The Power of Randomization: Distributed Submodular Maximization on Massive Datasets

11/08/2016

Sekiya Akira

Selected Paper

The Power of Randomization

Distributed Submodular Maximization on Massive Datasets*

Rafael da Ponte Barbosa[1], Alina Ene[1], Huy L. Nguyễn[2], and Justin Ward[†1]

[1]Department of Computer Science and DIMAP
University of Warwick
{rafael, A.Ene, J.D.Ward}@dcs.warwick.ac.uk
[2]Simons Institute
University of California, Berkeley
hlnguyen@cs.princeton.edu

April 23, 2015

- ICML, 2015

Background - distribution

- Because computers have limited amount of memory, when we want to solve larger problems, we need to distribute them
- In MapReduce model, each machines can only communicate and exchange data during the shuffle phage

Background - sub modular functions

- Wide variety of problems in machine learning / image clustering / sensor placement can be cast as sub modular function maximization
  - These problems sometimes too large to be solved on a single machine

Submodular Maximization

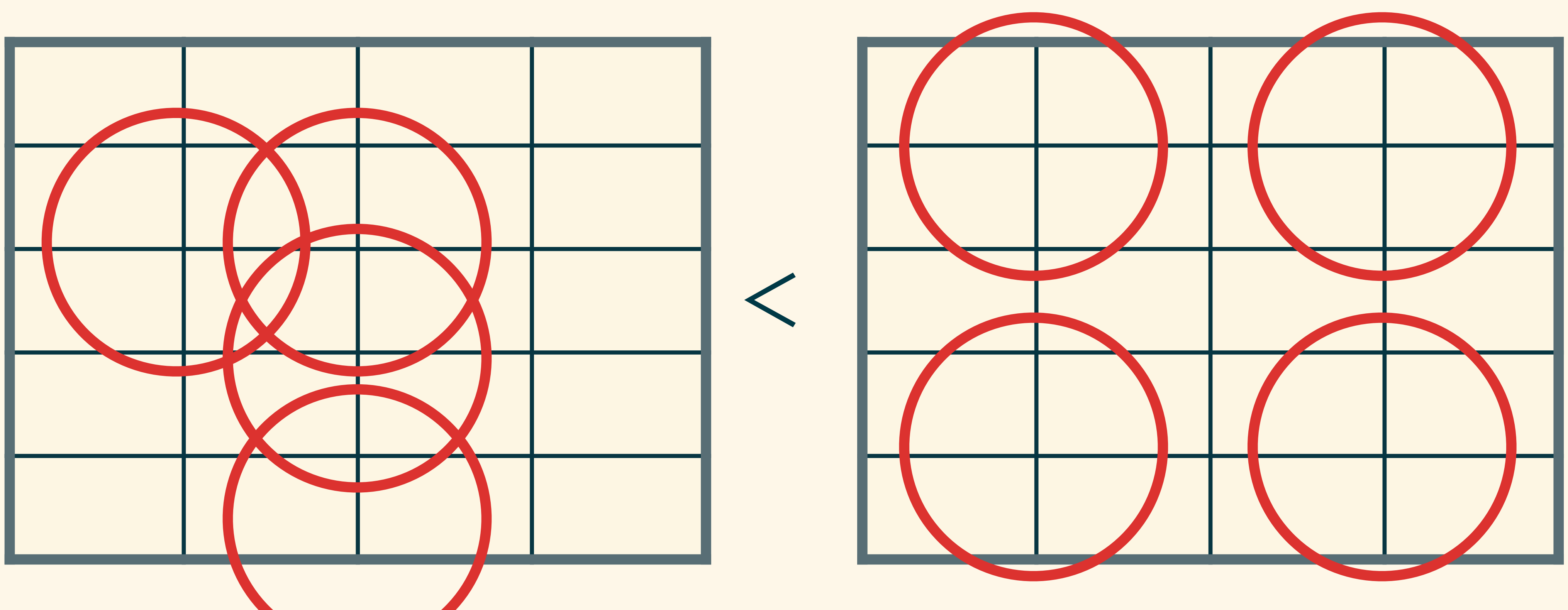
- Submodular function (劣モジュラ関数)
  - A set function $f : 2^V \to \mathbb{R}$ where For every $S, T \subseteq V$, $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$
  - i.e. for all $A \subseteq B \subseteq V, e \notin B$,

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

  - “A set function that the difference in the incremental value decreases as the size of the input set increases”

Submodular Maximization

- Maximization of Submodular function
- $$\underset{A \in C \subseteq 2^V}{\operatorname{argmax}} f(A)$$
  where C is the family of feasible solutions
- e.g. Sensor Placement Problem
  - Place sensor to measure the temperature of a board



Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S's dependency

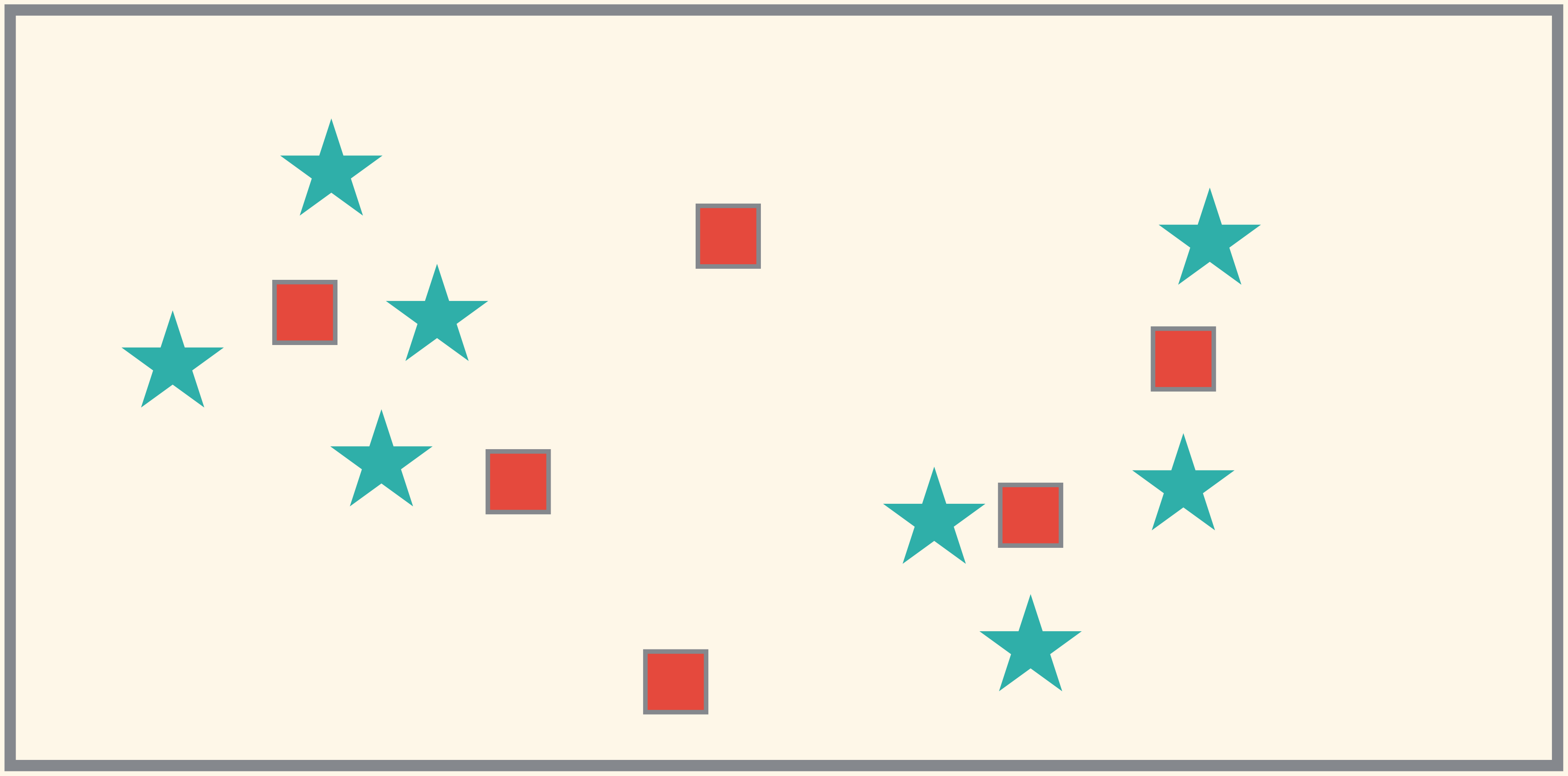
Algorithm 1 The standard greedy algorithm GREEDY

$S \leftarrow \emptyset$
loop
  Let $C = \{e \in V \setminus S : S \cup \{e\} \in \mathcal{I}\}$
  Let $e = \arg\max_{e \in C}\{f(S \cup \{e\}) - f(S)\}$
  if $C = \emptyset$ or $f(S \cup \{e\}) - f(S) < 0$ **then**
    return S
  end if
end loop

$S \leftarrow S \cup \{e\}$

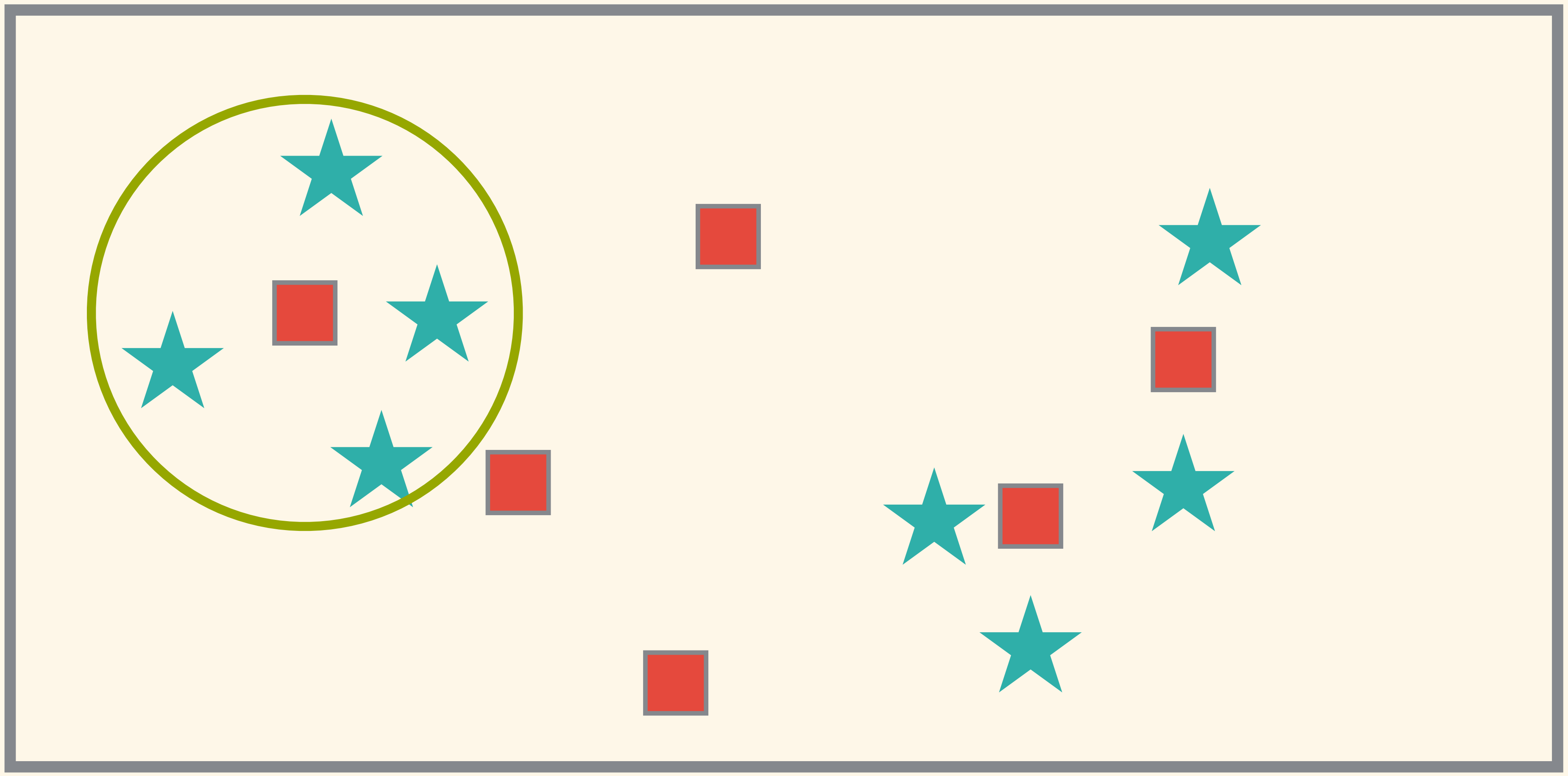
Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S's dependency



