Efficient Social Influence Maximization in Parallel

Chenhao Tan
Department of Computer Science
Cornell University
Ithaca, NY
chenhao@cs.cornell.edu

Wenlei Xie
Department of Computer Science
Cornell University
Ithaca, NY
wenleix@cs.cornell.edu

ABSTRACT

In this project, we study the classical problem of social influence maximization. Though the approximation algorithm via Monte Carlo simulation is polynomial, it is too slow on large scale graphs. We propose a faster algorithm by estimating the upper bound of the improvement a user can bring. This is done through the upper bounds of the user's neighborhood and the reachable probabilities of users. We implement a parallel version within MPI and Pthread to get a even larger speedup. It is shown that we run much faster on all the datasets. It can even scale to the Flixster dataset where the original algorithm cannot finish. Additionally, we conduct a bound analysis on the number of samplings to get an accurate approximation.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Non-numerical Algorithms and Problems

General Terms

Algorithms, Experimentation, Performance

Keywords

social networks, influence maximization, parallel, bound analysis

1. INTRODUCTION

With the success of many large-scale online social networks, such as Facebook, MySpace, Ning, and Twitter, social network analysis has become a popular research topic, attracting tremendous interests from mathematics, biology, physics, computer science, and sociology. One important aspect is that social network plays a fundamental role in the spread of information and ideas. We look at the problem of *influence maximization*.

Motivated by applications to marketing, Domingos and Richardsion posed a fundamental algorithmic problem for social networks [6, 16]. Suppose that we have data on a social network, with estimates for the extent to which individuals influence one another, we would like to market a new product that we hope will be adopted by a large fraction of the network. How should we choose the key few individuals to use for seeding the process? Kempe, Kleinberg and Tardos [13] are the first to formulate the problem as the following optimization problem. A social network is modeled as a graph with vertices representing individuals and edges representing connections or relationship between two individuals. Influence are propagated in the network according to a stochastic cascade model.

Kempe, Kleinberg and Tardos prove that the optimization problem is NP-hard and present a greedy approximation algorithm which guarantees that the influence spread is within $(1 - 1/e)$ of the optimal influence spread. Leskovec et al. [14] present an optimization in selecting new seeds, which is referred to as the "Cost-Effective Lazy Forward" (CELF) scheme. CELF does not does not affect the accuracy of the original algorithm proposed by Kempe et al. The recent work done by Wei Chen et al [3, 4] propose some heuristic methods, but as we later show, in some case, heuristic methods are outperformed by the original algorithm from Kempe et al. We believe that it is still worth running the original algorithm that guarantees an error bound. This is the major motivation of our project.

We focus on the independent cascade model with $p = 0.01$ in this project. We tackle the efficiency issue of influence maximization from two directions. First, following the idea in Leskovec et al. [14], we try to provide a better estimation of the upper bound of a user, considering previous estimations and previous reachable probabilities. Thus we can be even lazier compared with CELF. Second, we find that this algorithm is naturally parallelable. We implement a distributed algorithm to make use of multiple machines and multiple cores. Also, we use Chernoff bound and Markov Inequality to derive bounds on the number of sampling that we need in Monte Carlo estimation.

The rest of the report is organized as follows. Section 2 presents the related work. Section 3 formally defines our problem, explains our approach and provides a bound analysis on the number of samplings in Monte Carlo estimation. Section 4 presents the data used in our experiments and analyzes the experiment results. Section 5 shows some further discussion on some interesting observations. We conclude the paper in Section 6.

2. RELATED WORK

In this section, we present the related work. We categorize them into two parts: *Influence Maximization* and *Distributed Graph Processing*.

2.1 Influence Maximization

As we have mentioned in Section 1, Kempe et al. [13] introduce two diffusion models for the spread of influence.

Independent Cascade Model. *Independent Cascade Model* was investigated in the context of marketing by Goldenberg, Libai and Muller [9, 8]. We start with an initial set of active nodes S_0, and the process unfolds according to the following randomized rule. When node v first becomes active in step t, it is given a single chance to activate its inactive neighbor w; it succeeds with a probability $p_{v,w}$, a parameter of the system. (If w has multiple newly activated neighbors, their attempts are sequenced in an arbitrary order.) If v succeeds, then w will become active in step $t+1$; but whether or not v succeeds, it cannot make any further attempts to activate w in subsequent rounds.

Linear Threshold Model. Granovetter and Schelling were among the first to propose models based on node-specific thresholds [11, 17]. In *Linear Threshold Model*, a node v is influenced by each neighbor w according to a weight $b_{v,w}$ such that $\sum_{w\ neighbor\ of\ v} b_{v,w} \leq 1$. The process then proceed as follows. Each node v chooses a threshold θ_v uniformly at random from the interval $[0, 1]$; this represents the weighted fraction of v's neighbors that must become active in order for v to become active. Given a random choice of thresholds, and an initial set of active nodes S_0 (with of other nodes inactive), the diffusion process unfolds deterministically. In step t, all nodes that were active in step $t-1$ remain active, and we activate any node v for which the total weight of its active neighbors is at least θ_v.

