UNIVERSITY OF CALIFORNIA,
IRVINE

Influence Maximization in GOLAP

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by

Jennifer Kim Jin

Dissertation Committee:
Professor Phillip Sheu, Chair
Professor Jean-Luc Gaudiot
Professor Nikil Dutt

2019

TABLE OF CONTENTS

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor, Professor Phillip Sheu. Professor Sheu has been a tremendous mentor to me. I would like to thank him for encouraging my research and for allowing me to grow as a scholar. His strong guidance, scientific advice and knowledge, constructive criticism and many insightful discussions and suggestions have been invaluable throughout the years.

I am thankful to the members in my Lab. Guigang Zhang helped me on the thesis structure, provided me ideas, and helped me develop the use cases and examples. Shaoting Wang helped me brainstorm algorithm ideas. I would like to express my sincere gratitude to Charles Wang from Asia University, for collaborating with me on the biomedical project. His knowledge and expertise on bioinformatics has been extremely helpful. I would like to extend my sincere thanks to Bryan Chou, Jimmy Shih and Kyle Li for helping me with my theorems and proofs.

As a cooperator of our Lab, I am also thankful to the supports from NEC Solution Innovators, Ltd., Japan. Mr. Hiroyuki Shigematsu and Mr. Atsushi Kitazawa gave me many beneficial ideas on my research directions. I am also grateful to Mr. Masahiro Hayakawa for helping me polish my papers.

For this dissertation I would also like to thank my committee members: Professor Jean-Luc Gaudiot and Professor Nikil Dutt. With their expert knowledge in this area, they provided many useful comments so I could refine my dissertation and enlightened me to some future research directions.

I would also like to thank my friends for providing support and friendship that I needed. I am deeply thankful to my family. Words cannot express how grateful I am to my mom and dad for their love, invaluable advice, undying support and sacrifices. I am extremely grateful to my mother-in law and father-in-law for always supporting me with prayer and encouraging words. Your prayer for me was what sustained me thus far. The success would not have been possible without the support and nurturing of my beloved husband, Daniel Jin. I would also like to thank my best friend and my amazing sister, Mira Kim for supporting me in everything. I can't thank her enough for encouraging me throughout this experience. To my lovely daughters Natalie Grace Jin and Lauren Sophia Jin, I would like to say thank you for being such sweet girls and always cheering me up.

Finally, I thank my God for helping me through all the difficulties. I have experienced your guidance day by day. Thank you, Lord.

Jennifer Kim Jin
University of California, Irvine
2019

CURRICULUM VITAE

Jennifer Kim Jin

2009	B.A. in Computer Science, University of Texas, Dallas
2011	Master's in computer science, University of California, Los Angeles
2011-18	Teaching Assistant, University of California, Irvine
2019	Ph.D. in Computer Science, University of California, Irvine

FIELD OF STUDY

Semantic Computing and Graph Databases

PUBLICATIONS

Charles C.N. Wang, Jennifer Jin, Jan-Gowth Chang, Masahiro Hayakawa, Atsushi Kitazawa, Jeffrey J.P. Tsai and Phillip C.-Y. Sheu, "Identification of Most Influential Co-Occurring Gene Suites for GI Cancers using Biomedical Literature Mining and Graph-based Influence Maximization," Future Generation Computer Systems Special Issue on Data Exploration in the Web 3.0 Age, submitted on Feb 10, 2019.

Jennifer Jin and Masahiro Hayakawa, "Network Analysis for Graph-based OLAP (GOLAP)," International Journal of Semantic Computing, 13(1), 2018.

Jennifer Jin, "OLAP and machine learning," Encyclopedia with Semantic Computing and Robotic Intelligence, 1(1), 2017.

Shao-Ting Wang, Jennifer Jin, Pete Rivett, and Atsushi Kitazawa, "Technical Survey Graph Databases and Applications", International Journal of Semantic Computing 9(4), pp. 523-545, 2015.

Jennifer Jin, Mira Kim and Pete Rivett, "Technology Outlook: Semantic Computing for Education," International Journal of Semantic Computing, 9(3), 2015.

Jennifer Kim, George Wang and Sang Tae Bae, "A survey of Big Data Technologies and How Semantic Computing can Help," International Journal of Semantic Computing, 8(1), 2014.
Jennifer Kim, David Ostrowski, Hiroshi Yamaguchi and Phillip C.Y. Sheu, "Semantic Computing and Business Intelligence," International Journal of Semantic Computing, 7(1), August 2013.

Jennifer Kim, “Design and Evaluation of Mobile Applications with Full and Partial Offloadings,” The 7th International Conference on Grid and Pervasive Computing (GPC 2012), Hong Kong, May 2012.

Jennifer Kim, “Architectures with Full and Partial Offlaoding for Mobile Applications,” IEEE International Conference on Internet (ICONI 2010), Mactan Island, Philippines, Dec. 2010.

Jennifer Kim, “Architecture patterns for Service-based Mobile Applications,” IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2010), Perth, Australia, Dec. 2010

Jennifer Jin, “Augumented Influence Analytics on Social Network Services,” for Journal Publication (to be completed in Jan. 2019)

Jennifer Jin, “Software Framework for Gene Disease Relevance Analytics,” for Journal Publication (to be completed in Feb. 2019)

ABSTRACT OF THE DISSERTATION

Influence Maximization in GOLAP

By

Jennifer Kim Jin

Doctor of Philosophy in Computer Engineering

University of California, Irvine, 2019

Professor Phillip Sheu, Chair

The notion of influence among people or organizations has been the core conceptual basis for making various decisions and performing social activities in our society. With the increasing availability of datasets in various domains such as Social Networks and digital healthcare, it becomes more feasible to apply complex analytics on influence networks. However, there exist technical challenges of representing various types of influence networks, handling the variability on the analytics types, and optimizing the time complexity in running the analytics. We present a comprehensive approach to managing influence networks using a set of extended graph models, called Graph-based OLAP (GOLAP). The design space for GOLAP is defined by the incorporation of node types (i.e., colors), weights on relationships (i.e., edges), constraints on the number of nodes for a certain node type, and constraints on the percentage of nodes for a certain node type. We begin with defining a method to find a Strongest Influence Path (SIP) which is the strongest path from the source node to the target node. Then, we extend it with k-colors, a constraint on the number of nodes, and a constraint on the percentage of nodes. Hence, we can answer complex queries on influence networks such as "find the SIP with *t* nodes of color *c*" or "find the SIP with *t*% nodes of color *c."* Based on the SIP model, we present a set of Influence Maximization (IM)

methods which find a set of *s* seed nodes that can influence the whole graph maximally with various constraints such as having '*t* nodes of color *c*'. We apply the IM methods to Gastrointestinal (GI) cancer data and prove the proposed approach works well in the context of GI cancers. We use text mining to identify objects and relationships to construct a graph and use graph-based IM to discover the most influential co-occurring genes. We also address the methods for optimizing the time complexity of the analytics algorithms. We apply heuristic-based and graph reduction-based methods to reduce the time complexity. In addition to proving the proposed methods, we present the result of our implementation on the methods.

Chapter 1
Introduction

The research on influence analysis in social networks is becoming a hot topic. For example, in a social network a democrat or a republican can influence another democrat or a republican. In science, an author/scientist may influence another author or scientist and a publication can influence another publication.