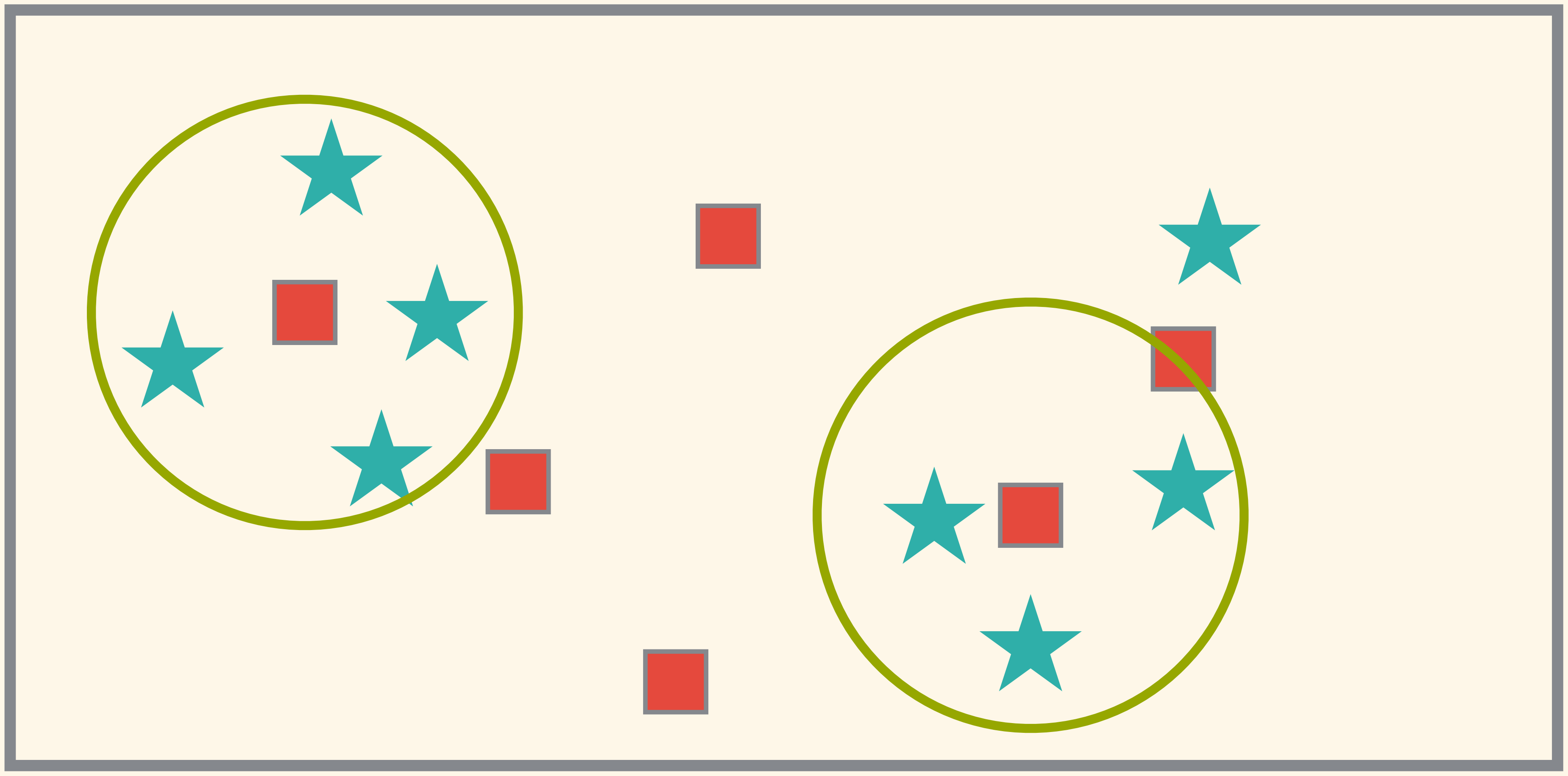
Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S's dependency



