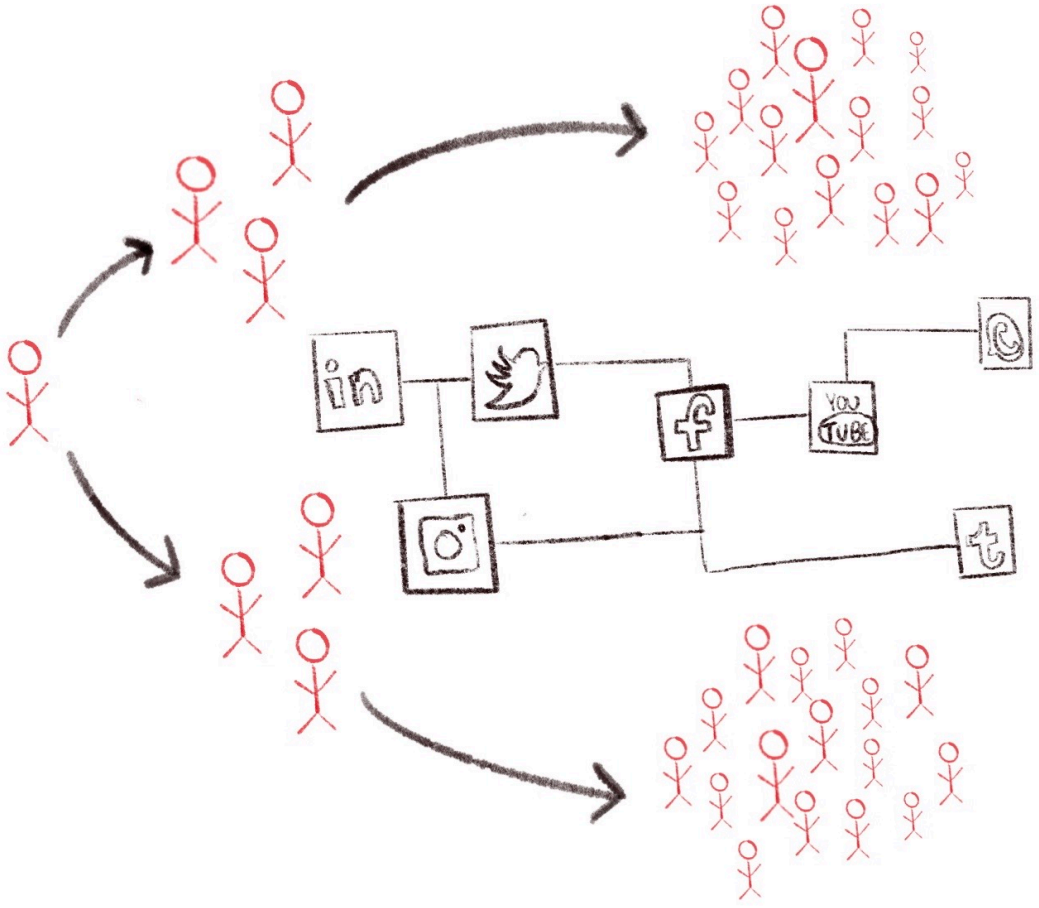
Overexposure in Viral Marketing

Kim Hancock

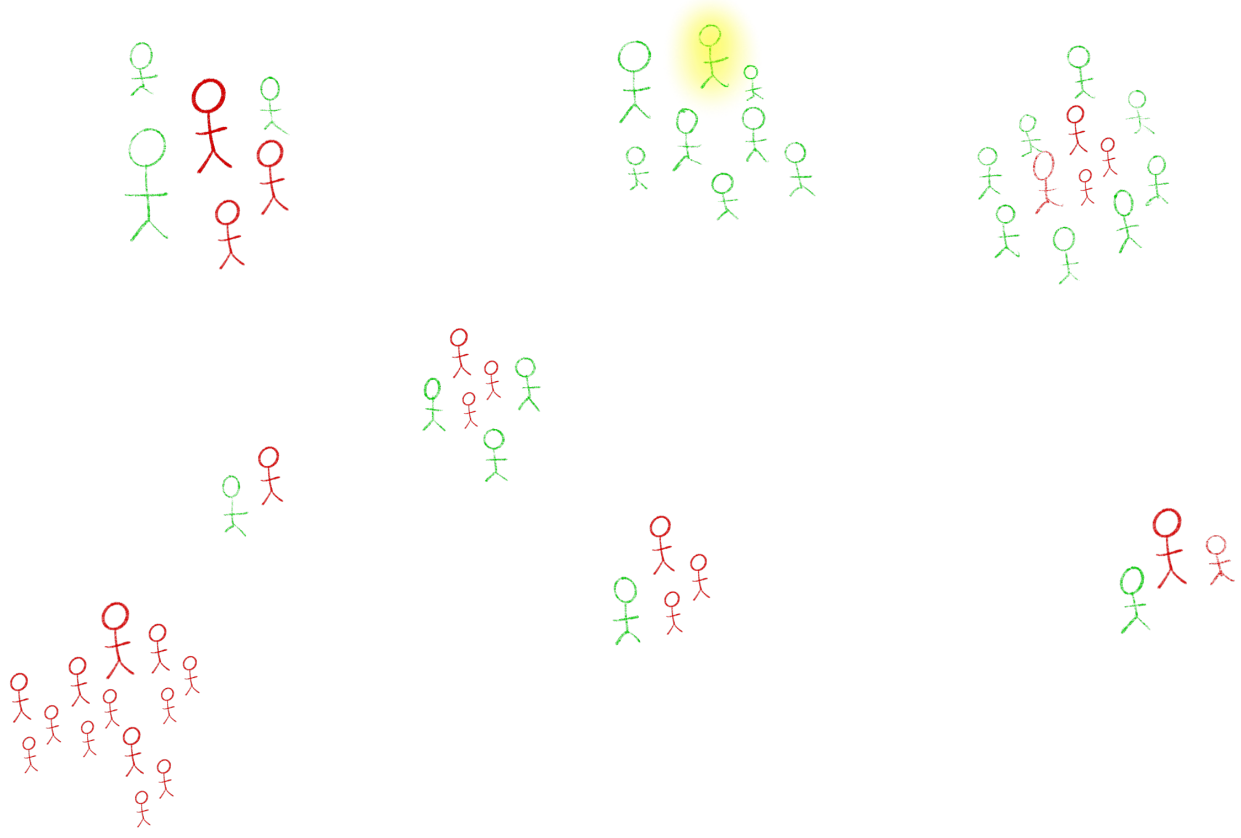
khancock@bowdoin.edu
Honors Project Advisor: Professor Irfan