There are many existing models for information spreading. The traditional models include Independent Cascade Model (ICM) [1], and Linear Threshold Model (LTM) [2]. However, these existing influence models are discrete. If one node accepts another node, it will have influence; if not, it has no influence. However, in the real world, if one node accepts or does not accept another node, it will have some possible influence to a certain degree. Therefore, we propose a new model based on the probability of influence. It is consequently not discrete. We define this model as the Strongest Influence Path (SIP) model.

Our definition for the SIP model is as follows. Given a graph *G*, assume an application is only interested in finding the strongest influence path from a node to another node. Can we design an algorithm to solve this problem? We call this problem the SIP (Strongest Influence Path) Problem. We can utilize this model to solve the Influence Maximization (IM) problem, the problem of finding the strongest seed nodes in a graph that can influence the whole graph maximally.

We also discovered that the traditional approaches to the IM problem are limited – They only consider traditional graphs. A traditional graph does not have any way of representing different classes of nodes. In many domains, queries involving various classes of nodes can produce useful information. In a social network, for example, the nodes could be classified based on gender, ethnicity, political party, area of residence, etc. The edges could be classified based on the

relationship between the nodes. By using classifications, we are able to answer class-level queries. Thus, with the combination of two sets of dimensions, we can post queries such as "Find a republican that has the highest influence on all the other people they work with." We propose using colors on nodes to represent different classes, and we study the IM problem in the context of "colored" graphs.

While reference [3] introduces Graph-based OLAP (GOLAP) that extends OLAP (On Line Analytical Processing) to address graph-based problems that involve object attributes, it does not address the IM problem.

Our contributions can be summarized as the following:

(1) We propose a new influence model: Strongest Influence Path (SIP) model.

(2) We propose algorithms to solve the *k*-colors constraint SIP problem and its variants.

(3) We propose the *k*-colors graph reduction methods based on the SIP model.

(4) We propose algorithms to solve the *k*-colors Influence Maximization problem based on the SIP model.

In Chapter 2, we summarize related works. We define the key concepts of this paper in Chapter 3. In Chapter 4, we propose the definition of SIP and its algorithms. In Chapter 5, we introduce Top-*m* SIP and its algorithms. In Chapter 6, we discuss SIP with *t* colored nodes. We present algorithms for SIP with *t*% colored nodes in Chapter 7 and a heuristic algorithm in Chapter 8. We introduce the *k*-colors influence maximization problem based on SIP in Chapter 9 and introduce a greedy heuristic algorithm in Chapter 10. In Chapter 11, we present *k*-colors influence maximization problem in Biomedical domain. We propose a *k*-colors graph reduction algorithm based on SIP in Chapter 12. Chapter 13 reports the experiment results that validate our methods.

Finally, in Chapter 14 we conclude the paper and discuss some possible research directions in the future.

Chapter 2
Related work

In this section, we summarize existing research that are related to GOLAP, the strongest path and influence in graphs.

2.1. Influence Research

Influence analysis has caught researchers' interest from the 1950s. More and more scientists and researchers begin to realize the importance of influence in social networks [4, 5]. In the early ages, several theories were proposed, such as the theory of six degrees of separation, the theory of four degrees of separation, and small world phenomenon [6, 7]. Analyzing influence on social networks [8, 9, 10] can be used to develop new applications, such as advertising, sentiment diffusion [11], link prediction [12], recommended systems and social communities finding.

Finding the most influential node in a social network such as the most important blogger [13] or an opinion leader [14] can be measured by degree centrality. High degree centrality of a node indicates the node's high importance. The PageRank algorithm [15] is used to find the importance of nodes in social networks. Betweenness Centrality [16] is another key factor to measure a node's importance. Closeness Centrality is also typically used to measure a node's importance. Besides these three methods, some others include HITS [17], K-Shell [18], and randomly walk algorithm [19]. There are several algorithms to measure a node's importance [20], but the main methods can be summarized into two categories: measurement based on topological structure, and measurement based on contents and measurement based on users' behaviors [21].

Influence maximization (IM) is one of the most studied topics in influence research, especially in social networks. The IM problem finds a set of seed nodes that can influence the whole graph

maximally. Most existing IM problems use diffusion models [22-24] such as ICM [1], LTM [2], SIS [25] and SIR [26].

IM is being applied to different types of network: labeled social networks [27,28], multiple labeled networks [29], and dynamic social networks [30-33]. Other than general graphs, special graphs such as Bipartite and Hierarchical IM [34] problems have been studied. Most current works focus on positive weighted networks and do not consider negative weighted graphs [35, 36]. IM is being applied to different social networks such as messenger-based social network [37], microblogging [38], and viral marketing social networks [39]. It can be applied to complex social networks [40], multiple social networks [41] and some unknown social networks [42]. These complex social networks may have multiple acceptance [43, 44]. Rahaman and Hosein [45] propose the multi-stage influence maximization problem. Some studies use the non-backtracking random walk method [46, 47]. Since greedy algorithms [48] have high time complexity, some obtain approximation results while sacrificing the accuracy [49]. K-Means based methods [50-55] are also used to solve IM problems. Lee and Chung [56] propose a query approach while Jiang *et al.* [57] propose the community detection method. Other methods include Memetic [58], SIMPATH [59], and the community discovery algorithm [60].

As most social networks are very large in size, scalability is a major concern. Some researchers propose scalable methods that do not lose efficiency as the size of the network grows. Li and Yu [60] propose a scalable community discovery algorithm to find the core members of the community. Wang *et al.* [82] present a bottom-ksketch (a summary is a set of items with nonnegative weights that supports approximate query processing) based Reverse Influence Sampling (RIS) framework, which brings the order of samples into the RIS framework. By applying the sketch technique, they derive early termination conditions to significantly accelerate

the seed set selection procedure. Chen *et al.* [62] propose a scalable IM algorithm tailored for linear threshold model based on the fast computation in directed acyclic graphs (DAGs). Mohan *et al.* [63] propose a model to find a set of influential nodes from a very large graph. By dividing the graph into communities and finding influential nodes within each of those communities, a better set of nodes for fast information diffusion is obtained. Chen *et al.* [64] design a new heuristic algorithm that is easily scalable to millions of nodes and edges. Their algorithm has a tunable parameter for users to control the balance between the running time and the influence spread of the algorithm.

As discussed, methods to solve classic IM problems have been proposed by many. Others propose special types of IM problems by adding special conditions or constraints, making it even more complex. Some special IM problems include privacy reserved influence maximization [65], online social networks [66-68] and location based social networks [69] influence maximization. Some social networks' nodes have several attitudes [70]. Some include time constraint IM [71, 72], topic-based IM [73], min-cut IM based on big data, budget IM [74-76] and profit IM [77] for multiple products.

As discussed, there are a lot of existing research done on influence research on social networks. However, to our knowledge most of the existing models do not consider partial influence from one node to another. The models are set up so a node either influences or does not influence another node. Another shortcoming of the existing work is that they do not consider different classes of nodes. They perform IM on just regular graphs, where nodes are not classified.

