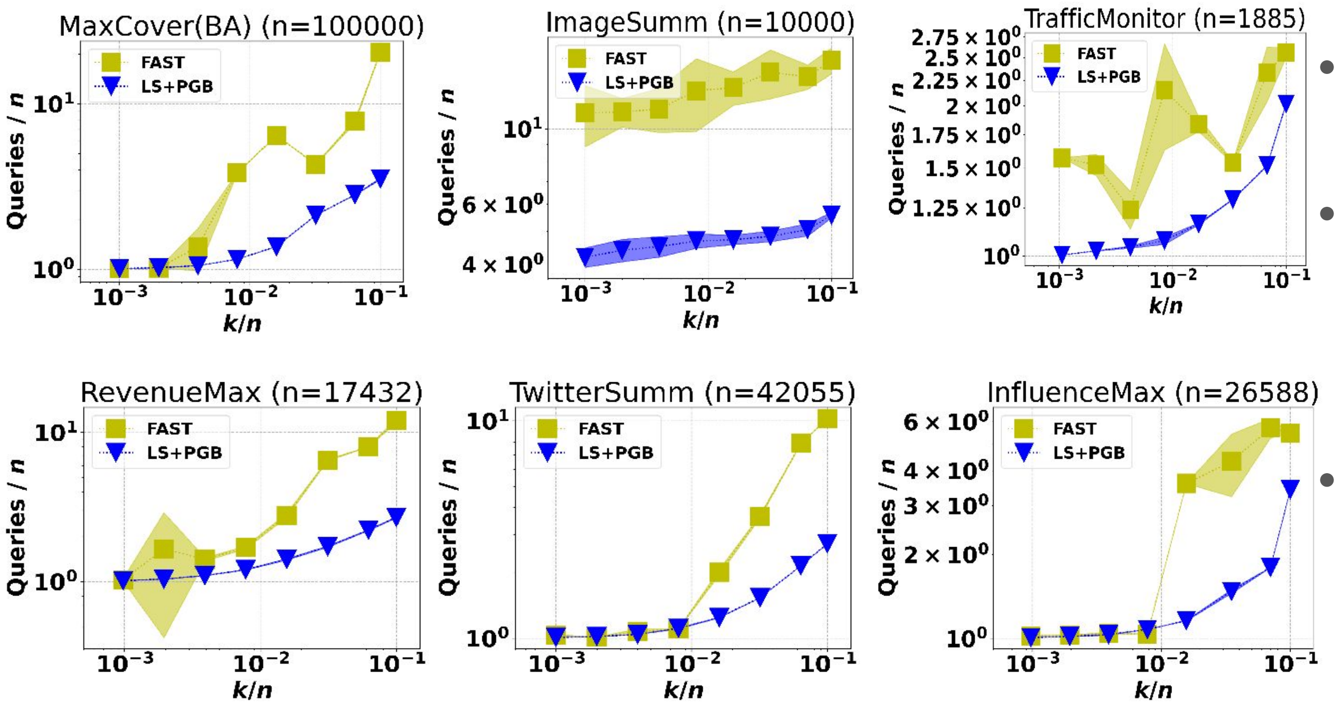
[1] **Adam Breuer, Eric Balkanski, and Yaron Singer. The FAST Algorithm for Submodular Maximization. In International Conference on Machine Learning (ICML), 2019.**

