DATA MINING
LECTURE 13

Absorbing Random walks

Coverage

Random Walks on Graphs

- Random walk:
  - Start from a node chosen uniformly at random with probability $\frac{1}{n}$.
    - Pick one of the outgoing edges uniformly at random
    - Move to the destination of the edge
    - Repeat.

Random walk

- Question: what is the probability p_i^t of being at node i after t steps?

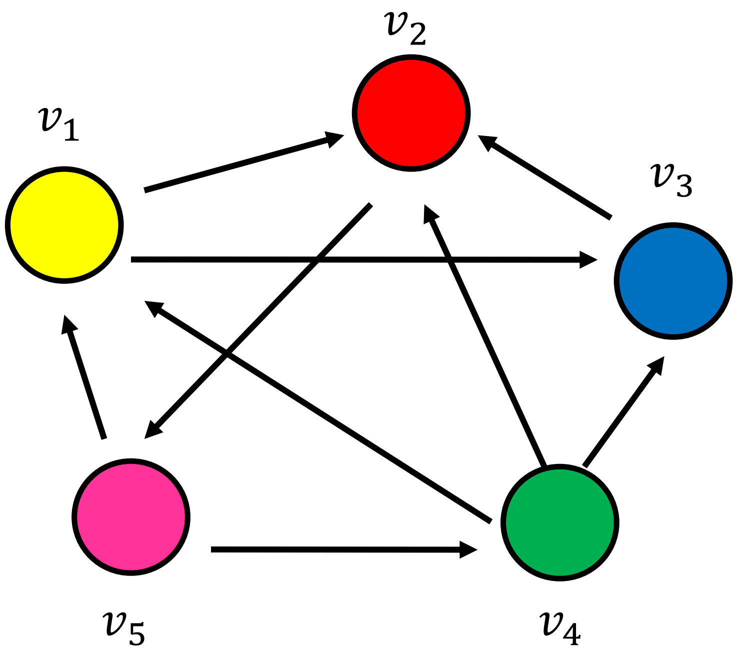
$$p_1^0 = \frac{1}{5} \qquad p_1^t = \frac{1}{3}p_4^{t-1} + \frac{1}{2}p_5^{t-1}$$

$$p_2^0 = \frac{1}{5} \qquad p_2^t = \frac{1}{2}p_1^{t-1} + p_3^{t-1} + \frac{1}{3}p_4^{t-1}$$

$$p_3^0 = \frac{1}{5} \qquad p_3^t = \frac{1}{2}p_1^{t-1} + \frac{1}{3}p_4^{t-1}$$

$$p_4^0 = \frac{1}{5} \qquad p_4^t = \frac{1}{2}p_5^{t-1}$$

$$p_5^0 = \frac{1}{5} \qquad p_5^t = p_2^{t-1}$$



Stationary distribution

- After many many steps ($t \to \infty$) the probabilities converge (updating the probabilities does not change the numbers)

- The converged probabilities define the stationary distribution of a random walk π

- The probability π_i is the fraction of times that we visited state i as $t \to \infty$

- Markov Chain Theory: The random walk converges to a unique stationary distribution independent of the initial vector if the graph is strongly connected, and not bipartite.

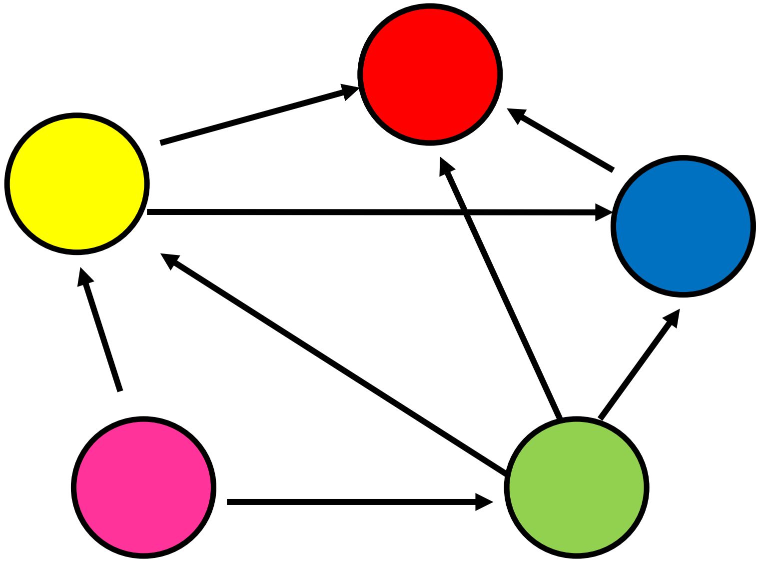
Random walk with Restarts

- This is the random walk used by the PageRank algorithm
  - At every step with probability $1-\alpha$ do a step of the random walk (follow a random link)
  - With probability α restart the random walk from a randomly selected node.
- The effect of the restart is that paths followed are never too long.
  - In expectation paths have length $1/\alpha$
- Restarts can also be from a specific node in the graph (always start the random walk from there)
- What is the effect of that?
  - The nodes that are close to the starting node have higher probability to be visited.

