SOCIAL DATA ANALYTICS USING TENSORS AND SPARSE TECHNIQUES

by

MIAO ZHANG

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2014

To my parents who support me as always.

Acknowledgements

I would like to thank my supervising professor Dr. Chris Ding for constantly teaching, motivating and encouraging me, and also for his invaluable advice during the process of my doctoral studies. I wish to thank my academic advisors Dr. Heng Huang, Dr. Chengkai Li, Dr. Jeff Lei for their interest in my research and for taking time to serve in my dissertation committee.

I would also like to extend my appreciation to my labmates in Computational Science Lab. Their constant thirst for knowledge and hard working spirit inspired me a lot. I am so honored to have this opportunity to work with them. I am grateful to all the teachers who taught me during the years I spent in school, first in China, then in the Unites States.

Finally, I would like to express my deep gratitude to my parents who have encouraged and inspired me and sponsored my undergraduate studies. I am extremely fortunate to be so blessed. I am also extremely grateful for their sacrifice, encouragement and patience. I also thank several of my friends who have helped me throughout my career.

Apr 4th, 2014

Abstract

SOCIAL DATA ANALYTICS USING TENSORS AND SPARSE TECHNIQUES

Miao Zhang, Ph.D.

The University of Texas at Arlington, 2014

Supervising Professor: Chris Ding

The development of internet and mobile technologies is driving an earthshaking social media revolution. They bring the internet world a huge amount of social media content, such as images, videos, comments, etc. Those massive media content and complicate social structures require the analytic expertise to transform those flood of information into actionable strategies, because mining those data can help organizations take control of those data, therefore organizations can improve customer satisfaction, identify patterns and trends, and make smarter marketing strategies. Mining those data can also help the consumers to grasp the most important and convenient information from the overwhelming data sea. By and large, there are three big constituents in social media content - users, resources/events and user's tags on those resources. In this thesis, we study three key technology areas to explore the social media data. The first is viral marketing (word of mouth) technology: we try to identify the most influential individuals on the social networks. We propose highly efficient and scalable methods to calculate the influence spread and then different greedy strategies will be applied to find the most influential users. Second, we live in a rich materialistic society: too main choices on everything. Recommender systems are the up-and-coming new information technology. Traditional recommender systems deal with users and items (books, movie, etc). New web 2.0 technology enables and encourages users to comment items (images) by assigning tags (key words). This social tagging recommendation helps new users (and existing users) to comment on more items with more tags — assist the users to communicate with each other — inciting more activities in the social network — thus attracting more users! The tagging information also helps web sites to organize their resources. We propose to use lower-order tensor decomposition techniques to tackle the extremely sparse social network data. Last but not least, in the social tagging area, there are many types of social media objects, data and resources; and image is the most overwhelming part. Fast automatic analysis of vast number of images is mostly based on image annotation and segmentation. We propose an efficient and robust image reconstruction model by applying L1 norm sparse coding techniques in the collection of images (a tenor); this help significantly the annotation and segmentation analysis. We did extensive experiments on several real world data sets to evaluate our proposed models to the above three social network tasks, and experimental results demonstrate that our methods outperform state-of-the-art approaches consistently.

Table of Contents

Acknowledgements iv

Abstract v

List of Illustrations xi

List of Tables xiv

Chapter Page

1. Introduction 1

    1.1 Introduction 1

2. Approximate and Exact Evaluation of Influence Propagation on Networks 4

    2.1 Introduction 4

        2.1.1 Related Work 6

    2.2 Independent Cascade Model 7

        2.2.1 Exact Influence Spread for Small Networks 8

            2.2.1.1 Solution for 3-node Network 9

            2.2.1.2 Solution for 4-node Network 11

            2.2.1.3 Solution for 5-node Network 14

    2.3 Inclusion-Exclusion Theorem 16

        2.3.1 Computing Activation Probability on a Single Node 18

        2.3.2 Computing Activation Probability on Entire Network 19

    2.4 Computing Exact Activations for DAG Networks 20

    2.5 Injection Point Algorithm for Non-DAG Networks: Decomposition 22

    2.6 Injection Point Algorithm for Non-DAG Networks: Exact Algorithm 26

        2.6.1 Illustration of Recursive Reduction 27

2.6.2 The Global Recursive Structure of Injection Point Strategy 28

2.7 Injection Point Algorithm for Non-DAG Networks: Approximate Algorithm 29

2.8 Selecting Seed Set for Viral Marketing 31

2.8.1 Greedy Method to Solve Social Influence Maximization 31

2.8.1.1 Greedy Method 1 31

2.8.1.2 Greedy Method 2 33

2.8.1.3 Probabilistic Additive 33

2.8.1.4 Incremental Search Strategy 34

2.9 Experiments 34

2.9.1 Data Sets 36

2.9.2 Inclusion-Exclusion Theorem V.S. Monte Carlo Simulation 37

2.9.3 Comparison of Injection Point Approximate Algorithm 40

2.9.3.1 Time Comparison 42

2.9.4 Comparison of Injection Point Exact Algorithm 43

2.9.4.1 Wiki-Vote data set 44

2.9.4.2 P2p data set 46

2.9.5 Seeds Selected by Different Methods 46

2.10 Conclusion 47

3. Social Tagging Recommendation 49

3.1 Introduction 49

3.2 Problem Definition 50

3.3 Low Order Tensor Decomposition 52

3.3.1 Motivation 52

3.3.2 Baseline Low Order Tensor Decomposition 55

3.4 Missing Value Problem 56

3.5 Tensor Fold-in Algorithm for New Users 57

3.5.1 Overview of Tensor Decomposition Models 57
3.5.2 Fold-in Algorithms 59
3.5.2.1 Tucker Decomposition Fold-in Algorithm 59
3.5.2.2 ParaFac Fold-in Algorithm 60
3.5.2.3 LOTD Fold-in Algorithm 61
3.6 Experiment Results 61
3.6.1 Evaluation Strategy and Metrics 63
3.6.2 Performance of the LOTD Model 64
3.6.3 Performance of the Fold-in Models 64
3.6.3.1 Efficiency of the Proposed Fold-in Algorithms 68
3.7 Conclusion 68
4. Robust Tucker Tensor Decomposition For Effective Image Representation 70
4.1 Introduction 70
4.2 Robust Tucker Tensor Decomposition (RTD) 72
4.3 Efficient Algorithm for Robust Tucker Tensor Decomposition 73
4.3.1 Solving the Sub-optimization Problems 74
4.3.2 Updating Parameters 76
4.4 Efficient Algorithm for L_1-PCA 77
4.5 Experiments 79
4.5.1 Data Description 79
4.5.2 Corrupted Images 80
4.5.3 Experiment Results 83
4.5.4 Reconstruction Images and Discussion 85
4.6 Conclusion 85
5. Conclusion and Future Work 88
References 90

Biographical Statement 95

List of Illustrations

Figure Page

2.1 Small networks: (a) 3-node network; (b) 4-node network; (c) 5-node network 9

2.2 Different stages of influence propagation for a 3-node network in (a). Graphs (b),(c),(d),(e) are first stage results of seed node 1 attempting to influence nodes $\{2, 3\}$ with corresponding probabilities given. Thick red edges indicate the influence action. Thick circle means the node is successfully influenced, also indicated by a number 1 or 0 underneath. From graph (c), node 2 tries to influence node 3; results are given in (f),(g). 10

2.3 Different stages of influence propagation from a 4-node network of (a). Node 1 is the seed. Symbols are same as Figure 2.2. From graph (c), node 2 attempts to influence node 4. Only successful result graph (g) is shown. Results of all unsuccessful attempts are skipped. 12

2.4 First stages of influence propagation of the 5-node network. Node 1 is the seed. Symbols are same as in Figure 2.2. 14

2.5 Illustration of the propagation process of IC model: v is the target node, Figure (a) is the first step; (b) is the second step; (c) is the third step. 18

2.6 In-links are deleted for an activated node v. 21

2.7 (a) A DAG network. Probabilities on the edges are shown. (b) Topological order of the seeded network. (c) Activations computed using DAG-IET algorithm. 22

2.8 Illustration of the injection point algorithm. Red-dashed circle indicates a strongly connected component (SCC). (a) is input network with influence weight on each edge. (b,c,d,f) are 4 case of influence spread from injection nodes v_{11}, v_{12}, with branching probabilities given in Eq.(2.19). In (b) in-links to activated v_1, v_4 are deleted. In (c) inlinks to v_4 are retained because v_4 could be activated by the influence coming from v_1. In (f), no influence passed on from v_{11}, v_{12} and thus all nodes have no possibility to be activated. (e) and (j) give the exact and approximate solutions of activation respectively. 23

2.9 A non-DAG seeded network decomposed into a collection of strongly connected components (SCC): the small circles are single nodes, and green ones are seed nodes. There is no activated in-bound neighbors for SCC_1, therefore nodes in SCC_1 cannot be activated thus could be deleted from the network 24

2.10 Injection equivalence 29

2.11 Activation probability comparison between IC Monte Carlo Simulation and Inclusion-Exclusion Theorem 38

2.12 Influence spreads computed by Inclusion-Exclusion Theorem and Monte Carlo Simulation given different sizes of seed sets 39

2.13 Influence spread comparison 41

2.14 RMSE comparison between our approximate algorithm and MIA 42

2.15 Accumulated absolute differences between exact solution and Monte Carlo simulation solutions when Monte Carlo simulations are run from 2000 to 20000 times 43

2.16 Influence spreads of seed sets selected by different methods. Note, the curves corresponding to incremental search 1 and incremental search 2 almost coincide with each other on p2p-223 network 44

2.17 Difference between exact solution and Monte Carlo simulations on wiki-Vote data set (number of seed nodes=5). Shown are the absolute value of the difference at each node. $\Delta\pi_{2000}$ is the results of Monte Carlo simulations of 2000 times. Similarly, for $\Delta\pi_{4000}$, $\Delta\pi_{8000}$, $\Delta\pi_{16000}$. 45

3.1 (a) An example of tensor X with 3 users and a post been masked. (b) The predicted results. 51

3.2 Tensor decomposition relationship 52

3.3 Three tensor models: (a) Tucker Model; (b) ParaFac Model; (c) LOTD Model. 58

3.4 The Precision-Recall curve for the Last.fm dataset 64

3.5 The Precision-Recall curve for the MovieLens dataset 65

3.6 The Precision-Recall curve for the bibsonomy dataset 65

3.7 Precision-Recall curves for three tensor Fold-in methods on three datasets: from left to right: Last.fm data, MovieLens data, Bibsonomy data. The performance gap between LOTD fold-in and ParaFac/Tucker fold-in methods increases from the smaller subset B to larger subset A. 67

4.1 Samples of occluded images and reconstructed images on AT&T face data. First row is the input occluded images; Second row is from RTD; Third row is from L_1PCA; Fourth row is from Tucker decomposition; Fifth row is from PCA. 86

4.2 Samples of type 2 (mixed) occluded images and reconstructed images using different methods of AT&T data set. The first row is from input occluded images; the second row is from RTDreconstructed images; the third row is from L_1 PCA; the fourth row is from Tucker tensor; and the fifth row is from PCA. The cross corruptions can only be removed by RTD. 87

List of Tables

Table Page

2.1 Description of data sets 37

2.2 Mean squared errors (MSE) and mean absolute errors (MAE) between the two activation probability vectors achieved Inclusion-Exclusion Theorem and Monte Carlo simulation on p2p-223 network 38

2.3 Mean squared errors (MSE) and mean absolute errors (MAE) between the two activation probability vectors achieved Inclusion-Exclusion Theorem and Monte Carlo simulation on p2p network 39

2.4 Mean squared errors (MSE) and mean absolute errors (MAE) between the two activation probability vectors achieved Inclusion-Exclusion Theorem and Monte Carlo simulation on wiki-Vote network 39

2.5 Time (sec) needed to compute the influence spread given different sizes of seed sets m on p2p-223 network 40

2.6 Time (sec) needed to compute the influence spread given different sizes of seed sets m on p2p network 40

2.7 Running time (seconds) comparison on p2p data set 42

2.8 Running time (seconds) comparison on wiki-Vote data set 43

3.1 Real world tag data set statistics, with number of tags, number of items, number of users, number of nonzero tensor elements (NNZ) and relative NNZ. 50

3.2 Symbols 52

3.3 Time for folding in one new user vs. original algorithm 68

4.1 Description of data sets 80

4.2 Performance comparison (storage, noise-free error and classification accuracy) on AT&T data with block occlusion 80
4.3 Performance comparison(storage, noise-free error and classification accuracy) on Yale data with block occlusion 81
4.4 Performance comparison(storage, noise-free error and classification accuracy) on CMU PIE data with block occlusion 81
4.5 Performance comparison(storage, noise-free error and classification accuracy) on AT&T data with mixed occlusion 82
4.6 Performance comparison(storage, noise-free error and classification accuracy) on Yale data with mixed occlusion 82
4.7 Performance comparison(storage, noise-free error and classification accuracy) on CMU PIE data with mixed occlusion 83
4.8 Parameters of different data sets 84

CHAPTER 1

Introduction

1.1 Introduction

Social network technologies have been seeing a lot of changes in this information technology era. Social media content are not static libraries for users to passively receive any more. They allow users to create their own content and communicate with each other. Every one can contribute to the web content. There are a huge number of users on the web and there are connections and communications/influences between them. The social web and mobile technologies have accelerated the speed at which information is shared and influence is propagated, and they also bring the internet world a huge amount of social media content, such as images, videos, comments, etc. Therefore, the development of internet and mobile technologies have helped to generate rich and big data to social networks. There are billions of users, billions of connections, billions of contents, which includes textual contents and multimedia contents (images, videos, audio, etc.). Those massive media content and complicate social structure can be transformed into actionable strategies by analytic expertise, Organizations can take control of those data by mining the latent information and intrigue structures, and furthermore can improve customer satisfaction, identify patterns and trends, and make smarter marketing strategies. Mining those data can also help the consumers to grasp important and convenient information to facilitate their life styles.

We will analyze the complex social media content from three different angles - users, resources/events and user's tag information on those resources, which we believe cover the most important factors of social networks. Therefore, in this thesis, we analyze the social networks from three key technology areas. First, for the user dimension, we try to identify the most influential individuals on the social networks, which can be applied to the viral marketing strategies. This problem is first defined as influence maximization problem in [1]. Kempe et al. proposed two basic stochastic models, which are extracted from previous studies on social network analysis, one is independent cascade (IC) model and the other is linear threshold (LT) model. We first concentrate on providing both exact and fast approximate solutions to IC model. And also we propose an greedy algorithm based incremental search strategies to find the most influential individuals. Second, for tag dimension, we tackle the 3D social tagging recommendation problem. Different from the traditional 2D recommender system, users are allowed to use short phrases, which refer to tags, to describe their social resources. Therefore, there are three dimensions involved in tagging recommendation - the three constituents (users, items, tags) mentioned above. Tag recommendation system helps the tagging process by advising a set of tags to the user that he may use for a specific item. The tagging information helps web sites to organize their resources, and also assist the users to communicate with each other. We propose to use lower-order tensor decomposition techniques to tackle the extremely sparse social network data. We also propose three tensor fold-in algorithms to deal with new user problems in tagging recommendation systems. Last but not least, for resources, there are many types of social media resources, and image is a big component part. We propose an efficient model to represent the gigantic amount of images in social media using tensor and L1 norm sparse techniques, which can be applied in image categorization problems. We did extensive experiments on several real world data sets to evaluate our proposed models to the above three social network tasks, and experimental results demonstrate that our methods outperform the state-of-art approaches consistently.

This thesis is organized as follows. Chapter II analyzes exact solutions of small networks; one key finding from these analysis is the inclusion-exclusion principle which we prove vigorously. We further propose exact probabilistic solutions to influence spread for both Directed Acyclic Graph (DAG) and non-DAG under IC model, and another fast and scalable linear order approximate algorithm for non-DAG graph. We also propose an incremental search strategy to continue refining the seed set, which is first obtained by greedy methods. After incremental search, the influence spread of the selected seed set is improved. Chapter III introduces the social tagging recommendation problem and our proposed low-order tensor decomposition models to deal with those sparse data in social networks specifically. Chapter IV gives an efficient and robust model using tensor and L1 norm based sparse techniques for image representation and image categorization problems. Chapter V proposes the future work and summarize the thesis.

CHAPTER 2

Approximate and Exact Evaluation of Influence Propagation on Networks

2.1 Introduction

Independent cascade (IC) model is widely used to model social influence propagation on social networks, such as opinions, information, ideas, innovations, etc. One important task is to identify the most influential nodes in these networks. This is especially useful for viral marketing (word-of-mouth marketing), which aims at a certain number of influential consumers at the beginning, and relies on communications and trust between individuals within close social networks [2] [3] to market some product. Web 2.0 enables convenient communications among people within or between different social circles through online social networks, such as Twitter, Facebook, Linkedin, and so on. Information can propagate from a small number of individuals to a huge number of users in social networks in a short time.

There are various research topics in viral marketing studies, such as, (1) how to determine the edge weights between different users; (2) how to calculate the social influence given a set of activated nodes (seed set); (3) how to select the optimal seed set, which has the maximum social influence, i.e., the number of activated nodes in the end are the largest. This problem is defined as influence maximization problem in [1]. The above three challenges rely on each other, such as, we need to know how to calculate the social influence given a seed set, if we want to find the most influential nodes. In this thesis, we concentrate on solving the second topic and third topic. To address the problem of how to calculate the social influence given each seed set, we first need to present a social influence model defining how the propagation proceeds under some circumstances. There are several influence models those have been proposed and studied, and the most popular ones are linear threshold model (LT) and independent cascade model (IC), which were presented by Kempe et al. in [1]. We study the influence propagation process under IC model in this thesis. IC Model can be described as a stochastic process based on some probabilistic settings. For details, social network can be modeled as one graph $G(V, E)$ with edge weights P. IC model starts with an initial active seed node set; in the first step, those active nodes try to influence their inactivated out-bound neighbors with probability of the corresponding edge weights; each active node only has one chance to influence its each inactivated out-bound neighbor; in next step, the newly activated nodes continue to influence their own inactivated out-bound neighbors with one single chance to each neighbor; this process proceeds until no more inactivated nodes become activated.

Kempe et al. proved the influence maximization problem under IC model is NP-hard in [1], and Wei Chen et al. proved that calculating the influence spread of a seed set under IC model is NP-hard too in [4]. Kempe et al. applied Monte Carlo simulation to approximate the influence spread, which is widely used in other papers to approximate the influence spread. which is time-consuming, because Monte Carlo simulation needs to be run at least thousands of times to reach a good approximation of the true influence spread. Therefore, as the first challenge, proposing an efficient approximation method to calculate the influence spread of a seed set is urgent. To that end, we present probabilistic solutions to calculate the influence spread under IC model both exactly and approximately. And then based on our fast and approximate solution given seed sets, incremental search strategies are proposed to continue refining the seed set, which is first obtained by two greedy methods. After incremental search, the influence spread of the selected seed set is improved comparing with the input seed set selected by greedy methods.

Our main contributions are listed as follows:

(1) First we analyze the exact solutions to small networks, and inspired by the analysis we propose to compute activation probability on each node using Inclusion-Exclusion theorem, then activation probabilities on entire network can be calculated by applying Inclusion-Exclusion theorem iteratively.
(2) Second we propose an injection point algorithm to compute the spread of the network exactly, and inspired by our exact solution, we also propose another approximate and fast algorithm to compute the influence spread given seed sets. Our approximate probabilistic solution significantly speeds up the calculation of propagation spread given seed sets.
(2) We also propose an incremental search strategy to continue refining the seed set, which is first obtained by two greedy methods. After incremental search, the influence spread of the selected seed set is improved comparing with the input seed set selected by greedy methods.

2.1.1 Related Work

There have been a lot of research work studying and analyzing different aspects of social influence, we group these related work into three categories. The first category includes research work on influence models. The second category includes the related work on how to compute social influence spread. The third category focuses on solving the ultimate viral marketing goal - find a set of seed nodes those have the maximal social influence.

For the first category, Domingos et al. [5] [6] first proposed to mine the customers' network value, and then based on customers' network value to solve the social influence maximization problem. Kempe et al. [1] first presented the two basic influence models - LT and IC model, extracted from previous work [7] [8]. Aggarwal et al. proposed a stochastic information flow model to determine the authoritative individuals in [9], which is closely related to IC model. Other aspects of influence models, such as the edge weights between individuals were also studied in [10]. Tang et al. [11] [12] proposed a Topical Factor Graph (TFG) model to analyze social influence on a specific topic.

For the second and third categories, Kempe et al. [1] presented to use Monte Carlo Simulation to estimate the influence spread for given seed set, and proposed a greedy method to find a good seed set, which is not scalable to large scale networks, because Monte Carlo Simulation needs to be run at least tens of thousand times to get a good estimation. Then many heuristic algorithms were introduced for the IC model. Kimura et al. proposed two influence cascade models based on shortest-path to approximate the influence spread of a seed set, and present algorithms to give good approximations to IC model for finding good seed sets [13]. Chen et al. proposed a heuristic algorithm using degree discount for a limited version of IC model, in which the edge weights/probabilities between any two connected individuals are the same in [14]. Chen et al. also proposed a maximum influence arborescence (MIA) heuristic model for the generic IC model in [4]. In MIA model, maximum influence paths (MIP) between every pair of two nodes need to be pre-computed, and then based on these MIPs, local MIA structures can be formed. There are other research work digging into this problem [15] [16] [17] [18].

2.2 Independent Cascade Model

In this section, we give a brief introduction to IC model. A social network can be represented by a directed graph $G(V, E)$ with edge weight/probability P, i.e., $P(u, v)$ or P_{uv} in short denotes the propagation probability through edge $(u, v) \in E$ from node u to node v. The total number of nodes in G is $n = |V|$.

Given an activated seed set S, the independent cascade model works as follows. $S_0 = S$ is the activated node set at step 0, and S_t denoted the activated node set at step t. At step $t+1$, every newly activated node u in S_t, i.e., $\{u | u \in S_t \backslash S_{t-1}\}$, is trying to influence its out-bound non-activated neighbors v, which don't belong to S_t, i.e. $\{v|(u \rightsquigarrow v) \cap (v \in V \backslash S_t)\}$ with probability $P(u, v)$. The process stops when an equilibrium state is reached, i.e. there are no more nodes being activated in next propagation step. In independent cascade model, once a node is activated (influenced), it will stay activated ever after. Also, each activated node $\{u|u \in S_t(t \geq 0)\}$ can only influence its out-bound neighbor v once.

Now, we are ready to define the probabilistic solution to IC model, which is the activation probability for each node in the graph in the final step (in the stationary/equilibrium state), given a seed set S.

The solution of a IC model on a network G is a probability distribution, i.e., the activation probability for each node. At present, the widely used method of computing this probability distribution is using Monte Carlo computer simulation method. Since the exact solution is NP-hard, efficient algorithms are the focus of current research.

2.2.1 Exact Influence Spread for Small Networks

In this section, we give three small network examples to illustrate the exact influence propagation process. The three small networks are shown in Figure 2.1, where node 1 is the seed (shaded in green) in each case. We present the exact propagation solution for each network. These exact solutions can be extended to larger networks.

From these exact results we obtain three important benefits:

(1) We learn the rules of adding contributions from different path of influence propagation. At first glance, these contributions seems to be statistically *independent*. But the exact results show they may not be independent and why. This introduces the inclusion-exclusion principle we found useful in correctly enumerating contributions from different paths.