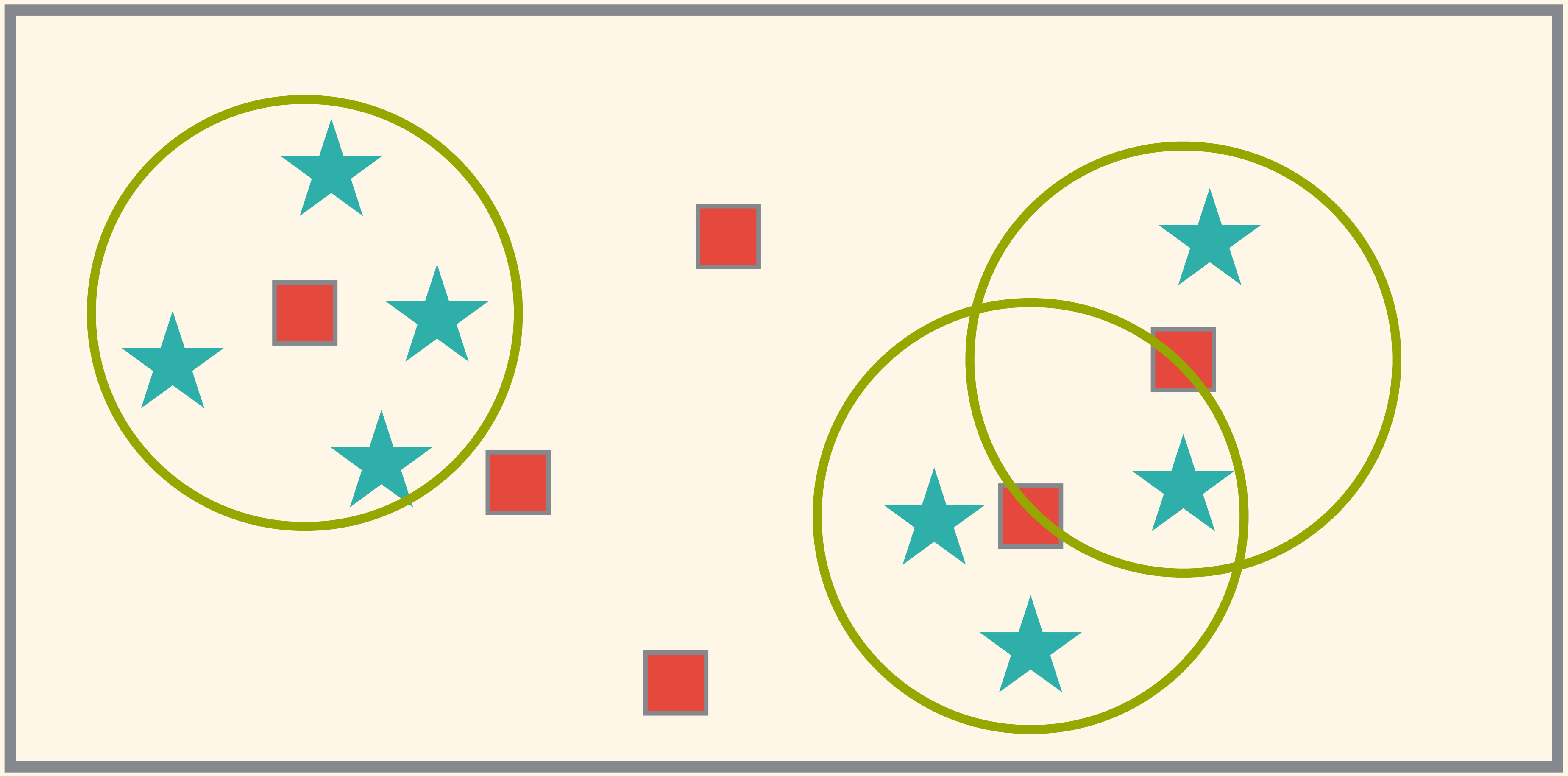
Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S's dependency



Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S's dependency



Related work

- GreeDi algorithm
  - for maximizing a monotone sub modular function with cardinality constraint
  - partitions data to each machine by block; runs Greedy algorithm in each machine; gather results in one machine; runs Greedy algorithm for these results
  - very simple and parallel, but worst case approximation guarantee is $1/\Theta\left(\min\left\{\sqrt{k}, m\right\}\right)$
    - k is cardinality constraint, m is number of machines

Related work

- Sample and Prune
  - for maximizing a monotone sub modular function with matroid constraint
  - runs greedy algorithm with small subset of dataset in single machine; prune some of the elements in dataset with results and reduce the data size
  - More general than GreeDi, but communication overhead is high

RandGreeDi Algorithm

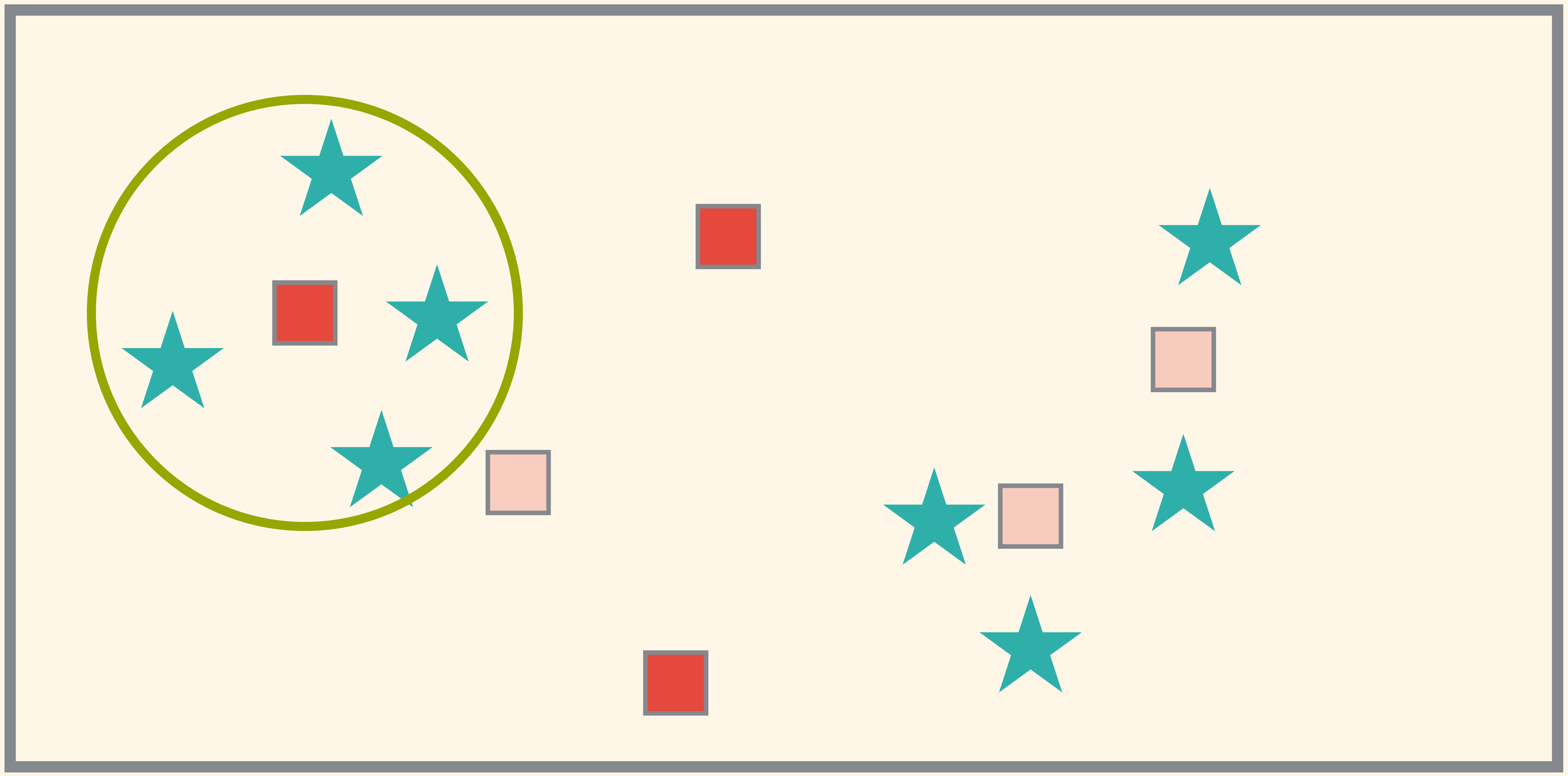
- Distribute input elements randomly to machines
- Run Greedy algorithm for each machines
- Combine them