Among them, Independent Cascade Models with uniform probability is the focus of our project. Kempe et al. [13] prove that with both models, the problem to find k users to maximize the influence is NP-hard. For Independent Cascade Model, they also show that it is approximation NP-hard[1]. They show that this problem is submodular. Thus the optimal solution for influence maximization can be efficiently approximated to within a factor of $(1 - 1/e - \epsilon)$ with a natural hill-climbing strategy. In each step, the algorithm needs to find a node which improves the current seed set most. The influence of a seed set is estimated by Monte Carlo simulation. Leskovec et al. [14] propose to speed up this algorithm by lazy estimation. The idea is that the result of previous estimation is an upper bound in the following steps. Chen et al. [4] proposes a simple change from the original sampling. Instead of sampling a graph each time, we can use the same sampled graph to estimate $\sigma_X(S)$ and $\sigma_X(S \cup v)$ at the same time. But they did not show theoretical validness of this approach. They also present a heuristic for Independent Cascade Model with uniform probability called DegreeDiscount, which is shown to outperform degree heuristic. But it seems that if the graph is large, it cannot approximate the results well. Chen et al. propose MPIA and PMPIA for Independent Cascade Model with weighted probability [3, 4]. This algorithm, on the other hand, considers the maximum influence path, and the subgraph consisting of all the nodes in that subgraph. And they argue that this is a proxy of the original problem. However, it is still NP-hard and submodular. They try to propose efficient algorithms in this new problem.

[1]It is NP-hard to gain any better approximation ratio that is larger than $1 - 1/e$.

2.2 Distributed Graph Processing

Pregel [15] is a framework for distributed processing on graphs. Programs are expressed as a sequence of iterations, in each of which a vertex can receive messages sent in the previous iteration, send messages to other vertices, and modify its own state and that of its outgoing edges or mutate graph topology. Goyal et al. [10] propose another data-driven approximation problem to the original problem. They introduce a new model called credit distribution, that directly leverages available propagation traces to learn how influence flows in the network and uses this to estimate expected influence spread. Chierchetti et al. [5] propose an algorithm on Map-reduce as an adaption of the original greedy algorithm. It can be easily expressed in the Map-Reduce programming paradigm, and requires only polylogarithmically many passes over the data. Zou et al. [18] propose a general parallel framework to process time-stepped applications in the distributed system environments that avoid latency spikes. Though they are quite different from our problem, we consider them as good efforts on the distributed system for graph processing.

3. APPROACH

In this section, we give the formal definition of our problem. We provide a summarization of previous approaches as a reference to our method. We show our further improvement by upper bound estimation, and describe our approach to parallel implementation. Finally, we provide a bound analysis on the number of samplings in each Monte Carlo simulation.

3.1 Problem Definition

Given a social network, modeled as a graph $G = (V, E)$, where the vertex set V represents the individual nodes and edge set E represents the potential influence among these individuals. The graph can be either directed or undirected. In the undirected version, an undirected edge (u, v) is equivalent to two directed edges (u, v) and (v, u). In the propagation process, each individual will be either *active* or *inactive*. For a given initial active vertex set S, in the independent cascade model *with uniform probability*, when node v first becomes active in step t, it can activate each inactive neighbor w, it succeeds with a probability p. If v succeeds, w will become active in the next step, but whether or not v succeeds, it cannot make any further attempts in $t+1$. This is equivalent to computing the reachable set in a sampled graph where each edge appears with probability p independently. We can compute the expected number of a set of active vertices, S.

$$\sigma(S) = \sum_{outcomes\ X} \Pr[X] \cdot \sigma_X(S) \tag{1}$$

Table 1: Important Variables used in this paper

Variables	Descriptions
n	number of vertices in G
m	number of edges in G
k	number of seeds to be selected
R	number of rounds of simulations
d_v	degree of vertex v in G
p	propagation probability
$\sigma(S)$	expected influence of S
$\Delta_S(u)$	the increase of expected influence adding u to S

Where X is a graph generated from sampling, $\sigma_X(S)$ is the reachable set of S in X.

So the formal problem is defined as:

Definition 1. *Given a graph $G = (V, E)$ and a number k, the task is to find the optimal set of k nodes S, to maximize the expected influence.*

$$argmax_{|S|=k}\sigma(S)$$

For convenience, Table 1 lists important variables used throughout the paper.

3.2 Existing Approaches

We summarize the existing approaches as follows.

- Greedy Algorithm [13].

  In each iteration, node u will be added such that $\sigma(S \cup \{u\})$ will be maximized. Let $\Delta_S(u) = \sigma(S \cup \{u\}) - \sigma(S)$, and this is equivalent to find the node u to maximize $\Delta_S(u)$. Algorithm 1 shows the Greedy Algorithm.