12/15/2020

First, some preliminaries...



Motivation for this project



- Information cascades in social networks
- Application: viral marketing
- Caution: "too much cascade" may be bad!
    - We call this *overexposure*

Central Research Question

How do we select a set of "initial adopters" that will maximize "payoff" of a cascade?

Preliminary Set-up

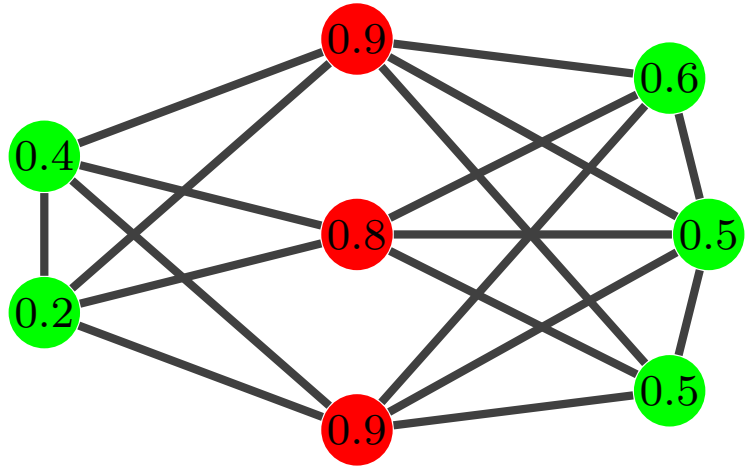


Figure: Graph Set-Up

We consider a graph $G = (V, E)$ where $|V| = n$ and $(u, v) \in E$

- A product has an "appeal" paramter $\phi \in [0, 1]$.
- Here, $\phi = 0.7$
- Each node i in the network has a criticality paramter θ_i
- If $\theta_i < \phi$ they accept the product
- If $\theta_i \geq \phi$ they reject it