Algorithm 2 The distributed algorithm RandGreeDi

for $e \in V$ **do**
  Assign e to a machine i chosen uniformly at random
end for
Let V_i be the elements assigned to machine i
Run Greedy(V_i) on each machine i to obtain S_i
Place $S = \bigcup_i S_i$ on machine 1
Run Alg(S) on machine 1 to obtain T
Let $S' = \arg\max_i \{f(S_i)\}$
return $\arg\max\{f(T), f(S')\}$

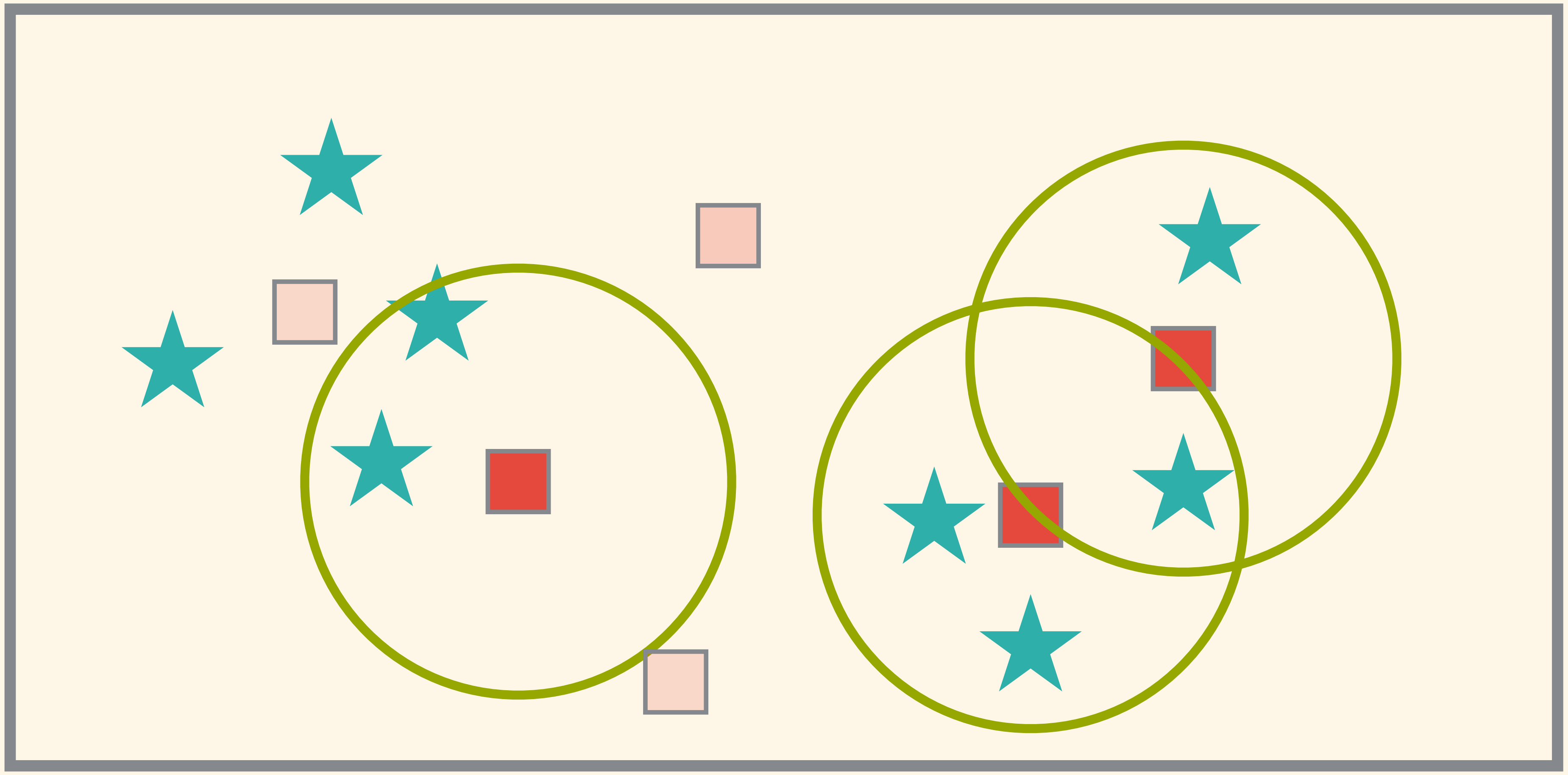
Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S's dependency