  However, as shown in [3], the calculation of function $\sigma(S)$ itself is #P-hard. In practice Monte Carlo method is used to estimate the function. One can simply sample each edge in the graph and then using BFS to compute the reachable set from S. Repeat this process for R times [2], and the average number of nodes activated will be used as the approximated value of the function. However, the reachable set is usually much smaller compared to the whole graph, especially when p is small. Thus there is no need to sample every edge in the graph. Instead, we sample the edge "on the fly" in the process of BFS. A random number will be generated to determine whether certain edge exists in the current sample graph only when the edge is used.

- Cost-Effective Lazy Forward [14]

  The standard greedy algorithm requires calculating $\Delta_S(u)$ for each node u in the graph in each iteration, which is quite time-consuming. Notice in the previous iterations, $\Delta_{S'}(u)$ has been calculated for each node u, where $S' \subsetneq S$. And by the submodular property we have $\Delta_{S'}(u) \geq \Delta_S(u)$. Thus if $\Delta_{S'}(u)$ is smaller than

[2]We take $R = 20000$ in most cases.

```
Input: $G$, $p$, and $k$
Output: $S$ with the maximal expected influence $M$
Initialize $S = \phi, M = 0$
for $i = 1$ to $k$ do
    Initialize $M_i = -\infty$
    foreach $u \in V - S$ do
        if $\Delta_S(u) > M_i$ then
            $M_i = \Delta_S(u)$
            $u_{\text{best}} = u$
        end
        $S = S \cup \{u_{\text{best}}\}$
        $M = M + M_i$
    end
end
```

Algorithm 1: Greedy Algorithm

the current maximum influence gain we have encountered in this iteration, there is no need to calculate $\Delta_{S'}(u)$. In other words, the influence gain computed previously can be considered as a bound later so that we can avoid a lot of expensive estimation of influence, especially when p is small. Algorithm 2 shows the CELF algorithm.

```
Input: $G$, $p$ and $k$
Output: $S$ with the maximal expected influence $M$
Initialize $S = \phi, M = 0$
foreach $v \in V$ do
    Calculate $\delta(v) = \Delta_\phi(v)$
end
Pick up node $u$ with the maximal $\delta(u)$. $S = S \cup \{u\}$
$M = M + \delta(u)$
for $i = 2$ to $k$ do
    repeat
        Pick up the node $u \in V - S$ with the maximal $\delta(u)$
        Update $\delta(u) = \Delta_S(u)$.
        if $\delta(u)$ is still the maximal then
            $S = S \cup \{u\}$
            $M = M + \delta(u)$
            The node is found
        end
    until The node is found;
end
```

Algorithm 2: CELF Algorithm

- Degree Discount Heuristic [4]

  The general idea is as follows. Let v be a neighbor of vertex u. If u has been selected as a seed, then when considering selecting v as a new seed based on its degree, we should not count the edge (v, u). Thus we should discount v's degree by one. Furthermore, since v is a neighbor of u that has been selected into the seed set, with probability at least p, v will be influenced by u, in which case we do not need to select v into the seed set. And they show that the expected number of additional vertices in v's neighborhood is $1 + (d_v - 2t_v - (d_v - t_v)t_v p + o(t_v)) \cdot p$, where t_v is number of neighbors of vertex v already selected as seeds. Algorithm 3 shows the Degree Discount Algorithm.

```
Input: $G$, $p$, and $k$
Output: $S$ with the maximal expected influence
Initialize $S = \phi$
foreach $u$ do
    Compute its degree $d_v$
    $dd_v = d_v$
    Initialize $t_v = 0$
end
for $i = 1$ to $k$ do
    Select $u = \text{argmax}_v\{dd_v | v \in V - S\}$
    $S = S \cup \{u\}$
    foreach $(u, v) \in E$ and $v \in V - S$ do
        $t_v = t_v + 1$
        $dd_v = d_v - 2t_v - (d_v - t_v) \cdot t_v p$
    end
end
```
Algorithm 3: Degree Discount Algorithm

3.3 Upper Bound Estimation

The bottleneck in the algorithm is the sampling subroutine. And estimating the upper bound of $\Delta_S(u)$ can help reduce the number of calling this subroutine . To achieve this, CELF uses the previously calculated $\Delta'_S(u)$ as the upper bound of current $\Delta_S(u)$ where $S' \subsetneq S$. However, it still has to calculate $\Delta_\phi(u)$ for every vertex $u \in V$ in the first step, which is too slow on networks with millions of nodes. In this section we will estimate the upper bounds in other ways.