Empirical Results - Objective Value



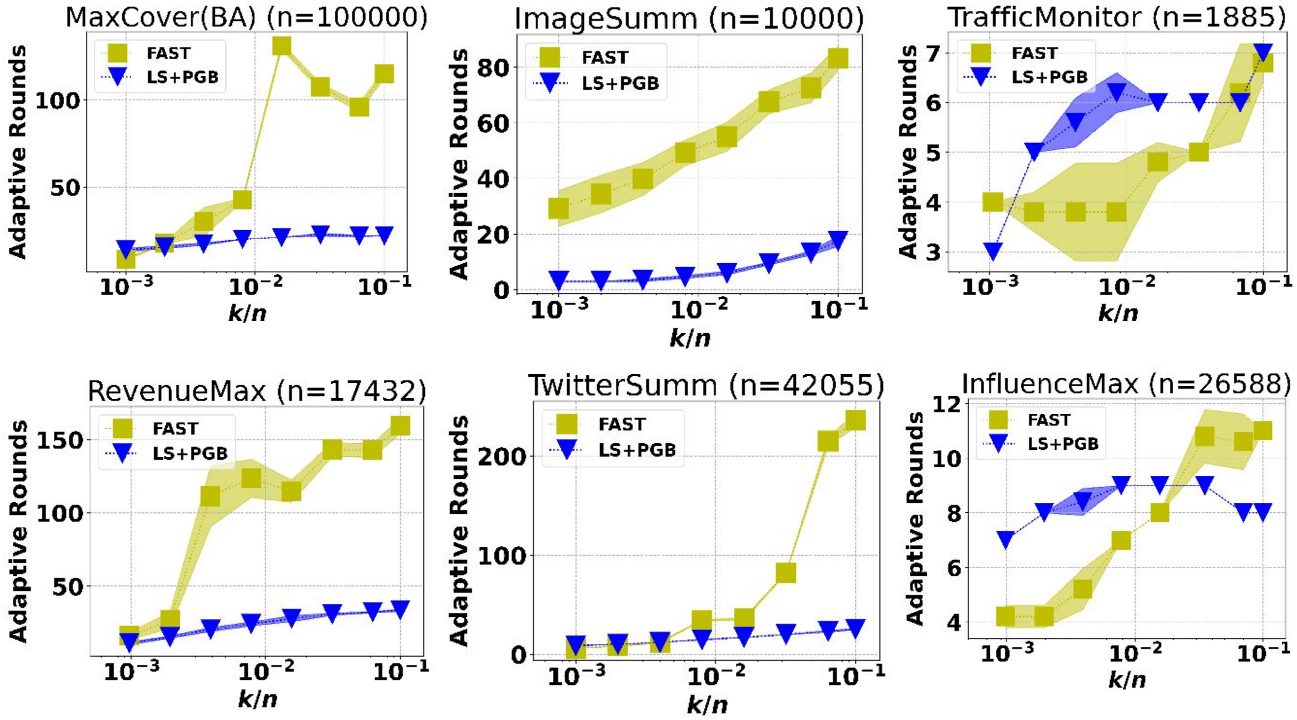
- The objective value is normalized by that of **Greedy**
- Overall **LS+PGB** either maintains or outperforms the objective obtained by **FAST** across all applications
- With the **TrafficMonitor** and **MaxCover (BA)** being the instances where it **exceeds the average objective value of FAST by 6% and 5% respectively**.

Empirical Results - Queries



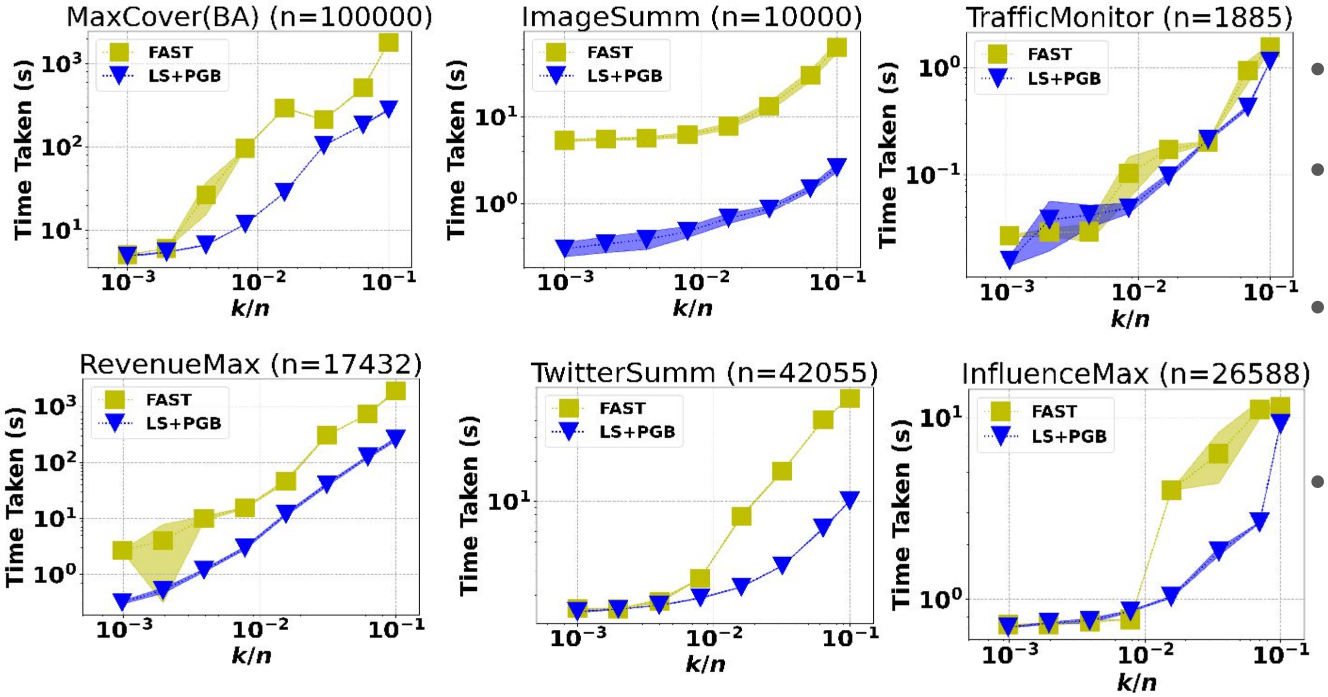
- Both **FAST** and **LS+PGB** exhibit a linear scaling behavior with the increasing k values
- Overall on an average **LS+PGB** achieves the objective in **less than half the total queries required by FAST** for all of the applications but TrafficMonitor and InfluenceMax.
- For **TrafficMonitor** & **InfluenceMax**, **FAST** requires **1.5 and 1.9 times** the **queries needed by LS+PGB**