(2) The rules we learned in this process are helpful to formulate an exact computational algorithm.

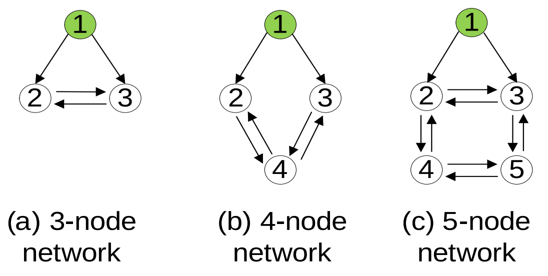


Figure 2.1: Small networks: (a) 3-node network; (b) 4-node network; (c) 5-node network

(3) Exact solutions obtained can be used to evaluate approximate algorithms in previous studies [14, 13, 9, 18]. This may lead to refined methods to further improve these existing approximate algorithms.

For those networks, we assume that transition probabilities on the edges already exist and remain fixed during the influence propagation.

2.2.1.1 Solution for 3-node Network

The IC influence propagation process for the 3-node network in Figure 2.1(a) can be illustrated in Figure 2.2.

We start with Figure 2.2(a), where node 1 is a seed node and thus always activated. The four networks of Figure 2.2(b, c, d, e) are the four possibilities of node 1 attempts to activate nodes 2 and 3. The 4 probabilities are indicated next to the arrow. For example the case where nodes 2 and 3 are both been successfully activated is shown in Figure 2.2(b), with probability $P_{12}P_{13}$.

The cases in Figure 2.2(b) and 2.2(e) are terminal, i.e., there are no further possibilities. In 2.2(c), node 2 (been successfully activated by node 1) will attempt to activate node 3. The results are shown in Figure 2.2(f) and Figure 2.2(g) with the appropriate probabilities indicated next to the arrows. Similarly, in Figure 2.2(d), node 3 (been successfully

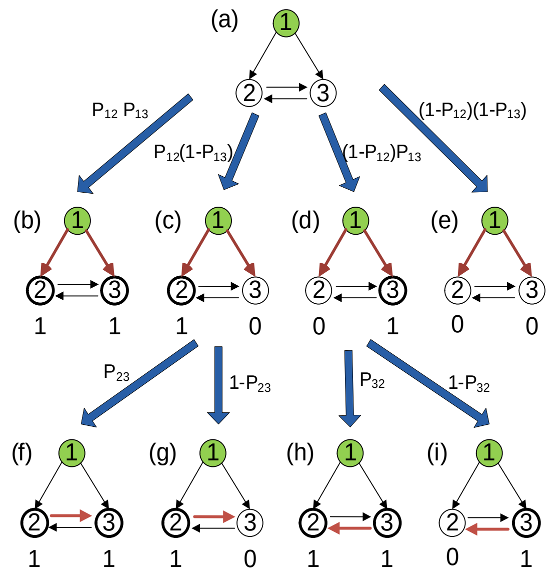


Figure 2.2: Different stages of influence propagation for a 3-node network in (a). Graphs (b),(c),(d),(e) are first stage results of seed node 1 attempting to influence nodes $\{2, 3\}$ with corresponding probabilities given. Thick red edges indicate the influence action. Thick circle means the node is successfully influenced, also indicated by a number 1 or 0 underneath. From graph (c), node 2 tries to influence node 3; results are given in (f),(g).

activated by node 1 will attempt to activate node 2. The results are shown in Figure 2.2(h) and Figure 2.2(i) with the appropriate probabilities indicated next to the arrows

Now, we can compute the activation probabilities. Let's consider node 2. There are 3 cases where node 2 becomes activated:

(i) Figure 2.2(b) with probability $P_{12}P_{13}$.

(ii) Figure 2.2(c) with probability $P_{12}(1 - P_{13})$. Note that this is equal to the sum of probabilities of Figure 2.2(f) and Figure 2.2(g).

(iii) Figure 2.2(h) with probability $(1 - P_{12})P_{13}P_{32}$. This probability for the influence flow path to Figure 2.2(h) equals to the probability to reach Figure 2.2(d) multiplied by the probability to further reach Figure 2.2(h).

Therefore, by adding these 3 probabilities, the activation probability for node 2 is,

$$\pi_2 = P_{12} + P_{13}P_{32} - P_{12}P_{13}P_{32}. \tag{2.1}$$

Another way to compute π_2 is by directly counting influence flow paths. First, node 2 can be influenced by node 1 directly, with probability P_{12} (this is the sum of probabilities of Fig.2.2(b),(c)). Second, If node 1 fails to influence node 2, there is another path node 2 can be activated, which is illustrated by Figure 2.2(a) $\rightarrow$ 2.2(d) $\rightarrow$ 2.2(h). For this influence path the probability is $(1-P_{12})P_{13}P_{32}$. Adding these two we get the same result in Eq.(2.1).

Inclusion-exclusion. The above two counting methods rely on the detailed influence propagation stages shown in Figure 2.2. The result of Eq.(2.1) can be obtained without relying on Figure 2.2. We compute probabilities of *different paths* together with an inclusion-exclusion principle. For node 2, there are two paths:

(i) $1 \rightarrow 2$, with probability $P_{1\rightarrow 2} = P_{12}$.

(ii)$1 \rightarrow 3 \rightarrow 2$, with probability $P_{1\rightarrow 3\rightarrow 2} = P_{13}P_{32}$.

These two events are not independent because in (ii) we did not include the factor $(1-P_{12})$.

We use inclusion-exclusion principle to correct for over-counting, i.e., we set

$$\pi_2 = P_{1\rightarrow 2} + P_{1\rightarrow 3\rightarrow 2} - P_{1\rightarrow 2}P_{1\rightarrow 3\rightarrow 2}. \tag{2.2}$$

This gives the same result of Eq.(2.1).

For node 3, the probability can be calculated similarly,

$$\pi_3 = P_{13} + P_{12}P_{23} - P_{12}P_{13}P_{23}. \tag{2.3}$$

2.2.1.2 Solution for 4-node Network

Let us look at a more complicated case — the 4-node network of Figure 2.1(b). The IC influence propagation process is illustrated in Figure 2.3. The settings in Figure 2.3 are the same as those of Figure 2.2.

We start with Figure 2.3(a), where node 1 is a seed node. The four networks of Figure 2.3(b, c, d, e) are the four possibilities of node 1 attempts to activate nodes 2 and 3. The 4 probabilities are indicated next to the arrow.

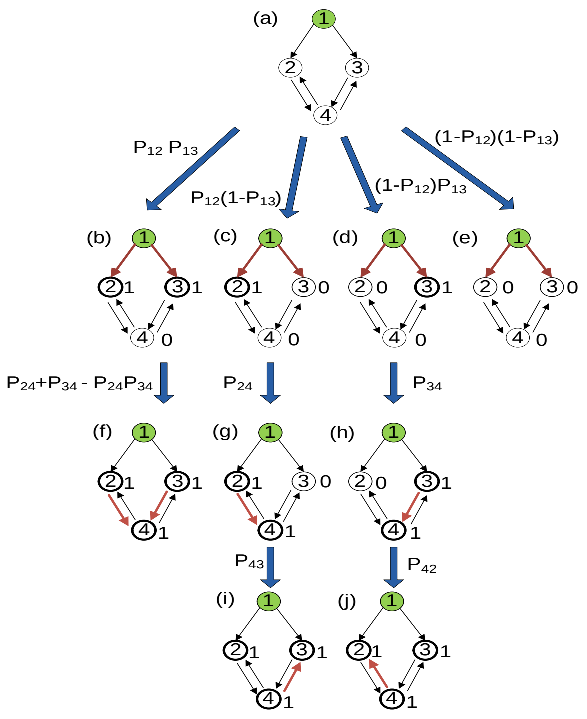


Figure 2.3: Different stages of influence propagation from a 4-node network of (a). Node 1 is the seed. Symbols are same as Figure 2.2. From graph (c), node 2 attempts to influence node 4. Only successful result graph (g) is shown. Results of all unsuccessful attempts are skipped.

The cases in Figure 2.3(e, f, i, j) are terminal, i.e., there are no further possibilities. In Figure 2.3(b), node 2 and 3 (been successfully activated by node 1) will attempt to activate node 4. The successful result is shown in Figure 2.3(f) with the appropriate probabilities indicated next to the arrows, figures for failure results are not shown here. We will compute the activation probabilities by directly counting influence flow paths, so the failure results will reach irrelevant terminal cases. In Figure 2.3(c), node 2 (been successfully activated by node 1) will attempt to activate node 4. The successful result is shown in Figure 2.3(g) with the appropriate probabilities indicated next to the arrows, and the failure result will reach an irrelevant terminal case, so we didn't show the figure here. In Figure 2.3(g), node 4 will attempt to activate node 3. The successful result is shown in Figure 2.3(i). Similarly, in Figure 2.3(d), node 3 (been successfully activated by node 1 will attempt to activate node 4. The successful result is shown in Figure 2.2(h) with the appropriate probabilities indicated next to the arrows, while the failure result is not shown here. In Figure 2.3(h), node 4 will attempt to activate node 2. The successful result is shown in Figure 2.3(j).

Now, we compute the activation probabilities. We compute π_2 by counting influence flow paths. For node 2, first, it can be influenced by node 1, with results given in Fig. 2.3(b, c). The corresponding probability is $\pi_2^{(1)} = P_{12}$.

If node 1 fails to influence node 2, there is another path node 2 can be activated, which is illustrated by Figure 2.3(a) $\rightarrow$ 2.3(d) $\rightarrow$ 2.3(h) $\rightarrow$ 2.3(j). For this influence path, the probability is $\pi_2^{(2)} = (1 - P_{12})P_{13}P_{34}P_{42}$. Adding these two we get the following result,

$$\pi_2 = P_{12} + P_{13}P_{34}P_{42} - P_{12}P_{13}P_{34}P_{42}. \tag{2.4}$$

We note again that we may directly compute the probabilities of two paths (i) $P_{1\rightarrow 2} = P_{12}$, and (ii) $P_{1\rightarrow 3\rightarrow 4\rightarrow 2} = P_{13}P_{34}P_{42}$ and use *inclusion-exclusion* to correct for the non-independence to obtain

$$\pi_2 = P_{1\rightarrow 2} + P_{1\rightarrow 3\rightarrow 4\rightarrow 2} - P_{1\rightarrow 2}P_{1\rightarrow 3\rightarrow 4\rightarrow 2}. \tag{2.5}$$

which gives the same result.

For node 3, the probability can be calculated symmetrically,

$$\pi_3 = P_{13} + P_{12}P_{24}P_{43} - P_{12}P_{13}P_{24}P_{43}.$$

For node 4, it can be activated by

(i) node 2 only, shown in Figure 2.3(g);

(ii) node 3 only, shown in Figure 2.3(h);

(iii) nodes 2 and 3 simultaneously, shown in Figure 2.3(f).

The total activation probability for node 4 is

$$\begin{aligned}\pi_4 =& (1 - P_{13})P_{12}P_{24} + (1 - P_{12})P_{13}P_{34} + P_{12}P_{13}(P_{24} + P_{34} - P_{24}P_{34}) \\ =& P_{12}P_{24} + P_{13}P_{34} - (P_{12}P_{24})(P_{13}P_{34}).\end{aligned} \tag{2.6}$$

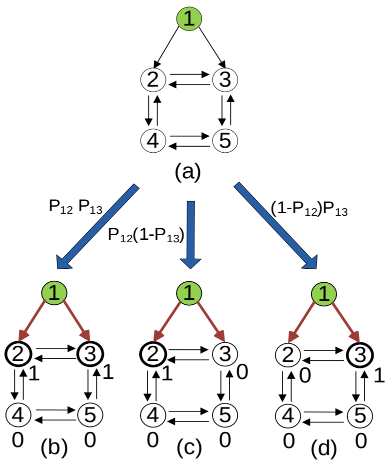


Figure 2.4: First stages of influence propagation of the 5-node network. Node 1 is the seed. Symbols are same as in Figure 2.2.

Once again, this results can be derived using the inclusion-exclusion principle mentioned above, without counting detailed influence propagation stages in Figure 2.3.

2.2.1.3 Solution for 5-node Network

As the last example, we compute activation probabilities for the 5-node network in Figure 2.1(c). The first stages of node 1 attempts to influence nodes 2,3 are shown in Figure 2.4(b,c,d).

Let us compute the activation probability for node 2. The contributions are shown in graphs Fig.2.4(b,c,d). The contributions of Fig.2.4(b,c) is P_{12}.

The contribution of Fig.2.4(d) is computed as the following. The probability to reach Fig.2.4(d) is $(1 - P_{12})P_{13}$. Starting from Fig.2.4(d), we may ignore node 1 and consider the remaining network with nodes $\{2, 3, 4, 5\}$, and node 3 is activated. This situation is identical to Figure 2.3, and we need to compute π_2. Following the results of Eq.(2.5), we obtain

$$P_{32} + P_{35}P_{54}P_{42} - P_{32}(P_{35}P_{54}P_{42}).$$

The contribution to node 2 of Fig.2.4(d) is a result of combining two paths $3 \rightarrow 2$ and $3 \rightarrow 5 \rightarrow 4 \rightarrow 2$ with the inclusion-exclusion principle. Therefore the final score for node 2 is

$$\pi_2 = P_{12} + (1 - P_{12})P_{13}\Big[P_{32} + P_{35}P_{54}P_{42} - P_{32}(P_{35}P_{54}P_{42})\Big]. \tag{2.7}$$

Now we compute activation probability for node 4. It can be activated by the following 3 paths:

(1) Starting from the situation in Figure 2.4(b) and activate node 4;

(2) Starting from the situation in Figure 2.4(c) and activate node 4; This is the same as the 4-node graph in Figure 2.3 and the node of interest is node 2.

(3) Starting from the situation in Figure 2.4(d) and activate node 4; This is the same as the 4-node graph in Figure 2.3 and the node of interest is node 4.

The total probability for node 4 being activated is

$$\begin{aligned}\pi_4 =& P_{12}P_{13}(P_{24} + P_{35}P_{54} - P_{24}P_{35}P_{54}) \\ &+ P_{12}(1 - P_{13})(P_{24} + P_{23}P_{35}P_{54} - P_{24}P_{23}P_{35}P_{54}) \\ &+ (1 - P_{12})P_{13}(P_{32}P_{24} + P_{35}P_{54} - P_{32}P_{24}P_{35}P_{54})\end{aligned} \tag{2.8}$$

We note that the 5-node network in Figure 2.1(c) includes both the 3-node network and the 4-node network cases, for example, 5-node network results become the results of the 3-node network, when $P_{24} = P_{35} = P_{54} = P_{42} = P_{53} = P_{45} = 0$; when $P_{54} = P_{45} = 1$, the results become those of the 4-node network.

When it comes to much more complicated large-scale network, the exact propagation solution is hard to derive. It's an exponential growth case along the number of nodes in V. Therefore, in this paper, we propose an approximation solution to IC model, which is exact solution for directed acyclic graph (DAG) and approximation solution for generic graphs. The exact solution for generic case is introduced later.

2.3 Inclusion-Exclusion Theorem

The lessons we learned in previous section on exact solution are useful. One of the most important lessons is the inclusion-exclusion principle that we briefly mentioned in previous section. Here we formalize the concept and prove it vigorously.

Let $\pi_v (0 \leq \pi_v \leq 1)$ denote the activation probability for each node $\{v | v \in V \backslash S_0\}$, which means to what extent v is activated.

We have the following Theorem, which is called inclusion-exclusion theorem,

Theorem 1 *Given fixed seed set S and edge weights P, for every non-seed node v and its in-bound neighbors $\mathcal{N}_v = (u_1, \cdots, u_k)$, i.e., $\mathcal{N}_v = \{u_i | u_i \rightsquigarrow v\}$, π_v, the stationary probability of v being activated, is related to $\{\pi_{u_i}\}$, stationary probabilities of its in-bound neighbors, with the following relationship,*

$$\begin{aligned}\pi_v = &\sum_{u_i \rightsquigarrow v} \pi_{u_i} P_{u_i v} - \sum_{\substack{u_i, u_j \rightsquigarrow v, \\ i<j}} (\pi_{u_i} P_{u_i v})(\pi_{u_j} P_{u_j v}) \\ &+ \sum_{\substack{u_i, u_j, u_l \rightsquigarrow v, \\ i<j<l}} (\pi_{u_i} P_{u_i v})(\pi_{u_j} P_{u_j v})(\pi_{u_l} P_{u_l v}) \\ &+ \cdots + (-1)^k (\pi_{u_1} P_{u_1 v})(\pi_{u_2} P_{u_2 v}) \cdots (\pi_{u_k} P_{u_k v}). \qquad (2.9)\end{aligned}$$

To better understand this result, we compare it to a simpler model of random walk. In this random-walk model, all neighbors of v can activate v any time it walks towards v. In this model, the activated probability would be

$$\pi_v^{(k)} = \sum_{u_i \rightsquigarrow v} \pi_{u_i} P_{u_i v}. \qquad (2.10)$$

In contrast to the random walk model, in the IC model, any actor can only attempt to affect v *once*. Thus the activation probability in IC model is lower than that in random walk model. Comparing Eq.(2.9) and Eq.(2.10), we see that the reduction from random walk model to IC model are the second term and later terms in Eq.(2.9). They are exactly the inclusion-exclusion principle.

Figure 2.5 illustrates the propagation process of IC model. In this figure, node v is the target node. Its in-bound neighbors $\mathcal{N}_v = \{u_1, \cdots, u_k\}$ attempts to influence it. Suppose there is only one activated neighbor u_1 in step 1, then u_1 tries to influence v. If u_1 fails to influence v in step 1, then newly activated u_2 tries to activate v in step 2. If u_2 fails to influence v in step 2, then newly activated u_3 tries to activate v in step 3, and so on so forth.

Proof of Theorem 1

To simplify the notations, we define $\sigma_{u_i} = \pi_{u_i} P_{u_i v}$.

Step 1 : The probability that v is activated by u_1 is

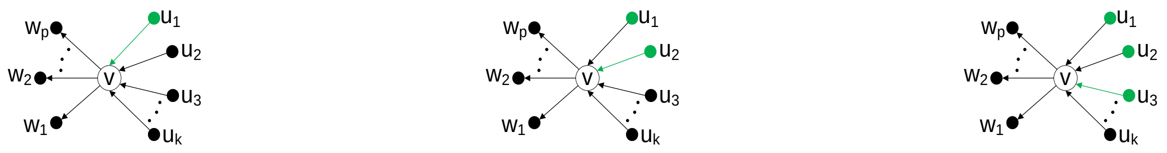
$$\pi_v^{(1)} = \pi_{u_1} P_{u_1 v} = \sigma_{u_1}. \tag{2.11}$$

Step 2 : If u_1 failed to activate v in step 1, the failure probability is $1 - \pi_v^{(1)}$. Now u_2 attempts to influence v; the probability that u_2 succeed in this is $\pi_{u_2} P_{u_2 v} = \sigma_{u_2}$; Therefore, the conditional probability that u_1 failed but u_2 succeed in activating v is $(1 - \pi_v^{(1)})\sigma_{u_2}$. This should be added to the total probability that v becomes activated. Thus

$$\pi_v^{(2)} = \pi_v^{(1)} + (1 - \pi_v^{(1)})\sigma_{u_2} = \sigma_{u_1} + \sigma_{u_2} - \sigma_{u_1}\sigma_{u_2}. \tag{2.12}$$

Step 3 : Now u_3 attempts to activate v under the condition that neither u_1 nor u_2 activated v. The probability that u_3 succeed in this is $\pi_{u_3} P_{u_3 v} = \sigma_{u_3}$; The probability that neither u_1 nor u_2 activated v is $1 - \pi_v^{(2)}$. Therefore, the conditional probability that u_1, u_2 failed but u_3 succeed in activating v is $(1 - \pi_v^{(2)})\sigma_{u_3}$. This should be added to the total probability that v becomes activated. Thus

$$\begin{aligned}\pi_v^{(3)} &= \pi_v^{(2)} + (1 - \pi_v^{(2)})\sigma_{u_3} \\ &= \sigma_{u_1} + \sigma_{u_2} + \sigma_{u_3} - \sigma_{u_1}\sigma_{u_2} - \sigma_{u_1}\sigma_{u_3} - \sigma_{u_2}\sigma_{u_3} \\ &+ \sigma_{u_1}\sigma_{u_2}\sigma_{u_3}.\end{aligned} \tag{2.13}$$



(a) step 1 (b) step 2 (c) step 3

Figure 2.5: Illustration of the propagation process of IC model: v is the target node, Figure (a) is the first step; (b) is the second step; (c) is the third step.

Using induction, we can include the value of π_v in step k.

Step k : Now u_k attempts to activate v under the condition that none of v's previous in-neighbors activated v. The total probability that v becomes activated is,

$$\begin{aligned}\pi_v^{(k)} &= \pi_v^{(k-1)} + (1-\pi_v^{(k-1)})\sigma_{u_k} \\ &= \sum_{u_i \in \mathcal{N}_v} \sigma_{u_i} - \sum_{\substack{u_i, u_j \in \mathcal{N}_v, \\ i<j}} \sigma_{u_i}\sigma_{u_j} + \sum_{\substack{u_i, u_j, u_l \in \mathcal{N}_v, \\ i<j<l}} \sigma_{u_i}\sigma_{u_j}\sigma_{u_l} \\ &+ \cdots + (-1)^{k-1}\sigma_{u_1}\sigma_{u_2}\cdots\sigma_{u_k}. \end{aligned} \tag{2.14}$$

This completes the proof.

2.3.1 Computing Activation Probability on a Single Node

Based on the inclusion-exclusion theorem, we can get the following Lemma,

Lemma 2 *Given a single node v's in-bound neighbors $\{u_i | u_i \rightsquigarrow v\}$ and edge weights P, to calculate the probability that node v becomes activated, we have the following iterative update equation,*

$$\pi_v^{(i+1)} = \pi_v^{(i)} + (1-\pi_v^{(i)})\sigma_{u_{i+1}}, \quad i = 1, \cdots, k-1. \tag{2.15}$$

Thus, now we can present the algorithm to compute the activation probability of single node v in the following,

```
Input: $\mathcal{N}_v = \{u_i | u_i \rightsquigarrow v\}$, $\{\sigma_{u_i}\}$, $k = |\mathcal{N}_v|$ denotes the number of in-bound
        neighbors of $v$
Output: $\pi_v^* = \pi_v^{(k)}$
Initialize $i = 1, \pi_v^{(1)} = \sigma_{u_1}$
for $i = 1 : k - 1$ do
    $\pi_v^{(i+1)} = \pi_v^{(i)} + (1 - \pi_v^{(i)})\sigma_{u_{i+1}}$
end
```

Algorithm 1: Activation Probability on v

2.3.2 Computing Activation Probability on Entire Network

Now we are ready to describe the algorithm to compute activation probability on entire network. Given fixed seed set S and edge weights P, the activation probability to IC model for each node v can be represented in the following,

$$\pi_v^* = F(\pi_{\mathcal{N}_v}^*), \quad v = 1, \cdots, n. \tag{2.16}$$

which can be obtained by the following updating strategy, where $\pi_{\mathcal{N}_v}$ denotes a vector whose elements are the activation probabilities of v's in-bound neighbors.

$$\pi_v^{(t+1)} = F(\pi_{\mathcal{N}_v}^{(t)}), \quad v = 1, \cdots, n. \tag{2.17}$$

where, $F(\cdot)$ denotes a function, which is represented in Eq.(2.15).

