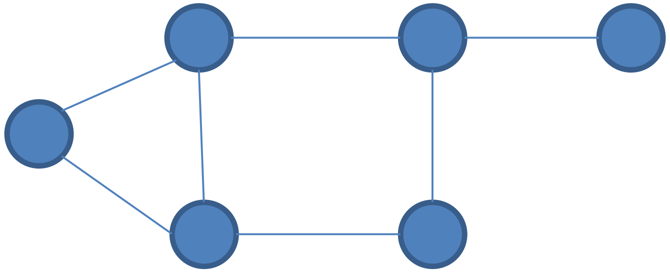
DATA MINING
LECTURE 12

Coverage

Approximation Algorithms

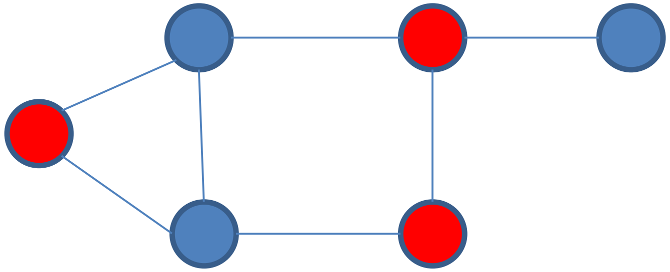
Example

- Promotion campaign on a social network
  - We have a social network as a graph.
  - People are more likely to buy a product if they have a friend who has the product.
  - We want to offer the product for free to some people such that every person in the graph is covered: they have a friend who has the product.
  - We want the number of free products to be as small as possible



Example

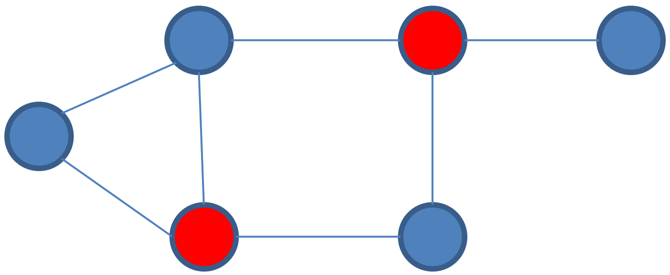
- Promotion campaign on a social network
  - We have a social network as a graph.
  - People are more likely to buy a product if they have a friend who has the product.
  - We want to offer the product for free to some people such that every person in the graph is covered: they have a friend who has the product.
  - We want the number of free products to be as small as possible



One possible selection

Example

- Promotion campaign on a social network
  - We have a social network as a graph.
  - People are more likely to buy a product if they have a friend who has the product.
  - We want to offer the product for free to some people such that every person in the graph is covered: they have a friend who has the product.
  - We want the number of free products to be as small as possible



A better selection

Dominating set

- Our problem is an instance of the dominating set problem

- Dominating Set: Given a graph $G = (V, E)$, a set of vertices $D \subseteq V$ is a dominating set if for each node u in V, either u is in D, or u has a neighbor in D.

- The Dominating Set Problem: Given a graph $G = (V, E)$ find a dominating set of minimum size.

Set Cover

- The dominating set problem is a special case of the Set Cover problem

- The Set Cover problem:
  - We have a universe of elements $U = \{x_1, \dots, x_N\}$
  - We have a collection of subsets of U, $\boldsymbol{S} = \{S_1, \dots, S_n\}$, such that $\bigcup_i S_i = U$
  - We want to find the smallest sub-collection $\boldsymbol{C} \subseteq \boldsymbol{S}$ of $\boldsymbol{S}$, such that $\bigcup_{S_i \in \boldsymbol{C}} S_i = U$
    - The sets in $\boldsymbol{C}$ cover the elements of U

An application of Set Cover

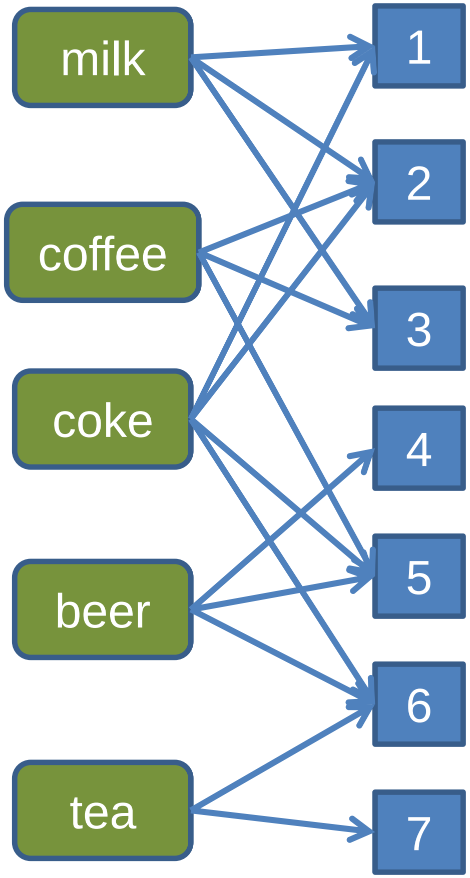
- Suppose that we want to create a catalog (with coupons) to give to customers of a store:
  - We want for every customer, the catalog to contain a product bought by the customer (this is a small store)
- How can we model this as a set cover problem?