Original Graph to Cluster Graph

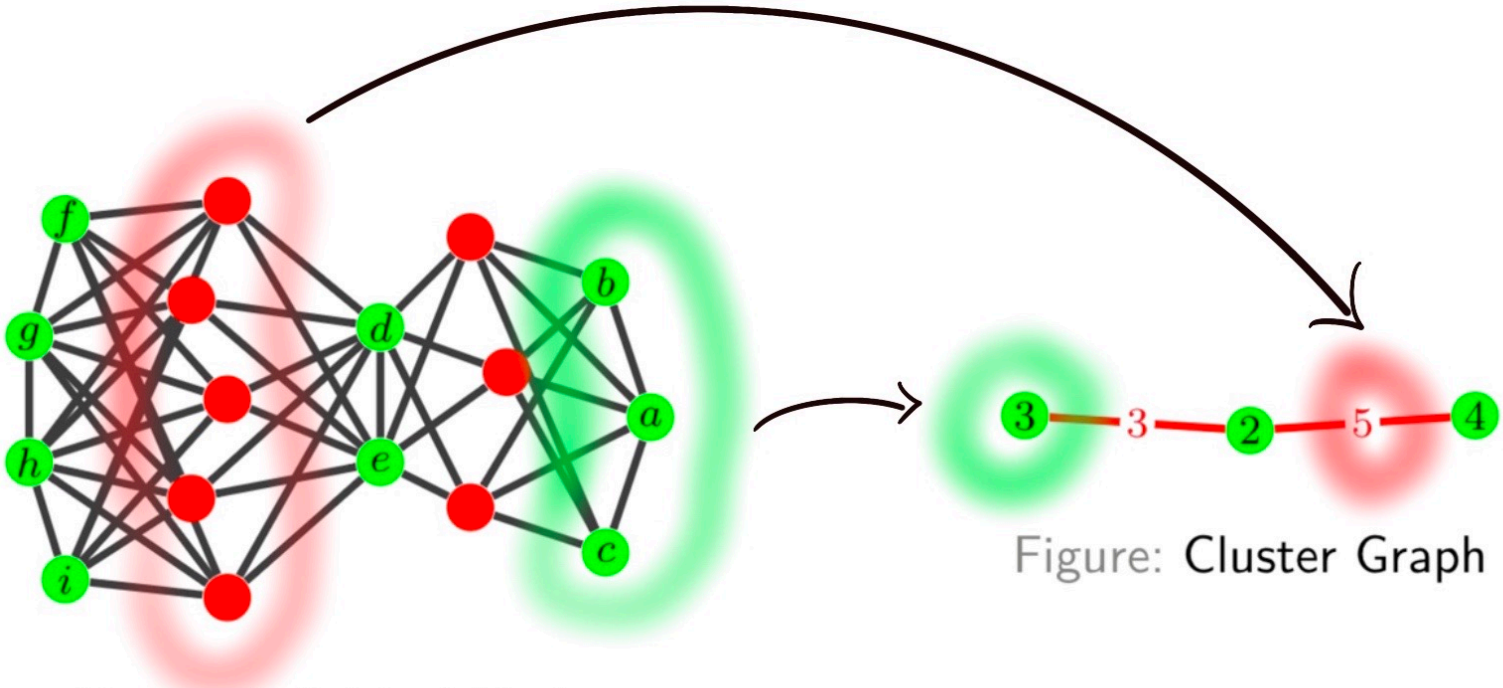


Figure: Individual Nodes

Figure: Cluster Graph

Original Graph to Cluster Graph

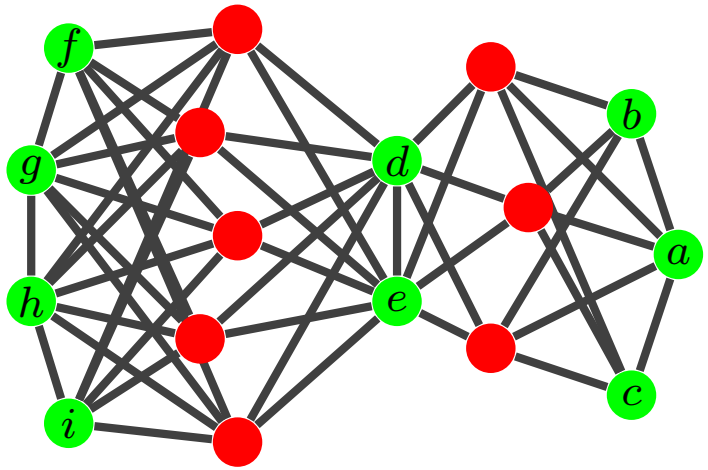


Figure: Individual Nodes

Figure: Cluster Graph

We identify *clusters of accepting nodes* which are groups of nodes surrounded by rejecting nodes. Each group of nodes becomes a cluster node in our new graph!

Previous Works

Models of propogation were first studied in the context of social networks by Kempe 2003 and experiment with both a *threshold* and *cascade* model of influence maximization.

More recent work by Cui et al. 2018 and Iyer and Adamic 2019 also use the cascade model but do not include overexposure explicitly.

Our Work

Abebe et al. 2018 considered overexposure for the first time.
They proved you could find an optimal seed set in polynomial time in the **unbudgeted** case

Theorem 1 (Budgeted Seeding Abebe et al. 2018)

The decision problem of whether there exists a seed set S with $|S| \leq k$ and $\pi(S) > 0$ is NP-complete.

We want to solve this budgeted case:

- Efficient algorithms for special cases
- Heuristics for general case

Our Algorithms

Overview of Algorithms we Have Designed so Far

	Greedy	**Recursive DP**	**Tree Decomposition**	**Integer Linear Program**
Overview	Greedily pick k highest payoff clusters	Recurse on up to k seeds, consider n choose k seeds	When we have cycles, do a tree decomposition	Get an exact answer an integer linear program
Pros	Fast, easy to implement	Gives exact answer for trees	Works on general graph	Exact answer
Cons	Inaccurate	Doesn't work in general case	Approx. factor not proven	Exponential run time