The detailed algorithm for computing activation probabilities for all nodes V is given in Algorithm 2, where π denotes the activation probability vector whose elements are activation probabilities for all nodes.

2.4 Computing Exact Activations for DAG Networks

Although, the above iterative strategy gives approximate solutions to the entire network, Independent Cascade Model only allows each node to be updated (influence) once. This can be achieved in DAG (directed acyclic graph), where a topological order can be established.

In this section, we describe the algorithm to compute the exact activations when the network is a directed acyclic graph (DAG) [1]. The algorithm for non-DAG network is presented in next section.

Given the network $G(V, E)$ and the seed set S. If $G\backslash S$ is a DAG, then we can compute all activations by passing through all the nodes. This is a linear time algorithm, linear in both $|V|$ and $|E|$.

[1] In fact, we only require $G\backslash S$ to be a DAG, because the seed set S are already activated.

```
Input: $G(V, E)$, $n = |V|$, edge weight $P$, activated seed set $S_0 = S$, $m = |S|$,
       $maxIter$
Initialize $\pi^{(0)}_{v \in S} = 1$, $\pi^{(0)}_{v \notin S} = 0$
for $l = 1 : maxIter$ do
    for $i = 1 : n$, $v_i \notin S$ do
        $\pi^{(l)}_{v_i} \leftarrow$ Activation probability on $v_i$
    end
    if $\|\pi^{(l)} - \pi^{(l-1)}\|_1 < \delta$ then
        break;
    end
end
Output: stationary activation probability for each node $\pi = \pi^{(l)}$
```

Algorithm 2: Activation probability on entire network

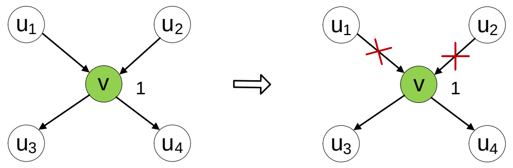


Figure 2.6: In-links are deleted for an activated node v.

In this algorithm, we utilize the above inclusion-exclusion theorem (IET). In IET the activation on v is easily computed from the activations on $\mathcal{N}_v$ (the in-bound nodes to v). A key observation is that if $G\backslash S$ is a DAG, then the ordering of nodes can be arranged in such a way that the activations on $\mathcal{N}_v$ are available (already computed) before v is visited (see Theorem 2). With this, activations are computed following the order in one pass.

We call this algorithm as DAG-IET algorithm. The main steps are:

(a) constructed the seeded network,

(b) compute the topological ordering of the seeded network,

(c) compute activations on nodes using the ordering.

Below, we discuss the algorithm in details. We first define the seeded network : A seeded network G_S is the original network G with seed set S activated and all in-links to these seed nodes are deleted.

The reason that in-links are deleted is that once a node is activated, it remains activated. Such in-links to this node will not affect the solution, and thus can be deleted. Figure 2.6 illustrates the situation.

If G is a DAG, the seeded network G_S is a DAG, and $G\backslash S$ is a DAG. If $G\backslash S$ is a DAG, G_S must be a DAG. Even if G is not a DAG, the seeded network G_S could be a DAG.

When G_S is a DAG, the topological ordering is obtained by the well-known topological sort algorithm. This is a order-$|E|$ algorithm.

After the topological ordering is computed, activations on non-seed nodes are computed using this ordering. One can easily prove the following

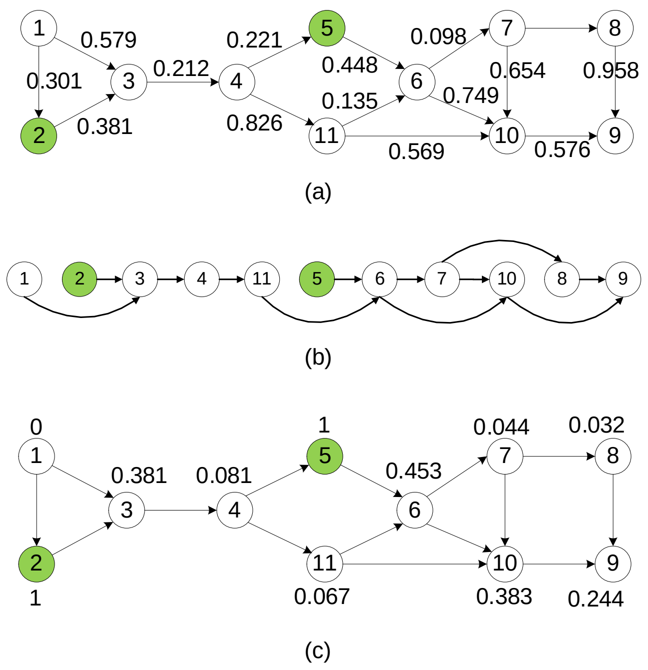


Figure 2.7: (a) A DAG network. Probabilities on the edges are shown. (b) Topological order of the seeded network. (c) Activations computed using DAG-IET algorithm.

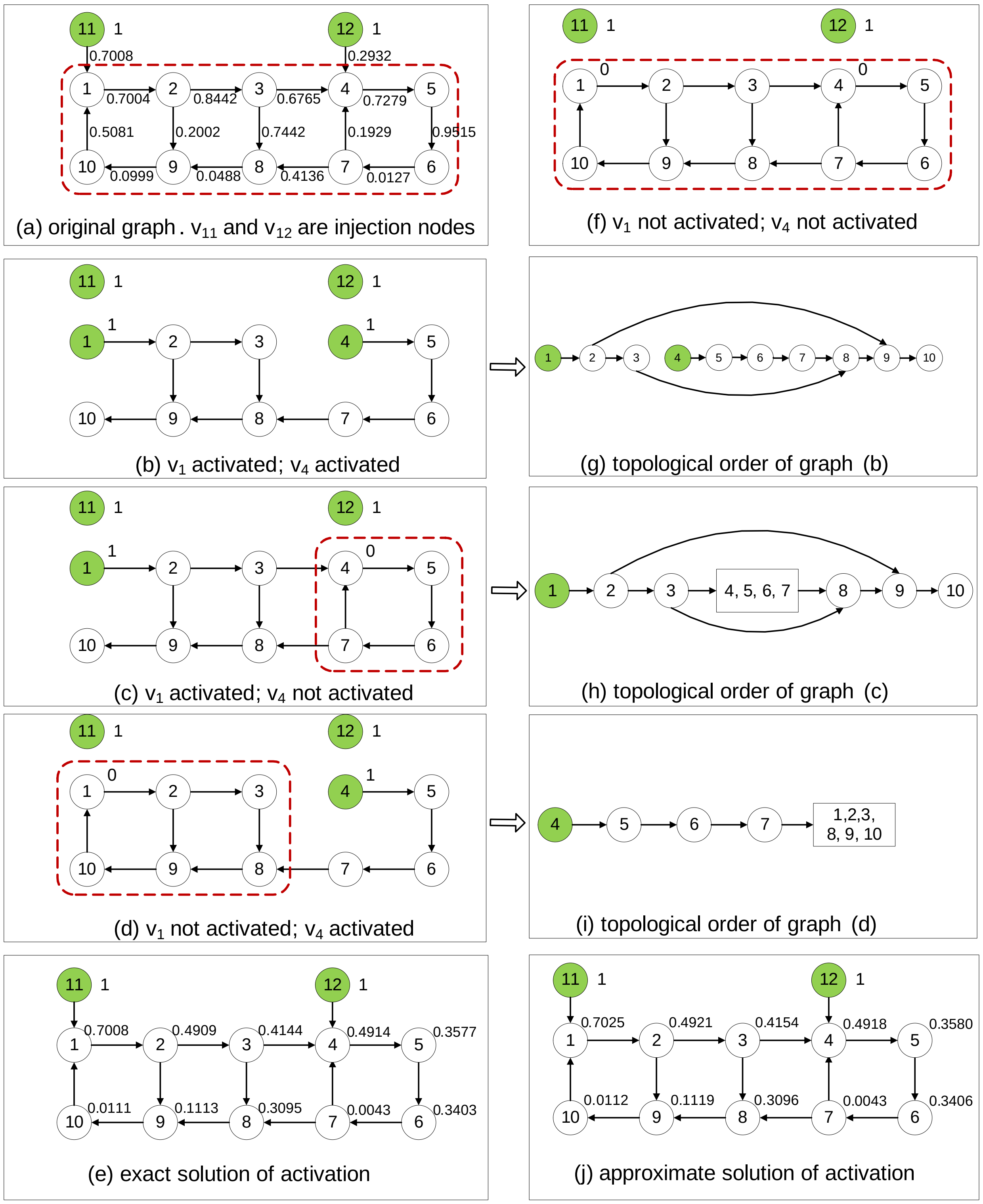
Theorem 3 *In the topological ordering of the DAG-IET algorithm, all nodes pointing to v are located before v.*

Figure 2.7 gives an illustration of the DAG-IET algorithm.

Illustration: The network in Figure 2.7 is a DAG. Node 2 and 5 are seed nodes. First, edges (1,2) and (4,5) are deleted to obtain the seeded network. This network is sorted to topological sort order, shown in Figure 2.7(b). Then we use DAG-IET algorithm in Algorithm 1 to compute the activation probabilities for each non-seed node, and the results are shown in Figure 2.7(c). Note that the final results for v_1 are $\pi_1 = 0$ because there are no activated nodes to influence it.

2.5 Injection Point Algorithm for Non-DAG Networks: Decomposition

In this and next two sections, we describe the algorithms for solving networks whose seeded networks are non-DAG.



(a) original graph. v_{11} and v_{12} are injection nodes
(f) v_1 not activated; v_4 not activated
(b) v_1 activated; v_4 activated
(g) topological order of graph (b)
(c) v_1 activated; v_4 not activated
(h) topological order of graph (c)
(d) v_1 not activated; v_4 activated
(i) topological order of graph (d)
(e) exact solution of activation
(j) approximate solution of activation
4, 5, 6, 7
1,2,3, 8, 9, 10
0.7008
0.2932
0.7004
0.8442
0.6765
0.7279
0.5081
0.2002
0.7442
0.1929
0.9515
0.0999
0.0488
0.4136
0.0127

Figure 2.8: Illustration of the injection point algorithm. Red-dashed circle indicates a strongly connected component (SCC). (a) is input network with influence weight on each edge. (b,c,d,f) are 4 case of influence spread from injection nodes v_{11}, v_{12}, with branching probabilities given in Eq.(2.19). In (b) in-links to activated v_1, v_4 are deleted. In (c) inlinks to v_4 are retained because v_4 could be activated by the influence coming from v_1. In (f), no influence passed on from v_{11}, v_{12} and thus all nodes have no possibility to be activated. (e) and (j) give the exact and approximate solutions of activation respectively.

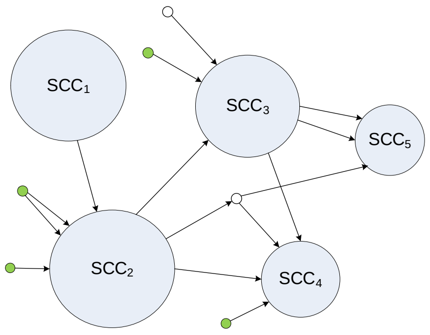


Figure 2.9: A non-DAG seeded network decomposed into a collection of strongly connected components (SCC): the small circles are single nodes, and green ones are seed nodes. There is no activated in-bound neighbors for SCC_1, therefore nodes in SCC_1 cannot be activated thus could be deleted from the network

In this section we describe how to decompose the non-DAG network to a series of logical units which is called *"injection-nodes + strongly connected component (SCC)" sub-network*. In next section, we describe how to solve these sub-networks *exactly*. Using the insights obtained there, in Section 7, we describe how to solve these sub-networks *approximately* using a linear algorithm.

A non-DAG seeded network can be decomposed into a collection of strongly connected components (SCC). The *component graph* G^{SCC} is obtained if we view each SCC as a single node (contracting all edges in a SCC). Figure (2.9) illustrates this process.

This component graph G^{SCC} is *acyclic* and we build a topological order on this component graph. Along this ordering on the component graph, the in-bound neighbors of a SCC (all nodes in the SCC) are all located before this SCC. Note that a seed node can not be inside a SCC because in-bound links to the seed nodes are deleted when constructing the seeded network. Thus seed nodes become 1-node SCCs on the component graph (we call them seed SCCs).

In general, on the topological ordering the first several SCCs before the first seed SCC are deleted since there is no activated nodes to influence them. The activations on them are set to zero.

Computing activations on the component graph are thus reduced to a series of tasks of computing activations on a
"in-bound neighboring nodes + SCC" sub-network.
The first of these sub-networks is the one where "in-bound neighboring nodes" are seed nodes. Here the activations on the in-bound neighboring nodes are known and our goal is to compute activations on the nodes in the SCC.

We further analyze the "in-bound neighboring nodes + SCC" sub-network, and reduce "in-bound neighboring nodes" on the component network to "injection nodes" in the original seeded network. Each of these in-bound neighboring nodes in the component graph could be: (a) a SCC, (b) a single non-seed node in the original network, (c) the seed node.

In case (b), single non-seed node will remain un-activated. Nodes like this are deleted from the network.

In case (a), we denote SCC$_1$ as the in-bound neighbor of SCC$_2$. This implies some nodes V_{1s} in SCC$_1$ points to some nodes in SCC$_2$ in the original network. Note that the activations on the nodes in SCC$_1$ are already computed. Nodes V_{1s} in SCC$_1$ are retained (since they will influence SCC$_2$) and other nodes in SCC$_1$ are deleted (since they will not influence SCC$_2$). Nodes V_{1s} are called injection nodes for SCC$_2$. The activations on these nodes are in general in $[0, 1]$.

In case (c), the seed node is an injection node for SCC$_2$.

In summary, we have decompose the original seeded non-DAG network to a series of logical units, "injection-nodes + SCC" sub-networks.

2.6 Injection Point Algorithm for Non-DAG Networks: Exact Algorithm

The key network structure we deal with is the ”injection-nodes + SCC” sub-network. Here there are several injection nodes that attempt to influence a SCC, as illustrated in Figure 2.8(a). Nodes v_{11}, v_{12} are injection nodes. They have been activated and they attempt to influence nodes $v_1 \cdots v_{10}$ which form a SCC.

Injection nodes: it is important to note that injection nodes v_{11}, v_{12} in Figure 2.8(a) could be either seed nodes, or intermediate nodes, such as v_3 in Figure 2.8(c). If injection nodes v_{11}, v_{12} are intermediate nodes, they have activation probabilities π_{11}, π_{12}. If v_{11}, v_{12} are seed nodes, we say they also have activation probabilities: $\pi_{11} = 1$ and $\pi_{12} = 1$, i.e., they are activated with probability 1.

Injection nodes standardization: we note there are several cases as shown in Figure 2.10. In Figure 2.10(a), node v has several injection nodes. This case can be transformed equivalently to “one injection-node” case where injection nodes combined into one node u which injects into v. Here $\pi_u = 1$ and the edge weight $P_{u,v}$ equals the activation probability of v computed using Theorem 1 (algorithm 1). In Figure 2.10(b), node u attempts to influence nodes $v_1, \cdots, v_k$. This can be equivalently viewed as each $v_1, \cdots, v_k$ has its own injection node u.

Injected nodes: inside the SCC, nodes who are immediately/directly influenced by the injection nodes. In Figure 2.8(a), nodes v_1, v_4 are injected nodes.

The computational algorithm is recursive reduction by enumerating all possible situations/states/cases of injected nodes. For each case, start anew on this reduced network. If the reduced network is a DAG, we use DAG-IET algorithm to solve it. If the reduced network is a non-DAG, start anew. This is repeated until the reduced network is small enough and the solution can be directly read from a recomputed and stored library. Solutions of different cases are combined together with appropriate probabilities.

2.6.1 Illustration of Recursive Reduction

Before we proceed further, we explain how our algorithm work for the network of Figure 2.8(a). This illustrates the essential elements of the algorithm.

Figure 2.8(a) has two injection nodes v_{11}, v_{12} with activation probability π_{11}, π_{12}. The injected nodes are v_1, v_4. The next stage of influence propagation are the 4 cases shown in Figure 2.8(b), (c), (d), (f). We first evaluate the activation probabilities $\pi^{(b)}_{1:10}$ for the case in Figure 2.8(b). This case is arrived with probability $P^{(b)} = \pi_{11}P_{11,1}\pi_{12}P_{12,4}$. Similarly, we evaluate $\pi^{(c)}_{1:10}$ for the case in Figure 2.8(c) and $\pi^{(d)}_{1:10}$ for the case in Figure 2.8(d). Note that $\pi^{(f)}_{1:10} = 0$ for the case in Figure 2.8(f) because there is no influence injection. The final activation probability are the weighted sum:

$$\pi_{1:10} = P^{(b)}\pi^{(b)}_{1:10} + P^{(c)}\pi^{(c)}_{1:10} + P^{(d)}\pi^{(d)}_{1:10} \tag{2.18}$$

where the branching probabilities are

$$\begin{aligned} P^{(b)} &= \pi_{11}P_{11,1}\pi_{12}P_{12,4}, \\ P^{(c)} &= \pi_{11}P_{11,1}\pi_{12}(1 - P_{12,4}), \\ P^{(d)} &= \pi_{11}(1 - P_{11,1})\pi_{12}P_{12,4}. \end{aligned} \tag{2.19}$$

The network of Figure 2.8(b) is a DAG and the activation probabilities are computed using DAG-IET algorithm.

The network of Figure 2.8(c) is non-DAG and its topological order is shown in Figure 2.8(h). We use DAG-IET algorithm to compute π_2, π_3. Note that $G_{4,5,6,7} \equiv (v_4, v_5, v_6, v_7)$ is a SCC. Thus $\{v_3 \bigcup G_{4,5,6,7}\}$ form an "injection-node + SCC" structure. We use the injection point algorithm to compute their activation probabilities.

The network of Figure 2.8(d) is non-DAG and its topological order is shown in Figure 2.8(i). We use DAG-IET algorithm to compute π_5, π_6, π_7. Now $G_{1,2,3,8,9,10} \equiv$

$(v_1, v_2, v_3, v_8, v_9, v_{10})$ is a SCC. Thus $\{v_7 \bigcup G_{1,2,3,8,9,10}\}$ form an "injection-node + SCC" sub-network. We use the injection point algorithm to compute their activation probabilities.

The essence of the injection point algorithm is a divide-and-conquer strategy: it repeatedly reduce an "injection-nodes + SCC" sub-network to smaller "injection-nodes + SCC" sub-network until these SCCs become small enough to be solved using the exact solutions explained in Section 2.2.1.

2.6.2 The Global Recursive Structure of Injection Point Strategy

As shown in Figure 2.8, the injection point algorithm is used to solve an "injection-nodes + SCC" sub-network in a recursive manner where the SCCs involved are gradually reduced in size until they are small enough and can be solved directly using the method of Section 2.2.1 (pre-implemented as a suit of library routines).

The Injection Point Algorithm is the following:

```
Injection-Point-Algorithm (IPA)
Input: "injection-nodes + SCC" sub-network
Output: activation probabilities of all nodes
Perform injection on "injection-nodes + SCC" sub-network.
FOR each injection result, DO
    Step 1. Find all SCCs of current network.
    Step 2. Contracting all SCCs and establishing a topological order.
    Step 3. Use DAG-IET to compute activation probabilities:
        IF no SCC, all activation probabilities on current network are computed
        IF encountering a SCC which is small enough, call a library to solve it
        IF the SCC is not small, identifying injection nodes to the SCC and call
            injection-point-algorithm on this sub-network.
        Continue to compute activation probabilities on the computing order until
```

the end or another SCC

Step 4. Accumulate activation probabilities from different injection branch.

RETURN

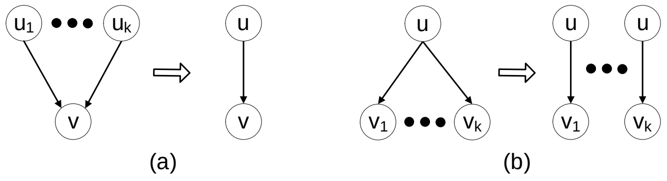


Figure 2.10: Injection equivalence

2.7 Injection Point Algorithm for Non-DAG Networks: Approximate Algorithm

In this section, we develop an approximate algorithm to solve the “injection-node + SCC” sub-network efficiently in two passes of the nodes and thus in linear time.

The algorithm is a modification of the DAG-IET algorithm of Section 4. When the sub-network is a DAG, the topological ordering ensures that when visiting each “un-touched” node v (where activation has not been evaluated so far), activations on the in-bound neighboring nodes $\mathcal{N}_v$ have already been correctly computed.

When the sub-network is non-DAG, the topological ordering does not exist. However, we can follow a DFS (depth-first-search) to visit each node and compute its activation. The problem here is that for some un-touched nodes, their in-bound neighboring nodes (or some of them) could also be un-touched, and thus the contribution from these un-touched neighboring nodes are unknown — the computation can not proceed.

We solve this problem by *pre-computing* the activations. We do the DFS-and-compute-activation pass twice. The first pass pre-computes the activations (approximately). In the second pass, for each node, all its in-bound neighboring nodes are already touched, i.e., have non-zero activation values (could be under-valued). Using IET, we compute the activation on this node (in fact, *updating* the value since this node is already touched).

The essence of the algorithm can be seen from computing π_2, π_3 of the 3-node non-DAG network of Figure 2.1(a). These activations will serve as values on $\mathcal{N}_v$ for all untouched node. At first iteration,

$$\pi_2^{(1)} = P_{12},\ \pi_3^{(1)} = P_{13}$$

With these values, activations on $\{\mathcal{N}_v\}$ are available for all v. At second iteration,

$$\pi_2^{(2)} = P_{12} + P_{13}P_{32} - P_{12}P_{13}P_{32},\ \pi_3^{(2)} = P_{13} + P_{12}P_{23} - P_{13}P_{12}P_{23}$$

At this point, the results are exactly correct.

The key observation is that even if there are SCCs in a non-DAG (such as SCC(v_2, v_3) in Figure 2.1(a)), influence does not propagate infinitely as cycles. This is because, in IC model, once a non-seed node is activated, it remains so forever. Once the node is activated, the in-link is effectively broken as shown in Figure 2.6 — it blocks the cycle.

Weighted DFS: intuitively, we wish to follow a traverse ordering that propagates influence most efficiently. A greedy approach is to follow the links with largest weight. This naturally motivates the weighted depth-first-search (DFS): when deciding who to visit among all links out of node v, we follow the link with highest edge weight (the activation probability). Standard DFS algorithm can be slightly modified to achieve this.

In summary, the algorithm is given below.

Approximate algorithm for "injection-node + SCC" sub-network:

(1) Pick the injected node with highest activation probability.

(2) Do weighted DFS pass and update/compute activations.

(3) Repeat step (2) once more.

Figure 2.8(j) gives the approximate activation probabilities for nodes in seeded graph of Figure 2.8(a), while Figure 2.8(e) shows the exact activation probability for each node. The differences between approximate solutions and exact solutions are all very small.