Applications

- The universe U of elements is the set of customers of a store.
- Each set corresponds to a product p sold in the store:

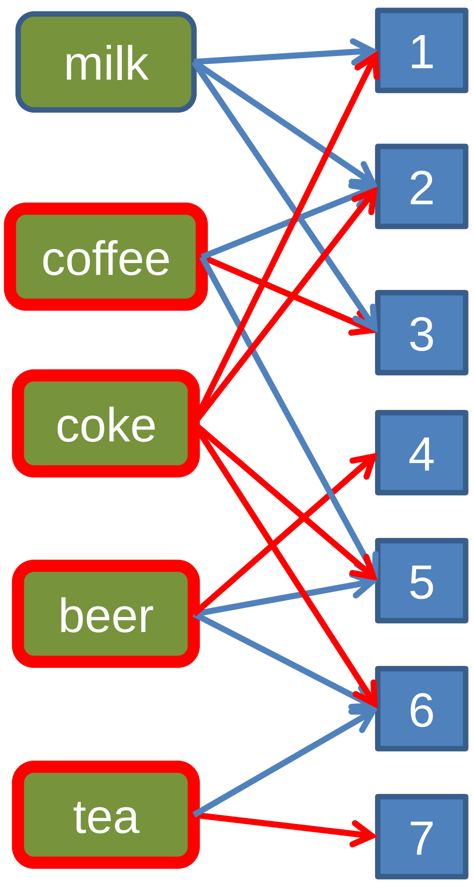
  $S_p = \{customers\ that\ bought\ p\}$

- Set cover: Find the minimum number of products (sets) that cover all the customers (elements of the universe)



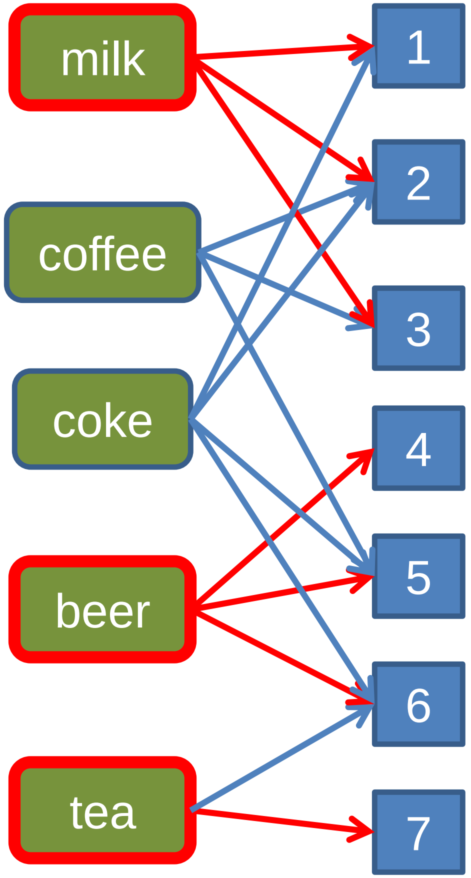
Applications

- The universe U of elements is the set of customers of a store.
- Each set corresponds to a product p sold in the store:
  $S_p = \{customers\ that\ bought\ p\}$
- Set cover: Find the minimum number of products (sets) that cover all the customers (elements of the universe)



Applications

- The universe U of elements is the set of customers of a store.
- Each set corresponds to a product p sold in the store:
  $S_p = \{customers\ that\ bought\ p\}$
- Set cover: Find the minimum number of products (sets) that cover all the customers (elements of the universe)



Applications

- Dominating Set (or Promotion Campaign) as Set Cover:
  - The universe U is the set of nodes V
  - Each node u defines a set S_u consisting of the node u and all of its neighbors
  - We want the minimum number of sets S_u (nodes) that cover all the nodes in the graph.
- Many more…

Best selection variant

- Suppose that we have a budget K of how big our set cover can be
  - We only have K products to give out for free.
  - We want to cover as many customers as possible.

- Maximum-Coverage Problem: Given a universe of elements $\boldsymbol{U}$, a collection $\boldsymbol{S}$ of subsets of $\boldsymbol{U}$, and a budget K, find a sub-collection $\boldsymbol{C} \subseteq \boldsymbol{S}$ of size at most K, such that the number of covered elements $\bigcup_{S_i \in \boldsymbol{C}} S_i$ is maximized.

Complexity

- Both the Set Cover and the Maximum Coverage problems are NP-complete
  - What does this mean?
  - Why do we care?

- There is no algorithm that can guarantee finding the best solution in polynomial time
  - Can we find an algorithm that can guarantee to find a solution that is close to the optimal?
  - Approximation Algorithms.

Approximation Algorithms

- For an (combinatorial) optimization problem, where:
  - X is an instance of the problem,
  - OPT(X) is the value of the optimal solution for X,
  - ALG(X) is the value of the solution of an algorithm ALG for X

ALG is a good approximation algorithm if the ratio of OPT(X) and ALG(X) is bounded for all input instances X

- Minimum set cover: input X = (U,**S**) is the universe of elements and the set collection, OPT(X) is the size of minimum set cover, ALG(X) is the size of the set cover found by an algorithm ALG.

- Maximum coverage: input X = (U,**S**,K) is the input instance, OPT(X) is the coverage of the optimal algorithm, ALG(X) is the coverage of the set found by an algorithm ALG.

Approximation Algorithms

- For a minimization problem, the algorithm ALG is an α-approximation algorithm, for $\alpha > 1$, if for all input instances X,

$$ALG(X) \leq \alpha OPT(X)$$

- In simple words: the algorithm ALG is at most α times worse than the optimal.