ABSORBING RANDOM WALKS

Random walk with absorbing nodes

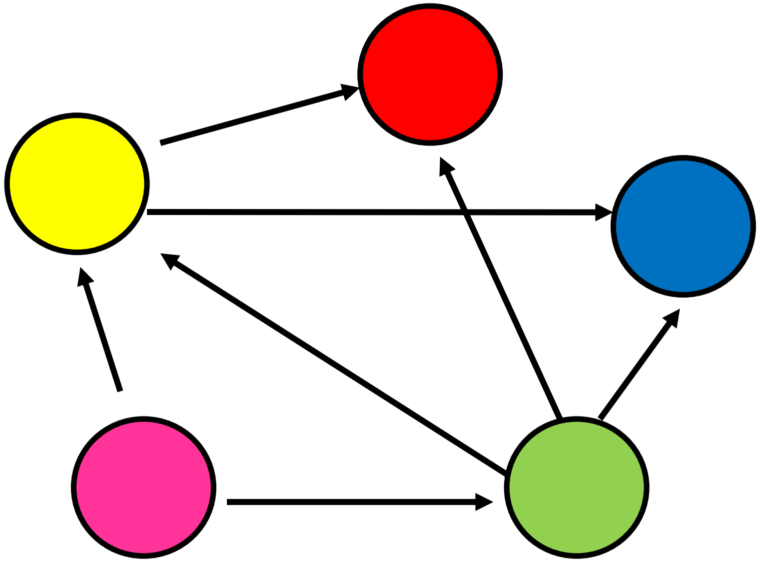
- What happens if we do a random walk on this graph? What is the stationary distribution?



- All the probability mass on the red sink node:
  - The red node is an absorbing node

Random walk with absorbing nodes

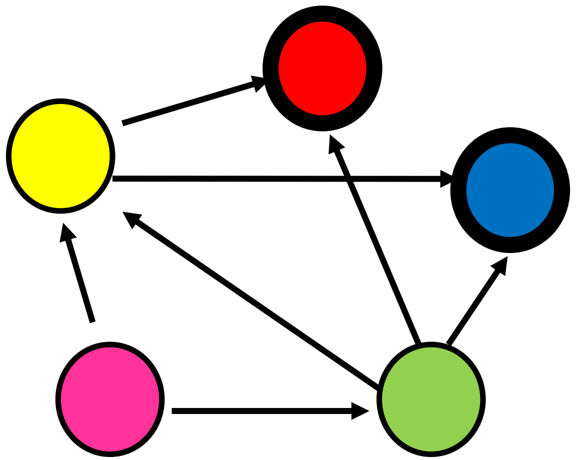
- What happens if we do a random walk on this graph? What is the stationary distribution?



- There are two absorbing nodes: the red and the blue.
- The probability mass will be divided between the two

Absorption probability

- If there are more than one absorbing nodes in the graph a random walk that starts from a non-absorbing node will be absorbed in one of them with some probability
  - The probability of absorption gives an estimate of how close the node is to red or blue



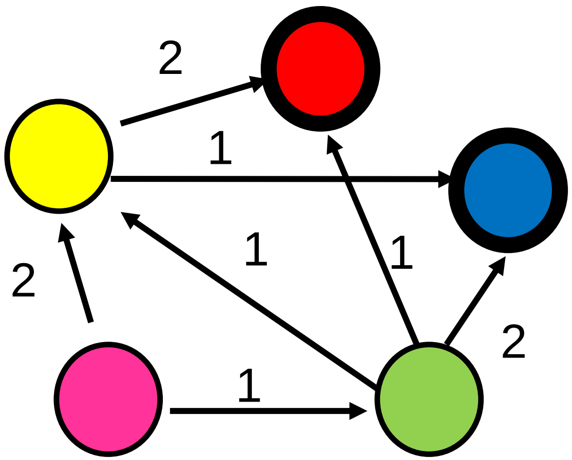
Absorption probability

- Computing the probability of being absorbed:
  - The absorbing nodes have probability 1 of being absorbed in themselves and zero of being absorbed in another node.
  - For the non-absorbing nodes, take the (weighted) average of the absorption probabilities of your neighbors
    - if one of the neighbors is the absorbing node, it has probability 1
  - Repeat until convergence (= very small change in probs)

$$P(Red|Pink) = \frac{2}{3}P(Red|Yellow) + \frac{1}{3}P(Red|Green)$$

$$P(Red|Green) = \frac{1}{4}P(Red|Yellow) + \frac{1}{4}$$

$$P(Red|Yellow) = \frac{2}{3}$$



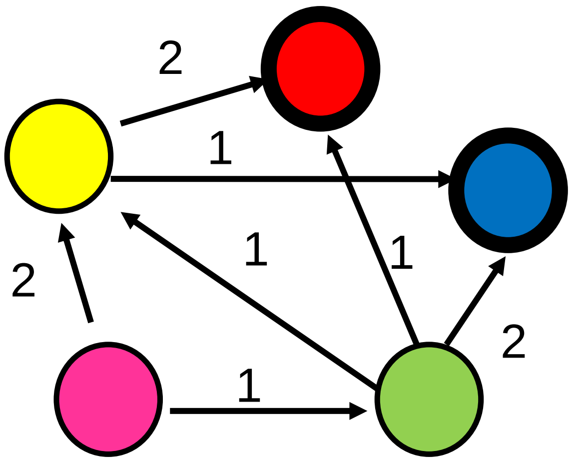
Absorption probability

- Computing the probability of being absorbed:
  - The absorbing nodes have probability 1 of being absorbed in themselves and zero of being absorbed in another node.
  - For the non-absorbing nodes, take the (weighted) average of the absorption probabilities of your neighbors
    - if one of the neighbors is the absorbing node, it has probability 1
  - Repeat until convergence (= very small change in probs)