2.8 Selecting Seed Set for Viral Marketing

To this end, we can define the objective function of maximizing the social influence as follows,

$$\begin{aligned} \max_{S} \quad & \sigma(S) = \sum_{v=1}^{n} \pi_v \\ s.t. \quad & |S| = m, \\ & \pi_v = F(\pi_{\mathcal{N}_v}), \quad v = 1, \cdots, n. \end{aligned} \tag{2.20}$$

where, m is the size of seed set. We will present an probabilistic additive strategy to solve the above social influence maximization problem using greedy methods.

2.8.1 Greedy Method to Solve Social Influence Maximization

As discussed above, influence maximization is to determine m activated seeds at the beginning of the information propagation, in order to maximize the social influence in the end. We first propose an probabilistic additive strategy and two greedy methods, then based on these two greedy methods, another efficient incremental search strategy will be introduced.

2.8.1.1 Greedy Method 1

The basic idea is to select each node v_i as a single seed, i.e., $S = \{v_i\}$, and then compute the stationary activation probability using Algorithm 2 for all the other nodes $\{u | u \notin S\}$. We let $\beta_{\{i\}}$ denote the stationary activation probability for every node when

v_i is selected as the seed node. Let $\mathbf{B} = (\beta_{\{1\}}, \beta_{\{2\}}, \cdots, \beta_{\{n\}})$ Obviously, the elements on diagonal of matrix $\mathbf{B}$ are all 1. After getting every stationary activation probability vector in $\mathbf{B}$, we calculate the influence spread of each node, denoted by $\sigma(v_i)$, which is the sum of $\beta_{\{i\}}$. The following Algorithm 3 describes the detailed process on how to calculate $\mathbf{B}$.

```
Input: Edge weight $P$
for $i = 1 : n$, $S = \{v_i\}$ do
    $\pi^{(0)}_{v_i} = 1$, $\pi^{(0)}_{V \setminus v_i} = 0$,
    $\pi \longleftarrow$ Call Algorithm 2($P$,$S = \{v_i\}$),
    $\beta_{\{i\}} = \pi$.
end
Output: Stationary activation probability matrix $\mathbf{B}$
```

Algorithm 3: Computing stationary activation probability vector when each node is selected as the seed node

After we get the social influence spread for each node being seed node, we sort the influence spread scores in descending order, i.e., $\sigma(v_1) > \sigma(v_2) > \cdots > \sigma(v_n)$, and then select the top m nodes with largest influence scores as the initialization seed set S.

Greedy method 1 is based on our inclusion-exclusion theorem, and it's much faster than greedy method using Monte Carlo Simulation, which needs at least thousands of simulations even for calculating the influence score of each v_i, not mention the entire stationary activation probability matrix $\mathbf{B}$. We will present the time needed for both methods in the experiment section.

2.8.1.2 Greedy Method 2

Before presenting Greedy Method 2, we first introduce a probabilistic additive strategy when adding more nodes to a current seed set.

2.8.1.3 Probabilistic Additive

Suppose we have a current seed set S_c and its activation probability vector β_{S_c}, calculated by Algorithm 2, and now we are going to add v_i to the current seed set, note, v's activation probability vector $\beta_{\{i\}}$ can be get from matrix $\mathbf{B}$. We have the following definition,

definition 4 *Probabilistic Additive of vector β_{S_c} and $\beta_{\{i\}}$ is defined as follows,*

$$\beta_{S_c \cup \{i\}} = \beta_{S_c} \biguplus \beta_{\{i\}} = 1 - (1 - \beta_{S_c}) .* (1 - \beta_{\{i\}}) \tag{2.21}$$

where $.*$ means element wise multiplication. Similarly, Probabilistic Additive of m vectors is defined as follows,

$$\beta_{\{1\} \cup \{2\} \cdots \cup \{m\}} = \beta_{\{1\}} \biguplus \beta_{\{2\}} \cdots \biguplus \beta_{\{m\}} = 1 - \Pi_{i=1}^{m} (1 - \beta_{\{i\}}) \tag{2.22}$$

where Π also means element wise multiplication.

We can use the Probabilistic Additive of each node in S $\beta_{\cup\{i|i \in S\}}$ as the initialization when calculating the activation probability for entire network using Algorithm 2 with seed set S, which is much faster than 0 or 1 initialization used in Algorithm 2.

As Greedy Method 1 did, Greedy Method 2 first uses algorithm 3 to calculate the stationary probability distribution matrix $\mathbf{B}$, and then calculate influence score for each node $\sigma(v_i)$. Different with Greedy Method 1, Greedy Method 2 adds only one node to seed set at a time, which is described in the following,

(1) Add node i_1 with the largest influence score to seed set S_1, $S_1 = \{i_1\}$, and the activation probability vector is $\beta_{S_1} = \beta_{\{i_1\}}$.

(2) Add node i_2, which can lead to the largest influence score of seed set S_2, $S_2 = S_1 \cup \{i_2\} = \{i_1, i_2\}$. Then we use probabilistic additive of β_{S_1} and $\beta_{\{i_2\}}$ to initialize π^0 when calculating the activation probability vector for S_2 using Algorithm 2, $\pi^{(0)} = \beta_{S_1 \cup \{i_2\}} = \beta_{S_1} \uplus \beta_{\{i_2\}}$;

(3) Repeat the above process until we need to add node i_m, and node i_m should lead to the largest influence score of seed set S_m, $S_m = S_{m-1} \cup \{i_m\} = \{i_1, \cdots, i_m\}$. Then we use probabilistic additive of $\beta_{S_{m-1}}$ and $\beta_{\{i_m\}}$ to initialize π^0 when calculating the activation probability vector for S_m using Algorithm 2, $\pi^{(0)} = \beta_{S_{m-1} \cup \{i_m\}} = \beta_{S_{m-1}} \uplus \beta_{\{i_m\}}$;

In this way, we get another seed set S. To get a better seed set, we do Incremental Search starting from the seed sets calculated from both greedy methods.

2.8.1.4 Incremental Search Strategy

After we get an initialization seed set by those two greedy methods introduced in last section, we want to keep digging more efficient seed set by replacing the nodes those have least contribution in the current seed set to the final influence score.

First, we give two important procedures, add k node to current seed set S_c and drop k node from S_c, which are listed in Algorithm 4 and Algorithm 5, respectively. For k, we normally choose $k = 1, 2, 3$.

Now, we are ready to present the incremental search strategy in Algorithm 6,

After using incremental search on seed set obtained from greedy methods, we get a seed set with larger influence sore.

2.9 Experiments

To validate the performance of our Inclusion-Exclusion Theorem, we conduct experiments on real data sets to compare the results using Inclusion-Exclusion Theorem (Algorithm 2) and that of using Monte Carlo Simulation. We conduct another two groups of experiments. One is to compare the solutions of our fast approximate algorithm with

```
Input: Current Seed Set: $S_c$, the size of $S_c$: $q = |S_c|$, Activation Probability
        Matrix: $\mathbf{B}$
for $i = 1 : C^k_{n-q}$ do
    Select nodes to add : $\{v_{i_1}, \cdots, v_{i_k}\}$,
    $S_n = S_c \cup \{v_{i_1}, \cdots, v_{i_k}\}$,
    $\pi^{(0)} = \beta_{S_c \cup \{i_1\} \cdots \cup \{i_k\}}$,
    $\pi \longleftarrow$ Call Algorithm 2($P$, $S_n$, $\pi^{(0)}$),
    $\sigma(S_n) = \sum_v \pi_v$.
end
$\{v_{i_1}, \cdots, v_{i_k}\} \longleftarrow \arg\max(\sigma(S_n))$,
Output: $S_c = S_c \cup \{v_{i_1}, \cdots, v_{i_k}\}$, $\beta_{S_c} = \pi$, $\sigma(S_c) = \sigma(S_n)$
```

Algorithm 4: Add k node

```
Input: Current Seed Set: $S_c$, the size of $S_c$: $q = |S_c|$, Activation Probability
        Matrix: $\mathbf{B}$
for $i = 1 : C^k_q$ do
    Select nodes to drop : $\{v_{i_1}, \cdots, v_{i_k}\}$,
    $S_n = S_c \backslash \{v_{i_1}, \cdots, v_{i_k}\}$,
    $\pi^{(0)} = \beta_{\cup\{j | j \in S_n\}}$,
    $\pi \longleftarrow$ Call Algorithm 2($P$, $S_n$, $\pi^{(0)}$),
    $\sigma(S_n) = \sum_v \pi_v$.
end
$\{v_{i_1}, \cdots, v_{i_k}\} \longleftarrow \arg\max(\sigma(S_n))$,
Output: $S_c = S_c \backslash \{v_{i_1}, \cdots, v_{i_k}\}$, $\beta_{S_c} = \pi$, $\sigma(S_c) = \sigma(S_n)$
```

Algorithm 5: Drop k node

those of other state-of-the-art approximate algorithms - (1) Monte Carlo simulations, (2) maximum influence arborescence (MIA) model in [4]. The other one is to demonstrate the correctness of our injection point algorithm for the exact solution to IC model. We also conduct experiments on real world data sets to compare the influence spread of seed set get from our incremental search strategy with those of seed sets get from various algorithms.

2.9.1 Data Sets

We use two real world data sets - p2p-Gnutella08 and wiki-Vote, which are two directed networks, and downloaded from SNAP [2]. Table 2.1 lists the detailed information of these two data sets.

p2p-Gnutella08 is a snapshot of Gnutella peer-to-peer file sharing network from August 2002. Nodes represent hosts in the Gnutella network topology and edges represent connections between the Gnutella hosts. We call this network p2p in this paper.

[2]http://snap.stanford.edu

```
Input: Current Seed Set: $S_c$, the size of $S_c$: $q = |S_c|$, $k$
for $i = 1 : maxIter$ do
    $S_a, \sigma(S_a) \longleftarrow$ Add $k$ nodes,
    $S_d, \sigma(S_d) \longleftarrow$ Drop $k$ nodes,
    if $\|\sigma(S_a) - \sigma(S_d)\| < \delta$ then
        break;
    end
end
Output: $S_c = S_d$
```

Algorithm 6: Incremental search strategy

Table 2.1: Description of data sets

Name	# Nodes	# Edges
p2p-Gnutella08	6301	20,777
wiki-Vote	7115	103,689

Wiki-Vote contains the Wikipedia voting data from the inception of Wikipedia till January 2008. Nodes in the network represent wikipedia users and a directed edge from node i to node j represents that user i voted on user j.

We also generate one subgraph from p2p-Gnutella08, which has 223 nodes and 954 edges. We call this subgraph p2p-223.

2.9.2 Inclusion-Exclusion Theorem V.S. Monte Carlo Simulation

First, we present the influence spread (activation probability for each node) comparison between IC Monte Carlo Simulation and Inclusion-Exclusion Theorem (Algorithm 2), to verify the effectiveness of Inclusion-Exclusion Theorem in approximating the influence spread for entire network. For demonstration purpose, we first present the results for each node on p2p-223 subgraph. We randomly selected 10 nodes as seed nodes. The results from Monte-Carlo Simulations are the average of 20000 simulations/realizations. Edge weight P is fixed for both methods. The activation probabilities for each node from two methods are shown in Figure 2.11. There are 223 nodes and 954 edges on p2p-223 network, and we omit the nodes with activation probability 0, which left us less than 90 nodes presented in the figure. For convenient comparison, we sort the activation probabilities of Monte Carlo Simulations, and present their corresponding activation probabilities calculated by Inclusion-Exclusion Theorem. Apparently, the two curves from these two methods almost coincide with each other.

We then present more comparisons between activation probabilities achieved by applying Monte Carlo simulation and those achieved by Inclusion-Exclusion Theorem. Fig-

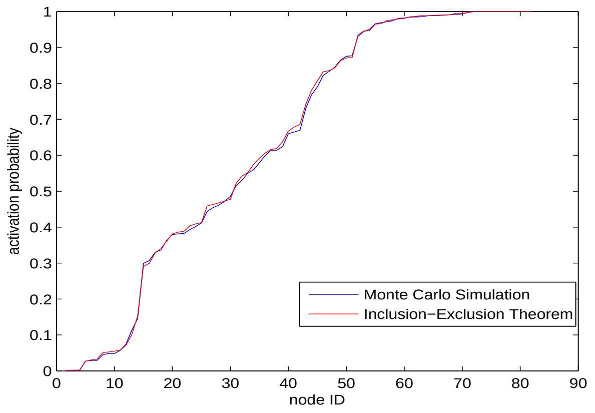
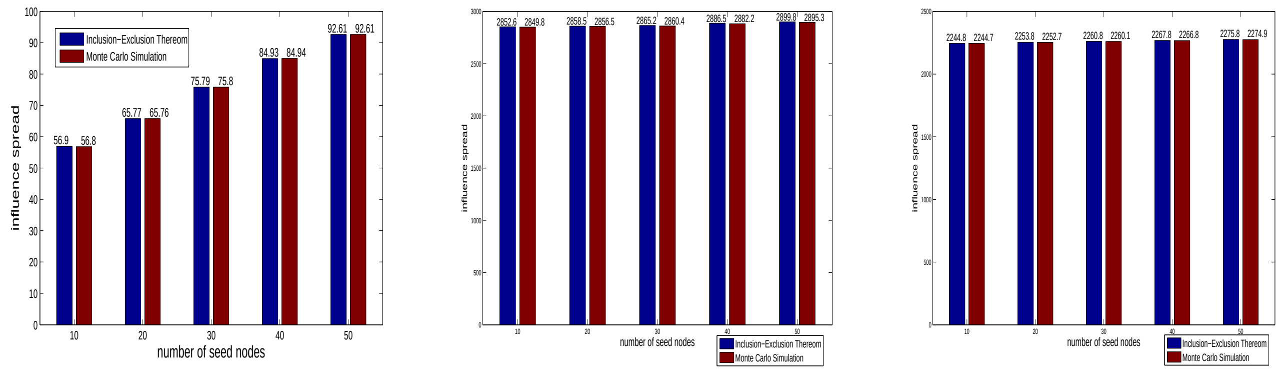


Figure 2.11: Activation probability comparison between IC Monte Carlo Simulation and Inclusion-Exclusion Theorem

Table 2.2: Mean squared errors (MSE) and mean absolute errors (MAE) between the two activation probability vectors achieved Inclusion-Exclusion Theorem and Monte Carlo simulation on p2p-223 network

	$m = 20$	$m = 30$	$m = 40$	$m = 50$
MSE	8.5×10^{-5}	7.9×10^{-5}	7.8×10^{-5}	7.1×10^{-5}
MAE	4.8×10^{-4}	4.7×10^{-4}	4.5×10^{-4}	3.8×10^{-4}

ure 2.12 shows the influence spreads by those two methods at different sizes of the same seed set - $m = \{10, 20, 30, 40, 50\}$ on the three data sets - p2p-223, p2p and wiki-Vote. The influence spreads are almost the same on p2p-223 subgraph, and are very close on p2p and wiki-Vote data sets. As shown on the figures, the altitudes of the histogram at the same m are almost the same for those two methods. We also show the mean squared errors (MSE) and mean absolute errors (MAE) between two activation probability vectors achieved by those two methods. Table 2.2, 2.3 and 2.4 show the MSE and MAE results at $m = \{20, 30, 40, 50\}$ (results at $m = 10$ are omitted due space limit) on the three data sets mentioned above, which demonstrate that the two vectors are almost the same when given the same seed set, i.e., the approximate results achieved by Inclusion-Exclusion Theorem are effective for generic graphs.



(a) p2p-223　　(b) p2p　　(c) wiki-Vote

Figure 2.12: Influence spreads computed by Inclusion-Exclusion Theorem and Monte Carlo Simulation given different sizes of seed sets

Table 2.3: Mean squared errors (MSE) and mean absolute errors (MAE) between the two activation probability vectors achieved Inclusion-Exclusion Theorem and Monte Carlo simulation on p2p network

	$m = 20$	$m = 30$	$m = 40$	$m = 50$
MSE	7.5×10^{-5}	8.5×10^{-5}	8.4×10^{-5}	8.3×10^{-5}
MAE	3.0×10^{-3}	3.8×10^{-3}	3.7×10^{-3}	3.3×10^{-3}

Second, we compare the time needed to compute the influence spread given different sizes of seed sets. Table 2.5 and Table 2.6 list the time needed for both Inclusion-Exclusion Theorem and Monte Carlo Simulation on two data sets - p2p-223 and p2p, at different m, where m denotes the number of seed nodes selected. And at the same m, the same seed set is selected for both methods. Let's look at Table 2.6, when $m = 50$, the time for Inclusion-Exclusion Theorem is just 2.8450 seconds, while Monte Carlo Simulation needs 36945.6 seconds for 20000 realizations. So the time for Inclusion-Exclusion Theorem is

Table 2.4: Mean squared errors (MSE) and mean absolute errors (MAE) between the two activation probability vectors achieved Inclusion-Exclusion Theorem and Monte Carlo simulation on wiki-Vote network

	$m = 20$	$m = 30$	$m = 40$	$m = 50$
MSE	5.0×10^{-6}	5.1×10^{-6}	5.2×10^{-6}	5.1×10^{-6}
MAE	8.1×10^{-5}	8.1×10^{-5}	8.2×10^{-5}	8.0×10^{-5}

Table 2.5: Time (sec) needed to compute the influence spread given different sizes of seed sets m on p2p-223 network

Methods	$m = 10$	$m = 20$	$m = 30$
Inclusion-Exclusion	0.1079	0.02531	0.01925
Monte Carlo Simulation (20000 times)	95.3935	94.0613	93.1055

Table 2.6: Time (sec) needed to compute the influence spread given different sizes of seed sets m on p2p network

Methods	$m = 10$	$m = 30$	$m = 50$
Inclusion-Exclusion	3.2012	3.0126	2.8450
Monte Carlo Simulation (20000 times)	32345.4	35372.2	36945.6

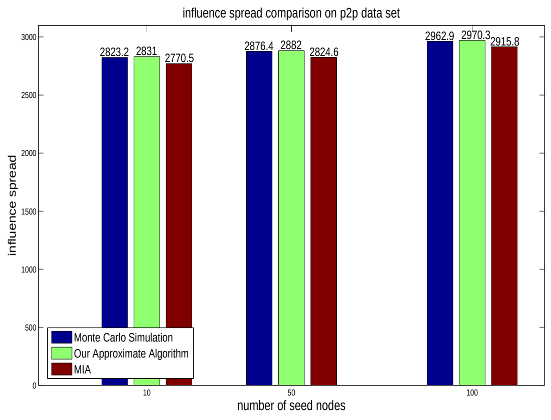
competitive with that of just one time Monte Carlo Simulation, however, Monte Carlo Simulations need thousands of times simulations to reach a steady solution. Therefore, our probabilistic solutions speed up the computation of influence spread, especially on large data sets, which make greedy methods to viral marketing scalable to large data sets.

2.9.3 Comparison of Injection Point Approximate Algorithm

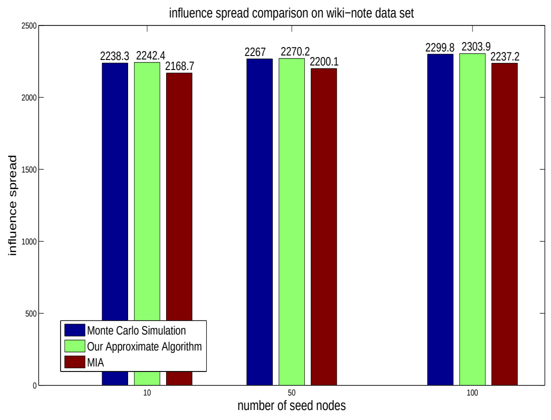
The purpose of experiments here is to validate the approximation accuracy of our fast algorithm introduced in Section 2.7. Monte Carlo simulation [1] is widely used and regarded as one good estimation of IC model. The maximum influence arborescence (MIA) model [4] is also a good way to approximate the spread given seed nodes. We compare the results of our approximate method to those two methods given the same seed sets. For each data set, given a seed set S, we run Monte Carlo simulations for 20000 times to get the final activation probability for each node. And for MIA model's threshold, we tune the value of threshold according to the papers suggestion - find a point where the change of arborescence size slows down.

We first compare the total influence spread given seed set S, which is the sum of the activation probability of each node, which is shown in Figure (2.13). As we can see, the spread of our approximate algorithm is more near to the spread of Monte Carlo simulations. For example, on wiki-Vote data set, when number of seed set is 50, the spread of our algorithm is 2267, which is near to Monte Carlo simulation's 2270, while MIA get 2200, underestimating the influence spread of seed nodes.

Next we compare root-mean-square error (RMSE) of our approximate solution with that of MIA, assuming Monte Carlo simulation gives the correct values. The results are shown in Figure (2.14). As we can see, our approximate algorithm gets more accurate approximation to the results of Monte Carlo simulations. RMSE of our approximate algorithm is less than 0.005, while RMSE of MIA is around 0.035.



(a) p2p



(b) wiki-Vote

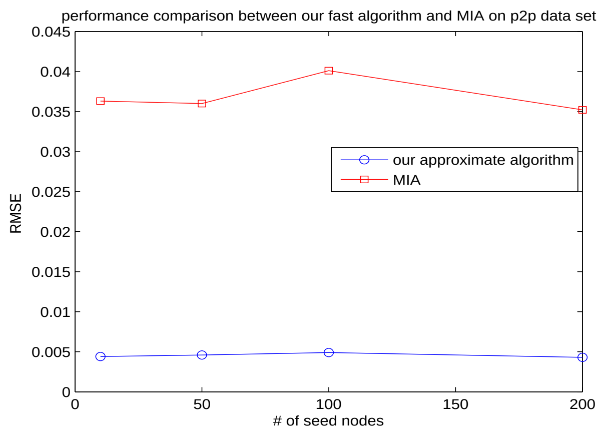
Figure 2.13: Influence spread comparison

2.9.3.1 Time Comparison

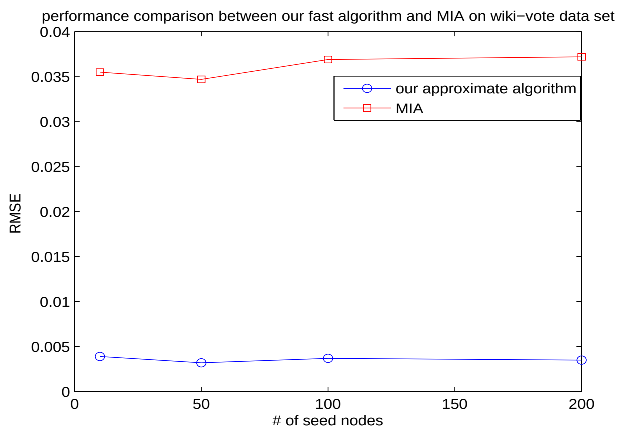
Our approximate algorithm is fast, since we follow a weighted DFS traverse ordering to update the activations twice. Monte Carlo simulations are time consuming, because they need to run 20000 times to get a good and steady approximation. MIA is also slower than our algorithm, because for each node, they need to build the maximum influence in-arborescence tree structure first. We list the running time comparison in Table 2.7 for p2p data set and Table 2.8 for wiki-Vote data set. Our experiments are run on a PC with a 3.0GHz Intel Core 2 Duo Processor and 12GB memory.