Recall that in the each step of Greedy Algorithm, given the current seed set S, we are trying to find the node u that maximizes $\Delta_S(u)$. Let $M = \max_u \Delta_S(u)$ be the maximal influence can be added by introducing a new node. According to the submodular property, we have:

$$\begin{aligned}\Delta_S(u) &\le 1 + p \sum_{(u,v)\in E} \Delta_S(v) \\ &\le 1 + p \cdot d_u \cdot M \qquad (2)\end{aligned}$$

Thus, if u is the node that maximizes $\Delta_S(u)$, which means $M = \Delta_S(u)$. We have $M \le 1 + p\sum_{(u,v)\in E} d_v \cdot M$. When $p \cdot d_v < 1$, we have

$$\Delta_S(u) = M \le 1/(1 - p \cdot d_u) \qquad (3)$$

Thus if the upper bound for $\Delta_S(u)$ is less than the maximal influence gain in the current iteration, we can skip computing $\Delta_S(u)$.

One drawback for Eq. 3 is that it cannot help to estimate the upper bound to nodes with degrees larger than $1/p$, even the node is connected to nodes with small degrees. This is because in Eq. 2, M is used to bound $\Delta_S(v)$, which might be too "loose" in some cases. To achieve a tighter upper bound estimation, we can refine the bound for the $\Delta_S(v)$ based on specific situations:

- If there is a good upper bound for $\Delta_S(v)$ (i.e. $\delta(v)$). We can use it directly. In specific, we can compare it to the current maximum influence gain in this iteration. $\delta(v)$ will be used if it is smaller than the current maximum influence gain.
- Otherwise, if $d_v < 1/p$. $\Delta_S(v)$ can be further expanded and bounded by the degree of v, i.e. $1+p{\cdot}d_v{\cdot}M$.
- Otherwise, $\Delta_S(v)$ will be bounded by M.

Algorithm 4 shows our upper bound estimation algorithm. Notice estimating all the nodes will traverse over all the nodes and edges, which result in time complexity of $O(|V| + |E|)$. Thus it adds $O(k(|V| + |E|)$, which is not dominant in the whole Greedy Algorithm.

```
Input: $G$, $p$, $u$, the current maximal influence gain
       $currentMax$
Output: Update upper bound $\delta(u)$
// $D \le A + BD$
Initialize $A = 1, B = 0$
foreach $(u, v) \in E$ do
    // Have a good estimation
    if $\delta(v) \le currentMax$ then
        $A = A + (1 - \text{visitedProb[v]}) \cdot \delta(v)$
    end
    else if $d_v < 1/p$ then
        $A = A + (1 - \text{visitedProb[v]})$
        $B = B + (1 - \text{visitedProb[v]}) \cdot p \cdot p \cdot d_v$
    end
    else
        $B = B + (1 - \text{visitedProb[v]}) \cdot p$
    end
end
```
Algorithm 4: Upper Bound Estimation Algorithm

3.4 Parallel Implementation

An independent approach to speeding up the algorithm is by parallel computation. In the experiments we found when the graph becomes large, calculating the reachable set for R time is quite time-consuming. The Monte Carlo simulation for each set of nodes can be easily distributed. Thus we can partition the sampling task into several machines, and each machine can run several threads. We implement the algorithm in the MPI framework. Each machine will run the same task with the same sampling number. In each iteration, the master node will broadcast the vertex to be estimated, and each slave machine will calculate it independently. As a global synchronization barrier, the master node will wait for the result from each individual machine, and either take the median or the mean of these results [3]. This procedure will happen many times in each iteration to choose the vertex to be added. Finally, at the end of each iteration, the master node will broadcast the decision, all the machines will then add the node into the current seed set and continue into the next iteration.

3.5 Bound Analysis

Now we provide a bound analysis on the number of samplings in Monte Carlo simulation. Though previous papers

[3]We will discuss the usage of mean and median in Section 5.

use $R = 20000$, it is an interesting question to ask how many rounds is enough. We develop the bound analysis from two angles.

Chernoff bound. According to the Chernoff bound, we can do the following analysis. Because the true solution $\sigma(S)$ is $M \in [0, |V|]$,

$$Pr[|\frac{1}{R}\sum_{i=1}^{R}\sigma_i - M| \leq M\epsilon] \geq 1 - 2e^{-\frac{\epsilon^2 RM}{3|V|}} \tag{4}$$

Thus if we want to get a (δ, ϵ) approximation,

$$R \geq \frac{3|V|\ln(\frac{2}{\delta})}{M\epsilon^2} \tag{5}$$

In this problem, suppose $\epsilon = 1$ is already enough for large graphs. Then we need $O(|V|/M)$. Usually, p is small, $M = o(V)$.

Markov Inequality + Chernoff Bound. In another bound analysis, we can use the similar analysis as Alon et al. did in [1]. We can first do several groups of random sampling and then take the median. However, it evolves analysis of $Var(\sigma_X(S))$, which we do not have a good theoretical bound. But we will provide empirical analysis in Section 5.