$$P(Blue|Pink) = \frac{2}{3}P(Blue|Yellow) + \frac{1}{3}P(Blue|Green)$$

$$P(Blue|Green) = \frac{1}{4}P(Blue|Yellow) + \frac{1}{2}$$

$$P(Blue|Yellow) = \frac{1}{3}$$

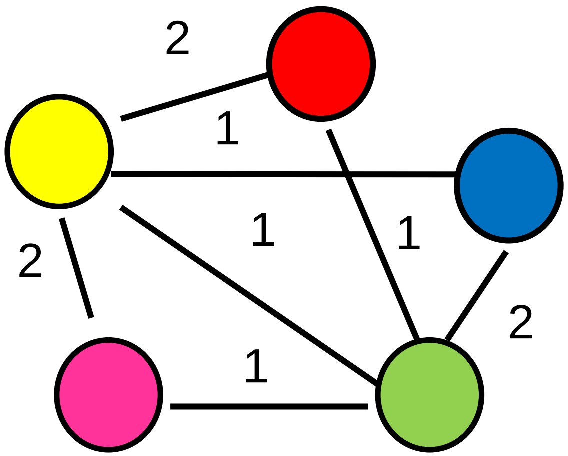


Why do we care?

- Why do we care to compute the absorption probability to sink nodes?
- Given a graph (directed or undirected) we can choose to make some nodes absorbing.
  - Simply direct all edges incident on the chosen nodes towards them and remove outgoing edges.
- The absorbing random walk provides a measure of proximity of non-absorbing nodes to the chosen nodes.
  - Useful for understanding proximity in graphs
  - Useful for propagation in the graph
    - E.g, some nodes have positive opinions for an issue, some have negative, to which opinion is a non-absorbing node closer?

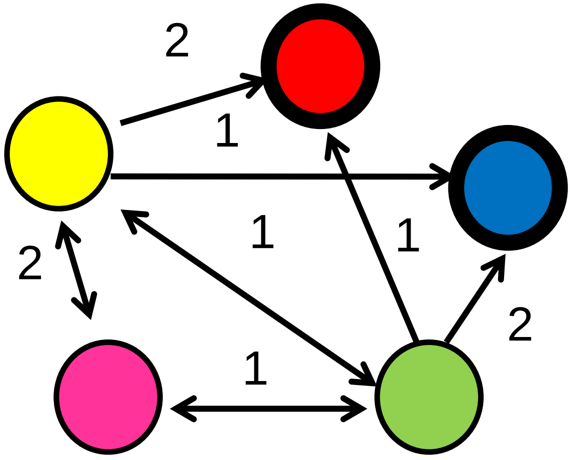
Example

- In this undirected graph we want to learn the proximity of nodes to the red and blue nodes



Example

- Make the nodes absorbing



Absorption probability

- Compute the absorbtion probabilities for red and blue

$$P(Red|Pink) = \frac{2}{3}P(Red|Yellow) + \frac{1}{3}P(Red|Green)$$

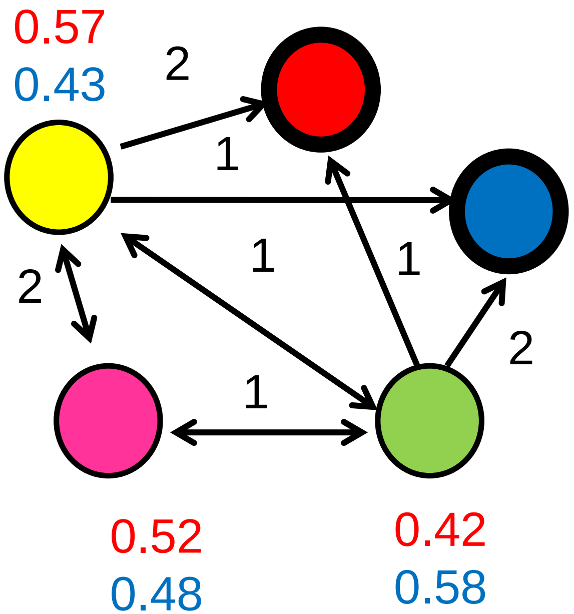
$$P(Red|Green) = \frac{1}{5}P(Red|Yellow) + \frac{1}{5}P(Red|Pink) + \frac{1}{5}$$

$$P(Red|Yellow) = \frac{1}{6}P(Red|Green) + \frac{1}{3}P(Red|Pink) + \frac{1}{3}$$

$$P(Blue|Pink) = 1 - P(Red|Pink)$$

$$P(Blue|Green) = 1 - \ P(Red|Green)$$

$$P(Blue|Yellow) = 1 - P(Red|Yellow)$$

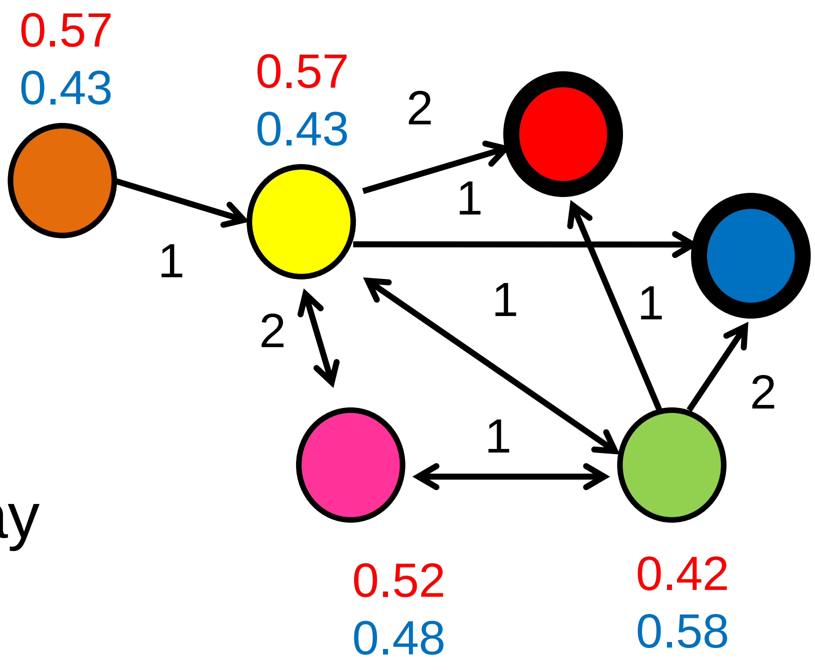


Penalizing long paths

- The orange node has the same probability of reaching red and blue as the yellow one