Table 2.7: Running time (seconds) comparison on p2p data set

seed nodes	$\|S\|$ = 10	$\|S\|$ = 50	$\|S\|$ = 100
Monte Carlo	1.62×10^4	1.59×10^4	1.58×10^4
MIA	2.53×10^3	2.61×10^3	2.64×10^3
Our approximate	1.04×10^2	0.93×10^2	1.01×10^2



(a) p2p

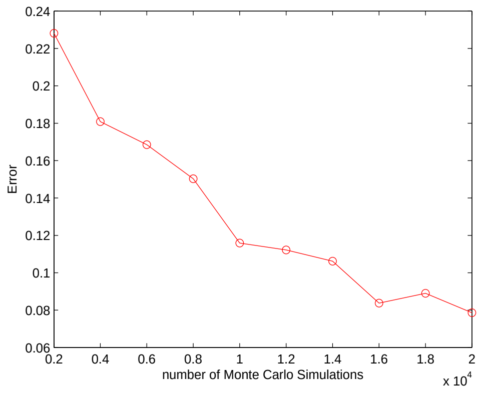


(b) wiki-Vote

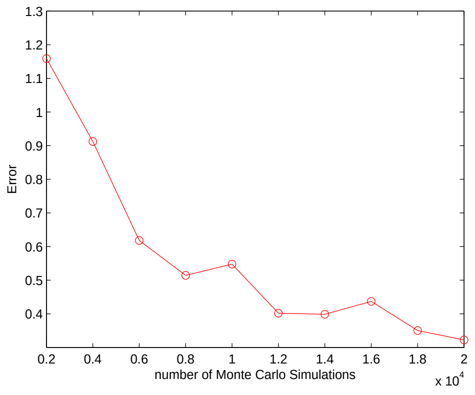
Figure 2.14: RMSE comparison between our approximate algorithm and MIA

Table 2.8: Running time (seconds) comparison on wiki-Vote data set

seed nodes	$\lvert S\rvert$ = 10	$\lvert S\rvert$ = 50	$\lvert S\rvert$ = 100
Monte Carlo	4.45×10^4	4.51×10^4	4.53×10^4
MIA	8.23×10^3	8.31×10^3	8.42×10^3
Our approximate	2.85×10^2	2.81×10^2	2.79×10^2



(a) p2p data set



(b) wiki-Vote data set

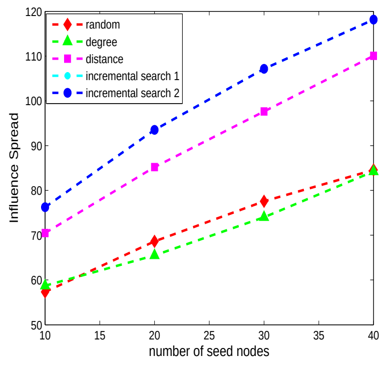
Figure 2.15: Accumulated absolute differences between exact solution and Monte Carlo simulation solutions when Monte Carlo simulations are run from 2000 to 20000 times

2.9.4 Comparison of Injection Point Exact Algorithm

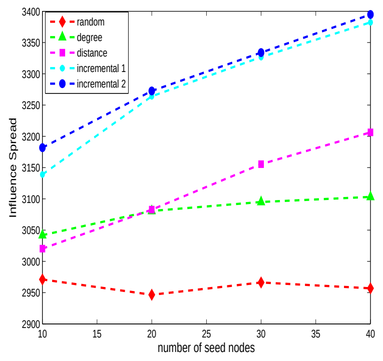
We do experiments here is to demonstrate that the solution from Monte Carlo simulations, as the number of simulations increase, approaches to the exact solution by injection point algorithm. For each data set, given a seed set S, we run Monte Carlo simulations from 2000 to 20000 times and check the differences between estimated solution by Monte Carlo simulations and our exact solution. Because the exact algorithm is slow, we do these experiments on sub-networks. Sub-network from wiki-Vote data set has 233 nodes and 456 edges and sub-network from p2p data set has 105 nodes and 151 edges.

The differences between Monte Carlo simulations solution and our exact algorithm solution are defined as

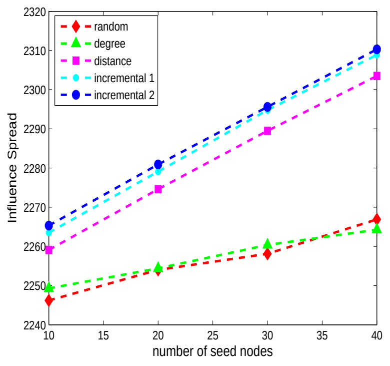
$$\Delta\pi^i = |(\pi_{exact})_i - (\pi_{mc})_i|, \text{ Error} = \sum_{i=1}^{n} \Delta\pi^i \tag{2.23}$$



(a) p2p-223



(b) p2p



(c) wiki-Vote

Figure 2.16: Influence spreads of seed sets selected by different methods. Note, the curves corresponding to incremental search 1 and incremental search 2 almost coincide with each other on p2p-223 network

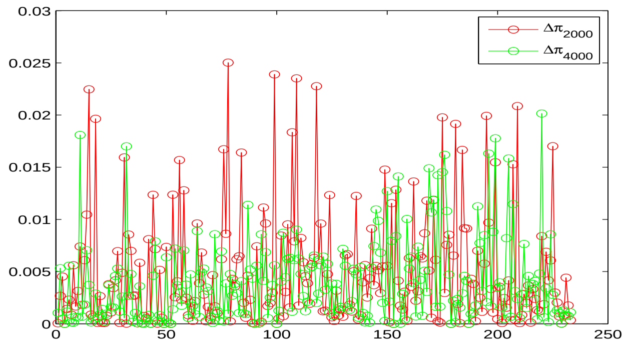
where, n is the number of nodes in the graph and π_{exact} is the exact activation probabilities calculated by our injection point algorithm; π_{mc} is the activation probabilities by Monte Carlo simulations (averaged over the specified number of MC simulations).

2.9.4.1 Wiki-Vote data set

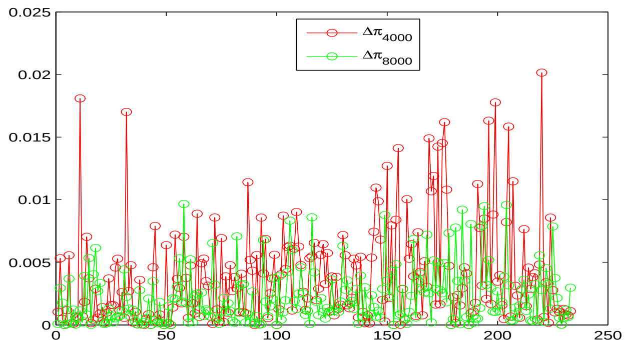
We randomly select 5 nodes as the seed nodes. Figure 2.15b shows the accumulated differences between our exact solution and Monte Carlo simulation solutions on wiki-Vote data set, which is calculated by Eq.(2.23)

On Figure 2.15b, red circle represents the differences between the exact solution and the Monte Carlo simulation solution at different number of times (indicated along horizontal axis). The curve is roughly descending, which is consistent with our intuition - the more times Monte Carlo simulations, the more accurate the Monte Carlo solution will be. This verifies the correctness of our exact solution to some extent.

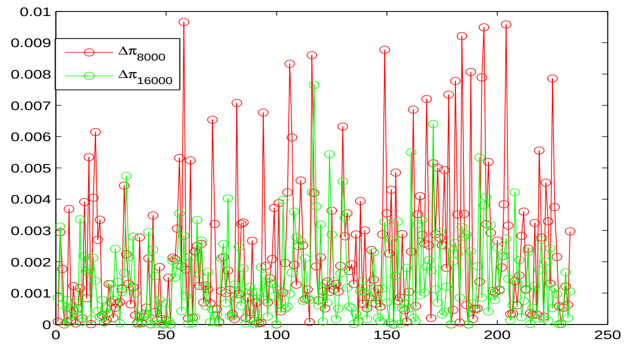
Figure 2.17 shows the difference of exact solution and Monte Carlo simulation solution on every node on wiki-Vote data set. The horizontal ordinate represents the index



(a) $\Delta\pi_{2000}$ vs. $\Delta\pi_{4000}$



(b) $\Delta\pi_{4000}$ vs. $\Delta\pi_{8000}$



(c) $\Delta\pi_{8000}$ vs. $\Delta\pi_{16000}$

Figure 2.17: Difference between exact solution and Monte Carlo simulations on wiki-Vote data set (number of seed nodes=5). Shown are the absolute value of the difference at each node. $\Delta\pi_{2000}$ is the results of Monte Carlo simulations of 2000 times. Similarly, for $\Delta\pi_{4000}$, $\Delta\pi_{8000}$, $\Delta\pi_{16000}$.

of each node. $\Delta\pi_{2000}$ means the absolute value of the difference between the two methods when Monte Carlo simulations are run 2000 times and similarly, $\Delta\pi_{4000}$, $\Delta\pi_{8000}$, $\Delta\pi_{16000}$ mean the absolute value of the difference between the two methods when Monte Carlo simulations are run 4000, 8000, 16000 times. To present the difference clearly, we show three 2-group comparison: $\Delta\pi_{2000}$ vs. $\Delta\pi_{4000}$, $\Delta\pi_{4000}$ vs. $\Delta\pi_{8000}$ and π_{8000} vs. $\Delta\pi_{16000}$. The overall difference is decreasing when the running times of Monte Carlo simulations are doubled. The biggest difference in $\Delta\pi_{2000}$ is around 0.025, and in $\Delta\pi_{4000}$, it's reduced to 0.02. In $\Delta\pi_{16000}$, it's reduced to 0.008, which is small enough to ignore.

2.9.4.2 P2p data set

We randomly select 5 nodes as the seed nodes. Figure 2.15a shows the accumulated differences between the two methods, which is calculated by Eq.(2.23). The curve is roughly descending too, similarly with that on wiki-Vote data set.

2.9.5 Seeds Selected by Different Methods

This part is to compare the social influence of seed sets selected by our incremental strategies with those of seed sets selected by other methods. We compare the following set of algorithms.

(1) Random selection: select m nodes randomly from V as the seed nodes.

(2) Degree selection: select m nodes with the largest out-degree as the seed nodes.

(3) Distance selection: select m nodes with smallest average shortest-path distances to all other nodes as the seed nodes.

(4) Incremental search 1: it's a combination of greedy method 1 and incremental search.

- Compute activation probability matrix $\mathbf{B}$ using Algorithm 3.
- Select m nodes with the largest influence score $\sigma(v_i)$.

- Apply incremental search strategy using Algorithm 6 on seed nodes selected by last step, with parameter $k = [1, 2, 3]$.

(5) Incremental search 2: it's a combination of greedy method 2 and incremental search.

- Select m nodes using greedy method 2 introduced in section 5.3.
- Apply incremental search strategy using Algorithm 6 on seed nodes selected by last step, with parameter $k = [1, 2, 3]$.

Note, for all the above 5 methods, we apply our Algorithm 2 to compute the influence spread for the entire network for a given seed set. And edge weight P remains fixed for the same data set. The experiments are run on a PC with a 3.0GHz Intel Core 2 Duo Processor and 12GB memory.

The results on 3 data sets are shown in Figure 2.16. Our methods outperform the other methods significantly. The point is that once we get the activation probability matrix $\mathbf{B}$, a lot of recalculation can be omitted by combining our proposed probabilistic additive strategy.

2.10 Conclusion

In this chapter, we first propose an inclusion-exclusion theorem to compute the activation probability for each node on the network, then we propose an injection point algorithm to compute the influence spread under IC model both approximately and exactly. The exact solution can provide guidance on developing efficient estimate solutions. The structures used in the exact algorithm provides a convenient way to design the approximate algorithm which runs linear in both number of edges and nodes. Experiments shown our approximate algorithm gives good approximations to activation probabilities, with RMSE about seven times smaller than the state-of-art MIA approximate algorithm while significantly faster than MIA algorithm. We believe the "injection-node + SCC" structure and their solution algorithms could be useful for solving a number of problems in IC model. We also propose an incremental search strategy to further refine the selected seed sets, which are first gained by greedy methods, and the incremental search strategies improve the final spread greatly.

CHAPTER 3

Social Tagging Recommendation

3.1 Introduction

Social Tagging is an important feature of Web 2.0, which enable many users add keywords (tags) to items/resources like music (last.fm), pictures(flickr), web pages(del.icio.us). This tagging information helps those websites organize their resources and assist the users to communicate with each other. Tag recommendation system helps the tag process by advising a set of tags to the user that he may use for this item. Just like the traditional recommendation system, the tag recommendation system is also based on the similarly observation, ”a user always marks an item with the tag which has already been used by the other users”. However, different from the traditional item-user (two dimensions) recommendation system, the tag recommendation system contains three dimensions - users, tags, items. Although some traditional methods, such as Collaborative Filtering, link mining, etc, can be directly extend to the tag recommendation by folding three dimensional space into three bipartite relationship item-tag, tag-user, user-item, these methods miss the holistic interactions between three dimensions. Symeonidis et al [19] first use the tensor model for the tag recommendation system and predict the tags to the users. Several studies using tensors also appeared in [20][21][22].

However, most of these tensor decomposition methods [23] suffer from several inherent weakness. One of the most well known challenge is the sparse data problem. In Table 3.1, we list the statistics of three widely used tensor data sets. We use tensor to model the data. The number of nonzero tensor elements show the number of tags actually been attached to items. (Note that a specific tag can be attached to many different items;

Table 3.1: Real world tag data set statistics, with number of tags, number of items, number of users, number of nonzero tensor elements (NNZ) and relative NNZ.

Dataset	$\#Item$	$\#Tag$	$\#User$	NNZ	Rel-NNZ
Last.Fm	1278327	272605	51992	10274749	5.6E-10
Movie Lens	7601	14810	4009	95499	2.1E-7
Delious	67	1472	2387	9212	3.9E-5

each of these attachment constituent a nonzero tensor element.) From the statistics, the *relative* number of nonzero tensor elements is defined as

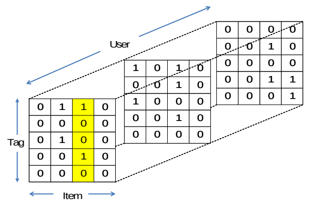
$$\text{Relative NNZ} = \frac{\text{number of tags actually attached}}{|items| \times |tags| \times |users|}$$

are extremely low, less than 1.0E-04=0.0001=0.01%, which indicates that all of these real world tag data sets are sparse networks. Many tensor decomposition algorithms are impeded by the sparse problem, hence can't handle users who have marked few items. Furthermore, previous tensor approaches simply set the masked-out values to zero and compute tensor decomposition once as the prediction. Thirdly, these tensor decomposition methods have high space and time complexity, and are impractical for large data sets.

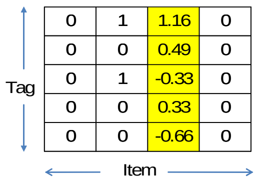
In this chapter, we present a new tensor decomposition model that specifically deals with very sparse data. The model utilizes low-order polynomials to improve/enhance statistics among users, items and tags. In contrast, traditional tensor decomposition methods such as Tucker and Parafac decompositions use only high order polynomials (3rd order polynomials for 3rd order tensor) which appear to overfit these very sparse social tagging data. Experiments on many social tagging data shows that our low order tensor decomposition model outperforms traditional decompositions consistently and significantly for the tag prediction problem. In addition, this low order tensor model has lower time complexity.

3.2 Problem Definition

Our research work focuses on how to provide a user with a ranked list of tags for a special item. For example, Figure 3.1a shows an example of the user-tag-item-relationship.



(a) Tensor X



0	1	1.16	0
0	0	0.49	0
0	1	-0.33	0
0	0	0.33	0
0	0	-0.66	0

(b) Result

Figure 3.1: (a) An example of tensor X with 3 users and a post been masked. (b) The predicted results.

If we want to recommend the tag for user 1 on item 3, we will mask the column of user 1 item 3, which has been drawn as the yellow color and set this column as the missing values. Then, we run tag recommendation algorithm which provide predicted values for each tag that may be used by this user. Figure 3.1b shows these predicted values. Then, we sort these values, and return the top N tags to the user. In this example, if N is equal to 1, we will return tag 2 to this user.

We formalize the notion of user-tag-item relationship and formulate the tag recommendation problem. The tag dataset is a tuple of $T = (U, T, I, R)$, where U, T and I are the subset of users, tags and items respectively and R is a relationship between I,U and T, which $R \in IUT$. For example, $(i, j, k) \in R$, that means user k mark item i with the tag j. For the tag recommendation, we will recommend for the special user-item pair(i, k) a list of tags $T(i, k)$. We define these user-item pairs as the post as following.

$$Post(i, k) = \{(i, k) | i \in I, k \in U, \exists j \in T : (i, j, k) \in I \times T \times U\}$$

In the real application, $T(i, k)$ is calculated by ranking on the set of tags by some criterion and quality and then select the top n unused tags to recommend the user.

We list all the notations in our paper in table 3.2.

Table 3.2: Symbols

Symbol	Definition
X	original tensor, $X = X_\Omega + X_m$
X^{old}	the original training tensor
X^{new}	the fold-in tensor
X_Ω	elements with observed value in X
X_m	elements with missing value in X
N_i	the total number of items
N_j	the total number of tags
N_k	the total number of users
$\|\|X\|\|$	Frobenius norm, $\|\|X\|\|^2 = \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} \sum_{k=1}^{N_k} X_{ijk}^2$
X_{i++}	$X_{i++} = \sum_{j=1}^{N_j} \sum_{k=1}^{N_k}$. X_{+j+},X_{++k} similarly defined
X_{i**}	$X_{i**} = X_{i++}/N_j N_k$. X_{*j*},X_{**k} are similarly defined
X_{ij+}	$X_{ij+} = \sum_{k=1}^{N_k} X_{ijk}$. X_{+jk},X_{i+k} are similarly defined
X_{ij*}	$X_{ij*} = X_{ij+}/N_k$. X_{*jk},X_{i*k} are similarly defined
X_{+++}	$X_{+++} = \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} \sum_{k=1}^{N_k} X_{ijk}$
X_{***}	$x_{***} = X_{+++}/N_i N_t N_k$
ε	the convergence factor, $\varepsilon = 0.001$

3.3 Low Order Tensor Decomposition

3.3.1 Motivation

Most social tagging data are very sparse. This refers to the relative number of nonzeros in the data are very low. For example, for the last.fm data, the percentage number of actually assigned tags are 0.1%.

We first mention an idea often used for improving rare statistics. Let us consider a rare disease such as a certain type of cancer. Suppose our task is to count the rate of its occurring per thousand people for a county. This rate will fluctuate significantly from

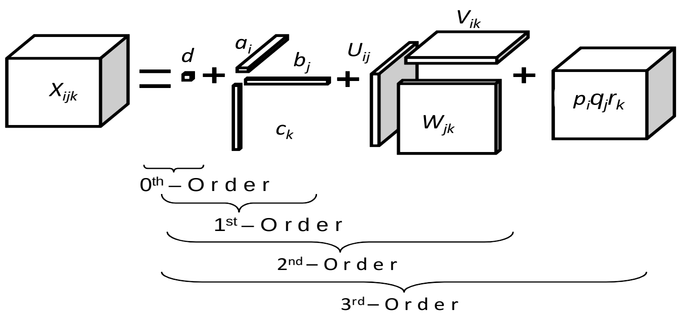


Figure 3.2: Tensor decomposition relationship

county to county because the number of samples are too small. Instead, we can average over a large population or a larger region (say both the county and its neighboring counties) to improve the statistics.

Now we demonstrate how to improve the statistics for tensor decomposition. The key idea is to use low-order polynomials. We will discuss the 0-th, 1st, 2nd order decompositions. Figure 3.2 shows the relationship between each tensor decompositions.

Consider the zero-th order polynomial. We set

$$Y_{ijk} = d \tag{3.1}$$

where d is a constant, and obtain the optimal solution by

$$\min_{d} J = \sum_{ijk} (X_{ijk} - d)^2 \tag{3.2}$$

Clearly,

$$J = \sum_{ijk} (X_{ijk}^2 - 2X_{ijk}d + d^2) = \sum_{ijk} X_{ijk}^2 - 2\sum_{ijk} X_{ijk}d + N_iN_jN_kd$$

Setting $\partial J / \partial d = 0$, we obtain the optimal solution

$$d = X_{***}, \tag{3.3}$$

where X_{***} is defined in Table 3.2. Clearly d is the average of the tensor X in all dimensions. Interestingly, this clearly matches our intuition.

Consider the first order polynomials. We set

$$Y_{ijk} = a_i \tag{3.4}$$

and obtain the optimal solution by

$$\min_{a_i} J = \sum_{ijk} (X_{ijk} - a_i)^2 \tag{3.5}$$

Clearly,

$$\begin{aligned} J &= \sum_{ijk}(X_{ijk}^2 - 2X_{ijk}a_i + a_i^2) && (3.6) \\ &= \sum_{ijk} X_{ijk}^2 - 2\sum_i X_{i++}a_i + N_jN_k\sum_i a_i^2 && (3.7) \end{aligned}$$

where X_{i++}, X_{i**} are defined in Table 3.2. Setting $\partial J/\partial a_i = 0$, we obtain the optimal solution

$$a_i = X_{i**}, \tag{3.8}$$

Clearly, a_i is the average over j, k dimensions. This significantly improve the statistics (by a factor of N_jN_k). We can do this for other dimension and the decomposition model of Eq.(3.4) can be expanded to general 1st order terms

$$Y_{ijk} = a_i + b_j + c_k \tag{3.9}$$

The optimal solution of this model is given similarly by the averages of other dimensions.

Consider the second order polynomials. We set

$$Y_{ijk} = U_{ij} \tag{3.10}$$

and obtain the optimal solution by

$$\min_{U_{ij}} J = \sum_{ijk}(X_{ijk} - U_{ij})^2 \tag{3.11}$$

Clearly, J can be expanded as

$$\sum_{ijk}(X_{ijk}^2 - 2X_{ijk}U_{ij} + U_{ij}^2) = \sum_{ijk} X_{ijk}^2 - 2\sum_{ij} X_{ij+}U_{ij} + N_k\sum_{ij} U_{ij}^2$$

where X_{ij+}, X_{ij*} are defined in Table 3.2. Setting $\partial J/\partial U_{ij} = 0$, we obtain the optimal solution

$$U_{ij} = X_{ij*}, \tag{3.12}$$

i.e., U_{ij} is the average over the k dimension. This clearly improve the statistics. We can do this for other dimension and the decomposition model of Eq.(3.10) can be expanded to generic 2nd order terms

$$Y_{ijk} = U_{ij} + V_{ik} + W_{jk} \tag{3.13}$$

The optimal solution of this model is given similarly by the averages of other dimensions.

3.3.2 Baseline Low Order Tensor Decomposition

From the above analysis, combing zero order Eq.(3.3) ,first order Eq.(3.8) and second order Eq.(3.13), our low-order model of tensor decomposition is defined as

$$Y_{ijk} = a_i + b_j + c_k + d + U_{ij} + V_{ik} + W_{jk} \tag{3.14}$$

Thus the model parameters are $\theta = (a, b, c, d, U, V, W)$.

