Optimizing Outbreak Detection for Real World Networks

Qianying Lin[*], Chuanqi Shen[†], Kaidi Yan[‡]

December 8, 2015

1 Introduction

Influence Maximization and outbreak detection are two problems that have seen massive research in recent years, for good reason. For example, targeted tweets by well-selected Twitter users can lead to massive reshares, resulting in a quick dissemination of information. Well-placed sensors can allow contamination in a water pipeline network to be detected quickly. Although the two problems may seem entirely different, they are in fact modeled under the same framework. However, it has been known for a while that the problem is intractable in general. Due to this, much research has been done in approximation algorithms in an attempt to find provably good solutions efficiently. We proceed down the same path, and attempt to design efficient outbreak detection algorithms and test it on synthesized network data.

2 Background

Given an underlying network, the influence maximization problem asks what is the optimal way to place "influencers" in a network under some budget constraints, so that the influence is maximized, while the outbreak detection problem asks what is the optimal way to place "detectors" such that the spread of influence is minimized. Despite seemingly opposite optimization aims, the two problems can be described using the same framework.

To model either problem, one needs to first define how information will spread over the network. Such models are called diffusion models, and two of them are popularly used in the context of influence maximization and outbreak detection. The first model is called the *Linear Threshold Model*, first proposed by Granovetter and Schelling [5, 6]. In this model, the activity of a node is controlled by a specified function; when value of the function goes above a defined threshold value, the node becomes active. The second model is called the *Independent Cascade Model*, where the influence process is simulated in discrete steps. In the first step (step 0), the a set of active nodes A_0 was set, and in step i, each neighbour of each node x in A_{i-1} can be activated with some independent probability dependent on x. The newly activated nodes forms A_i.

No matter which model is used, the influence maximization problem was known to be NP-hard. Hence, no tractable solution exists, and focus shifted towards designing efficient algorithms that can find good approximate solutions close to the optimal.

In the seminal and highly-cited paper by Kempe et al [3] in 2003, Kempe et al realized that many of the evaluation functions used in such problems exhibit the problem of submodularity.

[*]qlin1@stanford.edu, Stanford University
[†]shencq@stanford.edu, Stanford University
[‡]kaidi@stanford.edu, Stanford University

Given this, a greedy algorithm with a good approximation factor was created. This algorithm keeps picking the vertex that maximizes the marginal gain. Leskovec et al [1] implemented this idea to create CELF, an optimized version of the greedy algorithm that proved to work quickly in practice. Goyal et al [2] further built upon this framework and created CELF++, which ran even more quickly on networks with uniform node activation cost. CELF and CELF++ represent the current state-of-the-art algorithms for solving the influence maximization problem.

Despite this, the algorithms implemented in CELF and CELF++ are essentially the same greedy algorithm devised by Kempe at al. Borg et al [8] developed an influence maximization algorithm that is fundamentally different from the method used by Kempe et al, but is able to attain an approximation ratio of $O(1 - 1/e - \epsilon)$ and runs quickly in $O((k + \ell)(m + n) \log n/\epsilon^2)$ time. It makes use of the concept of a reverse-reachable set. However, it has large memory footprint and only works on the Independent Cascade Model. Tang et al [9] improved upon Borg et al's work and developed an algorithm with a smaller memory footprint, and also works for Linear Threshold models. However, Borg et al and Tang et al's approaches only work for influence maximization.

Research has also been done for special graphs. For example, Chen et al [7] formulated a linear-time algorithm that finds the exact solution for directed acyclic graphs. The methods however, cannot be generalized to other types of graphs.

In this project, we aim to derive methods that work better than the state-of-the-art methods on real world networks.

3 Preliminaries

We will investigate the problem on the Independent Cascade Model for an undirected graph $G = (V, E)$, and where the propagation probability is 1 (cases where the propagation probability is not 1 can be reduced to this case). In our model, nodes are either active or inactive. At the beginning, a set of nodes A_0 is selected and set to active. At time i, all inactive nodes neighouring to nodes in A_{i-1} are set to active and are defined as the set A_i.

Given our undirected graph G, and a certain budget B allocated to us, both the influence maximization and outbreak detection problem can be modeled as the following:

$$\text{optimize}_{A \subseteq V} f(A) \text{ subject to } c(A) \leq B$$

where $f(A)$ is a function that evaluates how good our selection of nodes is. In our model, we define that an outbreak can start uniformly randomly from a single node and spread according to the Independent Cascade Model defined above. The function $f(A)$ we use is defined to be the expected time step we detect the outbreak. Thus,

$$f(A) = \frac{1}{|V|} \sum_{v \in V} d_A(v), \text{ where } d_A(v) = \min_{a \in A} d(v, a)$$

and $d(u, v)$ denotes the distance between u and v. We want to minimize $f(A)$. We will call $f(A)$ the penalty of the set A.