$$P(Red|Orange) = P(Red|Yellow)$$

$$P(Blue|Orange) = P(Blue|Yellow)$$



- Intuitively though it is further away

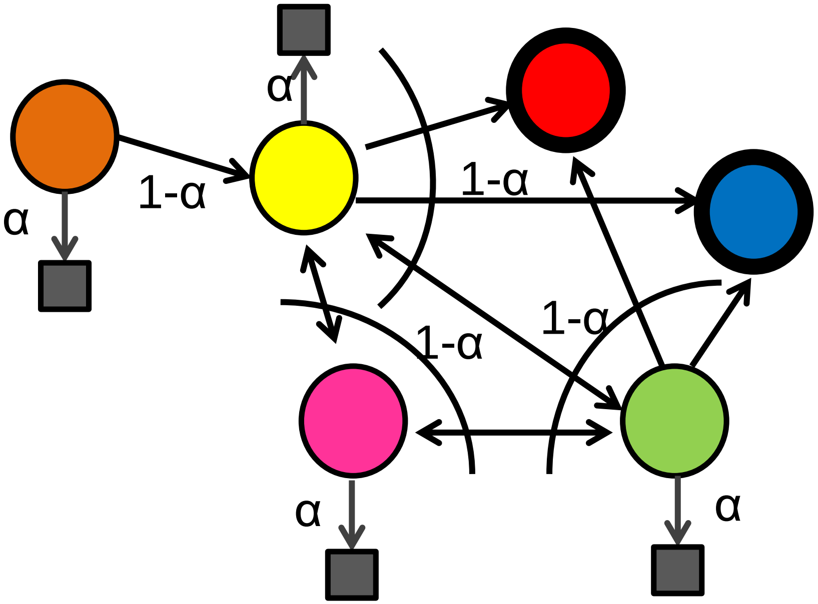
Penalizing long paths

- Add an universal absorbing node to which each node gets absorbed with probability α.

With probability α the random walk dies

With probability (1-α) the random walk continues as before

The longer the path from a node to an absorbing node the more likely the random walk dies along the way, the lower the absorbtion probability



e.g.

$$P(Red|Green) = (1-\alpha)\left(\frac{1}{5}P(Red|Yellow) + \frac{1}{5}P(Red|Pink) + \frac{1}{5}\right)$$

Random walk with restarts

- Adding a jump with probability α to a universal absorbing node seems similar to Pagerank

- Random walk with restart:
  - Start a random walk from node u
  - At every step with probability α, jump back to u
  - The probability of being at node v after large number of steps defines again a similarity between u,v

- The Random Walk With Restarts (RWS) and Absorbing Random Walk (ARW) are similar but not the same
  - RWS computes the probability of paths from the starting node u to a node v, while AWR the probability of paths from a node v, to the absorbing node u.
  - RWS defines a distribution over all nodes, while AWR defines a probability for each node
  - An absorbing node blocks the random walk, while restarts simply bias towards starting nodes
    - Makes a difference when having multiple (and possibly competing) absorbing nodes

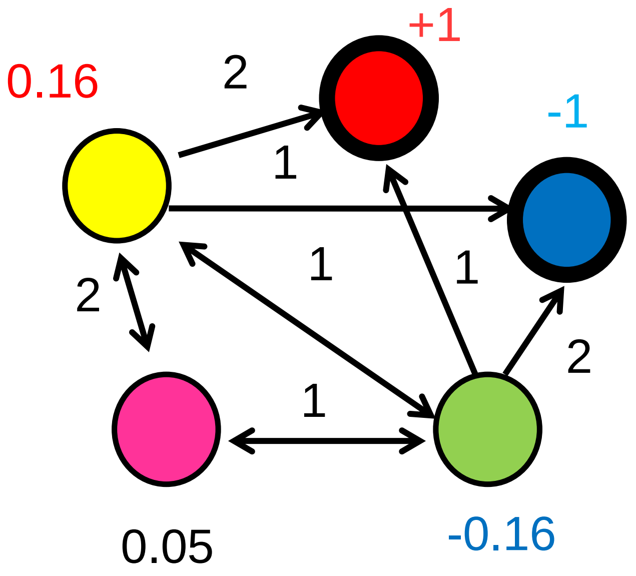
Propagating values

- Assume that Red has a positive value and Blue a negative value
    - Positive/Negative class, Positive/Negative opinion
- We can compute a value for all the other nodes by repeatedly averaging the values of the neighbors
    - The value of node u is the expected value at the point of absorption for a random walk that starts from u

$$V(Pink) = \frac{2}{3}V(Yellow) + \frac{1}{3}V(Green)$$

$$V(Green) = \frac{1}{5}V(Yellow) + \frac{1}{5}V(Pink) + \frac{1}{5} - \frac{2}{5}$$

$$V(Yellow) = \frac{1}{6}V(Green) + \frac{1}{3}V(Pink) + \frac{1}{3} - \frac{1}{6}$$