Greedy Approaches

Simple Greedy Algorithm

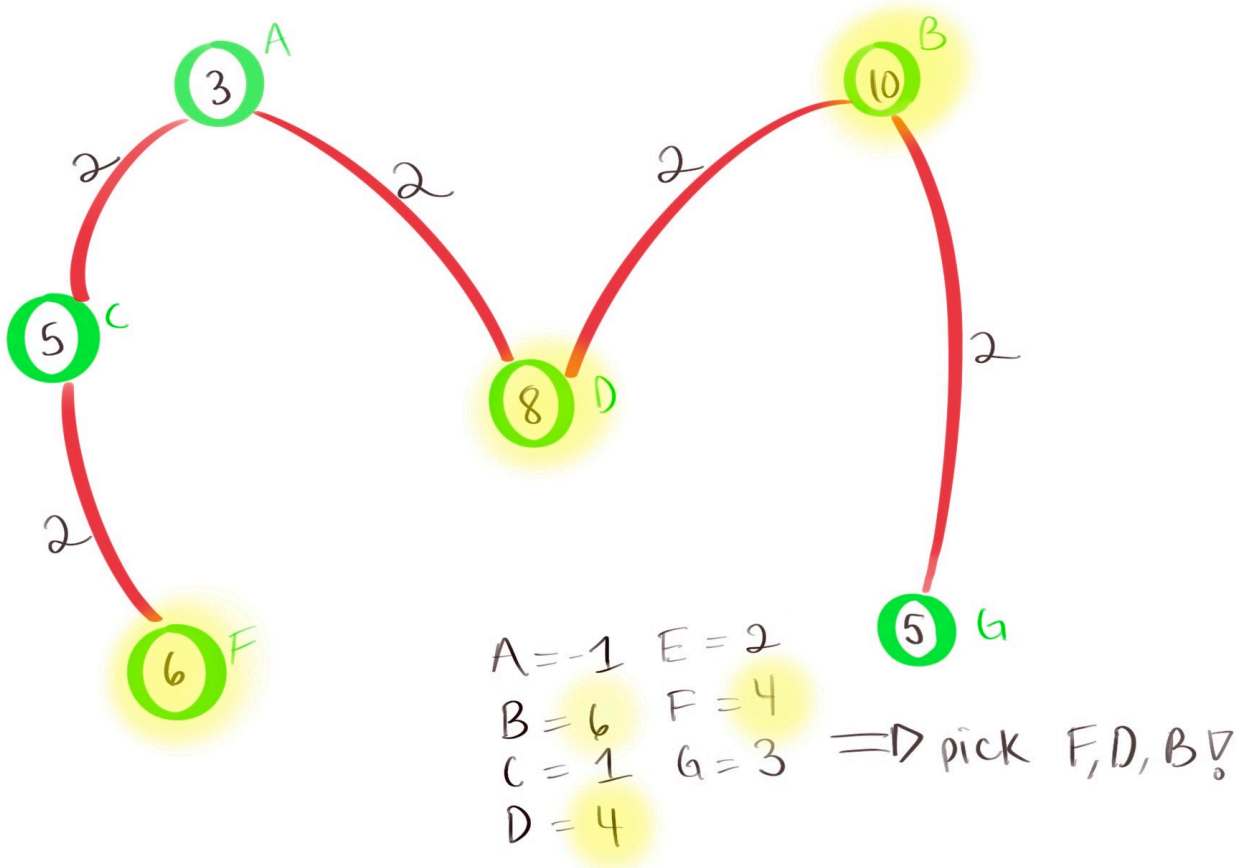
We pick the k highest payoff clusters!

Knapsack Dynamic Programming

We simply create a memoization table and pick clusters iteratively, recursing on *take* and *dont_take*