$c(A) = \sum_{a \in A} c(a)$ denotes the cost of placing detectors in the locations A. We are using uniform-cost i.e. $\forall v \in V, c(v) = k$ for some constant k. For simplicity, we set $k = 1$.

The function f we chose is special because it is submodular.

Definition 1. *A function $f : 2^V \to \mathbb{R}$ is called submodular if $\forall P, Q, V$ with $P \subseteq Q \subseteq V$ and $\forall v \in V \backslash Q$ we have*

$$f(P) - f(P + \{v\}) \geq f(Q) - f(Q + \{v\})$$

Qualitatively, a submodular function describes a function with diminishing returns i.e. the marginal gain received from adding a vertex decreases as our set grows larger.

Lemma 1. *If* $f(A) = \frac{1}{|V|}\sum_{v\in V} d_A(v)$*, then* f *is submodular.*

Proof. Pick any P, Q with $P \subseteq Q \subseteq V$ and pick $u \in V \backslash P$. Note that since $P \subseteq Q$, this means $\forall v, d_P(v) \geq d_Q(v)$. Then

$$\begin{aligned} f(P) - f(P + \{u\}) &= \sum_{v\in V} d_P(v) - \min(d_P(v), d(u,v)) \\ &= \sum_{v\ inV} \max(0, d_P(v) - d(u,v)) \\ &\geq \sum_{v\in V} \max(0, d_Q(v) - d(u,v)) \\ &= \sum_{v\in V} d_Q(v) - \min(d_Q(v), d(u,v)) \\ &= f(Q) - f(Q + \{u\}) \end{aligned}$$

□

Submodular functions are interesting because they allow the following greedy algorithm to work well. In high level, the algorithm continually adds the vertex v that maximizes the marginal gain $f(S \cup \{u\}) - f(S)$ until the budget is used up.

Algorithm 1 GREEDY

```
1: procedure GREEDY
2:     S ← ∅, T ← V
3:     while cost(S) < Budget do
4:         v ← argmin_{u∈T} f(S ∪ {u}) − f(S)
5:         S = S ∪ {v}, T = T \ {v}
6:     end while
7: end procedure
```

Theorem 1. *(Kempe et al) GREEDY has an approximation factor of* $1 - 1/e$*.*

In theory, this is in some sense the best approximation ratio that can be attained, and attaining a better approximation ratio would mean the collapse of the polynomial time hierarchy. In practice however, GREEDY often achieves better results. CELF and CELF++ are both optimized versions of GREEDY.

4 Algorithm

We came up with two possible improvements to current methods to solving the outbreak detection problem.

4.1 DEGREE

Algorithm 2 describes DEGREE algorithm. DEGREE is a modification of the original greedy algorithm implemented by CELF and CELF++. In our modified algorithm, after we include vertex v into our solution set, we remove all neighbors of v from our consideration, as shown in line 7 below. The idea here is that selecting a neighbor does not change the evaluation function by much, since the change in distance is at most 1. However, the time taken to compute the solution should improve by a lot because there are fewer vertices in consideration. We call this algorithm DEGREE, because the number of nodes we remove from consideration (and hence the speedup) is proportional to the degree of the nodes we add into our set of detectors.

Algorithm 2 DEGREE

```
1: procedure DEGREE
2:     S ← ∅, T ← V
3:     while cost(S) < Budget do
4:         v ← argmin_{u∈T} f(S ∪ {u}) − f(S)
5:         S = S ∪ {v}, T = T \ {v}
6:         for v′ : neighbor(v) do
7:             T = T \ {v′}
8:         end for
9:     end while
10: end procedure
```

4.2 Threshold-Outbreak-Detection (TOD)

Algorithm 3 describes TOD algorithm. This algorithm takes inspiration from Borgs et al.s method, which makes use of the reverse influence sampling. We first make the following definition:

Definition 2. *Let v be a node in graph G. The k-reverse-reachable (k-RR) set of v is defined as the set of nodes in G that can reach v in no greater than k steps.*

This definition is adapted from the definition used by Borg et al, which is for directed graphs. By definition, if a node u appears in a k-RR set of v, then by setting a detector at u, we can detect an outbreak starting from v in at most k units of time. In the start of TOD algorithm, we choose a random sample of θ nodes, and generate k-RR sets for each of these nodes. Then we try to solve the maximum cover problem on these k-RR sets using greedy algorithm. Intuitively, the set of nodes TOD algorithm returns roughly covers the largest number of k-RR sets. Thus, they are able to detect all the sample nodes in at most k units of time. If the initial sample of nodes are representative enough, we have high confidence that the expected time to detect an outbreak is around k.