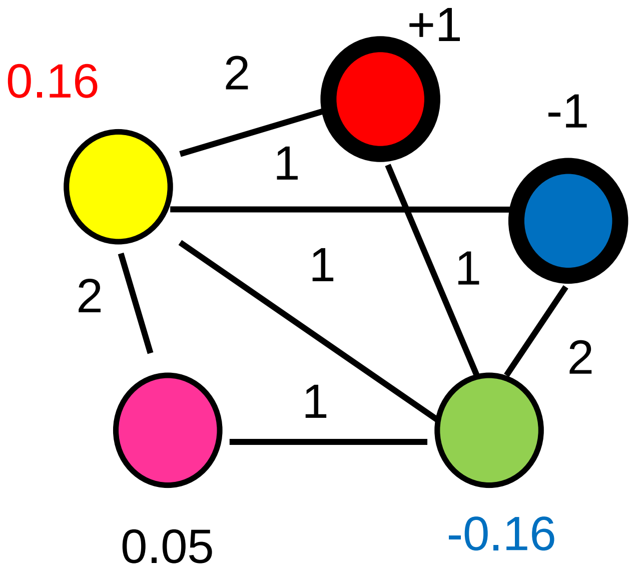
Electrical networks and random walks

- Our graph corresponds to an electrical network
- There is a positive voltage of +1 at the Red node, and a negative voltage -1 at the Blue node
- There are resistances on the edges inversely proportional to the weights (or conductance proportional to the weights)
- The computed values are the voltages at the nodes

$$V(Pink) = \frac{2}{3}V(Yellow) + \frac{1}{3}V(Green)$$

$$V(Green) = \frac{1}{5}V(Yellow) + \frac{1}{5}V(Pink) + \frac{1}{5} - \frac{2}{5}$$

$$V(Yellow) = \frac{1}{6}V(Green) + \frac{1}{3}V(Pink) + \frac{1}{3} - \frac{1}{6}$$



Opinion formation

- The value propagation can be used as a model of opinion formation.
- Model:
  - Opinions are values in [-1,1]
  - Every user u has an internal opinion s_u, and expressed opinion z_u.
  - The expressed opinion minimizes the personal cost of user u:

$$c(z_u) = (s_u - z_u)^2 + \sum_{v:v \text{ is a friend of } u} w_u (z_u - z_v)^2$$

    - Minimize deviation from your beliefs and conflicts with the society

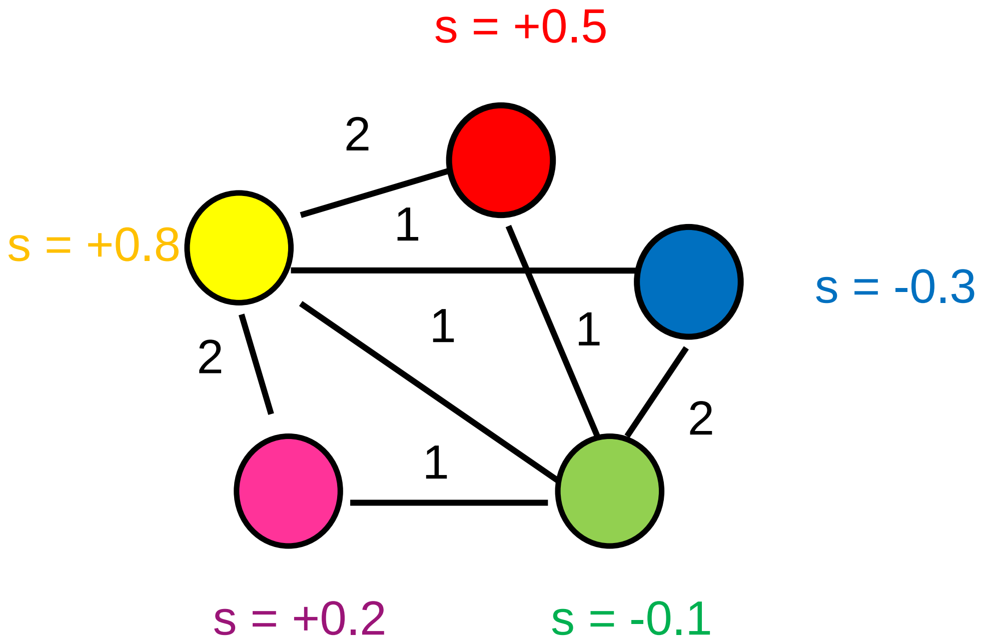
- If every user tries independently (selfishly) to minimize their personal cost then the best thing to do is to set z_u to the average of all opinions:

$$z_u = \frac{s_u + \sum_{v:v \text{ is a friend of } u} w_u z_u}{1 + \sum_{v:v \text{ is a friend of } u} w_u}$$

- This is the same as the value propagation we described before!