2.2. Strongest Path

As will be discussed later, the strongest path problem can be translated to a shortest path problem. In graph theory, the shortest path problem is a problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

Our SIP-based algorithms are based on Dijkstra's shortest path algorithm [78]. Classic shortest path algorithms can be used instead if desired. These include Dijkstra's algorithm [78], Floyd-Warshall algorithm [79], and A* algorithm [80]. Dijkstra's algorithm solves the single-source shortest path problem; Bellman–Ford algorithm [81] solves the single-source problem if edge weights may be negative; A* search algorithm solves single pair shortest path using heuristics to speed up the search; and Floyd–Warshall algorithm solves all pairs shortest paths.

2.3. Graph-based OLAP (GOLAP)

To our knowledge, not much work has been done for queries related to different classes of nodes. Chou *et al.* [3] propose GOLAP system with two dimensions: nodes and edges. For each dimension, different colors can be used to represent different classes. For example, the nodes in a social network may represent people. In Figure 1, the nodes are colored in blue for democrats and red for republicans. The edges in a social network may represent relationships. Based on the type of relationship, the edges could be colored in different colors. In Figure 1, green edges represent family, red edges represent friends and blue edges represent co-workers.

By using different colors for different types of people, we can visualize different classes and run class-specific queries. An example of a node dimensional query could be finding the top 3 democrats that are the most influential in the entire network. Another query may be finding the most influential people among co-workers. This would be an edge dimensional query.

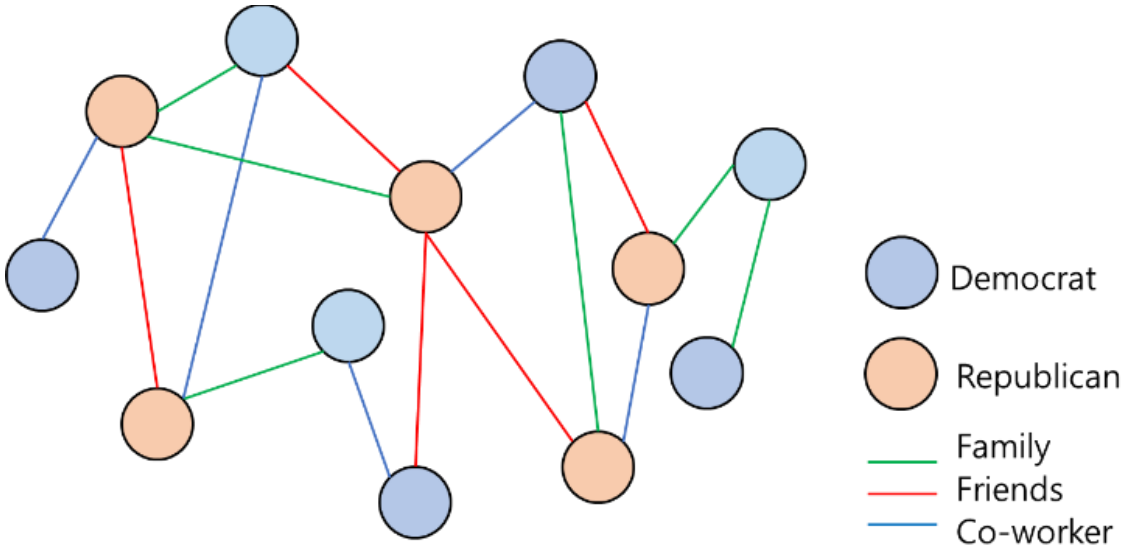


Figure 1. An example of node and edge hierarchy

Our model utilizes colors to classify the nodes. By applying Graph-based OLAP (GOLAP) proposed by Chou *et al.* [3], we are able to convey and apply this concept to solve problems using the node dimension.

2.4. Graph Reduction

With big data trending, graph reduction has been a topic of interest for quite some time. In certain applications, we only need a portion of a graph. We can reduce these graphs into small graphs and obtain the same or similar query results.

Wang *et al.* [82] propose graph reduction that can answer queries without losing information. Navlakha *et al.* [83] describe a graph summarization method with bounded errors. A graph can be reduced by a two-part representation: summary and correction. The summary is an aggregate graph that groups the nodes into sets with edges representing relations between sets. The corrections part specifies a list of edge-corrections that should be applied to the summary to recreate the graph. Toivonen *et al.* [84] compress a weighted graph by merging nodes and edges to achieve the target

reduction rate while minimizing the difference on the edge weights between the original graph and the compressed graph.

Most social networks are very large in scale. Some of the algorithms especially for solving influence maximization have high time complexity, making it challenging for users to obtain results in real-time. We propose graph reduction to alleviate this issue. Also, existing work do not consider reducing graphs with colored nodes. Being able to reduce a colored graph is more complex but useful in many domains.

Chapter 3
Definitions

Below we first review some terms in the graph theory and the strongest influence path (problems) discussed in this thesis.

(1) Weighted Graph: A weighted graph G is a triplet (V, E, w), where w: $E \rightarrow eVal$ is a function mapping an edge to its weight, and $eVal$ is the set of possible values. For a weighted graph, $eVal$ would usually be real numbers.

(2) Path: A path p from node u to v in G is a sequence of nodes $(u, v_0, v_1, \ldots, v_n, v)$ such that for every $i \in \{1, \ldots, n\}$, $(v_{i-1}, v_i) \in E$.

(3) Shortest Path: A shortest path from u to v is a path p= $(u, v_0, v_1, \ldots, v_n, v)$ of which the sum $\sum_{i=1}^{n} w(v_{i-1}, v_i)$ is minimal.We use the notation $\delta G(u, v)$ to denote the total weight of the shortest path from u to v. There may be multiple shortest paths between two nodes.

(4) Influence: The influence from a node to another node is depicted as an edge weight ranging between 0 and 1. The edge weight inf (u, v) represents the probability of node u influencing node v.

(5) Strongest Influence Path (SIP): The strongest influence path from a node to another node is the path that has the strongest influence value (calculated by multiplication of influence probabilities along the path). There are multiple paths between a node to another node but there exists at least one strongest influence path among all these paths.

(6) Influence Maximization (IM): The influence maximization problem finds a set of s seed nodes that can influence the whole graph maximally. It asks for a parameter s, to find a s-node set of maximum influence.

Chapter 4
Strongest Influence Path (SIP)

In this section, we present our approaches to find the strongest influence path from a node to another node in a graph.

4.1. The SIP problem

Given a directed graph *G* where each edge (*a*,*b*) carries a weight (0, 1), that designates the probability of influence from node *a* to node *b*. Note that the weight of an edge should not be 0 because if there was no influence between the two end nodes, that edge should not exist as edges represent influence. The Strongest Influence Path (SIP) problem finds the path from a given source node *s* to a given destination node *d* that has the strongest probability. The probability of influence for a path from *s* to *d* is calculated by multiplying the influence probabilities of all the edges on the path.

4.2. Use case of SIP

Consider a corporate network that consists of different companies. Each node represents a company and each edge represents influence. A small company *s* wants to influence a mega company *d* into investing on a business idea they have. *s* wants to know which companies to go through to have the highest chance of getting an introduction to the mega company *d*. The path highlighted in red is the strongest influence path from node *s* to node *d*. As shown in Figure 2, the path highlighted in red is the strongest influence path from node *s* to node *d*. A directed edge from a node to another is labelled by the probability that the first node can influence the second node. For simplicity, we combine two bidirectional edges between two nodes *a* and *b* (i.e., they can influence each other) into one edge without an arrow, and label the edge with two probabilities –

the first being the probability of node *a* influences node *b* and the second being the probability of node *b* influences node *a*. With such, the influence probability of the SIP from *s* to *d* is the multiplication of influences 0.6*0.4*0.7*0.5*0.9*0.8*0.5 which is 0.03024.