This model has a very important property that it's optimal solution can be found in closed form, as showing in the following theorem:

Theorem 5 *The optimization of*

$$\min_{\theta} J = \sum_{ijk} (X_{ijk} - Y_{ijk}(\theta))^2 \tag{3.15}$$

has the optimal solution

$$d = X_{***}, \;\; a_i = -X_{i**}, \; b_j = -X_{*j*}, \; c_k = -X_{**k}, \tag{3.16}$$

$$U_{ij} = X_{ij*}, \; V_{ik} = X_{i*k}, \; W_{jk} = X_{*jk}. \tag{3.17}$$

or combined together

$$Y_{ijk} = X_{***} - X_{i**} - X_{*j*} - X_{**k} + X_{ij*} + X_{i*k} + X_{*jk} \tag{3.18}$$

3.4 Missing Value Problem

In recommender systems for both rating data and social tagging data, there are missing values. Here we discuss the algorithm to deal with missing value in tensor and prove its convergence.

The tensor missing value problem can formulated as

$$\min_{Y} \|X - Y\|^2_\Omega = \sum_{(ijk)\in\Omega} (X_{ijk} - Y_{ijk})^2 \tag{3.19}$$

where Ω represent the collection of tensor elements which have been assigned values. The missing value problem is to solve for the tensor decomposition model Y such as Eq.(3.14) while only parts of X are known.

We now describe the solution algorithm. First, we use A_m to represent the elements of tensor A whose values are missing. Therefore for any tensor A, $A = A_\Omega + A_m$. Our approach is to iteratively compute $Y^0, Y^1, Y^2, \cdots$ by filling up the missing values and solving the standard tensor decomposition

$$\min_{Y^{t+1}} \|(X_\Omega + Y^t_m) - Y^{t+1}\|^2 \tag{3.20}$$

where Y^t is the solution at t-th iteration, and Y^t_m is the missing value part of Y^t. Note that $X_\Omega = X$ is the input tensor with missing values.

We now prove that this iterative algorithm converges:

Theorem 6 *The solution to the optimization of Eq.(3.20) satisfies*

$$\|X - Y^t\|^2_\Omega \geq \|X - Y^{t+1}\|^2_\Omega \tag{3.21}$$

for $t = 0, 1, 2, \cdots$.

Proof. We have

$$\|X - Y^t\|^2_\Omega = \|(X + Y^t_m) - Y^t\|^2 \geq \|(X + Y^t_m) - Y^{t+1}\|^2 \tag{3.22}$$

The first equality is due the definition of $\|\cdot\|_\Omega$. The second inequality is due to the way Y^{t+1} through Eq.(3.20). Furthermore, noting $Y^{t+1} = Y^{t+1}_\Omega + Y^{t+1}_m$ we have

$$\begin{aligned} \|(X + Y^t_m) - Y^{t+1}\|^2 &= \|(X - Y^{t+1})_\Omega + (Y^t - Y^{t+1})_m\|^2 \\ &= \|X - Y^{t+1}\|^2_\Omega + \|Y^t - Y^{t+1}\|^2_m \geq \|X - Y^{t+1}\|^2_\Omega \end{aligned} \tag{3.23}$$

Combining Eq.(3.22) and Eq.(3.23), we obtain Eq.(3.21). This completes the proof.

This theorem guarantees the convergence of the algorithm because the error goes down monotonically and but remains bigger than zero:

$$\|X - Y^0\|^2_\Omega \geq \|X - Y^1\|^2_\Omega \geq \|X - Y^2\|^2_\Omega \geq \cdots \tag{3.24}$$

3.5 Tensor Fold-in Algorithm for New Users

As social networks become popular, each day, thousands of new users are added to the system and the decompositions must be updated daily in an online fashion. In this section, we provide analysis of the new user problem, and present fold-in algorithms for Tucker, ParaFac, and Low-order tensor decompositions proposed in last section.

3.5.1 Overview of Tensor Decomposition Models

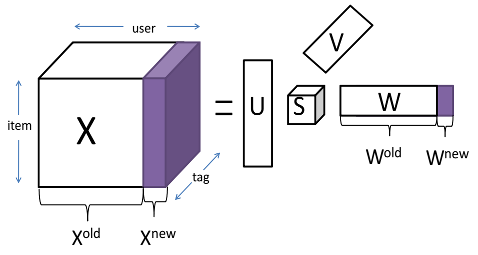
In tensor decomposition, we use Y_{ijk} to reconstruct/approximate the tensor X_{ijk},

$$X_{ijk} \approx Y_{ijk}, \tag{3.25}$$

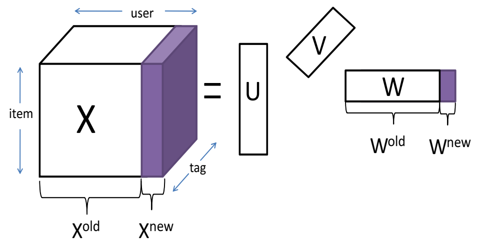
Y is formed using several factors/parameters, and the optimal values of those parameters are obtained by the following optimization,

$$\min_\theta J = \sum_{ijk} (X_{ijk} - Y_{ijk}(\theta))^2 \tag{3.26}$$

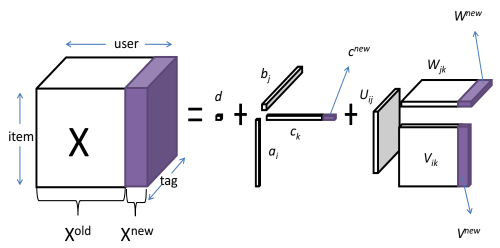
Different forms of Y_{ijk} constitute different models. Here we list three most important and widely used decomposition models in tag recommendation area.



(a) Tucker model



(b) ParaFac model



(c) LOTD model

Figure 3.3: Three tensor models: (a) Tucker Model; (b) ParaFac Model; (c) LOTD Model.

Tucker Model [24]: It can be exemplified in Fig. 3.3a. And the model parameters are $U_{n_i \times P}$, $V_{n_j \times Q}$, $W_{n_k \times R}$, $S_{P \times Q \times R}$. Note that matrix sizes are shown as subscripts. The optimal values of those parameters are obtained by the following optimization,

$$\min J = \sum_{ijk}^{n_i n_j n_k} (X_{ijk} - \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} U_{ip} V_{jq} W_{kr} S_{pqr})^2, \tag{3.27}$$

ParaFac Model [23]: It can be exemplified in Fig. 3.3b. And the model parameters are $U_{n_i \times R}$, $V_{n_j \times R}$, $W_{n_k \times R}$. The optimal values of those parameters are obtained by the following optimization,

$$\min J = \sum_{ijk}^{n_i n_j n_k} (X_{ijk} - \sum_{r=1}^{R} U_{ir} V_{jr} W_{kr})^2, \tag{3.28}$$

Low-order Tensor Decomposition Model: Both Tucker and ParaFac decomposition applied third order interpolation (three factors multiply), while Low-order Tensor Decomposition (LOTD) uses zeroth order, first order and second order interpolation. This low-order scheme has better performance for sparse tensor data, because it has lower chance to overfit the extremely sparse tensor data than higher order schemes. LOTD can be exemplified in Fig. 3.3c. The optimal values of the parameters are obtained by the following optimization,

$$\min J = \sum_{ijk}^{N_i N_j N_k} [X_{ijk} - (d + a_i + b_j + c_k + U_{ij} + V_{ik} + W_{jk})]^2, \tag{3.29}$$

This model has the following closed form solution,

$$\begin{aligned} d = X_{***},\ \ a_i = -X_{i**},\ b_j = -X_{*j*},\ c_k = -X_{**k}, \\ U_{ij} = X_{ij*},\ V_{ik} = X_{i*k},\ W_{jk} = X_{*jk} \end{aligned} \tag{3.30}$$

where the notations such as X_{***}, X_{i**} are explained in Table 3.2.

3.5.2 Fold-in Algorithms

As mentioned above, a large number of new users log on the social tagging systems everyday. To deal with the problem of recommending personalized tags to those new users efficiently, we propose fold-in methods for different tensor decompositions. This paper focuses on fold-in new users into the system.

Tensor factors of X^{old} are computed using models in last section. Based on those factors, we can fold in X^{new} without decomposing $X = (X^{old}, X^{new})$ all over again to get the prediction values of X^{new}.

3.5.2.1 Tucker Decomposition Fold-in Algorithm

For Tucker, the fold-in process is shaded in Fig. 3.3a. The shaded part of X represents new users X^{new}. Among model parameters, (U, V, S) remain unchanged, and the size of W in Eq. 3.27 will change from $n_k \times R$ to $(n_k + l) \times R$, where l is the number of new users. Then we split $W_{(n_k+l)\times R}$ as the following, W^{old} remains unchanged, so our task here is to compute W^{new}.

$$\begin{aligned} W_{(n_k+l)\times R} &= (W^{old}_{n_k\times R}, W^{new}_{l\times R}) \\ W^{old} &= (W_1, W_2, \cdots, W_{n_k}) \\ W^{new} &= (W_{n_k+1}, W_{n_k+2}, \cdots, W_{n_k+l}). \end{aligned} \tag{3.31}$$

Theorem 7 *Substituting $X = (X^{old}, X^{new})$ and $W = (W^{old}, W^{new})$ into Eq. 3.27, while fixing the old parameters, the optimal W^{new} is obtained by the following algorithm.*

(1) Rearrange the input X^{new} as a matrix

$$X^{new} = (\mathit{vec}(X^{n_k+1}), \cdots, \mathit{vec}(X^{n_k+l})),\ (X^k)_{ij} = X_{ijk}, \tag{3.32}$$

(2) Rearrange the computed core-tensor S as a series of matrices

$$S = (S^1, \cdots, S^R),\ (S^r)_{pq} = S_{pqr} \tag{3.33}$$

(3) Compute the following $n_i n_j$-by-R matrix

$$A = (\mathit{vec}(A^1), \cdots, \mathit{vec}(A^R)),\ A^r = US^rV^T, \tag{3.34}$$

(4) W^{new} is given by

$$W^{new} = X^{newT} A (A^T A)^{-1}. \tag{3.35}$$

3.5.2.2 ParaFac Fold-in Algorithm

For ParaFac, the fold-in process is shaded in Fig. 3.3b. The shaded part of X represents new users X^{new}. Among model parameters, (U, V) remain unchanged and W needs to be split as Eq.3.31.

Theorem 8 *Substituting $X = (X^{old}, X^{new})$ and $W = (W^{old}, W^{new})$ into Eq. 3.28, while fixing the old parameters, the optimal W^{new} is obtained by the following algorithm.*

(1) Rearrange input X^{new} as a matrix

$$X^{new} = (\textit{vec}(X^{n_k+1}), \cdots, \textit{vec}(X^{n_k+l})),\ (X^k)_{ij} = X_{ijk}, \tag{3.36}$$

(2) Compute the following $(n_i n_j)$-by-R matrix C, whose element is

$$C_{[ij]r} = U_{ir} V_{jr}, \tag{3.37}$$

(3) W^{new} is given by

$$W^{new} = X^{newT} C (C^T C)^{-1}. \tag{3.38}$$

3.5.2.3 LOTD Fold-in Algorithm

For LOTD, The fold-in process is shaded in Fig. 3.3c. The shaded part of X represents new users X^{new}. Among model parameters, (a, b, d, U) remain unchanged and c, V, W need to be split the same way as W does in Eq.3.31. $c^{old}, V^{old}, W^{old}$ remains unchanged. Our task is to compute $c^{new}, V^{new}, W^{new}$.

Theorem 9 *Substituting $X = (X^{old}, X^{new})$ and $c = (c^{old}, c^{new}), V = (V^{old}, V^{new}), W = (W^{old}, W^{new})$ into Eq. 3.29, while fixing the old parameters, the optimal $c^{new}, V^{new}, W^{new}$ is obtained by the following algorithm.*

(1) Compute a new tensor A as follows,

$$A_{ijk} = X^{new}_{ijk} - (d + a_i + b_j + U_{ij}) \tag{3.39}$$

(2) Compute $c^{new}, V^{new}, W^{new}$ as follows,

$$c_k = -A_{**k},\ V_{ik} = A_{i*k},\ W_{jk} = A_{*jk} \tag{3.40}$$

3.6 Experiment Results

We carried out experiments on several real world datasets to evaluate the performance of our methods.

DataSets: Our experiments use the dataset in table 3.1.

Many recommendation algorithms produce the bad recommendation results on the ”long tail” of users who marks only few items. All of the recommendation methods are based on the statistic method. If the data sample are very less, any method will not receive any good result. Thus, we restrict the evaluation on the ”dense” part of the tag dataset following the traditional approach. For each dataset, we introduce how we prune these dataset.

Last.fm DataSet[25]: Last.fm dataset consists of web pages crawled from the Last.fm web site which is a social media systems that provide the personalized media for its users and also promise the users to add the tags on that media. These dataset is crawled in the first half of 2009. We first choose the active users which mark more than 2400 tags but less than 5000 times on the items. The reason for us to delete the users who marks the tags more than 5000 times is that these users have a great chance to be a fake user. Then, we select the tags which is used by 1000 times but less than 4000 times. In the end, we choose the items which is marked more than 88 times but less than 3000 times.

MovieLens DataSet[26]: MovieLens dataset collect the item-tag-user information from the online movie recommender service MovieLens. We first choose the active users which mark more than 30 tags but less than 600 times on the items. Then, we select the tags which is used by 30 times but less than 1000 times. In the end, we choose the items which is marked more than 25 times but less than 1000 times.

Bibsonomy DataSet[27]: The bibsonomy dataset is download from bibsonomy.org. We first choose the active users which mark more than 1000 tags but less than 2600 times on the items. Then, we select the tags which is used by 420 times but less than 2000 times. In the end, we choose the items which is marked more than 76 times but less than 1000 times.

3.6.1 Evaluation Strategy and Metrics

We use the same evaluation protocol in [20] [22]. From the input tensor, for every user, we randomly mask one post, i.e., this post become missing values. (If there is only one post in this user, we don't move that post.) This forms the tensor on which we run various recommendation algorithms to fill up the masked values. The filled-up values are the predictions.

We measure the prediction quality using traditional Precision-Recall methods in a top-N fashion. For each post(i,k), we sort the predicted values, N_j values for N_j tags. We pick $N = 1, 2, 3, \cdots, 10$ top values and asset that the tags associated with these picked values are predicted as "positive". We assess the predicted tags with the known information which have been masked out. The precision and recall are defined as

$$Precision(Post(i,k)) = \frac{|t \in Top(i,k,N) \cap t \in Post(i,k)|}{N} \tag{3.41}$$

$$Precision(T_{test}, N) = \frac{Precision(Post(i,k))}{|Post|} \tag{3.42}$$

$$Recall(Post(i,k)) = \frac{|t \in Top(i,k,N) \cap t \in Post(i,k)|}{|\cap t \in Post(i,k)|} \tag{3.43}$$

$$Recall(T_{test}, N) = \frac{Recall(Post(i,k))}{|Post|} \tag{3.44}$$

$$F1(T_{test}, N) = \frac{2 \cdot Precision(T_{test}, N) \cdot Recall(T_{test}, N)}{Recall(T_{test}, N) + Recall(T_{test}, N)} \tag{3.45}$$

Clearly, at small N (less tags picked), precision is high and recall is low; At higher N (more tags picked) recall is higher and precision is lower. This forms the well-known ROC curve, as shown in Figures 3.4-3.6.

All the parameters for the different algorithms in these experiments are set according to the original papers. We note that many recommendation algorithm requires initialization of the missing values (masked out values). In our experiments, we use the average value of all 3 dimensions for initialization. This is better than using zero as for initialization.

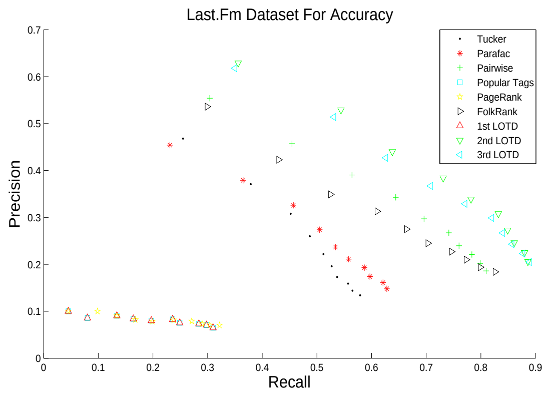


Figure 3.4: The Precision-Recall curve for the Last.fm dataset

3.6.2 Performance of the LOTD Model

We compare the prediction quality of LOTD to the other models. Figure 3.4, 3.5 and 3.6 show the comparison to (1) FolkRank, (2) PageRank, (2) Pairwise decomposition, (4) Tucker Decomposition, (5) Parafac Decomposition, (5) Popular Tags, together with our (6) 1st order LOTD, (7) 2nd order LOTD, (8) 3rd order LOTD methods (3rd order is illustrated in Figure 3.2).

The experiment results indicate : (A) In general, the 2nd order LOTD method has the best precision-recall curves for all datasets. (B) The performances of PageRank,Popular Tags and First Order Tensor decomposition is much lower than the other methods. The reason is these three methods only capture the global information and recommend different users the same tag sets. (C) Tucker and Parafac method are the traditional tensor decomposition methods, but the accuracy of these two methods are lower than the other factorization models.

3.6.3 Performance of the Fold-in Models

Considering the possible adoption of the social tagging prediction in real systems, a new user registers and logins the system and assigns some tags to some items. Based on

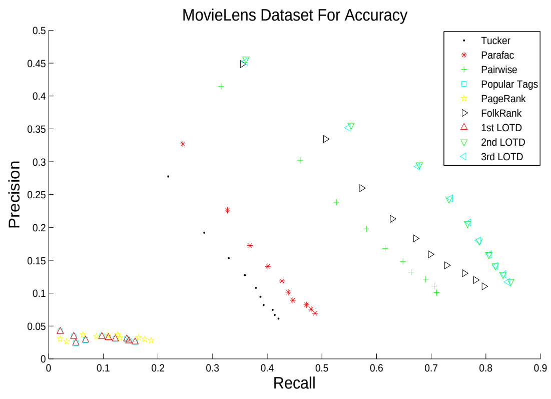


Figure 3.5: The Precision-Recall curve for the MovieLens dataset

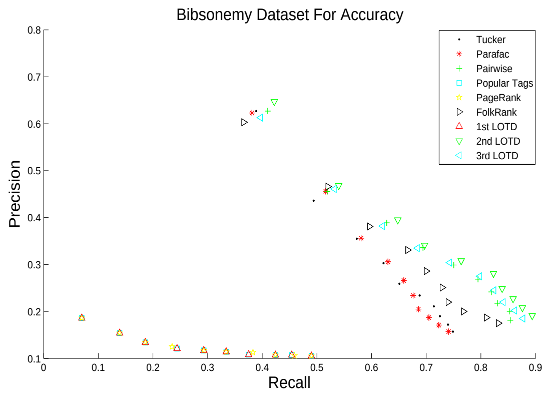


Figure 3.6: The Precision-Recall curve for the bibsonomy dataset

this information, the system makes a prediction/recommendation to this new user, that is, produces a ranked list of tags to this user. There could be multiple predictions occurring at the same time. For evaluation part, we evaluate the quality of the ranked list of tags. For example, if we want to provide user k a ranked list of tags for item i, we first mask the tagging activities regarding to user k towards item i and set them as missing values, then tensor fold-in algorithms predict these missing values. We define a user-item pair as a post.

10-fold cross-validation is adopted in our all experiments. For each dataset listed above, we randomly partition the input tensor into 10 parts. Each part is retained as the testing data X^{new} (fold-in tensor) for once. For every user in X^{new}, we randomly mask one post, i.e., this post becomes missing values. This forms the tensor defined as X^{new}_{masked} on which we run various fold-in algorithms to fill up the masked values. All the rest parts of input tensor constitute a training tensor, which is X^{old}. For each time, we use tensor factors of X^{old} and X^{new}_{masked} to predict the missing values in X^{new}_{masked}; the average of precision and recall results is the final prediction result.

For each fold-in prediction, we use traditional Precision-Recall methods in a top-N fashion. For each post(i, k), we sort the predicted values. We pick $N = 1, 2, 3, \cdots, 10$ top values and return the corresponding tags associated with these picked values.

Last.fm DataSet[25]: Last.fm is a social media system that provides the personalized media for its users and also promises the users to add the tags on that media.

Subset A: Subset A of Last.fm dataset has 203 items, 241 tags, 425 users and the Relative NNZ (density) is 0.11%.

Subset B: Subset B has 100 items, 157 tags, 280 users and the Relative NNZ is 0.27%.

MovieLens DataSet[26]: MovieLens dataset collects the item-tag-user information from the online movie recommender service MovieLens.

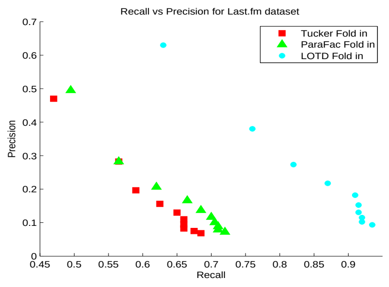
Subset A: Subset A of MovieLens dataset has 345 items, 369 tags, 465 users and the Relative NNZ is 0.0175%.

Subset B: Subset B has 98 items, 199 tags, 263 users and the Relative NNZ is 0.078%.

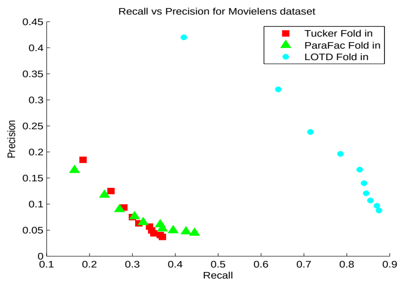
Bibsonomy DataSet[27]: The bibsonomy dataset is downloaded from bibsonomy.org.

Subset A: Subset A of Bibsonomy dataset has 362 items, 116 tags, 361 users and the Relative NNZ is 0.065%.

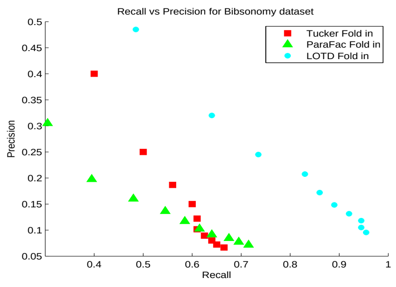
Subset B: Subset B has 117 items, 101 tags, 204 users and the Relative NNZ is 0.24%.



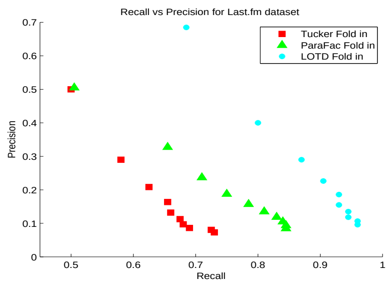
(a) Last.fm data : subset A



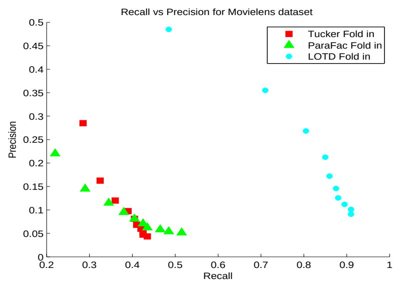
(b) MovieLens data: subset A



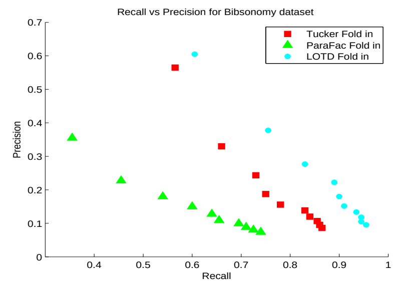
(c) Bibsonomy data: subset A



(d) Last.fm data : subset B



(e) MovieLens data: subset B



(f) Bibsonomy data: subset B

Figure 3.7: Precision-Recall curves for three tensor Fold-in methods on three datasets: from left to right: Last.fm data, MovieLens data, Bibsonomy data. The performance gap between LOTD fold-in and ParaFac/Tucker fold-in methods increases from the smaller subset B to larger subset A.