There are two key parameters in this algorithm - θ and k. We need to make sure θ is large enough so that the sample is representative of the entire graph, and k not too large so that our solution is more likely to be close to the optimal solution. Due to the complexity of the problem we are unable to produce a satisfactory probability analysis here. Instead, through empirical experiments we find that TOD produces the best result when we set k to be equal to half the diameter of the graph. Theoretically, it's best to include all nodes in the sample (this is the most representative). Considering the computational efficiency, however, we choose to let θ equal to a 1/10 the total number of nodes in the graph.

Algorithm 3 TOD

```
procedure TOD
    S ← ∅, R ← ∅
    Generate θ random k-RR sets and insert them into R
    while cost(S) < Budget do
        v ← node that covers most k-RR sets in R
        S = S ∪ {v}
        Remove all sets in R that are covered by v
    end while
end procedure
```

5 Experimentation

To determine how well our algorithms work, we test them on generated artificial graphs that have different properties. The list of graph types and their properties and parameters is shown in Figure 1 and Figure 2.

For each type of graph, we run CELF, CELF++, DEGREE, random and TOD and evaluate the algorithms based on the time taken and the quality of the solution found. For each graph, we assume a uniform cost scenario and vary the amount of budget available. Lastly, for each graph we also run the most naive random algorithm (choosing a random sample that has cost within budget) as a result comparison benchmark.

We use the following methodology to make sure that the number of nodes and edges for each graph is uniform. We set the number of nodes to be 1000 for each type of graphs and the total number of edges to be around 5000. For Erdos-Renyi Model and Configuration Model, we can set the number of nodes and total number of edges. For both Barabasi-Albert and Geometric Configuration Model, we set the number of nodes and the average number of edges. For Forest Fire Model and Stochastic Block Model, since only the probability of edges could be set, we run the data generation process, until the number of edges is within the range [4800, 5200].

We generate 50 graphs for each type of graphs. The rationale is that since we use artificially generated dataset compared to real world dataset we need to take account into possibilities of anomalies. When we generate the statistics for each type of graphs, we generate both the mean and the standard deviation to make sure each model is well represented.

Graph Type	Degree Distribution	Clustering effect
Erdos-Renyi	Poisson	No
Barabasi-Albert	Power law	No
Configuration model/ Chung-lu model	Power law	No
Geometric Configuration Model	Power law	No
Forest Fire Model	Power law	Yes
Stochastic Block Model	Poisson	Yes