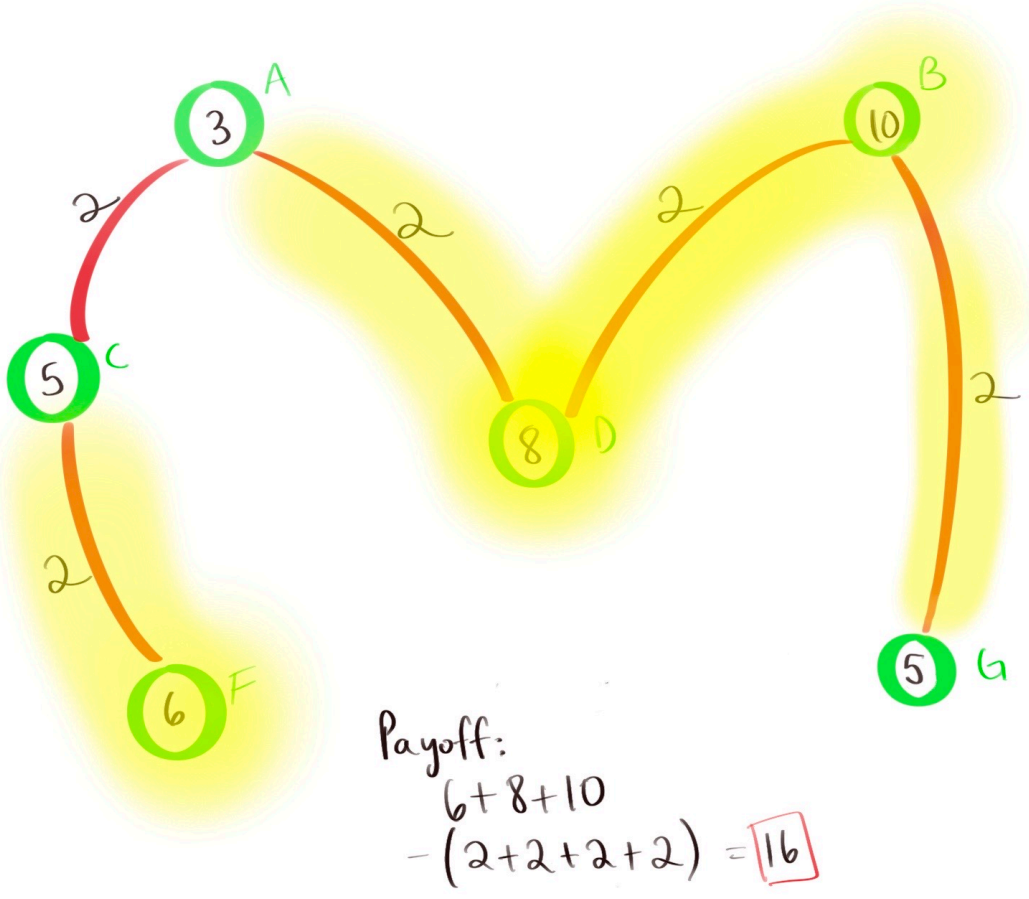
Greedy Example on Cluster Graph

Let's let $k = 3$ and pick the highest clusters x_i such that $x_i - \sum_e x_e$ is maximized



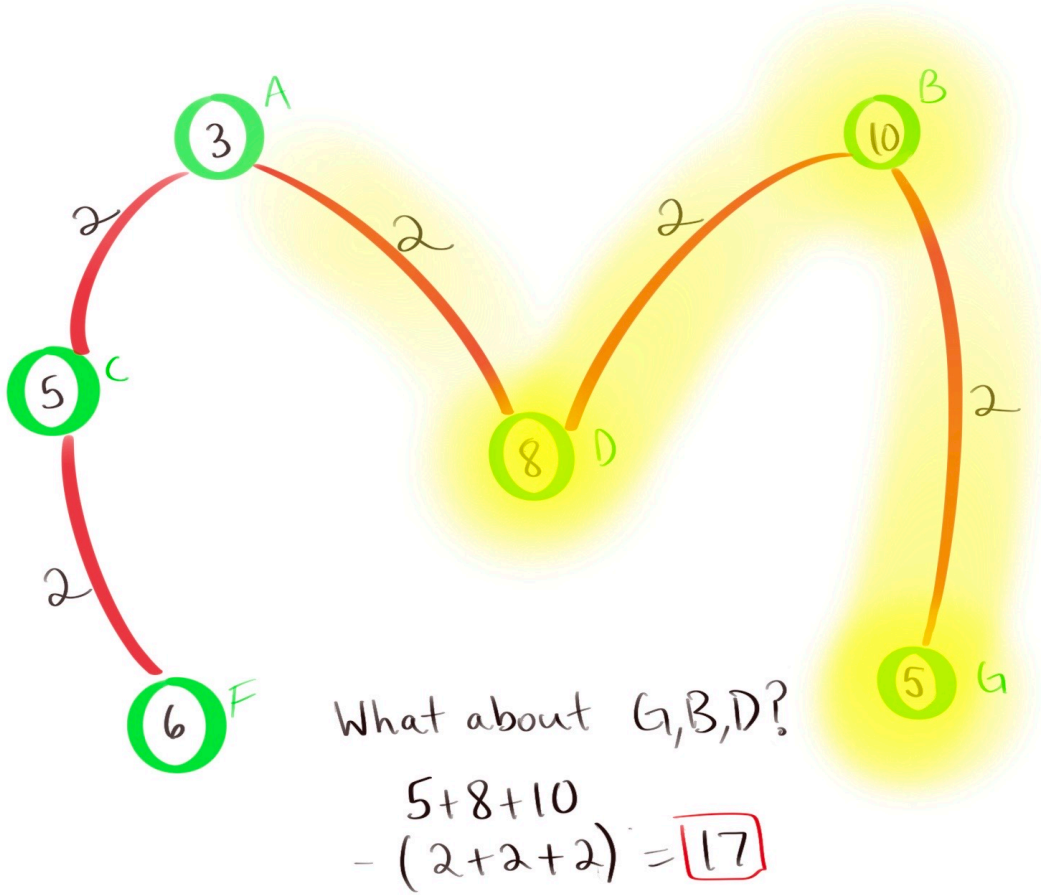
Greedy Example on Cluster Graph

Let's let $k = 3$ and pick the highest clusters x_i such that $x_i - \sum_e x_e$ is maximized



Greedy Example on Cluster Graph

But, we can easily see that there is a higher payoff!