First, we denote the number of rounds in each group as R. According to the Markov Inequality, we have,

$$Pr(|\sum_i \sigma_i \frac{1}{R} - M| \geq \epsilon M) \leq \frac{Var(\sigma)}{R\epsilon^2 M^2} \tag{6}$$

Thus the probability that a group provides a bad estimation is $p = \frac{Var(\sigma)}{R\epsilon^2 M^2}$. Consider variable $X_i = I(|\sum_i \sigma_i \frac{1}{R} - M| \geq \epsilon M)$ and the number of groups is denoted as R_1. According to the Chernoff bound, we have

$$Pr(\sum_i X_i \geq \frac{S_2}{2}) = Pr(\sum_i X_i \geq \frac{S_2}{p}(1 + (\frac{1}{2p} - 1))) \tag{7}$$

$$\leq e^{\frac{S_2 p(\frac{1}{2p}-1)^2}{3}} = \delta \tag{8}$$

Thus,

$$R_1 \geq \frac{3ln\frac{1}{\delta}}{p(\frac{1}{2p} - 1)^2} \tag{9}$$

Setting $p = \frac{1}{8}$, then we need in total $R \cdot R_1 = \frac{8Var(\sigma)}{E(\sigma)^2\epsilon^2}\frac{8ln\frac{1}{\delta}}{3}$ iterations.

Possibility of Using Monte Carlo Markov Chain

In the proposal, we mentioned using Monte Carlo Markov Chain. For example, by using Metropolis, we can design a Markov Chain, so that in each iteration we can save some computation on BFS and dynamically update the reachable set. On the other hand, this will cause more iterations to take. And Gillman [7] showed that the converging speed of the Markov Chain is proportional to $\frac{1}{\epsilon}$, where ϵ is the eigenvalue gap of the transition matrix. More analysis can also be found in [2].

We can view the Monte Carlo simulation as a Markov Chain with transition matrix I, thus the inverse of eigenvalue gap is 1. While in Metropolis, it is $O(|V|)$. It also takes additional cost to update the reachable set each time.

Therefore, Monte Carlo Markov Chain cannot bring benefits. It is actually natural, because MCMC is usually employed when the distribution cannot be directly sampled, while in this problem, we can sample easily according to the targeting distribution.

4. EXPERIMENT

In this section, we first introduce some basic statistics on the datasets we use in our experiment. We then list the baselines we compare with, as well as the measures we employ. Finally, we present the results of our experiments and validate the effectiveness of our approach.

4.1 Data

Now we show some basic statistics of the datasets we use in this project. In the experiment we use five real-world networks. Both of the first two datasets are from the e-print arXiv [4]. The first one denoted *NetHEPT*, is from the "High Energy Physics - Theory" sections with papers from 1991 to 2003. The second network is from the full paper list of the "Physics" section, denoted as *NetPHY*. The third network is published by Jure Lekovec. It is a Who-trust-whom network of Epinions.com, denoted as *Epinions*. The fourth network is a much larger collboration network, the DBLP database maintained by Michael Ley, denoted as *DBLP*. The fifth network is from flixster.com, provided by Jamali and Ester [12], denoted as *Flixster*. All the graphs are undirected, except for Epinions. Table 2 shows basic statistics on different datasets. Figure 1 shows the degree distributions in different networks. As expected, we can see that it follows power-law in all the networks.

4.2 Baselines and Measures

In the experiment, we did not compare our algorithm with the original greedy algorithm, because it is too slow. The following four algorithms are compared.

- Cost Effective Lazy Forward (CELF). As described in Section 3, this algorithm use previous estimations as upper bounds.
- Degree Discount Heuristic (DegreeDiscount). This heuristic considers the influence in one step as an approximation.
- Parallel Cost Effective Lazy Forward (PCELF). This algorithm implements CELF in parallel on 20 machines with 5 threads each.

[4]http://www.arXiv.org

Table 2: Statistics of five tested real-world networks (CC means Connected Component)

Dataset	#Node	#Edge	Avg Degree	Max Degree	#CC	Largest CC Size	Avg CC Size
NetHEPT	15,223	32,235	4.23	64	1,781	6,794	8.55
NetPHY	37,154	180,826	9.73	204	3,883	19,873	9.57
Epinions	75,879	508,837	6.71	1,801	2	75,877	37939.50
DBLP	967,534	3,524,868	7.29	1,060	43,132	837,134	22.43
Flixster	786,936	7,058,819	17.94	1,389	71	786762	11083.61