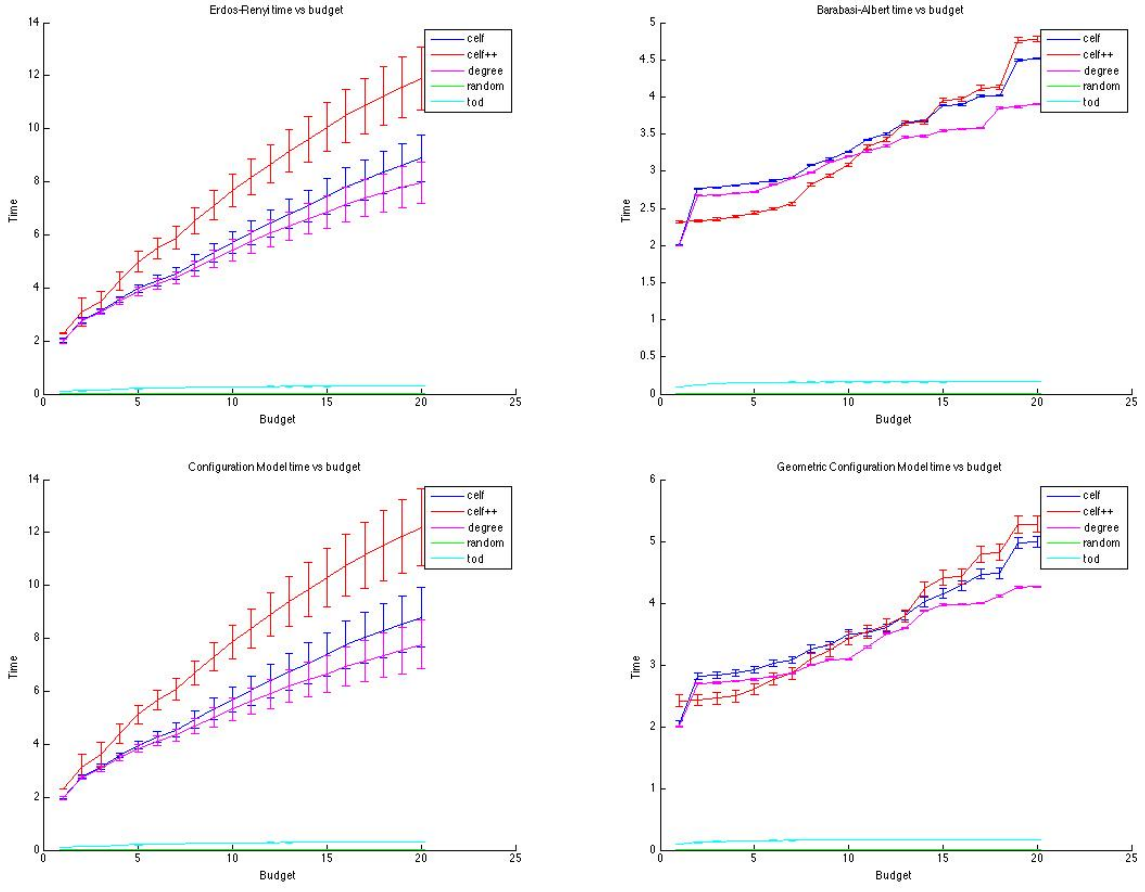
Figure 1: List of artificial graph types generated.

Graph Type	number of nodes	parameter
Erdos-Renyi	1000	5000 edges
Barabasi-Albert	1000	5 edges per node
Configuration model/ Chung-lu model	1000	5000 edges
Geometric Configuration Model	1000	5 edges per node
Forest Fire Model	1000	$f = 0.35, b = 0.354$
Stochastic Block Model	1000	4 partitions, $\alpha = 0.01, \beta = 0.01$

Figure 2: List of graph parameters. f is forward probability and b is backward probability, while α is the intra-probability and β is the inter-community probability

6 Results and Discussion

6.1 Runtime



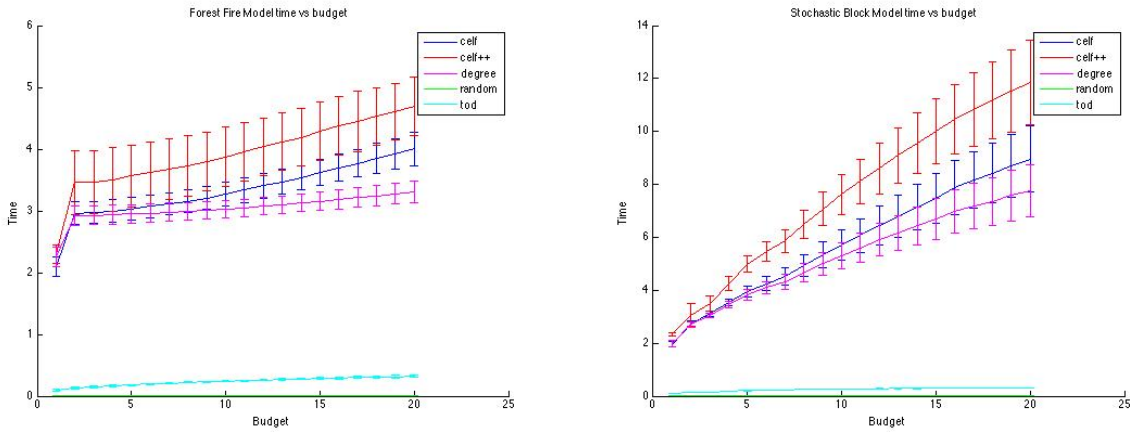


Figure 3: Results against time

Figure 3 shows the runtime of various algorithms on different graph types.

The first observation is that regardless of the types of graphs, CELF and CELF++ always have longer runtime than our own algorithms. DEGREE algorithm share similar runtime with CELF (DEGREE is slightly faster), but the gap between the two runtimes increases as budget increases. When budget reaches 20, the runtime of DEGREE is generally more than 10% less than that of CELF. TOD algorithm has even better runtime performance - its runtime is less than 5% that of CELF for all budgets we explored.