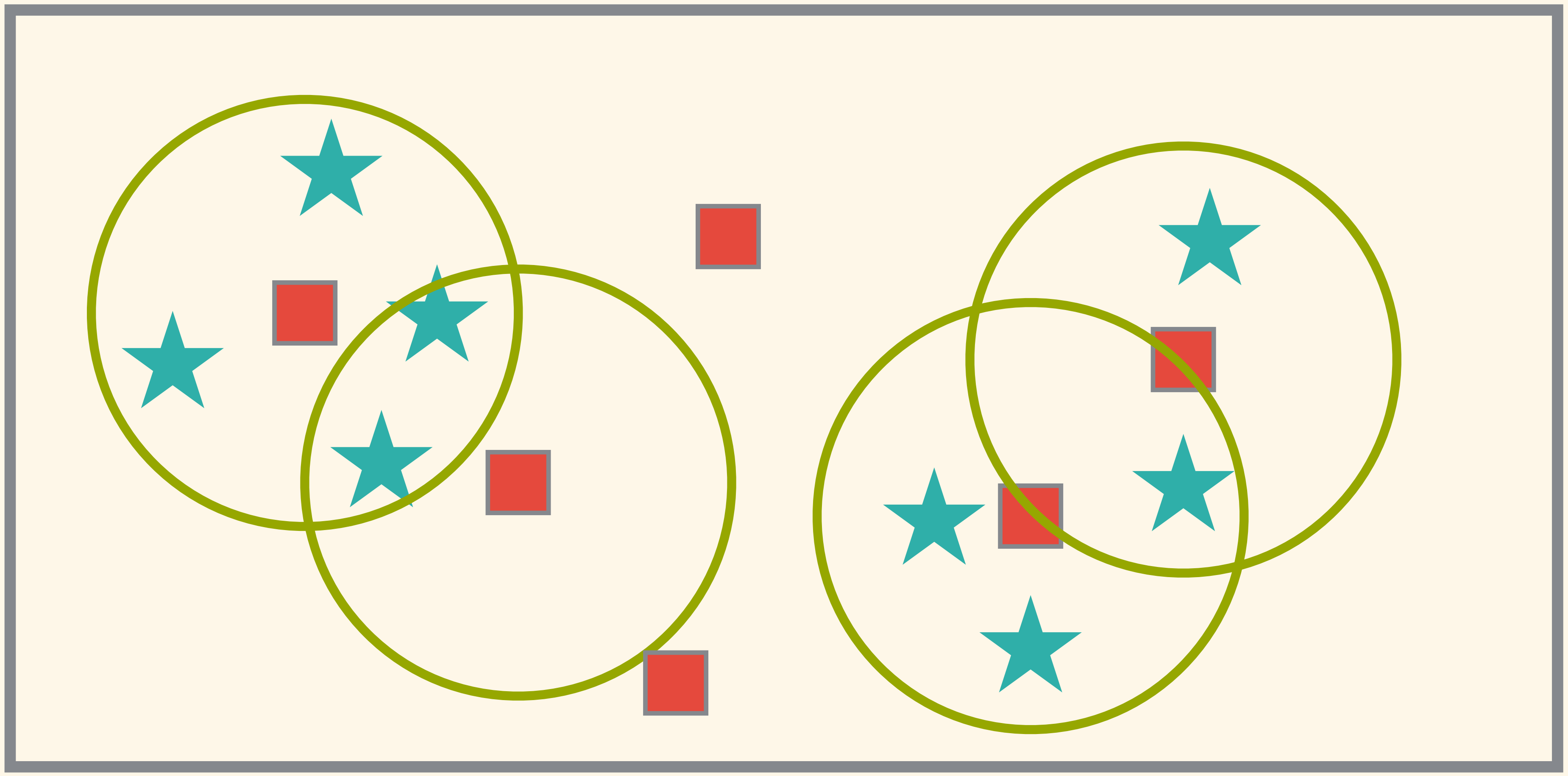
Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S's dependency



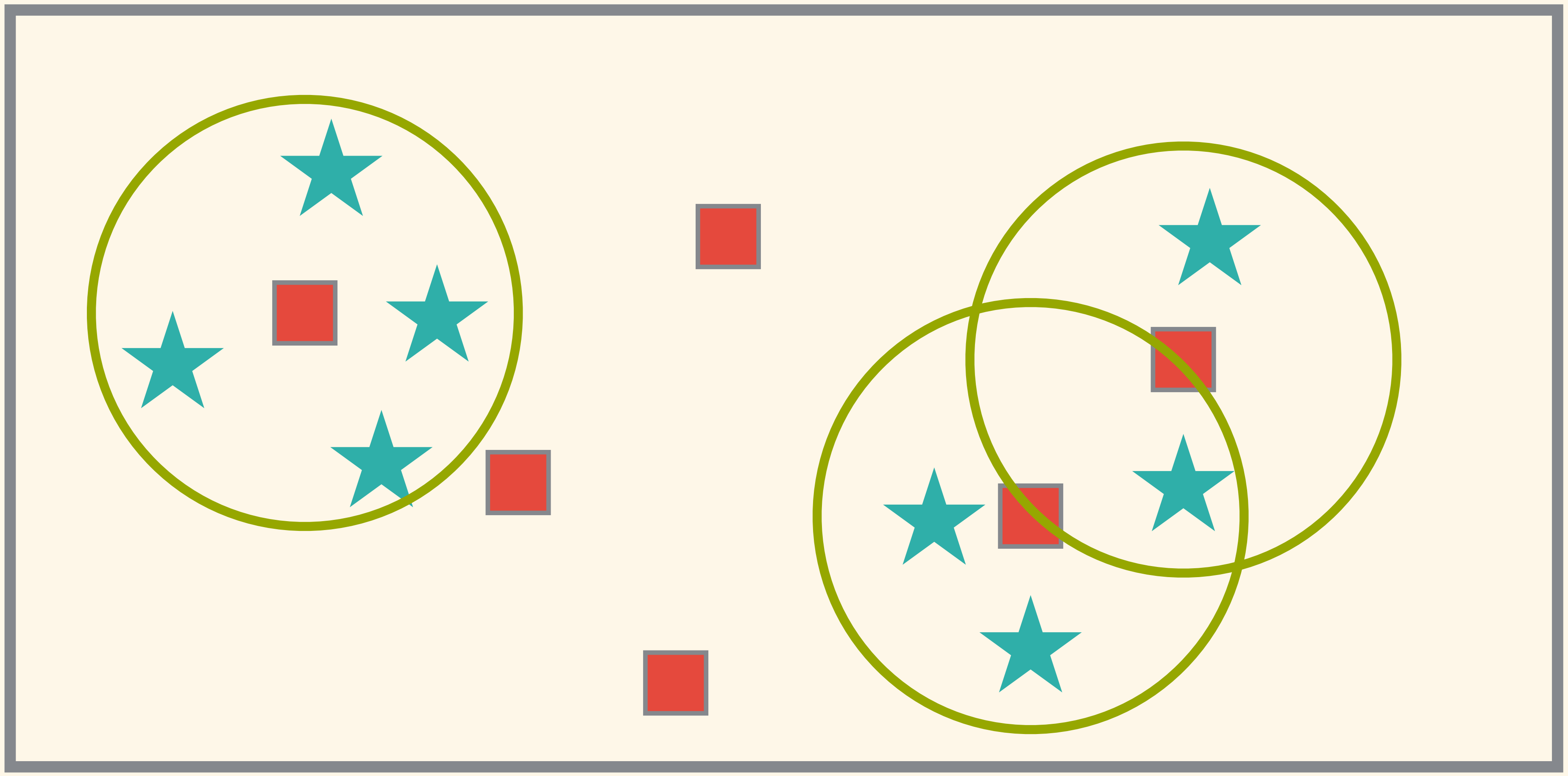
Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S’s dependency