Figure 1: Degree Distribution

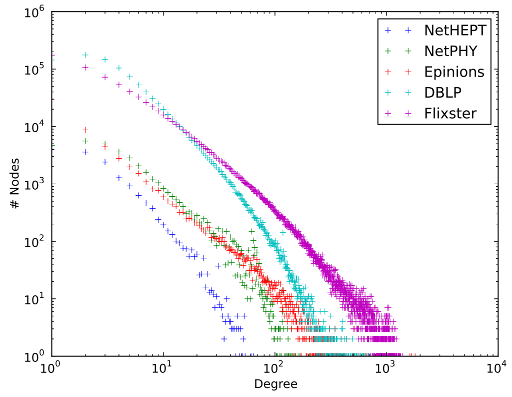
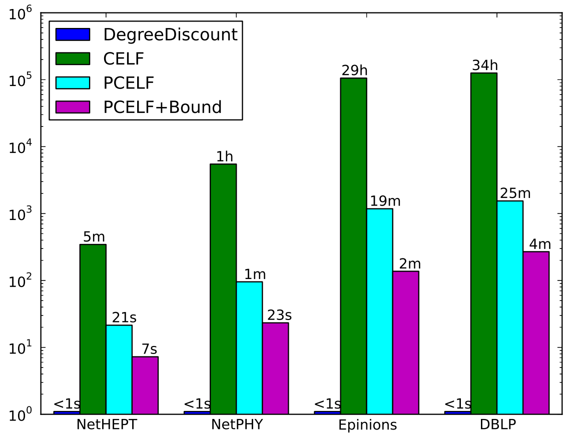


Figure 3: Running Time



- Parallel Cost Effective Lazy Forward with Upper Bound Estimation (PCELF+Bound). This algorithm also employs the upper bound estimation as mentioned in Section 3. It avoids even more estimation compared with CELF. [5]

We are interested in two measures, namely influence and running time.

- Influence. It is the expected influence after picking k nodes, we are also interested in the growing pattern of different networks.
- Running Time. It is the total running time used to pick k nodes.

Throughout this experiment, we use $p = 0.01$ and $k = 50$.

4.3 Results

Figure 2 and Figure 3 show the expected influence and running time on the five datasets. PCELF and PCELF+Bound is more than two orders of magnitude faster than CELF on the large datasets, while they still have almost the same influence as CELF. This demonstrates that our parallel implementation and upper bound estimation could scale up to much larger graphs without sacrificing the accuracy. On the other hand, DegreeDiscount, although fast enough (less than one second on all datasets), has sigfnificantly smaller influence on all dataset except the NetHEPT dataset. And there is no theoretic bound for DegreeDiscount.

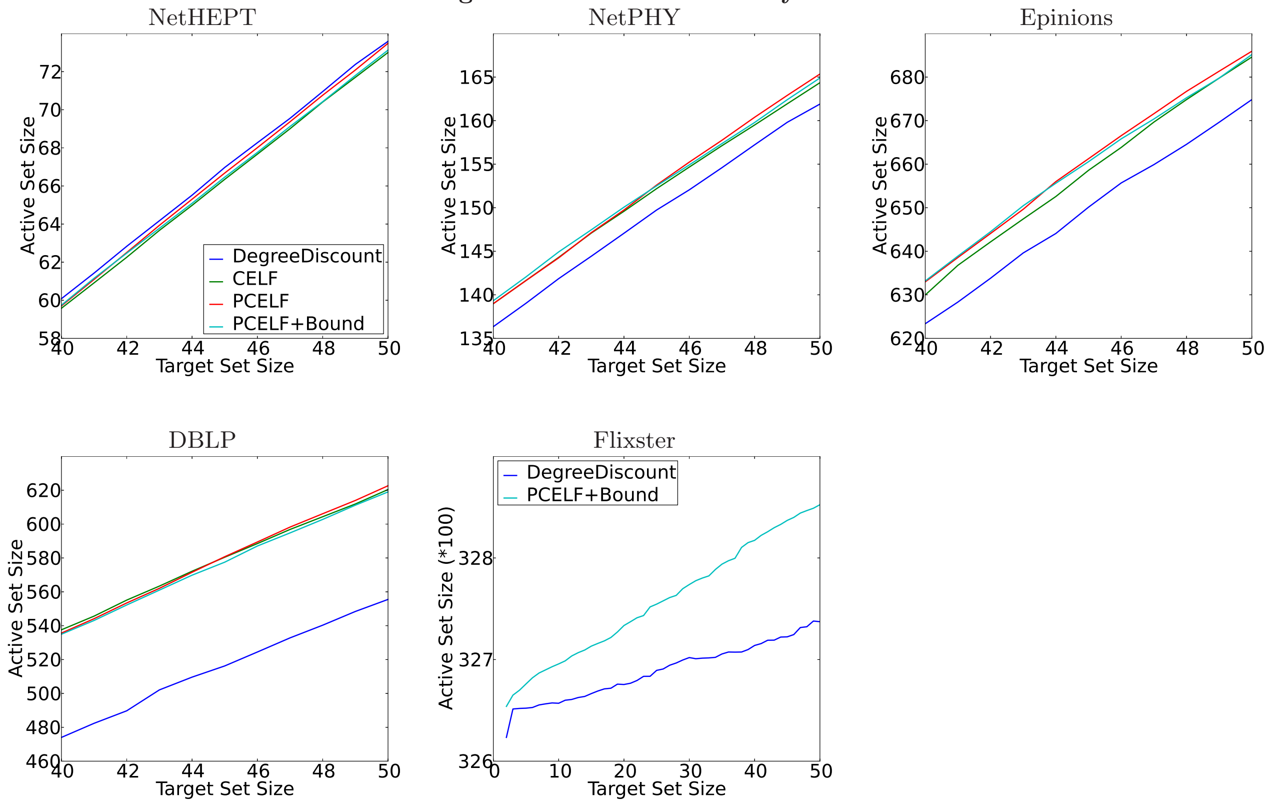
[5]It is a little different on flixster. Because the graph is too large, CELF and PCELF cannot finish. Even with PCELF+Bound, it is pretty slow. The version shown here is PCELF+Bound that only considers 10,000 nodes with the largest degree. It ran on 33 machines, 2 processes and 6 threads each. We have more discussion on the performance of Flixster in Section 5.