The second observation is that regardless of the graph types, TOD algorithm's runtime is almost constant - this is because the time spent on solving the set cover problem is much less than the time spent on generating k-RR sets. Therefore as budget increases, the runtime advantage of TOD over CELF, CELF++ and DEGREE is more obvious.

The last observation is that there is no observable pattern on the influence of degree distribution or clustering effect on the relative difference in runtime. In other words, graph types do not have a huge observable impact on the resultant runtime difference.

One surprising fact we observe is that CELF++ has worse runtime performance as compared with CELF for all types of graphs. The reason is that when performing the greedy choice, CELF++ performs additional marginal gain computations beforehand in the hope that future operations (marginal gain computations) can be reduced significantly. The number of nodes chosen (which is controlled by the budget) needs to be large enough before break even happens. We chose a maximum budget of 20 while Goyal et al chose a budget of 100 when running their experiments.

6.2 Penalty

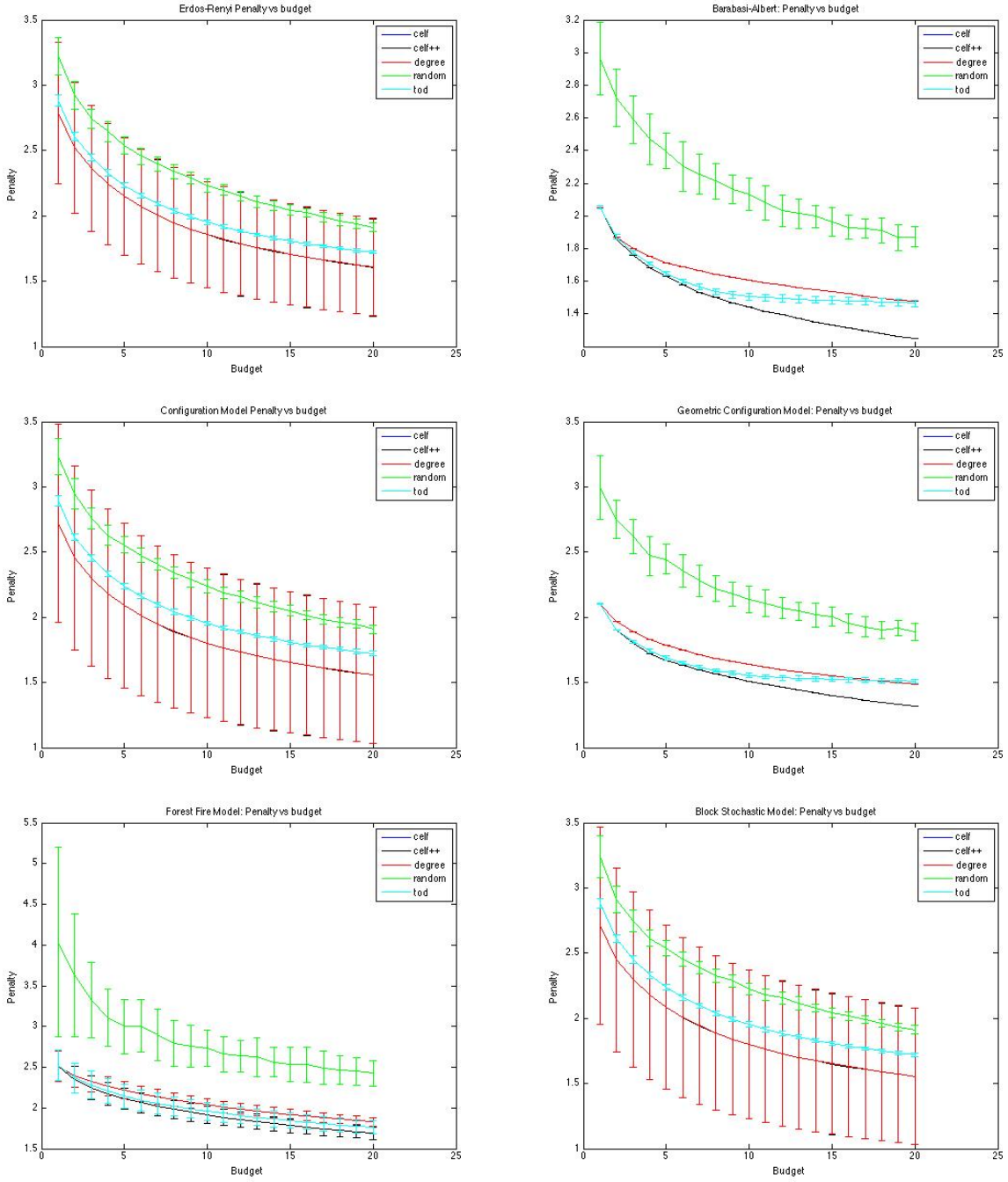


Figure 4: Results against penalty

Figure 4 shows the penalties of various algorithms on different graph types.