Greedy Example on Cluster Graph

We thus conclude that the greedy approach does not lead to an optimal solution! Neither does knapsack dynamic programming, as there is not optimal substructure. The order in which we consider each cluster matters, and determining an optimal order would be timely.

Linear Program on Cluster Graph

To get to an optimal solution in exponential time, we implemented a linear program.

$$\text{objective function} = \sum_{i=1}^{n} x_{i,i} * w_{i,i} - \sum_{j=1}^{n} x_{e,j} * w_{e,j}$$

Subject to constraints:

$1 \geq x_{i,j} \geq 0$, such that $x_{i,j} \in \mathbb{Z}$ for all $i, j \leq n$

If $w_{i,j} > 0$, then $x_{i,j} \geq x_{i,i}$ for all $i, j \leq n$ and $i \neq j$

$\sum_{i=1}^{n} x_{i,i} \leq k$, where k is the number of nodes we are choosing to seed.

Recursive Dynamic Programming

To approximate a better solution, we designed the following recursive dynamic programming algorithm:

Algorithm 2 Dynamic Programming Algorithm

procedure DP($G=(V,E,W), k, source, storePayoff, witness$)
▷ source is the root of subtree, storePayoff stores memoized payoffs
    precomputed_True_case = precomputed_False_case = False
    if storePayoff[source][k][True] != Null: **then** ▷ if we have already computed subproblem
        precomputed_True_case = True
    end if
    if storePayoff[source][k][False] != Null: **then** ▷ if we have already computed subproblem
        precomputed_False_case = True
    end if
    if $k == 0$ **then** ▷ meaning we have no seeds
        storePayoff[source][k][True] = $-\infty$
        storePayoff[source][k][False] = 0
        return
    end if
    if $G.children(source) = null$ **then** ▷ we are at a leaf node
        Set storePayoff[source][k][True] = W[source] - $W[(source, parent(source))]$
        Set storePayoff[source][k][False] = 0.
        return
    end if
    Case I. Leave source:
    if not preComputed_False_case **then**
        partitions = Set of all possible ways of dividing k seeds among the children of *source*.
        maxSum = -∞
        opt_allocation = Null ▷ allocation of seeds corresponding to max sum of payoffs of children
        opt_take_child? = Null ▷ take or leave the child node when seeds are allocated optimally?
        for p in partitions: **do**
            sum = 0
            Define a map *takeChild*? from each child of *source* to a boolean value (initialized to False).
            for each child i of *source*: **do**
                Call $DP(G, p[i], i, storePayoff, witness)$.
                if $storePayoff[i][p[i]][True] - W[(source, i)] \geq storePayoff[i][p[i]][False]$ **then**
                $sum+ = storePayoff[i][p[i]][True] - W[(source, i)]$
                $takeChild?[i] = True$
                else
                $sum+ = storePayoff[i][p[i]][False]$
                $takeChild?[i] = False$
                end if
            end for
            update maxSum, opt_allocation, and opt_take_child? if needed.
        end for
        $storePayoff[source][k][False] = maxSum$
        $witness[source][k][False] = (takeChild?[i], maxPartition[i])$ for each child i of source
    end if
    Case II. Take source:
    if not preComputed_True_case **then**
        partitions = Set of all possible ways of dividing $k-1$ seeds among the children of *source*.
        maxSum = -∞
        opt_allocation = Null
        opt_take_child? = Null ▷ take or leave the child node when seeds are allocated optimally?
        for p in partitions: **do**
            sum = 0
            Define a map *takeChild*? from each child of *source* to a boolean value (initialized to False).
            for each child i of *source*: **do**
                Call $DP(G, p[i], i, storePayoff, witness)$.
                if $storePayoff[i][p[i]][True] \geq storePayoff[i][p[i]][False]$ **then**
                $sum+ = storePayoff[i][p[i]][True]$
                $takeChild?[i] = True$
                else
                $sum+ = storePayoff[i][p[i]][False] - W[(source, i)]$
                $takeChild?[i] = False$
                end if
            end for
            update maxSum, opt_allocation, and opt_take_child? if needed.
        end for
        $storePayoff[source][k][True] = maxSum + W[source]$
        $witness[source][k][True] = (takeChild?[i], maxPartition[i])$ for each child i of source
    end if
end procedure

Recursive Dynamic Programming

Okay, let's break this down....

- Only works when the input graph is a tree
- Gives an exact answer (which is awesome!)
- Exponential in the number of seeds
- Compute all possible ways of seeding k clusters among j subtrees, where j is the number of child nodes.