Greedy Algorithm(貪欲法)

- Loop: Add one element which maximizes the function
- Unable to parallelize because of S’s dependency



Hereditary Constraints

- Consider submodular maximization:

$$\max\{f(s) : S \subseteq V, S \in \mathcal{I}\}$$

- where $f : 2^V \to \mathbb{R}_{\geq 0}$ is a sub modular function and $\mathcal{I} \subseteq 2^V$ is a family of subsets of V

- Hereditary Constrains: if some set is in $\mathcal{I}$, then all of its subsets is in $\mathcal{I}$.

RandGreeDi Algorithm

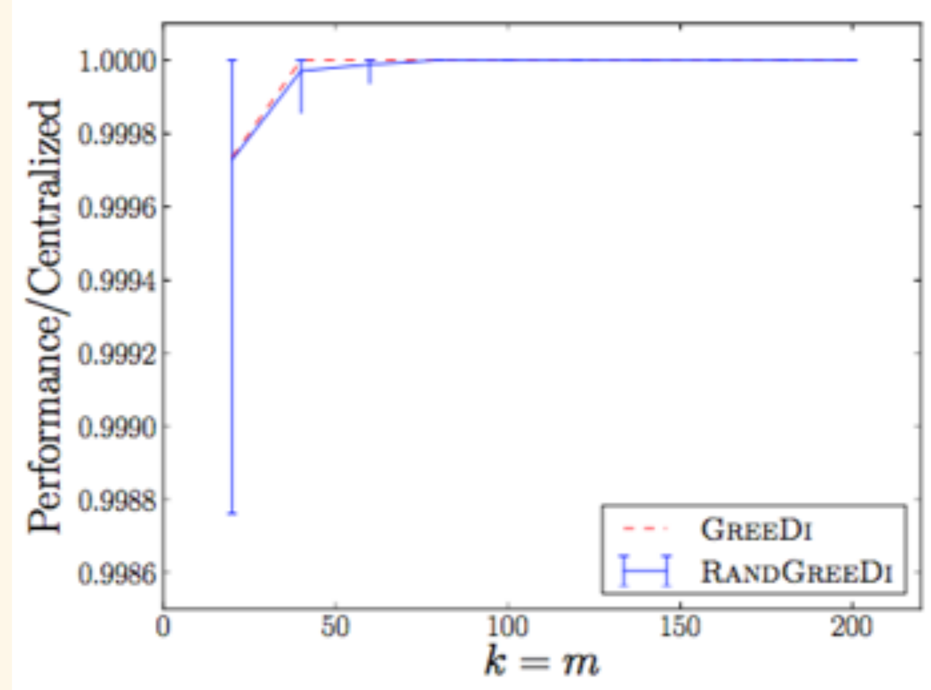
- When Greedy algorithm is α-approximation, RandGreeDi algorithm is:
  - for monotone submodular function: $\alpha/2$-approximation
  - for non-monotone submodular function: $\alpha/(4+2\alpha)$-approximation

Constraint	α	monotone approx. $\left(\frac{\alpha}{2}\right)$	non-monotone approx. $\left(\frac{\alpha}{4+2\alpha}\right)$
cardinality	$1-\frac{1}{e}\approx 0.632$	≈ 0.316	≈ 0.12
matroid	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{10}$
knapsack	≈ 0.35	≈ 0.17	≈ 0.074
p-system	$\frac{1}{p+1}$	$\frac{1}{2(p+1)}$	$\frac{1}{2+4(p+1)}$

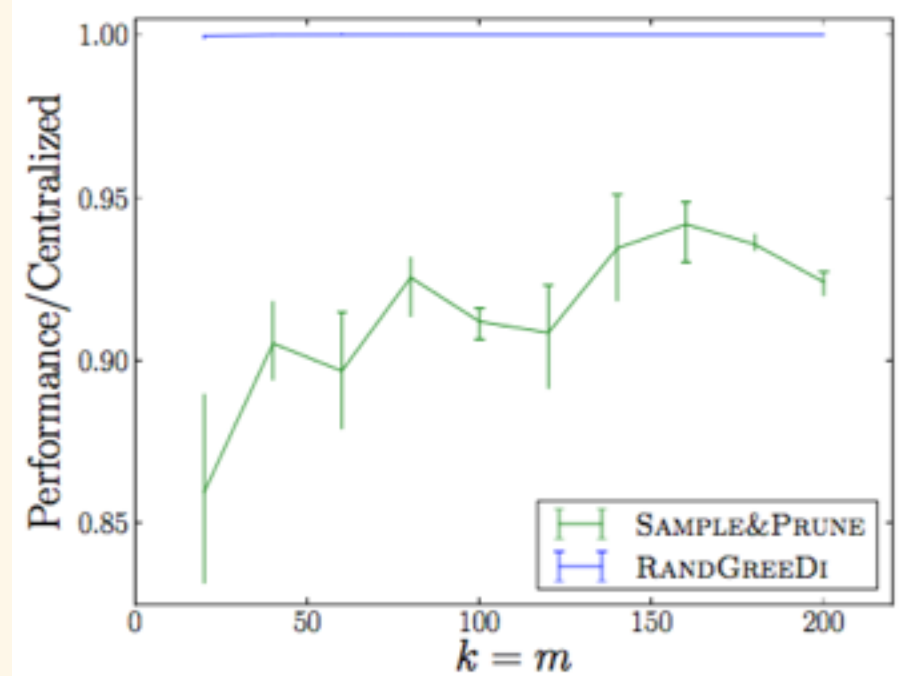
Table 1: New approximation results for randomized GreeDi for constrained monotone and non-monotone submodular maximization[3]