- α is the approximation ratio of the algorithm – we want α to be as close to 1 as possible
  - Best case: $\alpha = 1 + \epsilon$ and $\epsilon \to 0$, as $n \to \infty$ (e.g., $\epsilon = \frac{1}{n}$)
  - Good case: $\alpha = O(1)$ is a constant (e.g., $\alpha = 2$)
  - OK case: $\alpha = O(\log n)$
  - Bad case $\alpha = O(n^{\epsilon})$

Approximation Algorithms

- For a maximization problem, the algorithm ALG is an α-approximation algorithm, for $\alpha < 1$, if for all input instances X,

$$ALG(X) \geq \alpha OPT(X)$$

- In simple words: the algorithm ALG achieves at least α percent of what the optimal achieves.

- α is the approximation ratio of the algorithm – we want α to be as close to 1 as possible
  - Best case: $\alpha = 1 - \epsilon$ and $\epsilon \to 0$, as $n \to \infty$(e.g., $\epsilon = \frac{1}{n}$)
  - Good case: $\alpha = O(1)$ is a constant (e.g., $a = 0.5$)
  - OK case: $\alpha = O(\frac{1}{\log n})$
  - Bad case $\alpha = O(n^{-\epsilon})$

A simple approximation ratio for set cover

- Lemma: Any algorithm for set cover has approximation ratio $\alpha = |S_{max}|$, where S_{max} is the set in $\boldsymbol{S}$ with the largest cardinality
- Proof:
  - $OPT(X) \geq N/|S_{max}| \Rightarrow N \leq |S_{max}|OPT(X)$
  - $ALG(X) \leq N \leq |S_{max}|OPT(X)$

- This is true for any algorithm.
- Not a good bound since it may be that $|S_{max}| = O(N)$

An algorithm for Set Cover

- What is a natural algorithm for Set Cover?

- Greedy: each time add to the collection $\boldsymbol{C}$ the set S_i from $\boldsymbol{S}$ that covers the most of the remaining uncovered elements.

The GREEDY algorithm

GREEDY(U,S)