We compare the prediction qualities of the three fold-in algorithms. Figure 3.7a, 3.7b and 3.7c show the comparison between (1) Tucker Fold-in, (2) ParaFac Fold-in, (3) LOTD Fold-in on subset A of each dataset, and Figure 3.7d, 3.7e and 3.7f show the comparison on subset B of each subset.

The experiment results indicate: (A) LOTD Fold-in method has the best precision-recall curves for all datasets, because of the sparsity of real world dataset. (B) Tucker and ParaFac fold-in methods are based on the traditional tensor decomposition models, and they overfit these sparse tensor data. The results are consistent with the results in last section, in which, the original LOTD model gains better performance than the original Tucker and ParaFac decompositions. (C) For each dataset, we notice that the difference between LOTD fold-in and Tucker/ParaFac fold-in methods on subset A is much bigger than that of subset

B, which means that LOTD fold-in can gain relatively much better performance than the other two fold-in methods on sparser dataset. It is illustrated again that LOTD fold-in method is targeted at sparse data.

3.6.3.1 Efficiency of the Proposed Fold-in Algorithms

Table 3.3: Time for folding in one new user vs. original algorithm

Alg.	Time for one new user(sec)
Tucker	33
ParaFac	16000
LOTD	21
Tucker Fold-in	0.72
ParaFac Fold-in	0.0024
LOTD Fold-in	0.00077

Table 3.3 gives the runtime of the fold-in methods and original methods on Last.fm dataset. As the original algorithms, the same convergence tolerance factor $\varepsilon = 0.001$ is used. We can see that the consuming time for the fold-in methods is much less than the corresponding original algorithm, because we don't need to calculate the whole tensor decomposition again. We can see that LOTD Fold-in method is much faster than the traditional Tucker and ParaFac Fold-in methods.

3.7 Conclusion

In this chapter, we present a systematic study of low-order tensor decomposition approach that are specifically targeted at the very sparse data problem in tagging recommendation problem. We demonstrate that low-order polynomials are uniquely capable of enhancing statistics and avoids overfitting than traditional tensor decompositions such as Tucker and Parafac decompositions. We performed extensive experiments on several datasets and compared with six existing methods. Experimental results demonstrate that our approach outperform existing approaches.

We also propose three tensor fold-in techniques to deal with new user problem. The tensor factors coming from X^{old} carry the historic information and also the similar users activities. While factors from X^{new} indicate the new users' personalized preference and features those can help to detect their potential activities. When l gets bigger, social tagging system can combine current X^{new} into X^{old}, and then reproduce the decomposed factors. With new decomposed factors, which carry more latest information and trends, the system can do online recommendations again. The fold-in techniques proposed in this paper have fast online performance, requiring just a few simple matrix operations for new users. Meanwhile, the experiment results demonstrate that the fold-in methods can provide comparable prediction quality. Especially, LOTD fold-in method based on Low-order Tensor Decomposition model is specifically targeted at the sparsity challenge in tag recommendation systems, because low-order polynomials can enhance the statistics of the sparse tagging datasets. Therefore, it can gain better predicting accuracy than the other two fold-in methods, which are based on two traditional tensor decomposition models. The traditional tensor methods (Tucker and ParaFac) obviously overfit the sparse tensor decompositions. The fold-in methods can help social tagging recommendation systems achieve high scalability while providing good predictive accuracy.

CHAPTER 4

Robust Tucker Tensor Decomposition For Effective Image Representation

4.1 Introduction

The development of online social media provides tons of images and videos everyday, which makes image or video storage and denoising problems two important and urgent research topics.

In a typical image storage problem, an image is represented as a 1-d long feature vector, and then this long vector denotes one data point in a high dimensional space. But as we all know, an image can be naturally represented as a 2-d matrix, with each element denoting the feature value on that specific spot. The 1-d vector denotation of an image makes it convenient for subspace learning, such as principal component analysis (PCA)[28] and linear discriminant analysis (LDA)[29] used in face recognition area.

Recently, some of other subspace learning algorithms applied on 1-d vector data are studied, such as locality preserving projection (LPP)[30] and localized linear models (LLM)[31], which are proven to be efficient. However, the 1-d vector denotation strategy as a whole ignores the neighborhood feature information within one image, while 2-d matrix denotation retains the important spatial relationship between features within one image.

Therefore, a lot of tensor decomposition techniques are studied in computer vision applications. For example, Shashua and Levine [32] adopted rank-one decomposition to represent images, which was described in detail in [33]. Yang et al. [34] introduced a two dimensional PCA (2DPCA), in which, one-side low-rank approximation was applied. Generalized Low Rank Approximation of Matrices (GLRAM) was proposed by Ye et al. [35], and the method projected the original images onto one two dimensional space. Ding and Ye proposed a two dimensional singular value decomposition (2DSVD) [36], which computes principal eigenvectors of row-row and column-column covariance matrices. Other tensor decomposition methods are also proposed and some of them are proven to be equivalent to 2DSVD and GLRAM in [37]. High order singular value decomposition (HOSVD) [38] were proposed for higher dimensional tensor by Vasilescu and Terzopoulos [39].

In above tensor analysis algorithms, an image is denoted by a 2-d matrix or second order tensor as itself, which retains the neighborhood information within the image itself, and then a set of images can be denoted by a third-order tensor. They minimize the sum of squared errors, which is known as frobenious norm, in which large errors due to outliers and feature noises such as occlusion, after being squared, dominate the error function and force the low rank approximation to concentrate on these few data points and features, while nearly ignoring most of other data points.

Over the years, there are many different approaches proposed to solve this problem both on 1-d vector data and 2-d matrix data. [40] [41] [42] [43] [44] [45] [46] [47] [48]. The approach using pure L_1-norm is used widely because it offers an simple and elegant formulation [43] [44] [45] [47] to suppress the impact coming from noisy data or features.

A difficulty of pure L_1-based methods is that the optimization tends to be hard. Several computational methods have been proposed [43] [44] [45] [47]. These methods are either complicated or difficult to scale to large problems.

In this chapter, we propose a robust Tucker tensor decomposition (RTD) model to deal with images occluded by noisy information, and also propose a simple yet computationally efficient algorithm to solve the L_1-norm based Tucker tensor decomposition optimization. This method also provides some insights to the optimization problem such as the Lagrangian multiplier and KKT condition. We also carry out extensive experiments in face recognition, and verify the robustness of the proposed method to image occlusions. Both numerical and visual results demonstrate the effectiveness of our proposed method.

4.2 Robust Tucker Tensor Decomposition (RTD)

Standard Tucker tensor decomposition [38] uses reconstructed tensor Y to approximate the original tensor X,

$$Y_{ijk} = \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} U_{ip}V_{jq}W_{kr}S_{pqr} \tag{4.1}$$

where Y is a third order tensor, $Y \in \Re^{n_i \times n_j \times n_k}$, $U \in \Re^{n_i \times P}$, $V \in \Re^{n_j \times Q}$, $W \in \Re^{n_k \times R}$, $S \in \Re^{P \times Q \times R}$ is a core tensor, which couples different 3rd order multi-linear polynomials. Therefore, mathematically, Y can be expressed as the following (Eq.(4.2)), which simplifies the tensor constructing expressions in next sections.

$$Y = U \otimes_1 V \otimes_2 W \otimes_3 S \tag{4.2}$$

Tucker tensor decomposition has the following cost function [33],

$$\begin{aligned} &\min_{U,V,W,S} \|X - Y\|_F^2 = \sum_{i=1}^{n_i}\sum_{j=1}^{n_j}\sum_{k=1}^{n_k}\Big(X_{ijk} - Y_{ijk}\Big)^2 \\ &s.t. \quad U^T U = I, V^T V = I, W^T W = I \end{aligned} \tag{4.3}$$

It is well-known that the solution to the above optimization is given by high order singular value decomposition (HOSVD) [38], which will be introduced in the algorithm part. As we can see, the standard Tucker tensor decomposition uses Frobenius norm to decompose the original tensor. Frobenius norm is known for being sensitive to outliers and feature noises, because it sums the squared errors. While, L_1-norm just sums the absolute value of error, which reduces the influence of the outliers comparing to the Frobenius norm. So the more robust against outlier version of Tucker tensor decomposition is formulated using L_1-norm. L_1-norm of a third order tensor A with size $n_i \times n_j \times n_k$ is defined as $\|A\|_1 = \sum_{i=1}^{n_i}\sum_{j=1}^{n_j}\sum_{k=1}^{n_k}|a_{ijk}|$. Therefore, the robust Tucker tensor decomposition (RTD) is formulated as,

$$\begin{aligned} \min_{U,V,W,S}\|X-Y\|_1 &= \sum_{i=1}^{n_i}\sum_{j=1}^{n_j}\sum_{k=1}^{n_k}\Big|X_{ijk}-Y_{ijk}\Big| \\ s.t. \quad U^TU = I, V^TV &= I, W^TW = I \end{aligned} \tag{4.4}$$

Illustration: before going any further, we want to give a glance at the denoising effect by RTD first. Figure 4.1 and Figure 4.2 illustrate the reconstructed effect on AT&T data set, with existence of two different occlusion strategies, which will be explained in details in the experiment part. In both figures, images of the second row represent the reconstructed images by RTD and those of the fourth row represent images reconstructed by Tucker tensor decomposition. In both noise and corruption cases, Robust Tucker decomposition gives clearly better reconstruction.

4.3 Efficient Algorithm for Robust Tucker Tensor Decomposition

The standard Tucker decomposition can be efficiently solved using the HOSVD algorithm [38]. In this chapter, we propose an efficient algorithm to solve robust Tucker tensor decomposition. We employ the Augmented Lagrange Multiplier (ALM) method [49] to solve this problem. ALM has been successfully used in other L_1 related problems [50].

One important finding is that ALM is extremely well suited to this RTD model. The algorithm iteratively solves two sub-problems: One is a simplified LASSO (see Eq.(4.7)) with simple exact solution; Another is a standard Tucker tensor decomposition of Eq.(4.3). This enables us to utilize existing software to efficiently solve the RTD.

Outline of the algorithm: we first rewrite the objective function of robust Tucker tensor decomposition equivalently as

$$\begin{aligned} &\min_{U,V,W,S,E} \|E\|_1 \\ &s.t. \quad E = X - U \otimes_1 V \otimes_2 W \otimes_3 S \\ &\qquad U^T U = I, V^T V = I, W^T W = I \end{aligned} \tag{4.5}$$

Now we use ALM approach by enforcing the equality constraint $E = X - U \otimes_1 V \otimes_2 W \otimes_3 S$ using Lagrange multipliers (matrix Λ) and quadratic penalty. Then ALM becomes to solve the following problem,

$$\begin{aligned} \min_{E,U,V,W,S} &\|E\|_1 + \langle \Lambda, X - U \otimes_1 V \otimes_2 W \otimes_3 S - E \rangle \\ &+ \frac{\mu}{2} \|X - U \otimes_1 V \otimes_2 W \otimes_3 S - E\|_F^2 \\ s.t. \quad &U^T U = I, V^T V = I, W^T W = I \end{aligned} \tag{4.6}$$

where scalar μ is the penalty parameter, $\langle P, Q \rangle$ is defined as $\sum_{ijk} P_{ijk} Q_{ijk}$.

The ALM is an iteratively updating algorithm. There are two major parts, solving the sub-problems and updating parameters, which will be presented in the following sections.

4.3.1 Solving the Sub-optimization Problems

The key step of the algorithm is solving the two sub-programs of Eq.(4.6) for each set of parameter values of Λ, μ. Fortunately, this can be solved in closed form solutions for E and group of (U, V, W, S).

A: solve for E. First, we solve E while fixing U, V, W and S. From Eq.(4.6), the objective function becomes

$$\min_E \|E\|_1 + \frac{\mu}{2} \|E - P\|_F^2 \tag{4.7}$$

where P is a constant matrix independent of E:

$$P = X - U \otimes_1 V \otimes_2 W \otimes_3 S + \frac{\Lambda}{\mu}. \tag{4.8}$$

This problem has closed form solution

$$E^*_{ijk} = sign(P_{ijk}) \max(|P_{ijk}| - 1/\mu, 0). \tag{4.9}$$

B: solve for (U, V, W, S). In this step, we solve U, V, W and S together while fixing E. From Eq.(4.6), the objective function becomes

$$\begin{aligned} &\min_{U,V,W,S} \frac{\mu}{2}||Q - U \otimes_1 V \otimes_2 W \otimes_3 S||_F^2, \\ &s.t. \quad U^TU = I, V^TV = I, W^TW = I \end{aligned} \tag{4.10}$$

where

$$Q = X - E + \frac{A}{\mu}; \tag{4.11}$$

This is exactly the usual Tucker tensor decomposition. This is solved by the known HOSVD algorithm [38]. HOSVD is an iterative algorithm. Given initial guess of U, V, W we update U, V, W until convergence.

U is given by the P eigenvectors with largest eigenvalues of F, where

$$F_{ii'} = \sum_{jj'kk'} Q_{ijk} Q_{i'j'k'} (VV^T)_{jj'} (WW^T)_{kk'} \tag{4.12}$$

V is given by the Q eigenvectors with largest eigenvalues of G, where

$$G_{jj'} = \sum_{ii'kk'} Q_{ijk} Q_{i'j'k'} (UU^T)_{ii'} (WW^T)_{kk'} \tag{4.13}$$

W is given by the R eigenvectors with largest eigenvalues of H, where

$$H_{kk'} = \sum_{jj'ii'} Q_{ijk} Q_{i'j'k'} (VV^T)_{jj'} (UU^T)_{ii'}. \tag{4.14}$$

These steps are repeated until convergence. After (U^*, V^*, W^*) are obtained, S is given by

$$S_{pqr} = \sum_{ijk} Q_{ijk} U_{ip} V_{jq} W_{kr}. \tag{4.15}$$

4.3.2 Updating Parameters

In each iteration of ALM, after obtaining consistent E and (U, V, W, S), the parameters Λ and μ are updated as the following

$$\Lambda \Leftarrow \Lambda + \mu(X - U \otimes_1 V \otimes_2 W \otimes_3 S - E) \tag{4.16}$$

$$\mu \Leftarrow \mu\rho \tag{4.17}$$

where $\rho > 1$ is a constant.

The complete algorithm is described in Algorithm 1.

```
Input: $X$, $P$, $Q$, $R$
Output: $U, V, W, S$
Initialize $\mu = 1/||X||_F, \rho = 1.01, U_0, V_0, W_0$
repeat
    Compute $E$ using Eq.(4.9)
    Compute $U, V, W, S$ using Eq.(4.12 - 4.15)
    $\Lambda = \Lambda + \mu(X - U \otimes_1 V \otimes_2 W \otimes_3 S - E)$
    $\mu = \min(\mu\rho, 10^{10})$
until Converge
```

Algorithm 7: RTD Algorithm

We initialize (U, V, W) either by random or by the solution to the standard Tucker decomposition. In all these cases the ALM algorithm did converge. The converged solutions from different initialization are very close to each other[51], and there are no visible differences in the reconstructed images.

Convergence Analysis: by taking derivative of the Lagrangian function w.r.t. E, we obtain the Karush-Kuhn-Tucker (KKT) condition,

$$\Lambda_{ijk} = \begin{cases} sign(E_{ijk}) & \text{if } E_{ijk} \neq 0 \\ \partial |E_{ijk}| & \text{if } E_{ijk} = 0 \end{cases} \tag{4.18}$$

where $\partial|E_{ijk}| \in [-1, 1]$ is the subgradient of function $f(x) = |x|$.

here we view Λ_{ijk} as Lagrangian multipliers. We now verify the KKT condition of our algorithm. The following are examples from AT&T dataset, whose tensor size is 56x46x400. More detailed dataset information will be introduced in the experiment part.

At convergence, the first 25 elements of computed E_{ijk} are,

$$E = \begin{pmatrix} 0.0012 & -0.0003 & 0.0000 & -0.0005 & 0 \\ 0.0005 & -0.0005 & 0 & -0.0007 & -0.0011 \\ -0.0001 & 0 & -0.0005 & -0.0008 & 0 \\ -0.0002 & 0 & -0.0015 & -0.0001 & 0 \\ 0 & 0 & -0.0012 & 0 & 0.0001 \end{pmatrix}$$

The corresponding 25 elements Λ_{ijk} are

$$\Lambda = \begin{pmatrix} 1.0000 & -1.0000 & 1.0000 & -1.0000 & 0.2806 \\ 1.0000 & -1.0000 & -0.8213 & -1.0000 & -1.0000 \\ -1.0000 & -0.5164 & -1.0000 & -1.0000 & 0.3976 \\ -1.0000 & -0.2643 & -1.0000 & -1.0000 & -0.4540 \\ 0.0630 & 0.1762 & -1.0000 & 0.3274 & 1.0000 \end{pmatrix}$$

We see that the above KKT condition are satisfied for every elements. When E_{ijk} is nonzero, Λ_{ijk} is its sign. When E_{ijk} is zero, Λ_{ijk} is its subgradient (a value in $[-1, 1]$).

4.4 Efficient Algorithm for L_1-PCA

In standard computer vision problems, each image is converted to a vector and a set of images is represented by a matrix. Here PCA is mostly wide used. The advantage of tensor approach is that each image retains its 2D form in tensor representation and thus tensor analysis retains more information on image collections.

We need to compare the tensor approaches with matrix approaches. Thus we implement the algorithm for computing L_1PCA. L_1PCA is formulated as the following

$$\min_{U,V} \; \|X - UV\|_1 = \sum_{j=1}^{n} \sum_{i=1}^{p} |(X - UV)_{ij}|, \tag{4.19}$$

where $X = (x_1, \cdots, x_n)$ contain n images. $X \in \Re^{p \times n}$ where $p = rc$ for r-by-c images. The factor matrices U, V have sizes of $U \in \Re^{p \times k}, \; V \in \Re^{k \times n}$.

Similarly with solving RTD, Eq.(4.19) can be rewritten equivalently as

$$\min_{E,U,V} \; \|E\|_1, \; s.t. \; E = X - UV, \tag{4.20}$$

ALM solves a sequence of sub-problems

$$\min_{E,U,V} \; \|E\|_1 + \langle A, X - UV - E \rangle + \frac{\mu}{2} ||X - UV - E||_F^2 \tag{4.21}$$

where matrix A is the Lagrange multipliers.

A: solve for E. First, we solve E while fixing U and V. From Eq.(4.21), the objective function becomes

$$\min_{E} \; \|E\|_1 + \frac{\mu}{2} ||E - (X - UV + \frac{A}{\mu})||_F^2 \tag{4.22}$$

This problem has closed form solution:

$$E_{ij}^* = sign(P_{ij})(|P_{ij}| - 1/\mu)_+, \; P = X - UV + \frac{A}{\mu}. \tag{4.23}$$

B: solve for U, V. Next we solve U and V together while fixing E. From Eq.(4.21), the objective function becomes

$$\min_{U,V} \langle A, X - UV - E \rangle + \frac{\mu}{2} ||X - UV - E||_F^2. \tag{4.24}$$

Which is is equivalent to

$$\min_{U,V} \; \frac{\mu}{2} ||Q - UV||_F^2, \;\; Q = X - E + \frac{A}{\mu}; \tag{4.25}$$

The solution is given by standard PCA. Denote the singular value decomposition (SVD) of Q as

$$Q = F\Sigma G^T \tag{4.26}$$

Only first k largest singular values and associated singular vectors are needed. Then the solution of U, V are given by

$$\begin{aligned} U &= F_k, \\ V &= \Sigma_k G_k^T \end{aligned} \tag{4.27}$$

In each iteration of ALM, after obtaining consistent E and (U, V), the parameters A and μ are updated as the following

$$A \Leftarrow A + \mu(X - UV - E) \tag{4.28}$$

$$\mu \Leftarrow \mu\rho \tag{4.29}$$

where $\rho > 1$ is a constant.

4.5 Experiments

In this section, three benchmark face databases AT&T, YALE and CMU PIE are used to evaluate the effectiveness of our proposed RTD tensor factorization approach.

4.5.1 Data Description

The properties of the three data sets we used are summarized in Table 4.1, and the detailed information of each data set is given as the following.

AT&T: The AT&T face data contains 400 upright face images of 40 individuals, collected by AT&T Laboratories Cambridge. Each image is resized to 56x46 pixels in this experiment.

YALE: There are totally 38 classes (10 subjects in original database with 28 subjects in the extended database) under 576 viewing conditions (9 poses with 64 different illu-

Table 4.1: Description of data sets

Data set	#images n_k	#Dimensions $n_i \times n_j$	#Class K
AT&T	400	56×46	40
YALE	1984	48×42	31
CMU PIE	680	32×32	68

Table 4.2: Performance comparison (storage, noise-free error and classification accuracy) on AT&T data with block occlusion

Methods	Storage	Noise-free Error	Class ACC
Corrupted X	1,030,400	4.7269×10^4	0.6050
RTD	19,672	3.0457×10^4	0.7125
L_1PCA	119,040	3.1435×10^4	0.7025
Standard Tensor	19,672	3.3834×10^4	0.6775
Standard PCA	119,040	3.4959×10^4	0.6675

mination conditions). 64 images in different illumination conditions from 31 classes are selected for our experiment, so there are totally 1984 images.

CMU PIE: CMU PIE is a face database of 41,368 images of 68 people, collected by Carnegie Mellon Robotics Institute between October and December 2000. Each image is resized into 32x32 pixels in our experiment. We randomly select 10 images from each class with different combinations of pose, face expression and illumination condition.

4.5.2 Corrupted Images

For evaluation purpose, we generate occluded images from the above three image data sets. One added advantage of this approach is that we can compare the reconstructed images with the original uncorrupt images to assess the effectiveness of removing the corruption (occlusion).

We use two type of occlusions added to the original input images to evaluate the effectiveness of proposed RTD tensor method against outliers. First, square block occlusions with different size are added. The occlusion is generated as the following, given the size

Table 4.3: Performance comparison(storage, noise-free error and classification accuracy) on Yale data with block occlusion

Methods	Storage	Noise-free Error	Class ACC
Corrupted X	3,999,744	6.9070×10^4	0.3766
RTD	64,204	4.3685×10^4	0.3896
L_1PCA	124,000	4.6886×10^4	0.3311
Standard Tensor	64,204	4.8164×10^4	0.3831
Standard PCA	124,000	5.0806×10^4	0.2989

Table 4.4: Performance comparison(storage, noise-free error and classification accuracy) on CMU PIE data with block occlusion

Methods	Storage	Noise-free Error	Class ACC
Corrupted X	696,320	2.4501×10^4	0.4735
RTD	47,840	0.8578×10^4	0.5294
L_1PCA	115,872	1.0388×10^4	0.5279
Standard Tensor	47,840	1.7610×10^4	0.4926
Standard PCA	115,872	1.8419×10^4	0.4882

of occlusion d, we randomly pick up the $d \times d$ block position for each image, and we set pixels in this $d \times d$ area to zero. There are some examples of occluded images using this method in Figure 4.1.