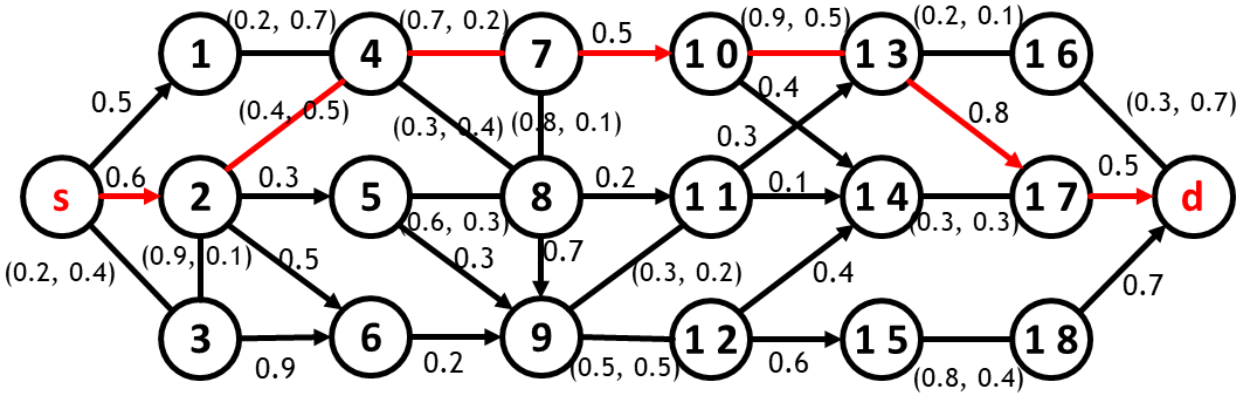


Figure 2. Example of SIP

4.3. Algorithm of SIP

Our SIP algorithm that finds the strongest influence between two nodes is based on Dijkstra's shortest path algorithm. To compute the strongest path between two nodes, we can adapt Dijkstra's algorithm as Hangal *et al.* does [85]. They define the influence of a path to be the product of the influence of each edge on the path.

For the SIP, let us say there are 3 edges in the path, *e1*, *e2* and *e3* whose weights are between 0 and 1. The total influence is calculated by *inf(e1)*inf(e2)*inf(e3)*, which is still between 0 and 1. Maximizing *inf(e1)*inf(e2)*inf(e3)* is equivalent to minimizing -log(*inf(e1)*inf(e2)*inf(e3)*) because -log(1) = 0 and -log(0) is ∞. Since -log(*inf(e1)*inf(e2)*inf(e3)*) = -(log (*inf(e1)*) + log (*inf(e2)*) + log (*inf(e3)*)) = (-log (*inf(e1)*))+ (-log (*inf(e2)*)) + (-log (*inf(e3)*)). It becomes a shortest path problem with all positive distances as -log(*x*) is positive when 0<*x*<1. Therefore, our SIP method can correctly find the strongest influence path as long as the edge weights stay in range (0, 1].

We repeat Dijkstra's algorithm in the following:

Partial snippet of Dijkstra's algorithm (finds the shortest path from a source node to a destination node)	
Input	Graph *G*, source node *s*, destination node *d*
Output	The shortest path from *s* to *d*
1	For each neighbor *v* of *u*; // where *v* is still in 1;
2	*alt* ←dist[*u*] + length(*u*,*v*)
3	if *alt* < dist[*v*]; // a shortest path to *v* has been found
	dist[*v*] ←*alt*;
4	prev[*v*] ←*u*;
5	return dist[], prev[]
6	

In the above, *dist*[*v*] designates the distance from the source node to *v*. *Prev*[*v*] is the previous node in the optimal path from the source. Node *u* is the node with the shortest distance that was selected. As shown in line 2, the distance is calculated by the sum of the weights.

The SIP algorithm is revised from the Dijkstra algorithm using *influence*[*v*] which is the total influence value from the source node to *v* instead of *dist*[*v*]. It is calculated by the product of the weights. The SIP algorithm finds the path with the highest overall weight as shown in line 3.

Partial snippet of SIP algorithm (finds the strongest influence paths from a source node to a destination node)	
Input	Graph *G*, source node *s*, destination node *d*
Output	The Strongest Influence Path from *s* to *d*
1	for each neighbor *v* of *u*; // where *v* is still in 1;
2	if inf[*u*] * inf(*u*,*v*)> inf[*v*]; // a strongest influence path to v has been found
	inf[*v*] = inf[*u*] * inf(*u*,*v*);
3	prev[*v*] =*u*;
4	Return inf[], prev[]
5	

The time complexity of the SIP algorithm is the same as that of the Dijkstra's algorithm which is *O(e log n).*

Chapter 5
Top-m SIP (Top-m Strongest Influence Path)

In this section, we present our approaches to find the top-m strongest influence paths from a node to another node in a graph.

5.1. The Top-m SIP Problem

Given a directed graph *G* where each edge (*a, b*) carries a weight between 0 and 1 that designates the probability of influence from node *a* to node *b*. *SIP-m* (*s, d*) is the *m* strongest influence paths from the source node *s* to the destination node *d*. The Top-m Strongest Influence Path (Top-m SIP) problem finds not just the strongest influence path but the top-m strongest influence paths from a node to another node (*SIP-1, SIP-2, …, SIP-m*).

5.2. Use case of Top-m SIP

Consider a corporate network that consists of different companies. Each node represents a company and each edge represents an influence. Word on the street is that a huge software company *d* is looking for a hardware company to collaborate on creating the next big tech gadget. A hardware company *s* would like a chance of working with company *d* but knows they have a lot of competition. *s* is interested to see which path of companies to go through to influence company *d* for picking *s* as their partner. To compare options, *s* would like to see what their top three path choices are.

Figure 3 shows an example of the top-m SIP where the value of *m* is 3. Shown in red is the *SIP-1*, which is the strongest influence path from node *s* to node *d*. The second strongest SIP (*SIP-2*) is shown in blue. The next strongest SIP (*SIP-3*) is shown in purple. Company *s* compares their choices and decides to avoid *SIP-1* since they had a bad business deal in the past with company 7.

Company 3, a rival of *s* with a potential to steal their opportunity is on *SIP-2* so *s* decides to skip this path. *s* chooses *SIP-3* to pass a good recommendation to *d*.