Performance. Figure 2 shows the expected influence of these four algorithms on the five datasets. One interesting thing is that on NetHEPT, DegreeDiscount even has slightly larger influence than the CELF-family algorithm (less than 1). This is because the Greedy Algorithm itself is an approximate algorithm (with the $1 - 1/e$ guarantee), and it is possible on some certain dataset it is not good as heuristics. However, as we show in our experiments, this only happens on the NetHEPT dataset where the influence is quite small. On all the other datasets, DegreeDiscount has the smaller influence. Thus the greedy algorithm is still the best approximation algorithm to the original problem.

We notice that on the different datasets, the difference of CELF and DegreeDiscount are quite various. For example, on the DBLP dataset, the influence of DegreeDiscount algorithm is about 55 less than the Greedy Algorithm. However on the NetPHY and Epinions, the difference is about 3 and 10, respectively. One interesting question would be what kind of properties in the graph guarantee good performance of DegreeDiscount Algorithm, and it may shed light on how to further optimize the Greedy Algorithm as well.

Scalability Our parallel implementation has very high speedup on large datasets. On the DBLP dataset, the PCELF is

Figure 2: Performance Analysis



81 times faster than the original algorithm, while on the Epinions dataset the speedup ratio is 89. Notice the upper bound for speedup ratio is 100. And they are quite close to the theoretic speedup ratio. On the smaller graph, the speedup ratio is lower but still good: 57 on NetPHY and 16 on NetHEPT. This is because on a large graph, the estimation over a single node will take considerable time and the synchronization and other overheads can be neglected. While on a small graph, the estimation over single node is so fast that other overheads cannot be overlooked. Also, notice PCELF+Bound is much faster than PCELF without sacrificing the accuracy. On Epinsions dataset it is 8.6 times faster than the PCELF, while on DBLP it is 5.7 times faster. On Flixster, it makes the impossible possible.

5. DISCUSSION

In this section, we discuss some interesting observations in the experiment.

5.1 Reachable Probability

As we can observe in the experiment, in Flixster, after selecting the first two nodes, adding a node cannot gain a large improvement any more. We had the hypothesis that it was because after picking a node, most of nodes are influenced with a large probability, and adding more nodes does not help very much. Table 3 shows the contributions from nodes with different influence probabilities.

Now we think that the real reason is that nodes with large degree not only are more probable to influence others, but also are more probable to be influenced. As we can see that Figure 4. Thus, after picking up a few nodes, these nodes with large degrees already have high probability to be influenced, while the influence probabilities for nodes with small degrees are low and will remain low in the future. It would be an interesting question to discover in what kind of graphs there is such phenomenon.

Table 3: Influence Contribution from Nodes with Different Probabilities

Probability	# of Nodes	Influence Contribution
$(0.0, 0.1]$	701302	12917.31
$(0.1, 0.2]$	43200	6033.35
$(0.2, 0.3]$	15691	3816.88
$(0.3, 0.4]$	7582	2617.67
$(0.4, 0.5]$	4265	1898.57
$(0.5, 0.6]$	2682	1466.20
$(0.6, 0.7]$	1765	1142.19
$(0.7, 0.8]$	1319	987.06
$(0.8, 0.9]$	1063	902.22
$(0.9, 1.0]$	610	571.16

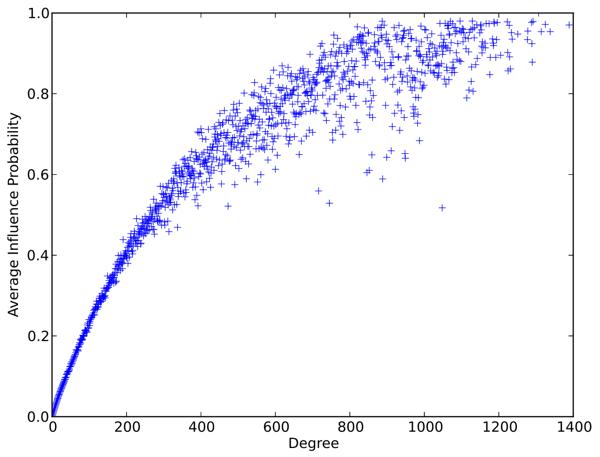
5.2 Number of Samples and Median vs. Mean