X= U

C = {}

while X is not empty do

For all $S_i \in \boldsymbol{S}$ let $\mathrm{gain}(S_i) = |S_i \cap X|$

Let S_* be such that $gain(S_*)$ is maximum

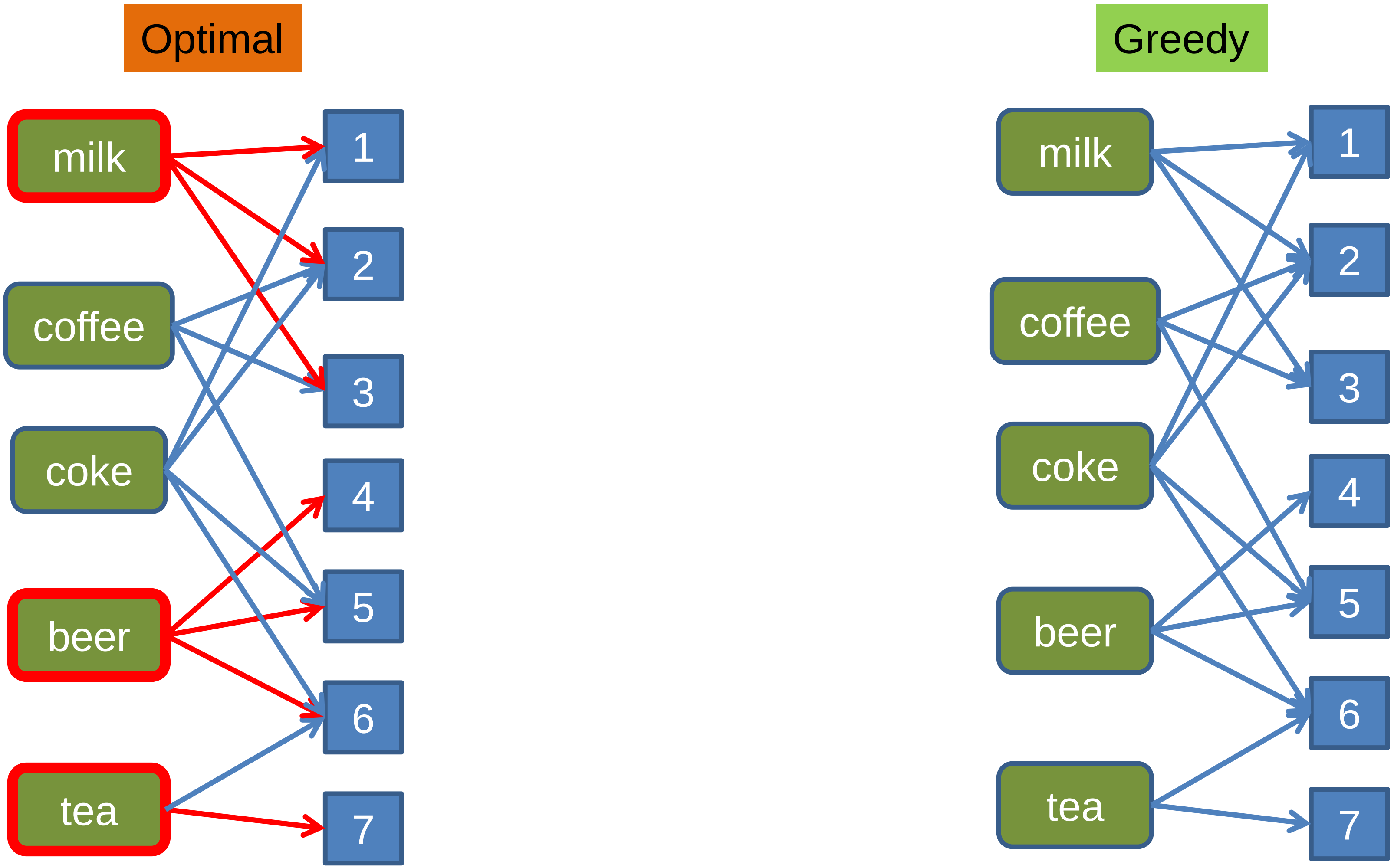
C = C U {S_*}

X = X\ S_*

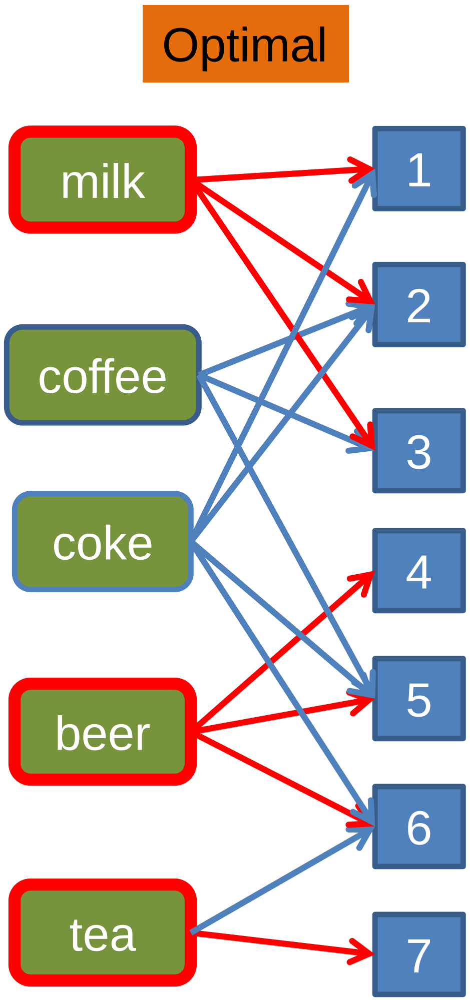
S = ***S***\ S_*

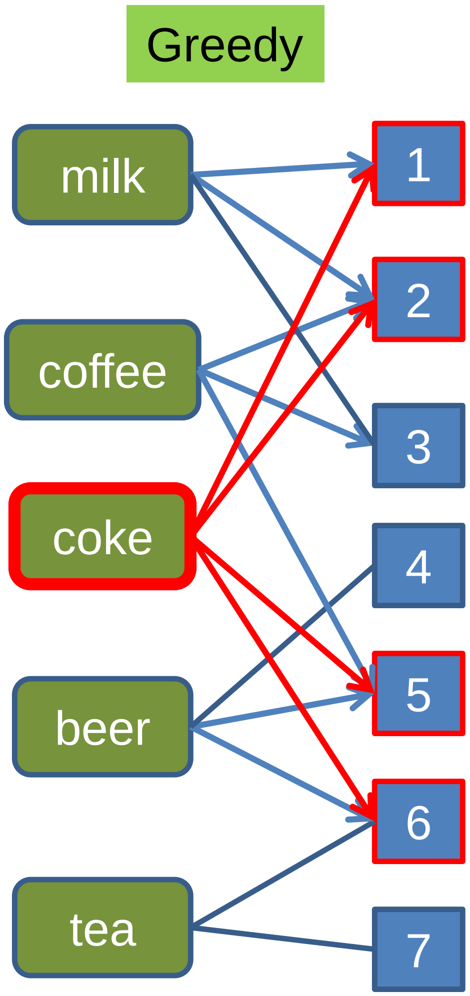
The number of elements covered by S_i not already covered by C.