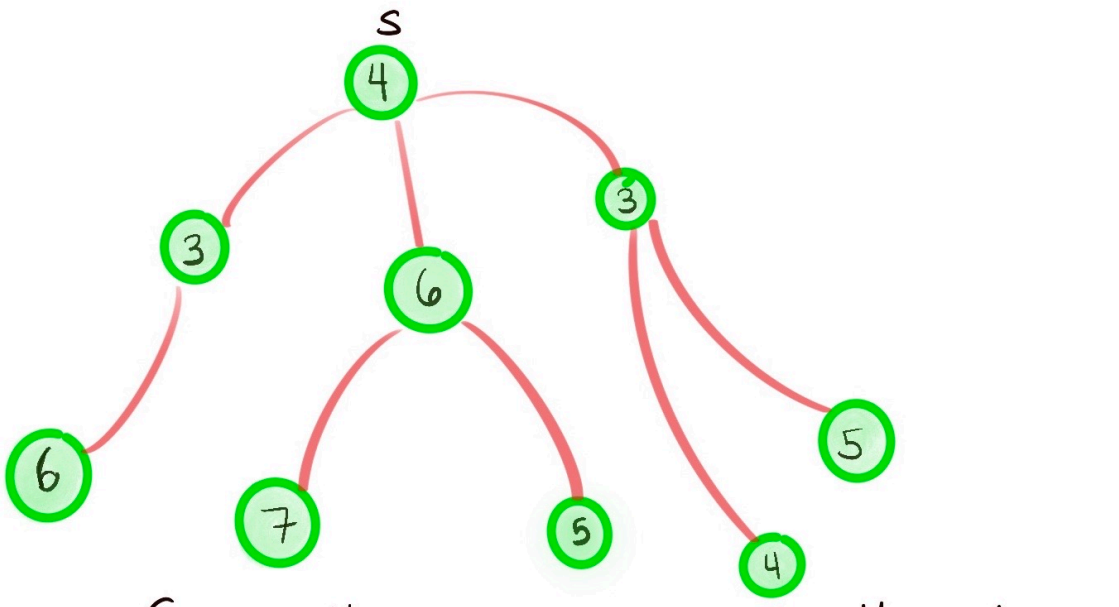
Recursive Calls

$DP(G, p[i], i, storePayoff, witness)$

Where

- $p[i]$ is the number of seeds we are allocating to the subtree rooted at node i
- storePayoff is our memoization table, and witness is a vector storing which seeds we are taking

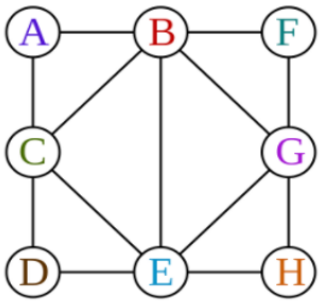
Recursive Dynamic Programming

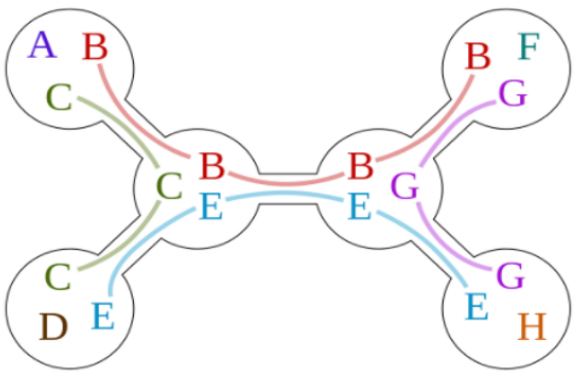


So, we have $\binom{n+k-1}{n-1}$ ways of allocating k seeds among n subtrees. For this example, that is $\binom{6}{2} = 15$ ways!

Tree Decomposition

What do we do when we have cycles in our input graph?
Answer: Tree decomposition!