Example

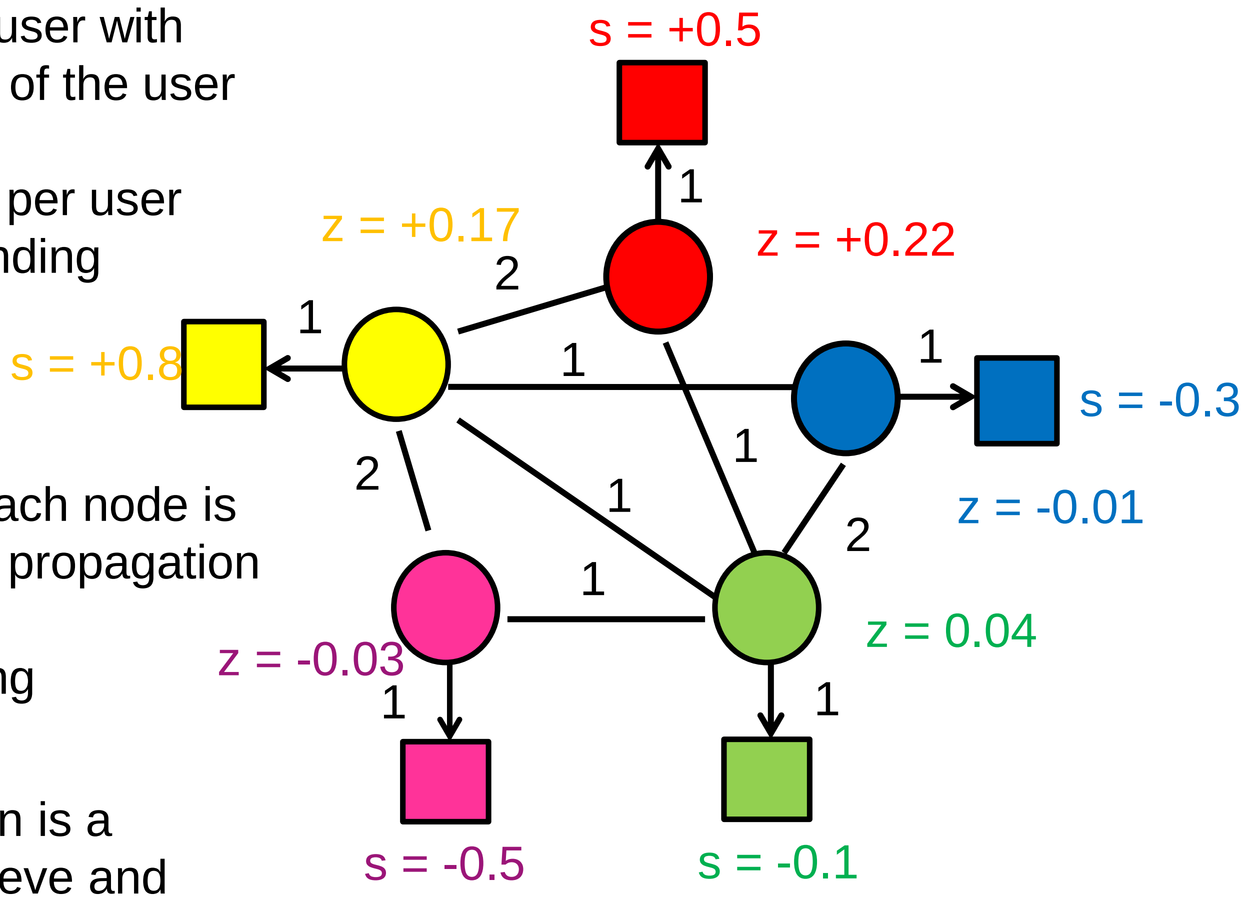
- Social network with internal opinions



Example

One absorbing node per user with value the internal opinion of the user

One non-absorbing node per user that links to the corresponding absorbing node



The external opinion for each node is computed using the value propagation we described before

- Repeated averaging

Intuitive model: my opinion is a combination of what I believe and what my social network believes.

Hitting time

- A related quantity: Hitting time H(u,v)
  - The expected number of steps for a random walk starting from node u to end up in v for the first time
    - Make node v absorbing and compute the expected number of steps to reach v
    - Assumes that the graph is strongly connected, and there are no other absorbing nodes.
- Commute time H(u,v) + H(v,u): often used as a distance metric
  - Proportional to the total resistance between nodes u, and v

Transductive learning

- If we have a graph of relationships and some labels on some nodes we can propagate them to the remaining nodes
  - Make the labeled nodes to be absorbing and compute the probability for the rest of the graph
  - E.g., a social network where some people are tagged as spammers
  - E.g., the movie-actor graph where some movies are tagged as action or comedy.
- This is a form of semi-supervised learning
  - We make use of the unlabeled data, and the relationships
- It is also called transductive learning because it does not produce a model, but just labels the unlabeled data that is at hand.
  - Contrast to inductive learning that learns a model and can label any new example

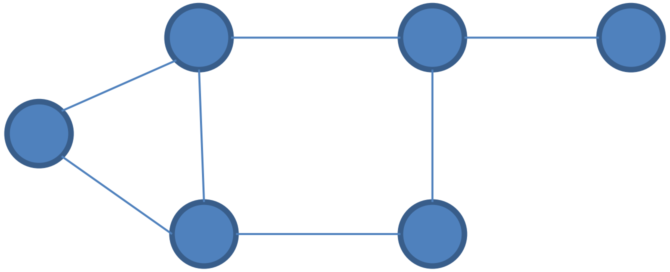
Implementation details

- Implementation is in many ways similar to the PageRank implementation
  - For an edge (u, v) instead of updating the value of v we update the value of u.
    - The value of a node is the average of its neighbors
  - We need to check for the case that a node u is absorbing, in which case the value of the node is not updated.
  - Repeat the updates until the change in values is very small.