Greedy is not always optimal

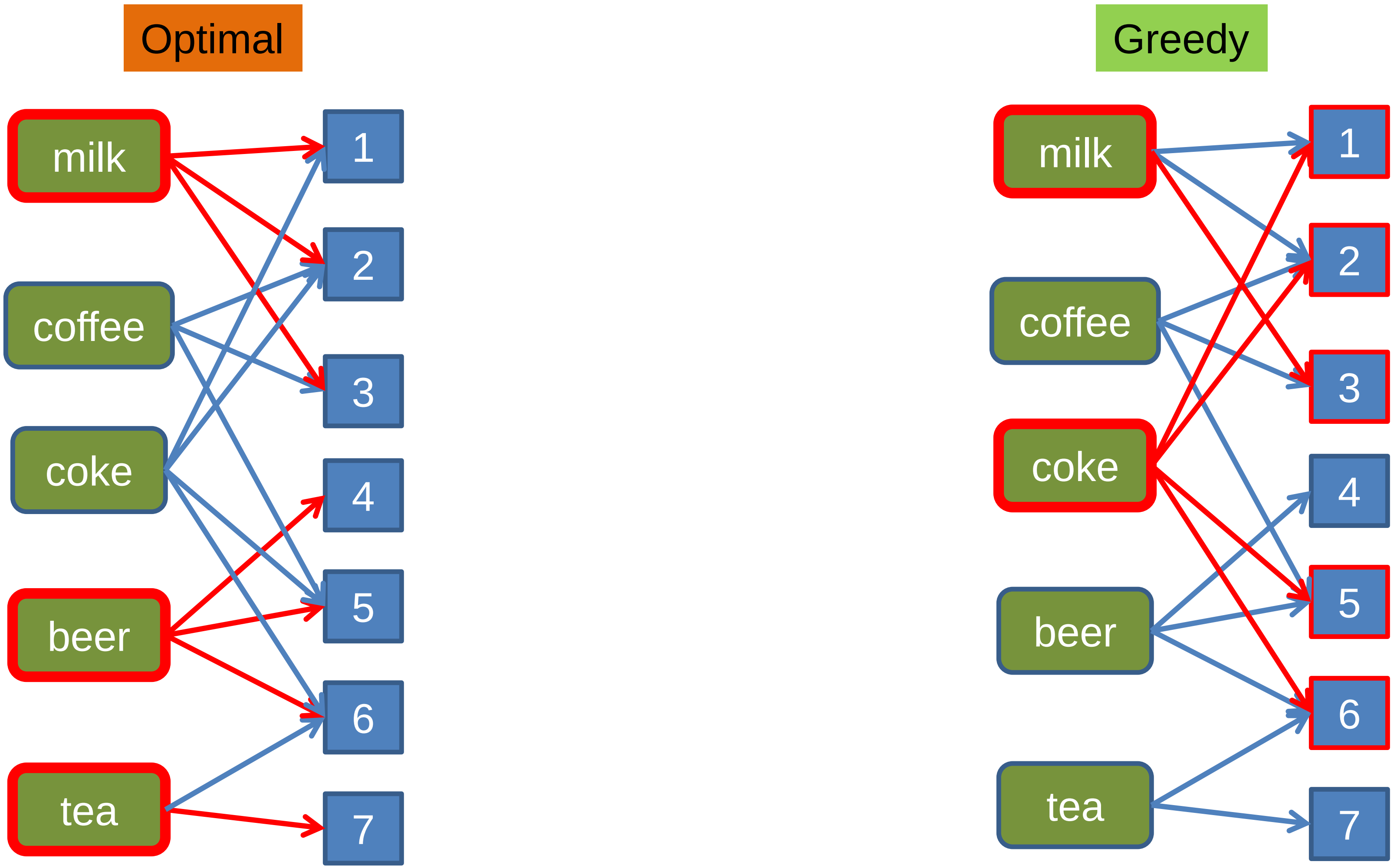


Greedy is not always optimal

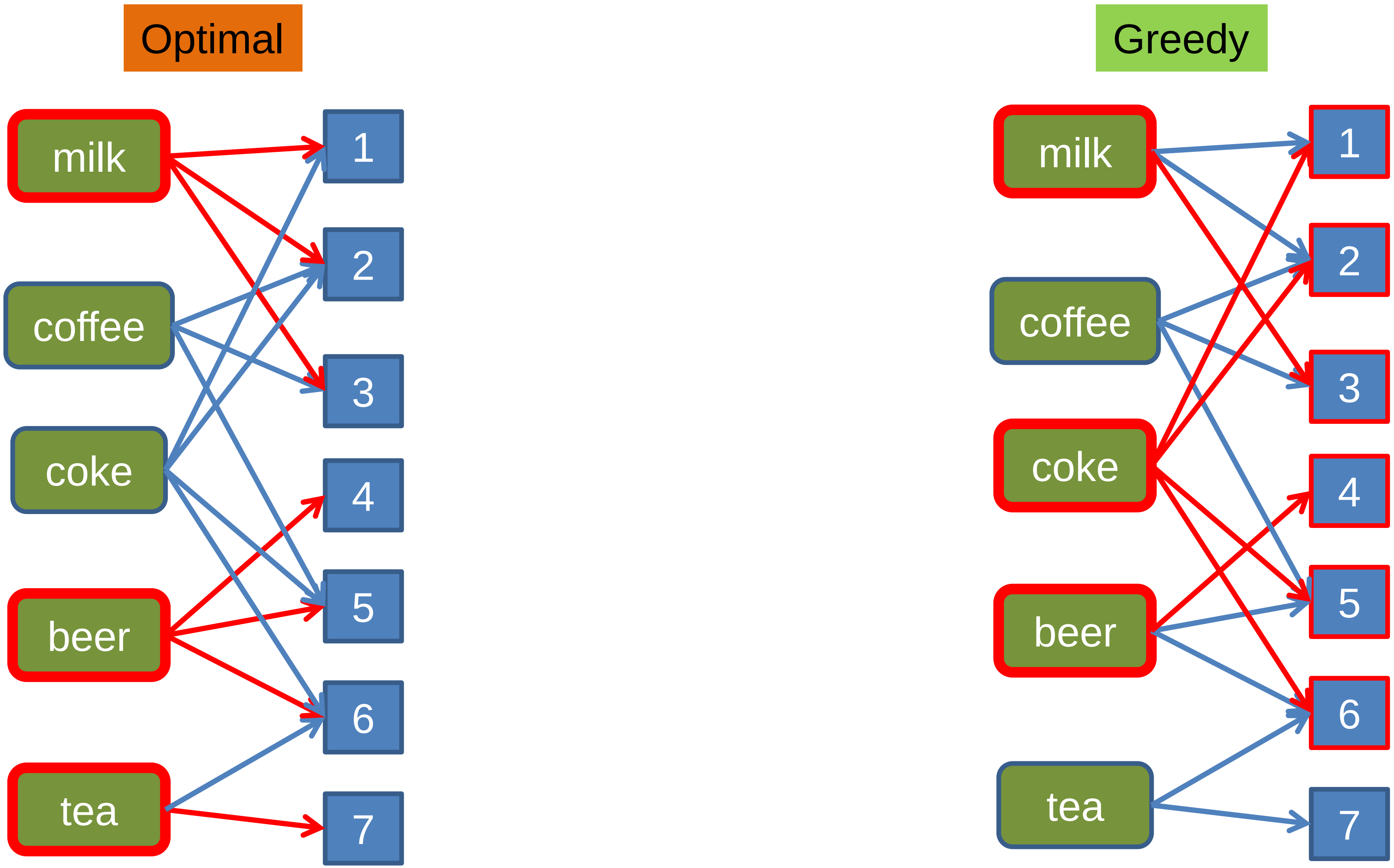




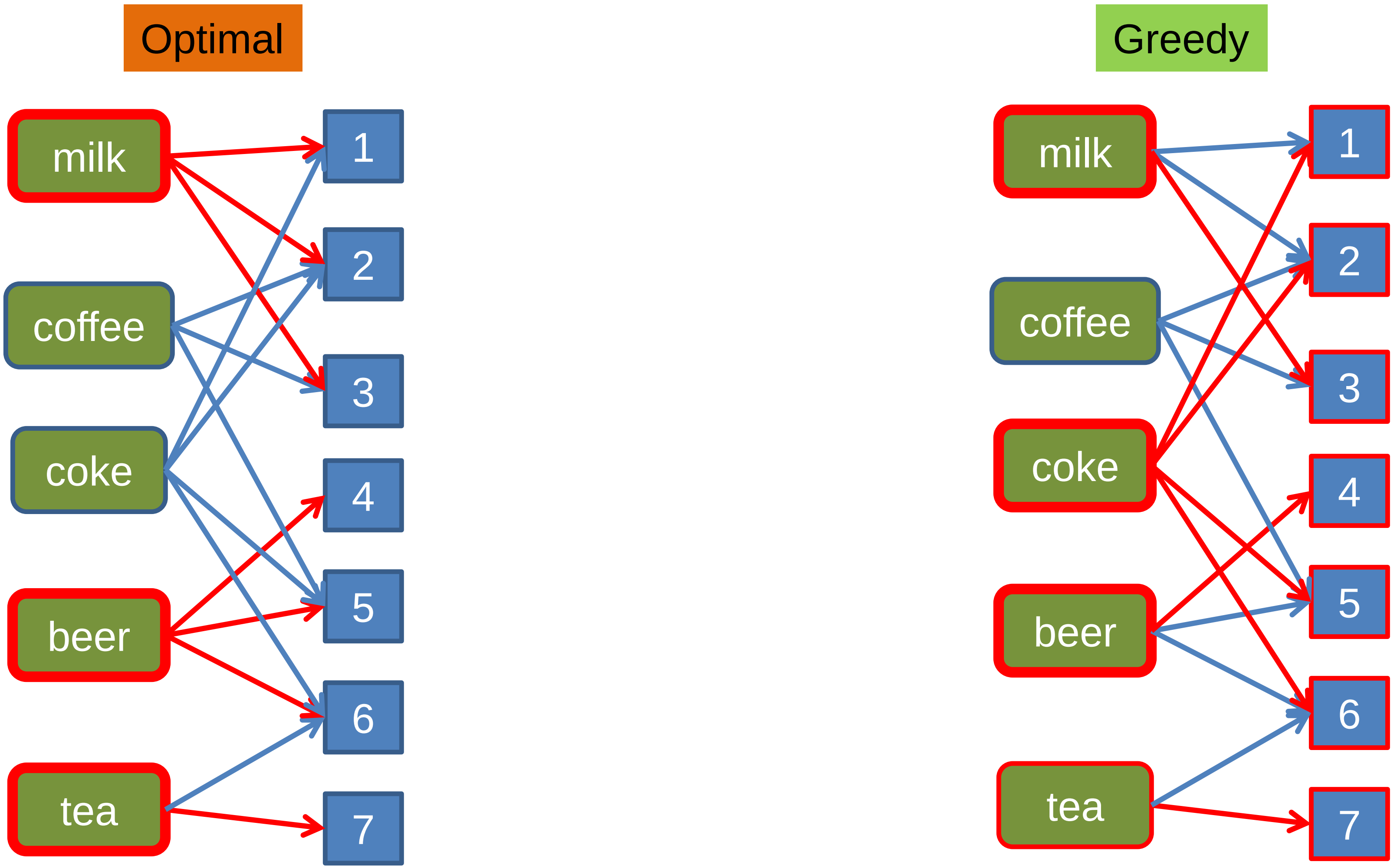
Greedy is not always optimal



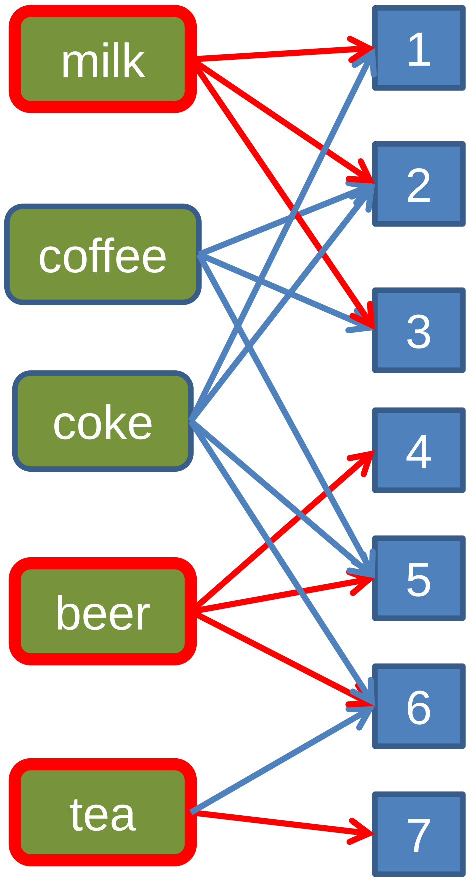
Greedy is not always optimal



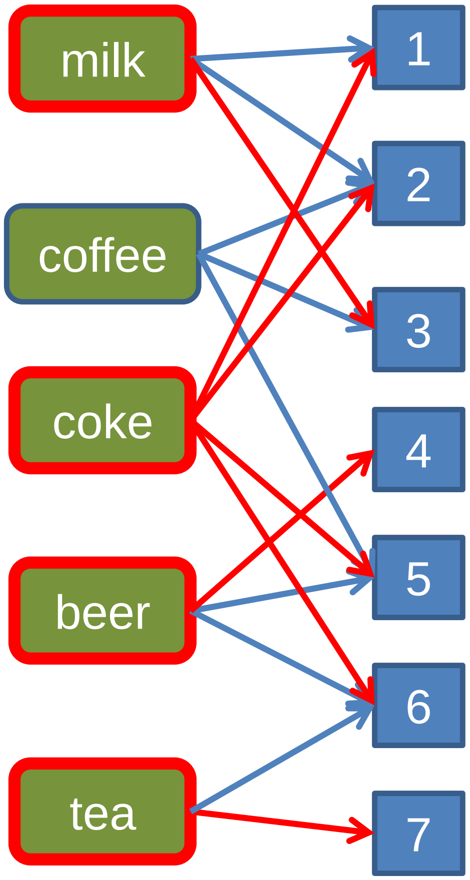
Greedy is not always optimal



Greedy is not always optimal



- Adding Coke to the set is useless.
- We need Milk (or Coffee) and Beer to cover all customers



Approximation ratio of GREEDY

- Good news: GREEDY has approximation ratio:

$$\alpha = H(|S_{\max}|) = 1 + \ln|S_{\max}|, \qquad H(n) = \sum_{k=1}^{n} \frac{1}{k}$$

$$GREEDY(X) \leq (1 + \ln|S_{\max}|)OPT(X),$$ for all X

Maximum Coverage

- Greedy is also applicable here

GREEDY(U,S,K)

X = U

C = {}

while |***C***| < K

For all $S_i \in \boldsymbol{S}$ let $\mathrm{gain}(S_i) = |S_i \cap X|$

Let S_* be such that $gain(S_*)$ is maximum

C = C U {S_*}

X = X\ S_*

S= ***S***\ S_*

The number of elements covered by S_i not already covered by C.

Approximation Ratio for Max-K Coverage

- Better news! The GREEDY algorithm has approximation ratio $\alpha = 1 - \frac{1}{e}$

$$GREEDY(X) \geq \left(1 - \frac{1}{e}\right) OPT(X),$$ for all X

- (e is the basis of the natural logarithm)

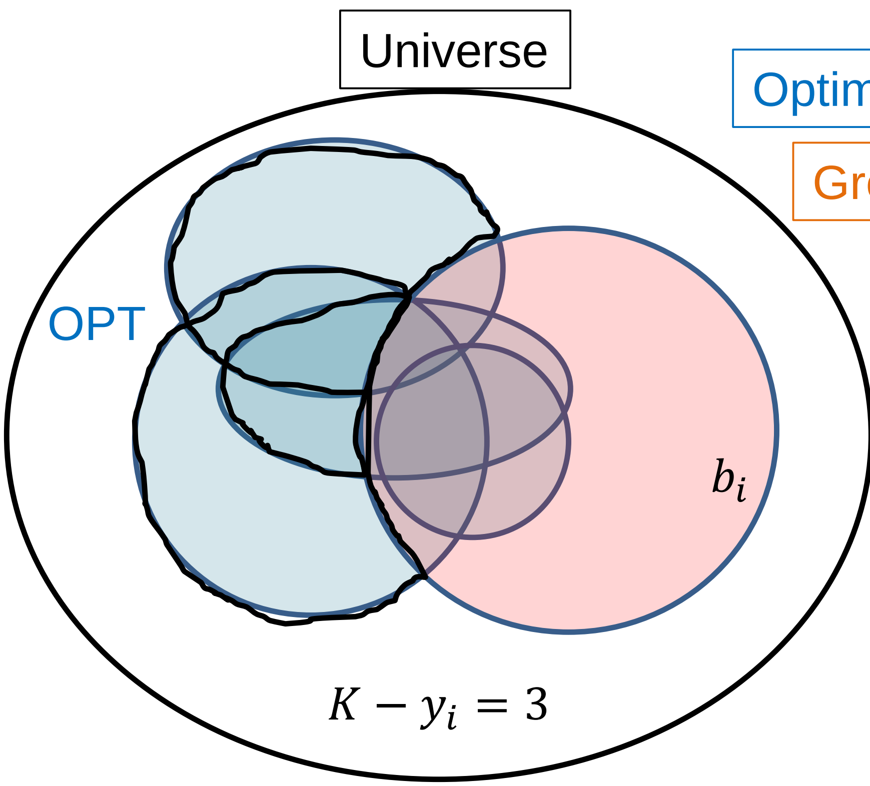
- The coverage of the Greedy solution is at least 63% that of the optimal

Proof of approximation ratios

- We will now give a proof of the approximation ratios for the SET-COVER and the MAX-COVERAGE
  - We start with MAX-COVERAGE

- Definitions:
  - OPT: size of the optimal solution
  - b_i: number of covered elements at iteration i of Greedy
  - a_i: number of newly covered elements at iteration i
  - $c_i = OPT - b_i$: difference between solution size of Optimal and Greedy solutions at iteration i.