Second, mixed occlusions with 3 different corrupting methods are added to the original images. First corruption methods are called cross occlusions, and the cross has specified length l and width w. For each class, we randomly select m images to add cross occlusions. We also randomly select the position of the cross, and set the pixels in the cross to the average pixel value of the whole data set. To make the occlusions realistic and diversified, for each class, on the basis of cross occlusions, we randomly select m images to add square block occlusions introduced above. In the end, rectangular occlusions are added. Similarly, for each class, we randomly select m images to add rectangular occlusions. We randomly set the sizes of each rectangle within a permitted range $[a, b]$, and within each rectangle,

Table 4.5: Performance comparison(storage, noise-free error and classification accuracy) on AT&T data with mixed occlusion

Methods	Storage	Noise-free Error	Class ACC
Corrupted X	1,030,400	2.9635×10^4	0.8725
RTD	19,672	1.8536×10^4	0.9450
L_1PCA	119,040	1.9924×10^4	0.9325
Standard Tensor	19,672	2.4942×10^4	0.8875
Standard PCA	119,040	2.5723×10^4	0.8800

Table 4.6: Performance comparison(storage, noise-free error and classification accuracy) on Yale data with mixed occlusion

Methods	Storage	Noise-free Error	Class ACC
Corrupted X	3,999,744	4.5618×10^4	0.3725
RTD	64,204	3.3482×10^4	0.4134
L_1PCA	124,000	3.6471×10^4	0.3916
Standard Tensor	64,204	4.1843×10^4	0.3678
Standard PCA	124,000	4.0981×10^4	0.3714

some of the pixels are set to 0, and the rest are set to 1. The first row in Figure 4.2 demonstrates this mixed occlusion method.

Figure 4.1 and Figure 4.2 only show 1 person of 400 people in AT&T data set due to space limitation. For AT&T data set, an 8×8 occlusion is added to every image of each class in the first type of occlusion. For the second type of occlusion, within each class of images, we first randomly select $m = 2$ images to add the cross, and for each selected image the length of cross is $l = 22$ and width is $w = 3$. Second, we randomly select $m = 2$ images to add the square block. Third, we randomly select $m = 2$ images to add the rectangle, and for each added rectangle, the sizes are random within a ranger of $[a, b]$ = $[4, 10]$. Similarly, for Yaleb data set, $d = 8$ and $l = 20$, $w = 3$, $m = 12$, $[a, b]$ = $[4, 10]$. For CMU PIE data set, we set $d = 6$ and $l = 15$, $w = 3$, $m = 3$, $[a, b]$ = $[3, 10]$.

Table 4.7: Performance comparison(storage, noise-free error and classification accuracy) on CMU PIE data with mixed occlusion

Methods	Storage	Noise-free Error	Class ACC
Corrupted X	696,320	2.4532×10^4	0.4562
RTD	47,840	1.7856×10^4	0.5332
L_1PCA	115,872	1.8442×10^4	0.5106
Standard Tensor	47,840	2.1427×10^4	0.4762
Standard PCA	115,872	2.1019×10^4	0.4632

4.5.3 Experiment Results

In this section, we compare the performance of our RTD method with standard Tucker tensor method, L_1-norm PCA method (L_1PCA) and standard PCA method at storage space, the noise reduction effect and classification accuracy.

One of the biggest advantage of our proposed RTD method is to save image storage space, because for Tucker tensor decomposition methods, to reconstruct the images, we only need to store U, V and W, the core tensor S can be calculated using U, V, W. The sizes of U, V, W are $n_i \times P$, $n_j \times Q$, $n_k \times R$, respectively. So the storage space for our L_1-norm tensor are

$$n_i \times P + n_j \times Q + n_k \times R$$

While for PCA based methods, U and V need to be stored, and the sizes of U and V are $p \times k$ and $k \times n$ respectively, and here $p = n_i \times n_j$ and $n = n_k$. So the storage space for PCA based methods would be

$$n_i \times n_j \times k + k \times n_k$$

The parameters we used in our experiment for each data set is given in Table 4.8. Accordingly, the needed storage space for each method on every data sets can be calculated, which are given in Table 4.2, 4.3, 4.4. Noise-free Reconstruction Error: let X be the original images and O be the occlusion. Then $X + O$ are the input data to tensor decompositions

Table 4.8: Parameters of different data sets

Data set	$P \times Q \times R$	k
AT&T	$36 \times 36 \times 40$	40
YALE	$30 \times 30 \times 31$	31
CMU PIE	$25 \times 25 \times 68$	68

and PCA. Let Y be the reconstructed images from Eq.(4.1). All tensor analysis and PCA minimize $\|(X + O) - Y\|_F$. However, our goal is to recover the *true, noise-free* images. For occluded data, we take the original images as the approximation of the true noise-free images, and consider $\|X - Y\|_F$ as a measure of the ability to recover the noise-free images. We thus call $\|X - Y\|_F$ as the noise-free reconstruction error. It can be computed for PCA and tensor decompositions.

The noise-free error for each method is listed in Table 4.2, 4.3, 4.4 for the first type of occlusion and Table 4.5, 4.6, 4.7 for the second type of occlusion. We can see (1) the noise-free errors for RTD and L_1PCA are *always* smaller than those for Tucker decomposition and PCA; This shows the effectiveness of L_1 norm for removing corruptions. (2) Noise-free errors for RTD are always smaller than those for L_1PCA; This demonstrates the advantage of Tensor decomposition approach.

A byproduct of image denoising is improved classification accuracy. Here we perform classification as the demonstration and evaluation of denoising effectiveness of the proposed RTD. We use k nearest neighbor (kNN) (we use 1NN here) as the multi-class classifier. Classification accuracy on occluded image data are listed in Table 4.2, 4.3, 4.4 for the first type of occlusion and Table 4.5, 4.6, 4.7 for the second type of occlusion. All classification results are based on 2-fold cross-validation. For each class, we randomly split the images into 2 parts, and then we set each of the two parts as training set and the rest part as testing set. The reported accuracy is the average of 100 times of cross validations.

4.5.4 Reconstruction Images and Discussion

Figure 4.1 and Figure 4.2 demonstrate the sample occluded images and the corresponding reconstructed images from different methods. As we can see, the reconstructed images from our RTD method reduce the occlusion more successfully than other methods, which is also shown by the noise-free error in Table 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, the noise-free error of our methods are smaller than other methods. Our method needs far less storage space than PCA based methods, for example, the storage for PCA based method is 119,040 for AT&T data set, while for our RTD method, the storage is only 19,672, that is to say, PCA based methods need 6 times bigger storage than tensor methods do on AT&T data set. Classification accuracies on the reconstructed images from RTD method are higher in most cases, which demonstrated the effectiveness our method.

4.6 Conclusion

In this chapter, we propose an L_1-norm based robust Tucker tensor decomposition (RTD) method, which is effective for correcting corrupted images. Our method requires far less storage space than PCA based methods. We also propose a computationally efficient algorithm to solve the proposed RTD model. Extensive experiments are carried out to evaluate the proposed RTD. Both numerical and visual results are consistently better for images with outliers or noisy features than standard PCA, L_1PCA and standard Tucker tensor decomposition methods. This validates the effectiveness of the proposed RTD.

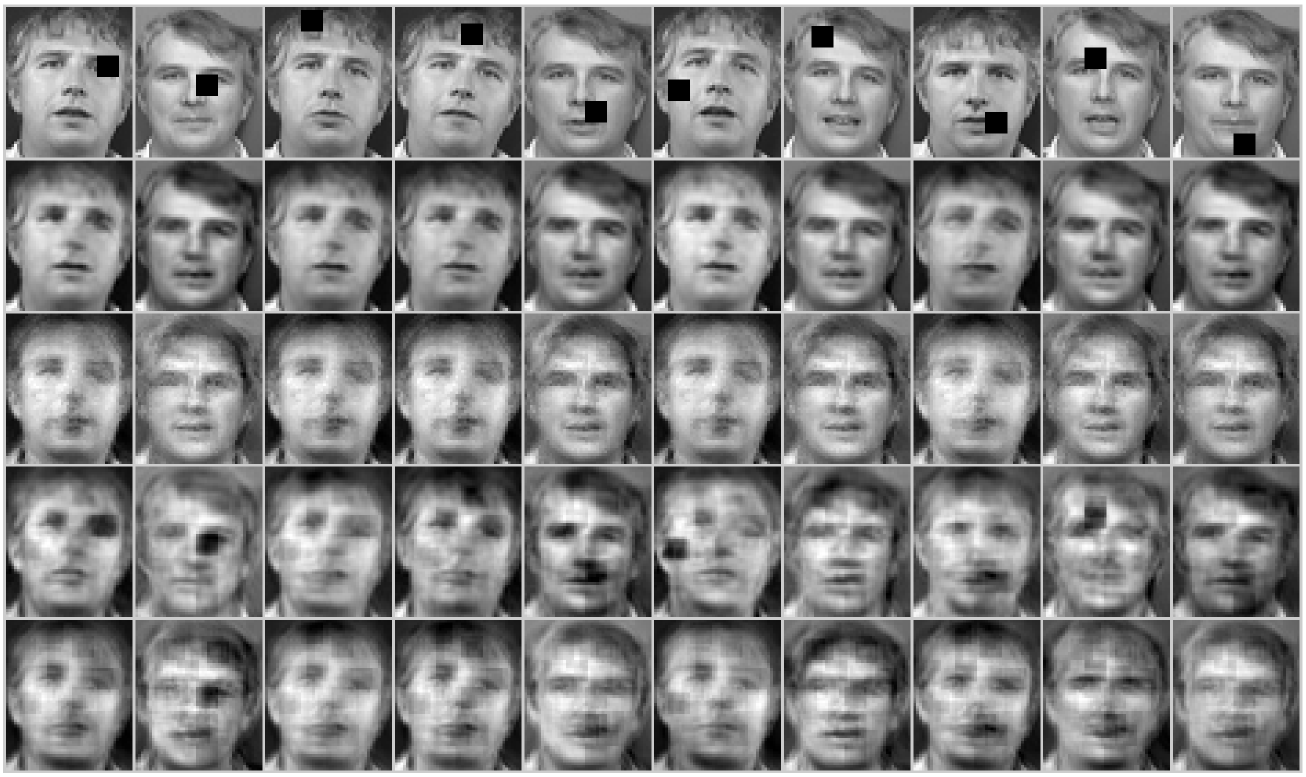


Figure 4.1: Samples of occluded images and reconstructed images on AT&T face data. First row is the input occluded images; Second row is from RTD; Third row is from L_1PCA; Fourth row is from Tucker decomposition; Fifth row is from PCA.

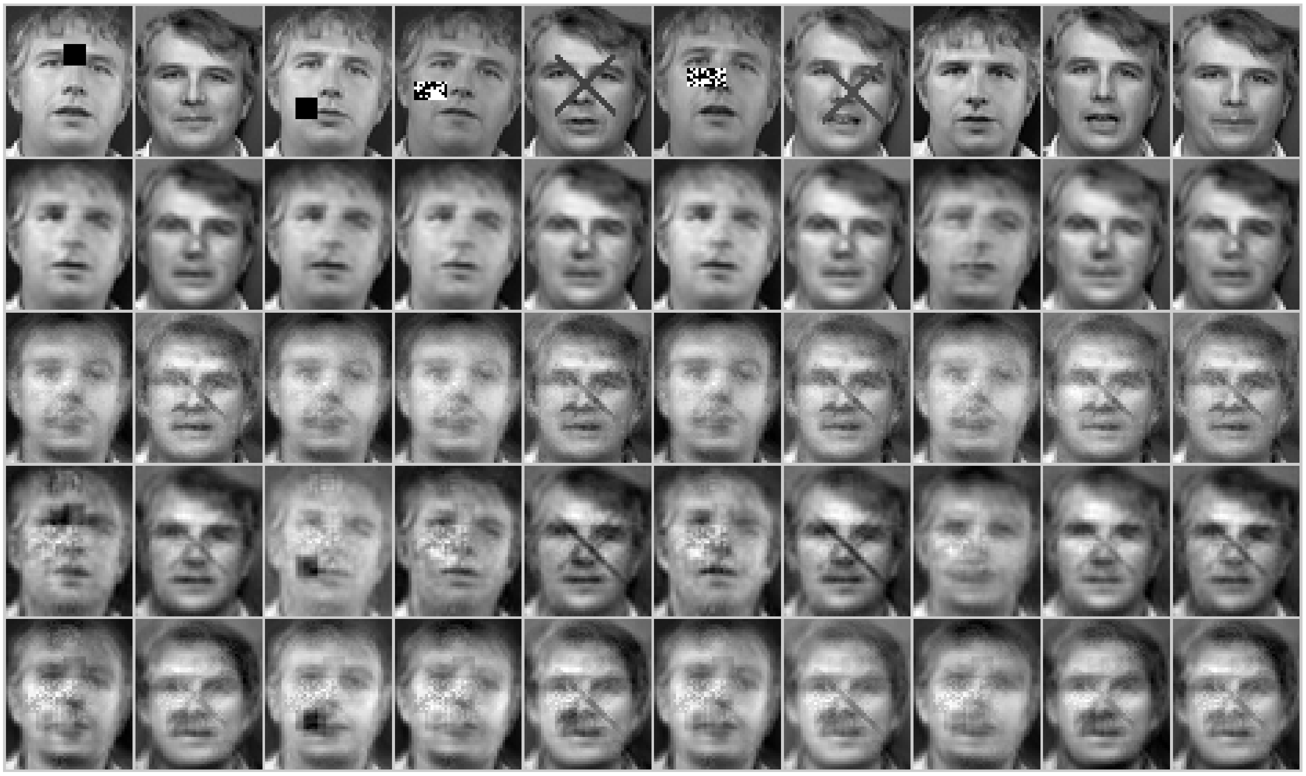


Figure 4.2: Samples of type 2 (mixed) occluded images and reconstructed images using different methods of AT&T data set. The first row is from input occluded images; the second row is from RTDreconstructed images; the third row is from L_1 PCA; the fourth row is from Tucker tensor; and the fifth row is from PCA. The cross corruptions can only be removed by RTD.

CHAPTER 5

Conclusion and Future Work

In this dissertation, we study three key technology areas to explore the social media data. The first is viral marketing (word of mouth) technology: we try to identify the most influential individuals on the social networks. We propose highly efficient and scalable methods to calculate the influence spread and then different greedy strategies will be applied to find the most influential users. We do extensive experiments on real world data sets to justify the effectiveness and efficiency of our algorithms. Second, we tackle the 3D social tagging recommendation problem. Different from the traditional 2D recommender system, users are allowed to use short phrases, which refer to tags, to describe their social resources. Therefore, there are three dimensions involved in tagging recommendation - the three constituents (users, items, tags) mentioned above. Tag recommendation system helps the tagging process by advising a set of tags to the user that he may use for a specific item. The tagging information helps web sites to organize their resources, and also assist the users to communicate with each other. We propose to use lower-order tensor decomposition techniques to tackle the extremely sparse social network data. Experiments on real world data sets demonstrate the better performance of our proposed models comparing to state-of-the-art methods. Last but not least, in the social tagging area, there are many types of social media resources, and image is a big component part. We propose an efficient and robust model by applying tensor and L_1 norm sparse coding techniques for effective image representations and image categorization. Experiments show the effectiveness of our models.

We mainly focused on independent cascade model in the study of viral marketing part. We will keep studying and solving other models in the social influence area, such as linear threshold model.

References

[1] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003, pp. 137–146.

[2] I. R. Misner, *The Worlds Best Known Marketing Secret: Building Your Business with Word-of-Mouth Marketing*. Bard Press, 2nd edition, 1999.

[3] J. Nail, "The consumer advertising backlash," Forrester Research and Intelliseek Market Research Report, Tech. Rep., May 2004.

[4] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *KDD*, 2010, pp. 1029–1038.

[5] P. Domingos and M. Richardson, "Mining the network value of customers," in *KDD*, 2001, pp. 57–66.

[6] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *KDD*, 2002, pp. 61–70.

[7] M. Granovetter, "Threshold models of collective behavior," *The American Journal of Sociology*, vol. 83, pp. 1420–1443, 1978.

[8] T. Liggett, *Interacting Particle Systems*, ser. Classics in Mathematics. Springer, 1985.

[9] C. C. Aggarwal, A. Khan, and X. Yan, "On flow authority discovery in social networks," in *SDM*, 2011, pp. 522–533.

[10] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*, 2010, pp. 241–250.

[11] J. Tang, J. Sun, C. Wang, and Z. Yang, "Social influence analysis in large-scale networks," in *KDD*, 2009, pp. 807–816.

[12] J. Tang, S. Wu, B. Gao, and Y. Wan, “Topic-level ssocial network search,” in *KDD*, 2011, pp. 769–772.

[13] M. Kimura and K. Saito, “Tractable models for information diffusion in social networks,” in *PKDD*, 2006, pp. 259–271.

[14] W. Chen, Y. Wang, and S. Yang, “Efficient influence maximization in social networks,” in *KDD*, 2009, pp. 199–208.

[15] A. Anagnostopoulos, R. Kumar, and M. Mahdian, “Influence and correlation in social networks,” in *KDD*, 2008, pp. 7–15.

[16] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan, “Group formation in large social networks: Membership, growth, and evolution.” in *KDD*. ACM, 2006, pp. 44–54.

[17] E. Bakshy, I. Rosenn, C. Marlow, and L. A. Adamic, “The role of social networks in information diffusion,” in *WWW*, 2012, pp. 519–528.

[18] Y. Yang, E. Chen, Q. Liu, B. Xiang, T. Xu, and S. A. Shad, “On approximation of real-world influence spread,” in *ECML/PKDD (2)*, 2012, pp. 548–564.

[19] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos, “Tag recommendations based on tensor dimensionality reduction,” in *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 2008, pp. 43–50.

[20] S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme, “Learning optimal ranking with tensor factorization for tag recommendation,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2009, pp. 727–736.

[21] S. Rendle and L. Schmidt-Thieme, “Pairwise interaction tensor factorization for personalized tag recommendation,” in *Proceedings of the third ACM International Conference on Web Search and Data Mining*. ACM, 2010, pp. 81–90.

[22] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos, “A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis,” *IEEE Transactions on Knowledge and Data Engineering*, 2009.

[23] G. Tomasi and R. Bro, “Parafac and missing values,” *Chemometrics and Intelligent Laboratory Systems*, vol. 75, no. 2, pp. 163–180, 2005.

[24] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, “A multilinear singular value decomposition,” *SIAM J. Matrix Anal. Appl*, vol. 21, pp. 1253–1278, 2000.

[25] R. Schifanella, A. Barrat, C. Cattuto, B. Markines, and F. Menczer, “Folks in folksonomies: Social link prediction from shared metadata,” in *Proceedings of the Third ACM International Conference on Web Search and Data Mining*. ACM, 2010, pp. 271–280.

[26] “Movielens dataset, http://www.grouplens.org.”

[27] “Bibsonomy dataset, http://www.kde.cs.uni-kassel.de/bibsonomy/dumps.”

[28] M. Turk and A. Pentland, “Eigen faces for recognition,” *Journal of Cognitive Neuroscience*, vol. 3, pp. 71–86, 1991.

[29] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 711–720, 1997.

[30] X. He, S. Yan, Y. Hu, and H. jiang Zhang, “Learning a locality preserving subspace for visual recognition,” in *in Proc. IEEE International Conference on Computer Vision*, 2003, pp. 385–393.

[31] Y. Fu, Z. Li, J. Yuan, Y. Wu, and T. S. Huang, “Locality versus globality: Query-driven localized linear models for facial image computing,” *IEEE Trans. Circuits Syst. Video Techn.*, vol. 18, no. 12, pp. 1741–1752, 2008.

[32] A. Shashua and A. Levin, “Linear image coding for regression and classification using the tensor-rank principle.” in *CVPR*, 2001, pp. 42–49.

[33] L. R. Tucker, “Some mathematical notes on three-mode factor analysis,” *Psychometrika*, vol. 31, pp. 279–311, 1966c.

[34] J. Yang, D. Zhang, A. F. Frangi, and J.-Y. Yang, “Two-dimensional pca: A new approach to appearance-based face representation and recognition.” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 1, pp. 131–137, 2004.

[35] J. Ye, “Generalized low rank approximations of matrices,” in *ICML*, 2004.

[36] C. H. Q. Ding and J. Ye, “2-dimensional singular value decomposition for 2d maps and images,” in *SDM*, 2005.

[37] K. Inoue and K. Urahama, “Equivalence of non-iterative algorithms for simultaneous low rank approximations of matrices,” in *CVPR (1)*, 2006, pp. 154–159.

[38] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, “A multilinear singular value decomposition,” *SIAM J. Matrix Anal. Appl*, vol. 21, pp. 1253–1278, 2000.

[39] M. A. O. Vasilescu and D. Terzopoulos, “Multilinear analysis of image ensembles: Tensorfaces,” in *ECCV (1)*, 2002, pp. 447–460.

[40] H. Zou, T. Hastie, and R. Tibshirani, “Sparse principal component analysis,” *J. Computational and Graphical Statistics*, vol. 15, pp. 265–286, 2006.

[41] F. D. Torre and M. J. Black, “A framework for robust subspace learning,” *Int’l J. Computer Vision*, pp. 117–142, 2003.

[42] J. Galpin and D. Hawkins., “Methods of l1 estimation of a covariance matrix,” *Computational Statistics and Data Analysis*, vol. 5, pp. 305–319, 1987.

[43] A. Baccini, P. Besse, and A. D. Falguerolles, “An L1-norm PCA and a heuristic approach,” *Ordinal and Symbolic Data Analysis, edited by E. Diday, Y. Lechevalier and O. Opitz, Springer*, 1996.

[44] J. Bolton and W. J. Krzanowski, “A characterization of principal components for projection pursuit,” Mar. 26 2001.

[45] Q. Ke and T. Kanade, “Robust l1 norm factorization in the presence of outliers and missing data by alternative convex programming,” in *IEEE Conf. Computer Vision and Pattern Recognition*, 2004, pp. 592–599.

[46] N. Kwak, “Principal component analysis based on L1-norm maximization,” *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 30, no. 9, pp. 1672–1680, 2008. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2008.114

[47] J. Gao, “Robust L1 principal component analysis and its bayesian variational inference,” *Neural Computation*, vol. 20, no. 2, 2008.

[48] E. J. Candes, X. Li, Y. Ma, and J. Wright, “Robust principal component analysis,” Dec. 18 2009.

[49] D. P. Bertsekas, *Nonlinear Programming, 2nd Ed.* MIT Press, 1998.

[50] A. Y. Yang, A. Ganesh, Z. Zhou, S. Sastry, and Y. Ma, “A review of fast l1-minimization algorithms for robust face recognition,” *CoRR*, vol. abs/1007.3753, 2010.

[51] D. Luo, C. Ding, and H. Huang, “Are tensor decomposition solutions unique? on the global convergence of hosvd and parafac algorithms,” 2008.

Biographical Statement

Miao Zhang was born in Shandong, China, in 1984. She received her B.S. degree from Shanghai Jiao Tong University, China, in 2007, and her Ph.D. degree from The University of Texas at Arlington in 2014 in Computer Science and Engineering. Her current research interest is in the area of data mining, machine learning and their applications in social network areas.