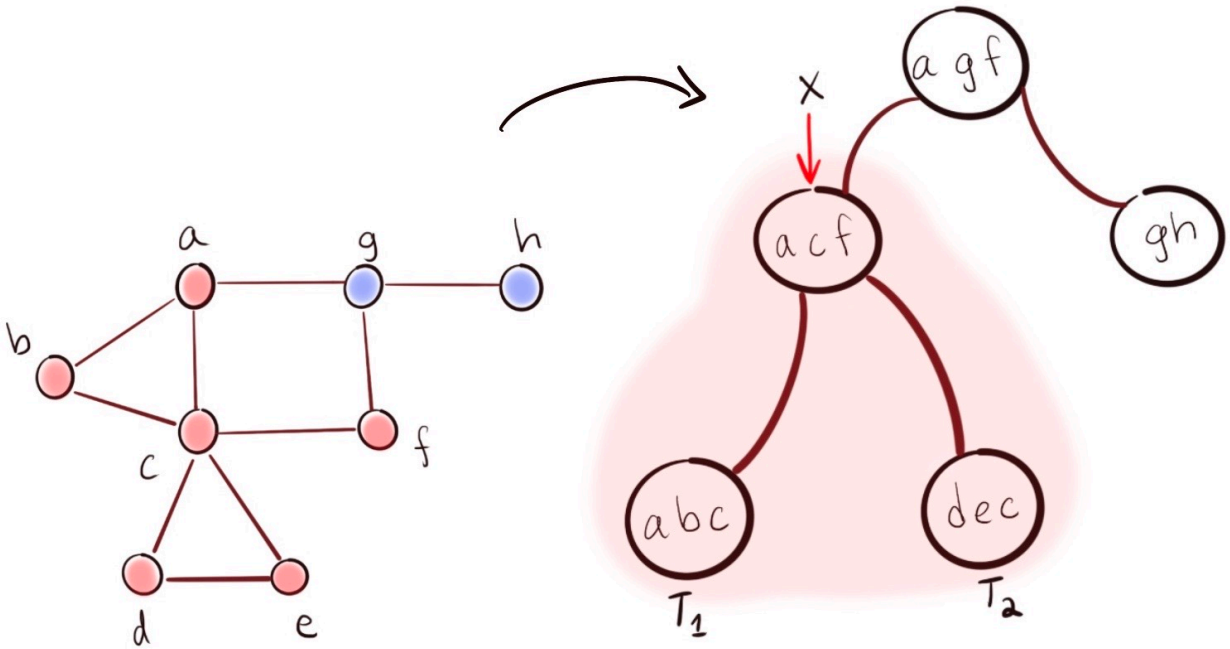
Tree Decomposition

We denote the "bags" of the tree decomposition T X_i

For every edge $(u, v) \in G$ there must be some bag X_i which includes both u and v

If X_i and X_j contain v, then all nodes between them contain v

Tree Decomposition Algorithm



The optimal value at the subroot X for including a subset of nodes in the subtree is computed by considering the optimal values in T_1 and T_2

Details of Recursive Formulation

Then, we have two recursive calls. The first is the max payoff for selecting $S \subseteq X_i$:

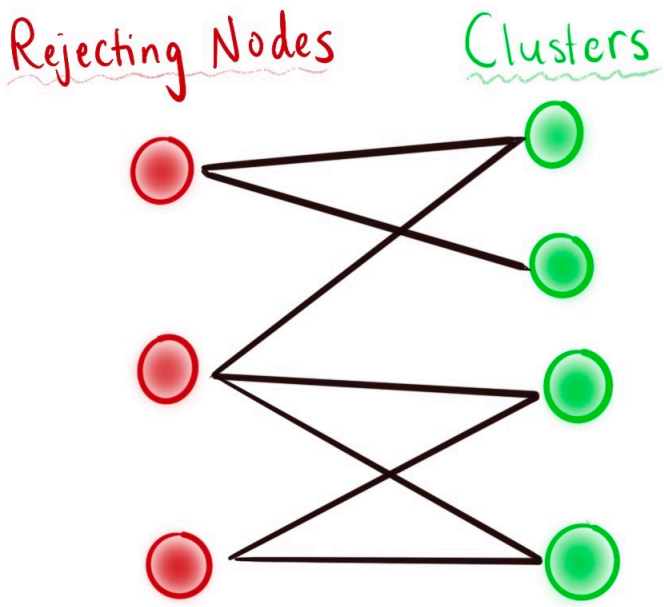
$$A(S, i, k) = \max_p \left[w_S - w_{E_S} + \sum_j \left[B(S \cap X_j) - w_{S \cap X_j} + w_{E_{S \cap X_j}} \right] \right] \tag{1}$$

where w_S is the total sum of the weights in S and w_{E_S} is the sum of the edges incident on S. We also have that p is where we enumerate all possible seeds.
Finally, we have our second recursive call:

$$B(T, j, i) = \begin{cases} -\infty & |T| > l_j \\ \max(A(T', i)) & |T| \leq l_j \end{cases} \tag{2}$$

where $T \subseteq T' \subseteq X_j$. Thus, T' is a superset of $T \subseteq X_i \cap X_j$.

Generalized Linear Program



Generalized Linear Program

$$\text{Max}\left(\sum_{i \in C} w_i x_i - \sum_{r \in R} y_r\right)$$

such that:

1) $\sum_i x_i \leq k$ BUDGET

2) $x_i \in \{0,1\}$ TAKE or NOT

3) $y_r \geq x_i \quad \forall (r,i) \in G$

IF WE TAKE A CLUSTER, MUST TAKE ALL REJ NODES ITS ATTACHED TO

Future Work

General Case

Improve a linear program that works in the most general case which

Real-life examples

Compare the outcomes of these algorithms on real life networks of > 2000 nodes

Formally Writing Proofs

We are confident these algorithms work but want to prove approximation factors

Special Thanks

Special thanks to the computer science department at Bowdoin and professor Irfan for giving me the opportunity to do this project!

Bibliography

Bibliography I


Abebe, Rediet, Lada A Adamic, and Jon Kleinberg (2018). "Mitigating overexposure in viral marketing". In: *Thirty-Second AAAI Conference on Artificial Intelligence*.


Cui, Fang, Hai-hua Hu, Wen-tian Cui, and Ying Xie (2018). "Seeding strategies for new product launch: The role of negative word-of-mouth". In: *PloS one* 13.11, e0206736.


Iyer, Shankar and Lada A Adamic (2019). "When can overambitious seeding cost you?" In: *Applied Network Science* 4.1, p. 38.


Kempe Kleinberg, Tardos (2003). "Maximizing the Spread of Influence through a Social Network". In: *ACM*.