The first observation is that random sampling algorithm always has the worst penalty for all types of graphs. This is an expected result since we only use it as a lower bound for our algorithms' performance. Another observation is that CELF and CELF++ always share the same penalty - this is also expected as CELF++ only does (supposedly) runtime improvement over CELF without modifying the output result.

For Erdos-Renyi Model, Configuration Model and Block Stochastic Model, the penalties of CELF, CELF++ and DEGREE overlap with each other. TOD's penalties are worse than that of CELF, CELF++ and DEGREE. Nevertheless for all these models, TOD's penalty is only less than 10 % higher than that of CELF when budget reaches 20, which is within tolerance range.

For Barabasi-Albert Model and Geometric Configuration Model, both DEGREE algorithm and TOD are similar and close to the penalty of CELF and CELF++. Another observation is that as budget increases, TOD performs even better than DEGREE algorithm. One possible reason is that as budget increases, many more neighbors are removed in graphs of these two models; hence DEGREE's performance is likely to hurt more, resulting in worse penalty.

For Forest Fire Model, TOD performs better than DEGREE algorithm throughout different budgets.

Overall, we find that the penalty performance patterns are dependent on different types of graphs; yet none of the the two attributes, degree distribution and clustering effect, seem to play a dorminant role in the performance of different algorithms.

6.3 Performance Discussion

Ultimately, any algorithm is a trade-off between penalty and runtime, one extreme being the random algorithm (very low runtime and very high penalty). Based on this criterion, TOD algorithm is a viable alternative to CELF and CELF++. This is because the penalty performance of TOD algorithm is not much worse compared to CELF and CELF++ yet the improvement in runtime performance is more than 20 times on average based on empirical data. Furthermore, the drastic improvement in runtime performance applies to all graph types. DEGREE provides a small speedup without affecting the penalty when the budget is small, and can be considered when the quality of solution is of bigger importance.

7 Conclusion

While we hypothesize the difference in qualities of our solutions on different types of graphs, we cannot find a single characteristic that defines the quality of solutions. In particular, neither degree distribution nor clustering pattern could have a observable huge impact on the penalty level and the runtime performance of our algorithms.

Generally as a trade-off between quality and speed of solutions, TOD algorithm is much faster than CELF and CELF++ while its penalty is higher only with a reasonable margin. Therefore, it could be regarded as a good approximation algorithm to replace CELF and CELF++ when the graph size is moderate.

8 Future Plan

There are several directions for future work.

1. Since TOD algorithm performs extremely well on Forest Fire Model, one future task could be to explore the particular structure of Forest Fire Model by varying the forward and backward probabilities in graph generation.

2. We can explore other graph properties such as betweenness centrality to understand the performance of DEGREE and TOD on these characteristics.

3. We ran our algorithms on artificially generated data because we want to know the average performance of our algorithms on graphs with specific properties. In the future, we can test our algorithms on actual networks and test how they perform in specific real-world instances.

9 Individual Contributions

Qianying	data generation, graph plotting, data analysis, implemented random, report writing
Chuanqi	implement CELF/++,library code,devised DEGREE, report writing,poster,running tests
Kaidi	devised and implemented TOD, testing framework, report writing, poster

References

[1] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 420429. ACM, 2007

[2] A. Goyal, F. Bonchi, and L. Lakshmanan. Learning influence probabilities in social networks. In Proceedings of the third ACM international conference on Web search and data mining, pages 241250. ACM, 2010

[3] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 137146. ACM, 2003.

[4] W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In Data Mining (ICDM), 2010 IEEE 10th International Conference on, pages 8897. IEEE, 2010.

[5] M. Granovetter. Threshold models of collective behavior. American Journal of Sociology 83(6):1420-1443, 1978.

[6] T. Schelling. Micromotives and Macrobehavior. Norton, 1978

[7] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 199208. ACM, 2009.

[8] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In SODA, pages 946957, 2014

[9] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In SIGMOD, pages 7586, 2014.