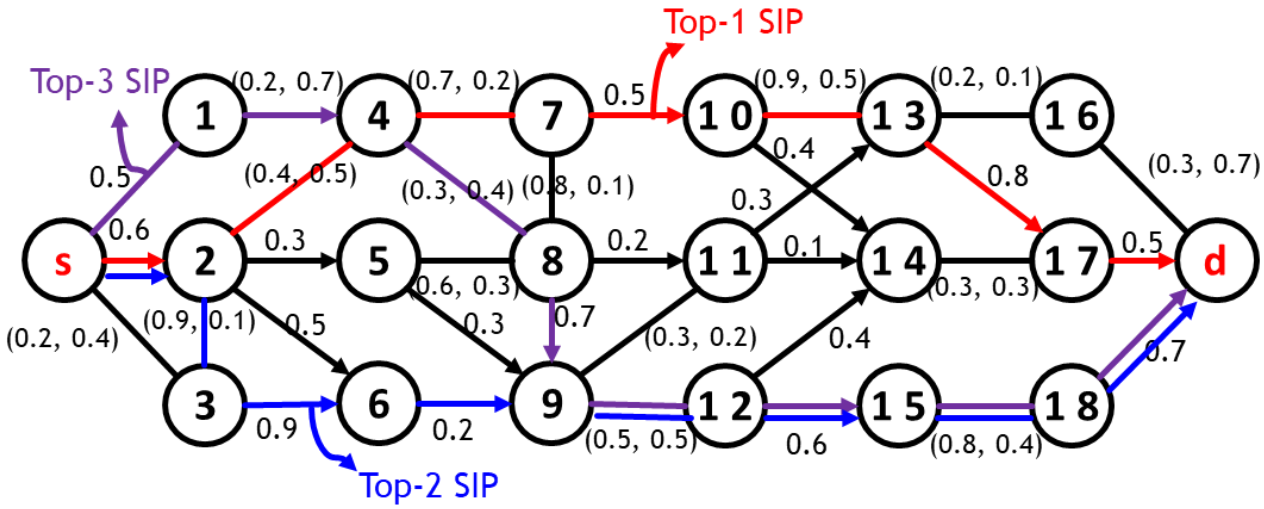


Figure 3. Example of Top-*m* SIP

5.3. Algorithm of Top-m SIP

The basic idea of the top-*m* SIP algorithm is that the m^{th} SIP will be a deviation from the previously discovered SIPs. It first calls the regular SIP algorithm to compute SIP (*SIP-1*). In order to find *SIP-k*, where *k* ranges from *2* to *m*, the algorithm assumes that all the paths from the source to the destination have previously been found. The algorithm iterates through the edges of the SIP found, removing one of the edges in each iteration and calculating alternative SIPs. Among these alternative routes, the one with the strongest influence is the next SIP.

Our algorithm is similar to Yen's algorithm [85] which computes the top-*m* shortest paths from a source node to a destination node. Yen's algorithm utilizes the idea that the m^{th} SIP will be a deviation from the previously discovered SIP. It holds two lists; list *A* holds the permanent *m* SIPs while list *B* holds the candidate *m* SIPs.

For example, the path *SIP-1* in Figure 3 (marked in red) is computed by the SIP algorithm and placed on list *A*. Subsequently, the SIP calculated without *e*(s,2) in graph G, the SIP calculated

without $e(2,4)$ in graph G, the SIP calculated without $e(4,7)$, …, and the SIP calculated without $e(17,d)$ are placed on list B. Finally, list B (containing the candidate paths) is sorted in decreasing order so among all the paths in list B, the one with the highest path weight becomes *SIP-2* and is moved to list A.

Next, the SIP calculated without $e(2, 3)$ in graph G, the SIP calculated without $e(3, 6)$ in graph G, the SIP calculated without $e(6,9)$, …, and the SIP calculated without $e(18, d)$ are placed on list B. Finally, list B (containing the candidate paths) is sorted in decreasing order so among all the paths in list B, the one with the highest path weight becomes *SIP-3* and is moved to list A. The process is repeated to find *SIP-4, SIP-5* etc. until *SIP-m* is found and placed on list A.

Similar to Dijkstra's algorithm which takes the summation of weights ranging from 1 to ∞ to find the minimum path weight, Yen's algorithm returns the top m paths with the smallest path weight. Yen's algorithm can be converted to top-m algorithm, in the same way Dijkstra's algorithm is converted to the SIP algorithm explained in Section IV-C. As long as the edge weights are between 0 and 1, we can find the m paths whose path weights are the maximum.

The Top-m SIP algorithm is summarized in the following.

Top-m SIP algorithm (finds the top m strongest influence paths from a source node to a destination node)

Input	Graph G, source node s, destination node d, m
Output	Top-m Strongest Influence Paths
1	Function TopmSIP(G, s, d, m):// Determine top m longest paths from the source to the destination.
2	$A[0]$ = SIP(G, s, d);// Initialize the heap to store the potential kth longest path.
3	B = [];
4	for m from 1 to M:// The spur node ranges from the first node to the next to last node in the previous m-longest path.
5	for i from 0 to size ($A[m-1]$) − 1:// Spur node is retrieved from the previous m-longest path, $m-1$.
6	*spurNode* = A[m-1].node(i);// The sequence of nodes from the source to the spur node of the m-longest path.
7	*rootPath* = A[m-1].nodes(0, i);
8	for each path p in A:

```
9           if rootPath == p.nodes(0, i):// Remove the links that are part of the
        previous longest paths which share the same root path.
10              remove p.e (i, i+1) from G;
11          for each node rootPathNode in rootPath except spurNode:
12              remove rootPathNode from G;
13          spurPath = SIP(G, spurNode, destination);// Calculate the spur path from
        the spur node to the destination.
14          totalPath = rootPath + spurPath;// Entire path is made up of the root path
        and spur path.
15          B.append(totalPath);// Add the potential m-longest path to the heap.
16          restore edges to G;// Add back the edges and nodes that were removed
        from the graph.
17          restore nodes in rootPath to G;
18      if B is empty:
        // This handles the case of there being no spur paths, or no spur paths left. This could
        happen if the spur paths have already been exhausted (added to A), or there are no
        spur paths at all - such as when both the source and destination vertices lie along a
        "dead end".
19          break;
20      B.reverseSort();// Sort the potential m-longest paths by cost in decreasing order.
21      A[k] = B[0];// Add the highest cost path becomes the m-longest path.
22      B.pop();
23  return A;
```

The time complexity of the top-*m* SIP algorithm is the same as that of Yen's algorithm which is $O(mn^3)$. The top-*m* SIP algorithm calls the SIP algorithm to obtain the longest path. The time complexity of the SIP algorithm is $O(n^2)$. Since the top-m SIP algorithm makes m*l calls to the SIP algorithm in computing the spur paths, where l is the length of the spur paths and the expected value of l in a condensed graph is O(log n) [85], while in the worst case it is n. So the best case time complexity is $O(m \log n \times n^2)$ and the worst case time complexity is $O(mn^3)$.

Chapter 6
k-Colors Strongest Influence Problem with *t* constraint (*k*-Colors/SIP-*t)*

In this section, we present our approaches to find the strongest influence path with at most/exactly/at least *t* nodes of color *c* from a node to another node in a graph.

6.1. k-Colors t-SIP (k-Colors Strongest Influence Path with t Constraint)

Given a directed *k*-colors graph *G* where each edge (*a, b*) carries a weight (0, 1) that designates the probability of influence from node *a* to node *b*. The nodes are one of *k* different colors and the colors represent different types of nodes, such as gender, ethnicity, age group and political preference. The *k*-colors SIP-*t* problem finds the SIP from the source node *s* to the destination node *d* which passes through exactly/at least/at most *t* nodes that have the same color (called *constraint color*).