Another interesting problem is that how many times we need in each graph according to our empirical result. Table 4 shows that the results of two methods of bound analysis

Table 4: Bounds of the number of sampling rounds ($\delta = 0.01$)

Dataset	$M\epsilon$	Variance	Expected Influence (M)	Chernoff Bound	Markov Inequality + Chernoff Bound
NetHEPT	0.1	72.96	5.507282	1,765,404,773	54,034
NetPHY	1	164.28	19.557152	97,017,369	1,922
Epinions	1	684.200	50.599	825, 208, 878	4,972
DBLP	1	619.607	57.550	9,528,878,172	5,653
Flixster	1	32849.087	474.411455	410,886,564,250	46,607

Figure 4: Average Influence Probability for Nodes with Different Degree



in Section 3, with reasonable ϵ and $\delta = 0.01$[6]. Though the bound from Chernoff bound alone tells us that we only need linear time on the number of nodes ($|V|$), the estimation is not practical at all. While with variance, the bound from *Markov Inequality + Chernoff Bound* makes much more sense. Actually, we ran another version of setting on Flixster with $50 \cdot 50 = 2,500$. The performance is very bad. This shows that $2,500$ is not enough.

We also tried to use 27 groups of 500 iterations and took the median, which was shown to perform worse in NetHEP. Because the program ran the fastest, we tested a lot on it. And 27×500 is not enough for this problem. When using 27 groups of 500 iterations and taking the median, it was shown to perform worse in NetHEP. However, taking the mean of $20,000$ iterations works well and stably. Though Chernoff bound gives theoretical bound on the median method, when computation is costly, we think it is still more robust to take the mean.

6. CONCLUSION AND FUTURE WORK

In this project, we propose a further optimization on CELF by estimating the upper bounds of nodes. We also implement the algorithm parallelly in MPI. We show that this method can significantly make the algorithm faster, more than two orders of magnitude speedup on Epinions and DBLP, and even larger on Flixster. Experiments have shown that greedy algorithm with Monte Carlo simulation does significantly outperform the DegreeDiscount heuristic in most cases. It means that it is worth exploring faster computation of the greedy algorithm. Additionally, we provide a bound analysis on the number of samplings in Monte Carlo simulation.

As for future work, there are many interesting directions to work on. It is challenging but useful if we can provide an even better estimation of the upper bound. Theoretically, it is intriguing to find out whether there are some properties of graphs that determine the variance of Monte Carlo estimation, which can enable the estimation of sampling rounds before empirical analysis. It is also interesting to study in what kind of graphs that DegreeDiscount can provide a good error bound. Exploring the possibility of better polynomial algorithm in Linear Threshold Model is another promising direction.

[6] In practice, normally we do not need to be so accurate.

7. ACKNOWLEDGMENTS

Robert Kleinberg and David Kempe provide very helpful discussion with us and lead us to some literatures we did not find by ourselves. David Kempe also provides the original code, which we appreciate very much.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 20–29, New York, NY, USA, 1996. ACM.

[2] Stephen Boyd, Persi Diaconis, and Lin Xiao. Fastest mixing markov chain on a graph. *SIAM Rev.*, 46:667–689, April 2004.

[3] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, 2010.

[4] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *KDD*, 2009.

[5] Flavio Chierichetti, Ravi Kumar, and Andrew Tomkins. Max-cover in map-reduce. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 231–240, New York, NY, USA, 2010. ACM.

[6] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 57–66, New York, NY, USA, 2001. ACM.

[7] David Gillman. A chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 27(4):1203–1220, 1998.

[8] J. Goldenberg, B. Libai, and E. Muller. Using Complex Systems Analysis to Advance Marketing Theory Development: Modeling Heterogeneity Effects on New Product Growth through Stochastic Cellular Automata. *Academy of Marketing Science Review*, 1, 2001.

[9] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 2001.

[10] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. A data-based approach to social influence maximization. *Proc. VLDB Endow.*, 5:73–84, September 2011.

[11] M. Granovetter. Threshold models of collective behavior. *The American Journal of Sociology*, 83(6):1420–1443, 1978.

[12] Mohsen Jamali and Martin Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 135–142, New York, NY, USA, 2010. ACM.

[13] David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.

[14] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 420–429, New York, NY, USA, 2007. ACM.

[15] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD Conference*, pages 135–146, 2010.

[16] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 61–70, New York, NY, USA, 2002. ACM.

[17] Thomas C. Schelling. *Micromotives and macrobehavior*. Fels lectures on public policy analysis. Norton, New York [u.a.], 1. ed edition, 1978.

[18] Tao Zou, Guozhang Wang, Marcos Vaz Salles, David Bindel, Alan Demers, Johannes Gehrke, and Walker White. Making time-stepped applications tick in the cloud. In *SOCC*, 2011.