Experiments

- Exemplar based clustering
  - Clustering by minimizing distances between images and ‘exemplar’
  - Solving k-medoid problem, which is sub modular function with cardinality constraint
  - Better performance than Sample & Prune



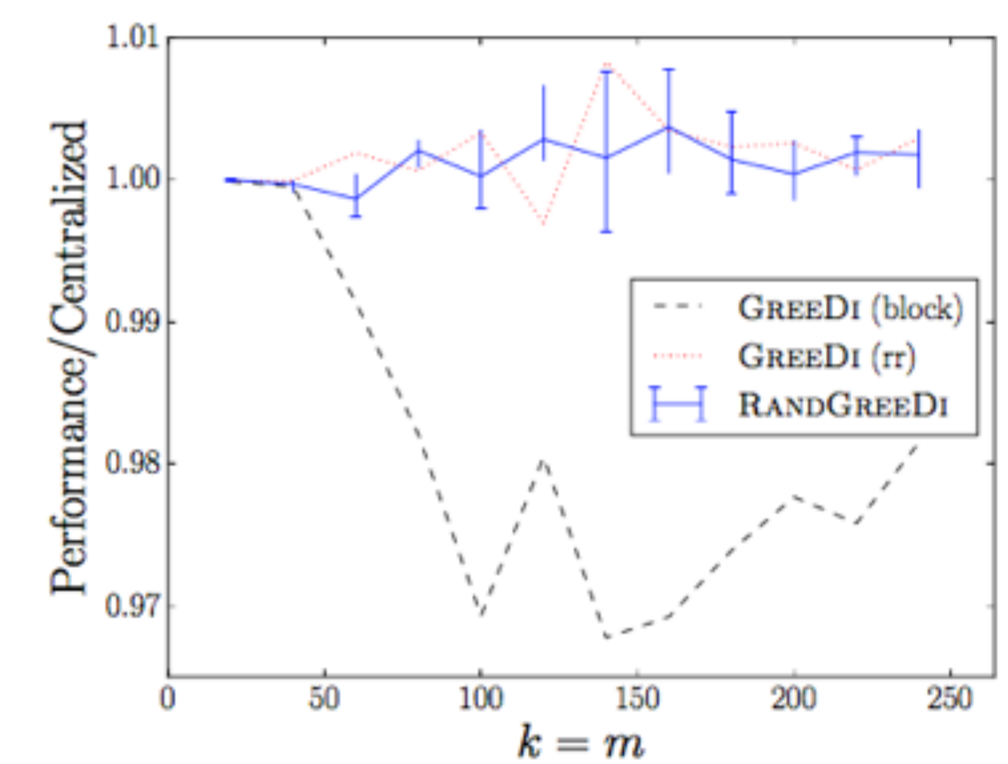
(c) 10K tiny images



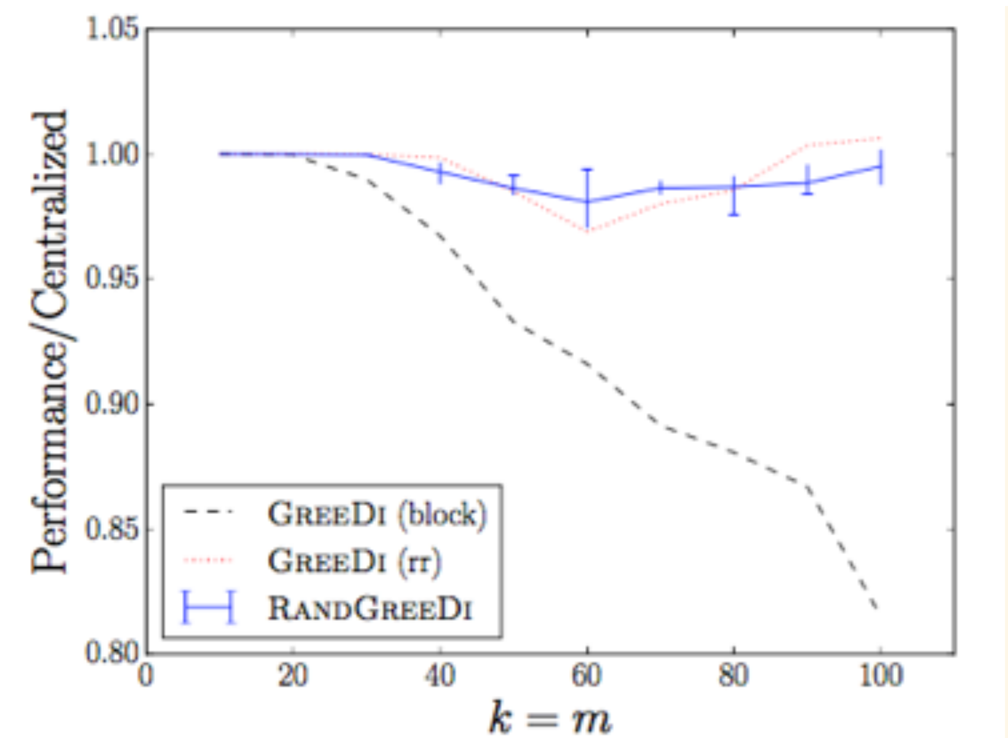
(f) 10K tiny images

Experiments

- Matroid constraints
  - ‘sensor placement problem’
  - randomized / round-robin distribution is better than block distribution
  - This is because each machine receives elements from several distinct partitions; which allows them to return a solution which is more nearer to optimal



(j) matroid coverage ($n = 900, r = 5$)



(k) matroid coverage ($n = 100, r = 100$)

Conclusion

- RandGreeDi is distributed Greedy algorithm with high approximation rate
- By using randomizing, Greedy algorithm allows each machine to return more usable solutions after reduce