Empirical Results - Adaptive Rounds



- **LS+PGB** exhibits a very good scaling behavior with the increasing k values **with at most 5 fold increase in adaptive rounds with 100 fold increase in k value**.
- Overall on an average **FAST** requires more than **3.5 times** the **adaptive rounds needed by LS+PGB** to achieve the objective.
- For **MaxCover, RevenueMax, TwitterSumm** and **ImageSumm** FAST requires **3.5, 4.3, 4.8** and **7.2 times more adaptive rounds.**

Empirical Results - Time Taken



- Both algorithms exhibit **linear scaling of runtime with k**
- On many instances, **LS+PGB is faster by more than an order of magnitude**
- Overall on an average **FAST** requires almost **4.8 times** the **time needed by LS+PGB** to achieve the objective.
- For **TwitterSumm, MaxCover, RevenueMax** and **ImageSumm** FAST is on average **4.6, 4.7, 6.8** and **19.2 times slower than** LS+PGB.

Empirical Results - Overall Result

Application	Runtime (s)		Objective Value		Queries	
	FAST	LS+PGB	FAST	LS+PGB	FAST	LS+PGB
TrafficMonitor	3.7×10^{-1}	$\mathbf{2.1 \times 10^{-1}}$	4.7×10^{8}	$\mathbf{5.0 \times 10^{8}}$	3.5×10^{3}	$\mathbf{2.4 \times 10^{3}}$
InfluenceMax	4.4×10^{0}	$\mathbf{2.3 \times 10^{0}}$	1.1×10^{3}	1.1×10^{3}	7.7×10^{4}	$\mathbf{4.0 \times 10^{4}}$
TwitterSumm	1.6×10^{1}	$\mathbf{3.5 \times 10^{0}}$	3.8×10^{5}	3.8×10^{5}	1.5×10^{5}	$\mathbf{6.2 \times 10^{4}}$
RevenueMax	3.9×10^{2}	$\mathbf{5.4 \times 10^{1}}$	1.4×10^{4}	1.4×10^{4}	7.6×10^{4}	$\mathbf{2.7 \times 10^{4}}$
MaxCover (BA)	3.7×10^{2}	$\mathbf{7.6 \times 10^{1}}$	6.0×10^{4}	$\mathbf{6.3 \times 10^{4}}$	5.8×10^{5}	$\mathbf{1.8 \times 10^{5}}$
ImageSumm	1.6×10^{1}	$\mathbf{8.1 \times 10^{-1}}$	9.1×10^{3}	9.1×10^{3}	1.3×10^{5}	$\mathbf{4.8 \times 10^{4}}$

Conclusion

- In this paper, we made the following contributions:
  - Theoretical
    - **LinearSeq**: A constant-factor algorithm for SM with smaller adaptivity than any previous algorithm, especially for values of *k* that are large relative to *n*.
    - **ThresholdSeq**: An algorithm that adds elements that have a gain of a specified threshold with expected linear query complexity and logarithmic adaptive rounds.
    - **LS+PGB**: An parallelized greedy algorithm which is used in conjunction with LinearSeq and ThreshoulSeq. It obtains nearly the optimal result, in terms of ratio, adaptivity and query complexity.
  - Empirical
    - LS+PGB is faster than the state-of-art algorithm FAST in an extensive empirical evaluation.