6.2. Use case of K-colors t-SIP

Consider a corporate map that consists of different companies based on supply chains (Figure 4). Each node represents a company, each edge represents the strength of supply, and each node is colored based on its size: blue for large business, green for medium-size business, and red for small business). A large company *d* wants to buy a small company *s* and at the same time acquire the intermediate suppliers with the strongest supplies (influence) between the two companies and at least 4 of them are small businesses. So the problem is to find the strongest path from *s* to *d* that includes at least 4 intermediate companies (red nodes) that are small businesses (exemplified by the red path in Figure 3).

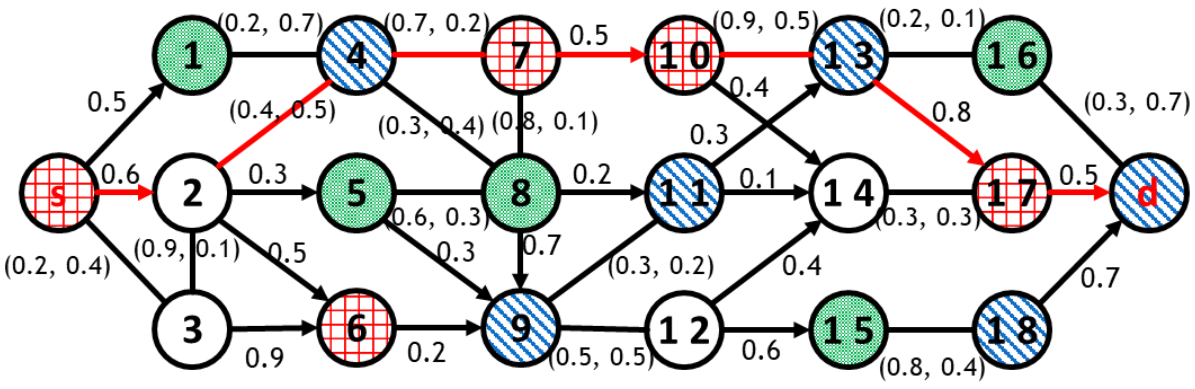
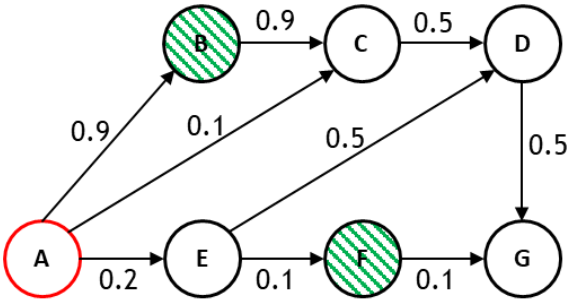


Figure 4. Example of *k*-colors SIP-*t*

6.3. Algorithm of k-Colors t-SIP

The basic idea of our algorithm is that it computes the paths by layers. Assume the constraint color is green. First, we compute the paths containing zero green node (*inf*[0]). Then we compute the paths containing one green node (*inf*[1]) based on *inf*[0], the paths containing two green nodes (*inf*[2]) based on *inf*[1], and so on. We keep going until no more paths can be computed. The visited nodes are reset when starting a new layer of path calculation. To avoid loops in a path, we do not visit a neighbor if it is already in the path. For example, if *B* is a green node and the path of *C*'s *inf*[1] is $A \rightarrow B \rightarrow C$, we do not compute *B*'s *inf*[2] from *C* because it will cause a loop. Note that we assume there only exists one SIP.

The following is an example. With *A* as the source node and *t*=1, it keeps track of the paths from node *A* to all the other nodes with zero green node and one green node. For example, *inf*[1].*C* designates the influence of the strongest path from the source node *A* to node *C* with 1 green node.

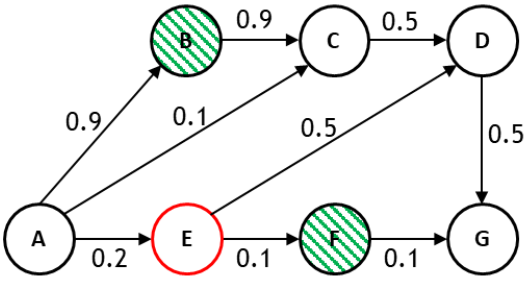


	B	C	D	E	F	G
inf[0]		.1		.2		
inf[1]	.9					
pre[0]		A		A		
pre[1]	A					

Grey: visited nodes
Red: maximum inf[0]

Figure 5(a) Step 1

Figure 5(a) shows the layer *inf*[*0*] with zero green node. We start with the source node which is *A* and visit its outgoing neighbors *B*, *C* and *E*. *B* is green so it will be excluded from *inf*[0]. However it is recorded in *inf*[1] because we will not visit *A* again (to avoid a loop.) Therefore *inf*[1].*B*=0.9 (*A*→*B*). *C* is white so *inf*[0].*C*=0.1 (*A*→*C*). *E* is white so *inf*[0].*E*=0.2 (*A*→*E*). Among all the *inf*[0] values, *E* has the maximal influence so the next node to visit is *E*.

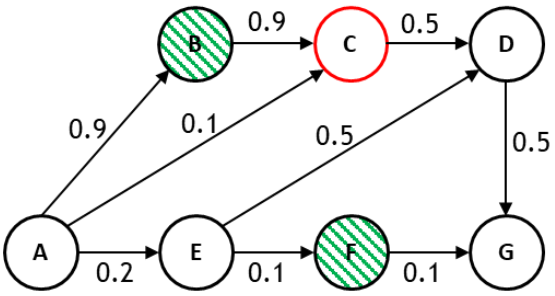


	B	C	D	E	F	G
inf[0]		.1	.1	.2		
inf[1]	.9				.02	
pre[0]		A	E	A		
pre[1]	A				E	

Grey: visited nodes
Red: maximum inf[0] in unvisited nodes

Figure 5(b) Step 2

Figure 5(b) computes the influences of node *A* on *E*'s outgoing neighbors *D* and *F*. *D* is white so inf[0].D=0.2*0.5=0.1 (A→E→D). F is green so inf[1].F=0.2*0.1=0.02 (A→E→F). Among the unvisited nodes, both nodes C and D have the maximal inf[0] so we randomly choose C.

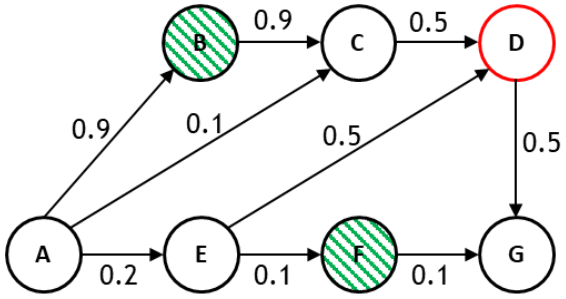


	B	C	D	E	F	G
inf[0]		.1	.1	.2		
inf[1]	.9				.02	
pre[0]		A	E	A		
pre[1]	A				E	

Grey: visited nodes
Red: maximum inf[0] in unvisited nodes

Figure 5(c) Step 3

Next, shown in Figure 5(c), we compute the influences of node *A* on *C*'s outgoing neighbor *D*. The influence 0.1*0.5 (*A*→*C*→*D*) is less than the current *inf*[0].*D*=0.1 so there is no need to update its value. Now only node *D* is visited because it is the only unvisited node with an *inf*[0] value.