- Lemma: $a_{i+1} \geq \frac{c_i}{K}$
- Proof:
  - For $i = 0$, it is simple to see since one of the K sets in the optimal solution has size at least $\frac{OPT}{K}$.
  - For larger i



Optimal solution for K = 5

Greedy solution at iteration i

x_i: intersection of optimal and Greedy solutions

$$x_i \leq b_i$$

y_i: number of optimal subsets fully included in the Greedy solution:

$$y_i \geq 0$$

There must be a set with

$$a_{i+1} \geq \frac{OPT - x_i}{K - y_i} \geq \frac{OPT - b_i}{K} = \frac{c_i}{K}$$

- Lemma: $c_{i+1} \leq \left(1 - \frac{1}{K}\right)^{i+1} OPT$

- Proof: By induction on i.
- Basis of induction: $c_0 \leq \left(1 - \frac{1}{K}\right) OPT$
  - Use the fact that $c_0 = OPT$, and $b_1 = a_1$
- Inductive Hypothesis: $c_i \leq \left(1 - \frac{1}{K}\right)^{i} OPT$
- Inductive step: $c_{i+1} \leq \left(1 - \frac{1}{K}\right)^{i+1} OPT$
  - Use the inductive hypothesis and that $b_{i+1} = \sum_{j=1}^{i+1} a_j$ and $c_{i+1} = c_i - a_i$

- Theorem: The Greedy algorithm has approximation ratio $\left(1-\frac{1}{e}\right)$
- Proof:

$$c_K \leq \left(1-\frac{1}{K}\right)^K OPT \leq \frac{1}{e} OPT$$

- The size of the Greedy solution is b_K

$$b_K = OPT - c_K \geq \left(1-\frac{1}{e}\right) OPT$$

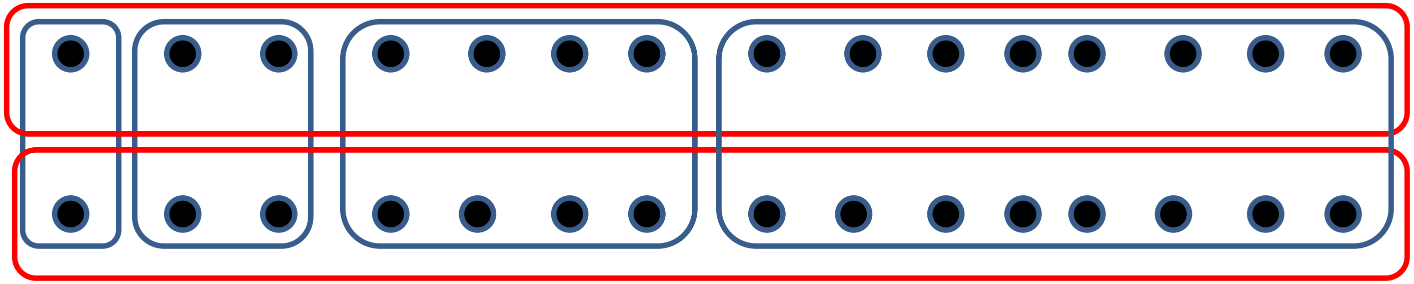
Proof for SET COVER

- In the case of SET COVER, we have that $OPT = n$
- Let k^* be the size of the optimal solution.

- We know that after i iterations: $c_i \leq \left(1 - \frac{1}{k^*}\right)^i n$.
- After $t = k^* \ln \frac{n}{k^*}$ iterations $c_t \leq k^*$ elements remain to be covered
  - We can cover those in at most k^* iterations
  - Total iterations are at most $k^*(\ln \frac{n}{k^*} + 1) \leq k^*(\ln n + 1)$

Lower bound

- The approximation ratio is tight up to a constant
  - Tight means that we can find a counter example with this ratio



- $OPT(X) = 2$
- $GREEDY(X) = \log N$
- $\alpha = \frac{1}{2}\log N$

Another proof of the approximation ratio for MAX-K COVERAGE

- For a collection of subsets C, let $F(C) = \left|\bigcup_{S_i \in \boldsymbol{C}} S_i\right|$ be the number of elements that are covered.
- The set function F has two properties:

- F is monotone:

$$F(A) \leq F(B) \; if \; A \subseteq B$$

- F is submodular:

$$F(A \cup \{S\}) - F(A) \geq F(B \cup \{S\}) - F(B) \;\; if \; A \subseteq B$$

- The addition of set S to a set of nodes has greater effect (more new covered items) for a smaller set.
  - The diminishing returns property

Optimizing submodular functions

- Theorem:

If we want to maximize a monotone and submodular function F under cardinality constraints (size of set at most K),

Then, the greedy algorithm that each time adds to the solution $\boldsymbol{C}$, the set S that maximizes the gain $F(\boldsymbol{C} \cup$

True for any monotone and submodular set function!

Other variants of Set Cover

- Hitting Set: select a set of elements so that you hit all the sets (the same as the set cover, reversing the roles)

- Vertex Cover: Select a set of vertices from a graph such that you cover all edges (for every edge an endpoint of the edge is in the set)
  - There is a 2-approximation algorithm

- Edge Cover: Select a set of edges that cover all vertices (for every vertex, there is one edge that has as endpoint this vertex)
  - There is a polynomial algorithm

OVERVIEW

Class Overview

- In this class you saw a set of tools for analyzing data
  - Frequent Itemsets, Association Rules
  - Sketching
  - Recommendation Systems
  - Clustering
  - Singular Value Decomposition
  - Classification
  - Link Analysis Ranking
  - Random Walks
  - Coverage
- All these are useful when trying to make sense of the data. A lot more tools exist.
- I hope that you found this interesting, useful and fun.