COVERAGE

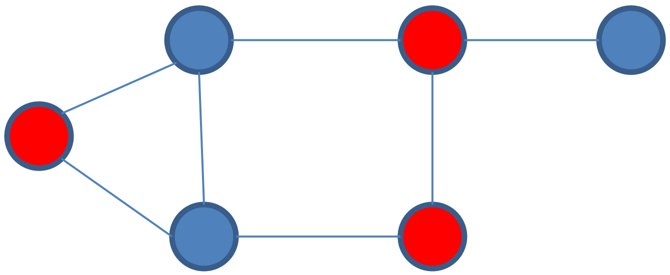
Example

- 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

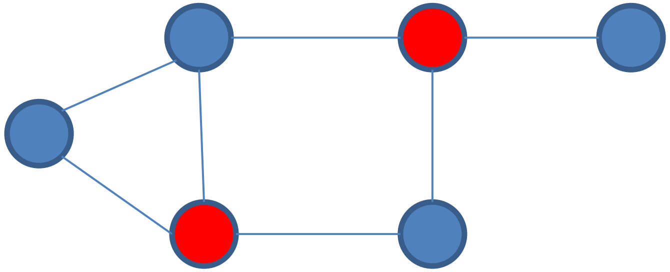
- Promotion campaign on a social network
  - We have a social network as a graph.
  - People are more likely to buy a product if they have a friend who has the product.
  - We want to offer the product for free to some people such that every person in the graph is covered: they have a friend who has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

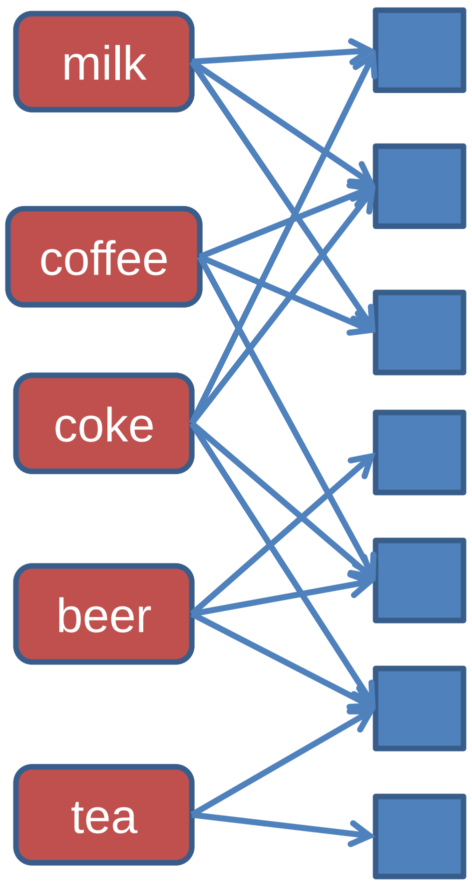
Applications

- 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

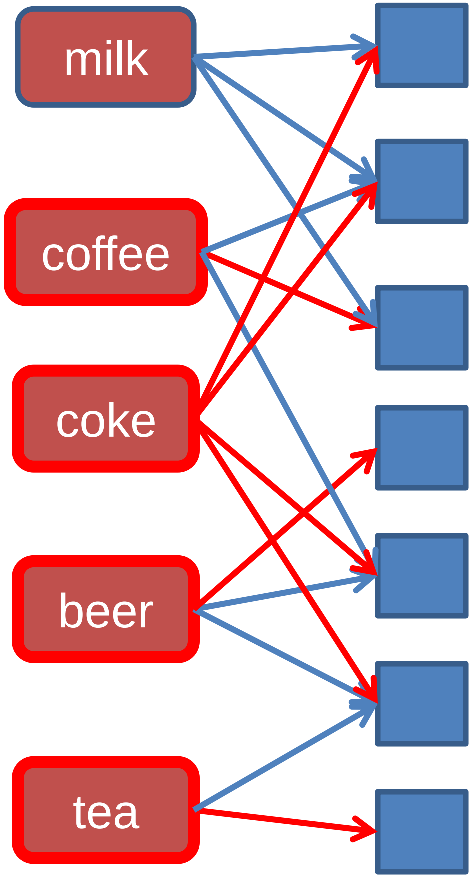
- The universe U of elements is the set of customers of a store.
- Each set corresponds to a product p sold in the store:

  $S_p = \{customers\ that\ bought\ p\}$
- Set cover: Find the minimum number of products (sets) that cover all the customers (elements of the universe)



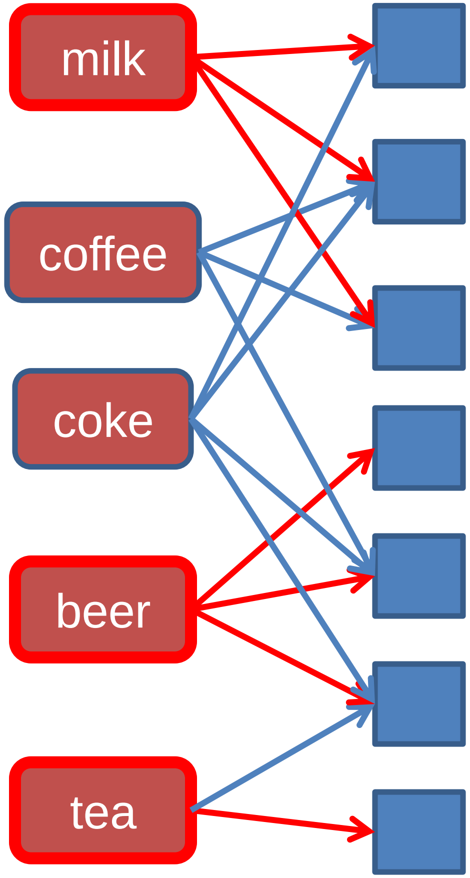
Applications

- 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 U, a collection of $\boldsymbol{S}$ of subsets of U, and a budget K, find a sub-collection $\boldsymbol{C} \subseteq \boldsymbol{S}$ of size 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 to find 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: X = G is the input graph, OPT(G) is the size of minimum set cover, ALG(G) is the size of the set cover found by an algorithm ALG.