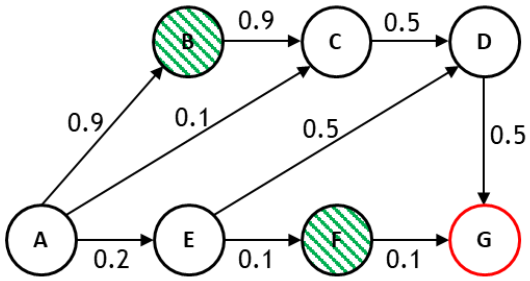


	B	C	D	E	F	G
inf[0]		.1	.1	.2		.05
inf[1]	.9				.02	
pre[0]		A	E	A		D
pre[1]	A				E	

Grey: visited nodes
Red: maximum inf[0] in unvisited nodes

Figure 5(d) Step 4

In Figure 5(d), we compute node *A*'s influence on *D*'s outgoing neighbor *G*. *G* is white so *inf*[0].*G* = 0.1*0.5 =0.05(*A*→*E*→*D*→*G*). Next, we choose the only unvisited node with an *inf*[0] value, which is *G*, and mark node *D* visited.

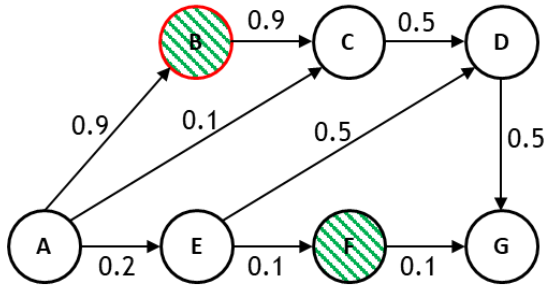


	B	C	D	E	F	G
inf[0]		.1	.1	.2		.05
inf[1]	.9				.02	
pre[0]		A	E	A		D
pre[1]	A				E	

Grey: visited nodes
Red: maximum inf[0] in unvisited nodes

Figure 5(e) Step 5

In Figure 5(e), *G* has no outgoing neighbors. There are no nodes with an *inf*[0] value that has not been visited anymore, so we can start the path layer with one green node (*inf*[1]). Node *B* has the maximum *inf*[1] value so we start with *B*, and mark node *G* visited.

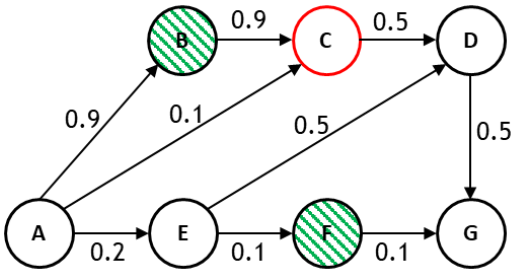


	B	C	D	E	F	G
inf[0]		.1	.1	.2		.05
inf[1]	.9	.81			.02	
pre[0]		A	E	A		D
pre[1]	A	B			E	

Grey: visited nodes
Red: maximum inf[1] in unvisited nodes

Figure 5(f) Step 6

Figure 5(f) shows the *inf*[1] layer. We first reset all the nodes as unvisited, and compute the influences of node *A* on *B*'s outgoing neighbor *C*. *C* is white so *inf*[1].*C* = 0.9*0.9 =0.81 (*A*→*B*→*C*). So the next node with the maximum *inf*[1] value is *C*, and we mark node *B* visited.



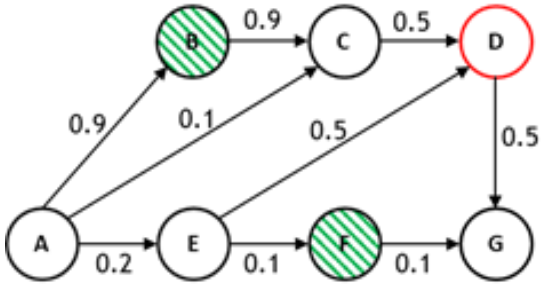
	B	C	D	E	F	G
inf[0]		.1	.1	.2		.05
inf[1]	.9	.81	.405		.02	
pre[0]		A	E	A		D
pre[1]	A	B	C		E	

Grey: visited nodes

Red: maximum inf[1] in unvisited nodes

Figure 5(g) Step 7

In Figure 5(g), we compute node *A*'s influence on *C*'s outgoing neighbor *D*. *C* is white so *inf*[1].*D* = 0.81*0.5 =0.405 (*A*→*B*→*C*→*D*). Therefore the next node with the maximal *inf*[1] is *D*, and we mark node *C* visited.



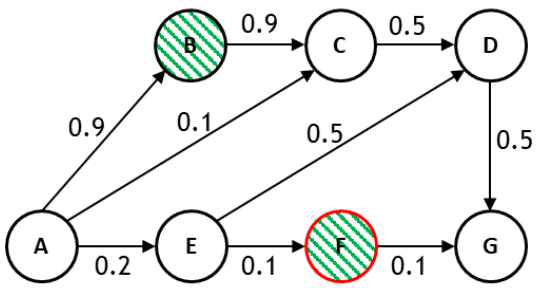
	B	C	D	E	F	G
inf[0]		.1	.1	.2		.05
inf[1]	.9	.81	.405		.02	.203
pre[0]		A	E	A		D
pre[1]	A	B	C		E	D

Grey: visited nodes

Red: maximum inf[1] in unvisited nodes

Figure 5(h) Step 8

In Figure 5(h), we compute the influence of node *A* on *D*'s outgoing neighbor *G*. *G* is white so *inf*[1].*G* = 0.405*0.5 =0.203 (*A*→*B*→*C*→*D*→*G*). *G* has no outgoing neighbor. We mark nodes *D* and *G* as visited. Now the only unvisited node with an *inf*[1] value is *F*.



	B	C	D	E	F	G
inf[0]		.1	.1	.2		.05
inf[1]	.9	.81	.405		.02	.203
pre[0]		A	E	A		D
pre[1]	A	B	C		E	D

Grey: visited nodes

Red: maximum inf[1] in unvisited nodes

Figure 5(i) Step 9

Figure 5. Exactly SIP-*t*

Finally, in Figure 5(i), we compute the influence of node *A* on *F*'s outgoing neighbor *G* which is 0.02*0.1=0.002. Because it is less than *inf*[1].*G* which is 0.203, we do not need to update its value and can mark node *F* as visited. At this point, every node that has an *inf*[1] value has been visited, so we can go to the next layer *inf*[2]. Since no path contains 2 green nodes, there are no values for *inf*[2], so the algorithm stops.

This algorithm computes the SIP including zero green node, the SIP including one green node, up to the SIP including *t* green nodes from a source node to all the other nodes in the graph. In our example, the source node is *A*. Given *t*=1 and destination node *d* = *C*, we look at the value for *inf*[1].*C* which is 0.405 (*A*→*B*→*C*). Given *t*=0 and destination node *d* = *D*, we look at the value for *inf*[0].*D* which is (*A*→*C*→*D*).