- Maximum coverage: X = (G,k) is the input instance, OPT(G,k) is the coverage of the optimal algorithm, ALG(G,k) 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,

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

- 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
  - 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,

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

- 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 $\epsilon \rightarrow 0$, as $n \rightarrow \infty$(e.g., $\epsilon = \frac{1}{n}$)
  - Good case: $\alpha = O(1)$ is a constant
  - OK case: $\alpha = O(\frac{1}{\log n})$
  - Bad case $\alpha = O(n^{-\epsilon})$

A simple approximation ratio for set cover

- Any algorithm for set cover has approximation ratio $\alpha = |S_{max}|$, where S_{max} is the set in **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 can be that $|S_{max}|=O(N)$

An algorithm for Set Cover

- What is the most 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 elements.

The GREEDY algorithm

GREEDY(U,S)

X= U

C = {}

while X is not empty do

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

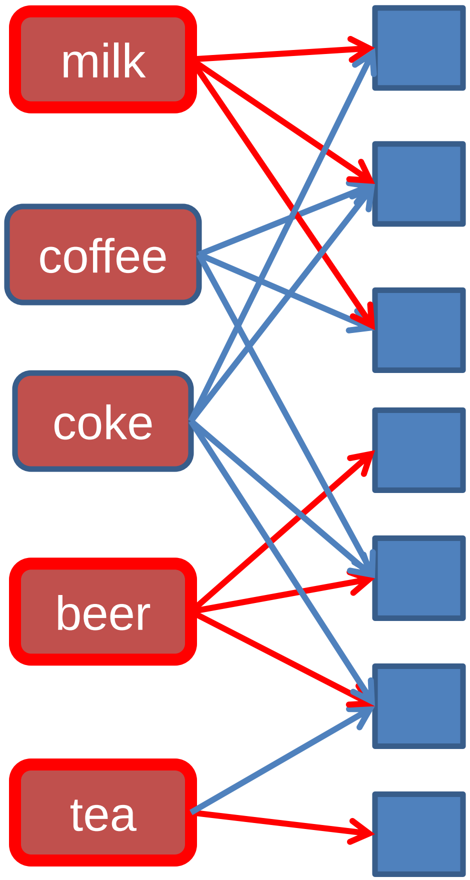
Let S_* be such that $gain(S_*)$ is maximum

C = C U {S_*}

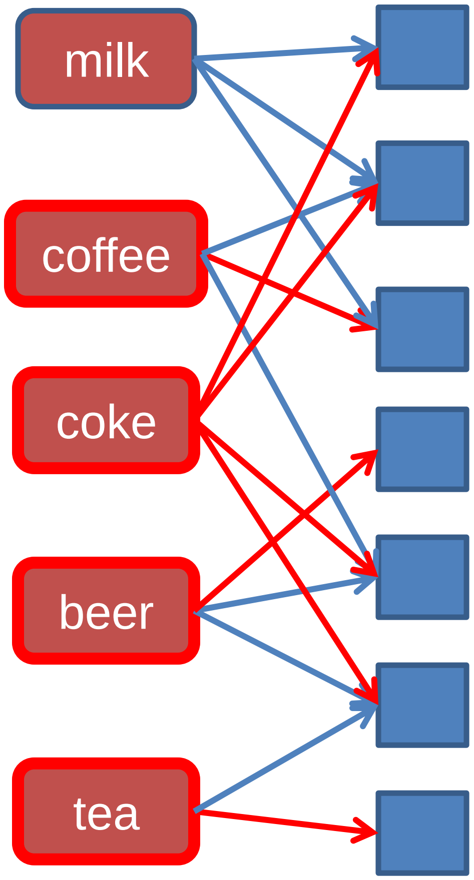
X = X\ S_*

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

Greedy is not always optimal



- Selecting Coke first forces us to pick coffee as well
- Milk and Coffee cover more customers together



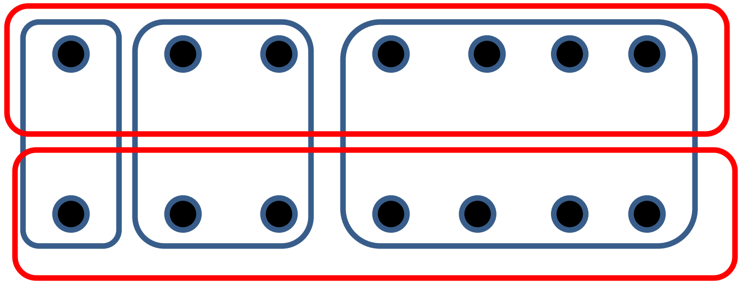
Approximation ratio of GREEDY

- Good news: GREEDY has approximation ratio:

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

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

- 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) = logN
α=½logN

Maximum Coverage

- What is a reasonable algorithm?

GREEDY(U,S,K)

X = U

C = {}

while $|\boldsymbol{C}| < K$

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

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

C = C U {S_*}

X = X\ S_*

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

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

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

Proof of approximation ratio

- For a collection C, let $F(C) = \left|\bigcup_{S_i \in C} S_i\right|$ be the number of elements that are covered.
- The 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: A greedy algorithm that optimizes a monotone and submodular function F, each time adding to the solution C, the set S that maximizes the gain $F(C \cup \{S\}) - F(C)$ has approximation ratio $\alpha = \left(1 - \frac{1}{e}\right)$

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 subset of vertices such that you cover all edges (an endpoint of each edge is in the set)
  - There is a 2-approximation algorithm
- Edge Cover: Select a set of edges that cover all vertices (there is one edge that has endpoint the vertex)
  - There is a polynomial algorithm

THIS IS THE END…

Parting thoughts

- In this class you saw a set of tools for analyzing data
  - Frequent Itemsets, Association Rules
  - Sketching
  - Clustering
  - Minimum Description Length
  - 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.