The k-Colors SIP-*t* algorithm is summarized as follows.

t-SIP-Path constraint Algorithm (Find all strongest influence paths form a node to any other nodes with exactly t nodes of color c.)

Input: Graph *G*, source node *s*, destination node *d*, number of nodes of color *c* constraint *t*, color *c*

Output: All strongest influence paths from *S* to any other nodes containing exactly *t* nodes of color *c*

1. Create vertex set *R* //object nodes
2. int *i, j*;
3. for each vertex *v* in *G*: //Initialization

```
4.        for i=0 to t:
5.            inf[i].v← -∞ //Unknown influence from source to v going through i nodes of
      color c
6.            prev[i].v← UNDEFINED //Previous node in optimal path from source
7.       for j=0 to t:
8.          while R ≠G:
9.               u ← vertex in O with max inf[j] //Node w/ the strongest influence will be
     selected next
10.              R =R ∪{u}
11.              for each neighbor w of u
12.             if j==0:
13.                  if color[w]== c:
14.                      inf[1].w=inf(u,w)
15.                      prev[1].w=u
16.                  else
17.                      inf[0].w=inf(u,w)
18.                      prev[0].w=u
19.             else
20.                  if color[w]== c:
21.                    if inf[j+1].w<inf[j].u*inf(u,w)
22.                      inf[j+1].w=inf(u,w)*inf[j].u
23.                      prev[j+1].w=u
24.                 else
25.                      if inf[j].w<inf[j].u*inf(u,w)
26.                          inf[j].w=inf[j].u*inf(u,w)
27.                          prev[j].w=u
28. return inf[t].d, prev[t].d
```

The time complexity of SIP-t is *O(tenlogn)*. This is because the complexity of Dijkstra's Algorithm is *e logn*, we need to keep track of *t* different influence levels and reset the unvisited nodes for each influence level which is another *n.*

Theorem 1: The *k*-Colors SIP-*t* algorithm returns the SIP with the *t* nodes of color *c* constraint.

Proof:

Let findSIPcConst(s, t, c) be a function that returns the SIP from the seed node s containing exactly t nodes of color c. We use proof of induction to prove the correctness. Our base case is to show findSIPcConst(s, 0, c) holds. The base case finds the SIP from the source to every

other node including exactly ‘0’ nodes of color c. That is, the SIP does not contain any node of c color.

Let inf[0].v be the label found by the algorithm and let δ[0].v be the strongest path influence from s-to-v. We want to show that inf[0].v = δ[0].v for every vertex v at the end of the algorithm, showing that the algorithm correctly computes the distances. We prove this by induction on $|R|$ via the following lemma: For each $x \in R$, inf[0].x = δ[0].x. We use proof by induction to prove the base case. The base case is when $|R| = 1$. Since R only grows in size, the only time $|R| = 1$ is when $R = \{s\}$. If color[s]==c, then inf[0].s remains - ∞ since the path from s to s including 0 node of color c does not exist. inf[0].s = 1 = δ[0].s, which is correct.

Let u be the last vertex added to R. Let $R' = R \cup \{u\}$. Our inductive hypothesis is for each $x \in R'$, inf[0].x = δ[0].x. By the inductive hypothesis, for every vertex in R’ that isn not u, we have the correct distance label. We need only show that inf[0].u = δ[0].u to complete the proof. If color[u]==c, we would not consider it because that path would contain at least 1 node of color c and wouldn’t meet the inf[0] constraint. If color[u]!=c, suppose for a contradiction that the strongest path from s-to-u with no node of color c is Q and has influence inf(Q) > inf[0].u.

Q starts in R' and at some leaves R'(to get to u which is not in R'). Let xy be the first edge along Q that leaves R'. Let Q_x be the s-to-x subpath of Q. Clearly: inf(Q_x) * inf(x, y) ≥ inf(Q). Since inf[0].x is the influence of the strongest s-to-x path by the I.H., inf[0].u ≥ inf(Q_x) , giving us inf[0].x * inf(x, y) ≥ inf(Q_x) . Since y is adjacent to x, inf[0].y must have been updated by the algorithm,

so inf[0].$y \geq$ inf[0].x * inf(x, y) . (Since inf[0].y is the length of the strongest s-to-y path by the I.H.)

Finally, since u was picked by the algorithm (It picks the node with the biggest distance label to be the next node to visit), u must have the biggest distance label out of all the unvisited nodes left (u, y, etc.): inf[0].$u \geq$ inf[0].y. Combining these inequalities in reverse order gives us the contradiction that inf[0].$u >$ inf[0].u. Therefore, no such stronger path Q must exist and so inf[0].u = δ[0].u. This lemma shows the algorithm is correct by "applying" the lemma for $R = V$. Therefore, the base case holds.

Our inductive hypothesis is that findSIPcConst(s, k, c) where $1 \leq k <$ t holds. This means inf[k].u contains the strongest influence value from s to u with exactly k nodes of color c. Using I.H., we show that findSIPcConst(s, k+1, c) holds. According to our algorithm, for any node v in graph G, where edge(u, v) exists, there are two cases. The first case is if the new visited neighbor v is not color c, this part of the algorithm is the same as the base case where color[v]!=c. Therefore, the same proof can be applied here. The second case is if the new visited neighbor v is color c, we use proof of contradiction to prove the correctness.

Suppose for a contradiction that the strongest path from s-to-v with k+1 node of color c is Q and has influence inf(Q) > inf[k+1].v. Inf[k].u is the strongest influence path from s to u with exactly k nodes of c based on our I.H. If inf[k].u * inf(u,v) is greater than inf[k+1].v, then inf[k+1].v is updated to inf[k].u * inf(u,v). A subpath of a strongest path is a strongest path. That means if inf[k+1].v is the strongest path from s-to-v, then inf[k].u is the strongest path from s-to-u. Since inf(Q) > inf[k+1].v, inf[k+1].v which is inf[k].u * inf(u,v) is not the strongest path. That means inf[k].u is not the strongest path. But based on our inductive hypothesis, inf[k].u is indeed the

strongest path from *s*-to-*u*. This gives us a contradiction. In both, findSIPcConst(*s*, *k*+1, *c*) returns a SIP with '*k*+1' nodes of *c* color. Consequently, findSIPcConst(*s*, *t*, *c*) holds for any *t*.

■

Chapter 7
k-Colors Strongest Influence Problem with *t* %constraint(*k-Colors*/ SIP-*t%)*

In this section, we present our approaches to find the strongest influence path with at most/exactly/at least *t* nodes of color *c* from a node to another node in a graph.

7.1. k-Colors SIP-t% (k-Colors Strongest Influence Path with t% Constraint)

In Chapter 6, we discuss the k-Colors SIP algorithm that includes exactly/at most/at least t nodes of a constraint color c. However, some applications might be interested in finding SIPs with a percentage constraint on the nodes of the same color.

Given a directed *k*-colors graph *G* where each edge (*a*,*b*) carries a weight (0, 1) that designates the probability of influence from node *a* to node *b*. The nodes are one of *k* different colors and the colors represent different types of nodes. The *k*-colors SIP-*t*% problem finds the SIP from the source node *s* to each destination node *d* which passes through exactly/at least/at most *t%* nodes that have the same color.

7.2. Use case of k-colors t%-SIP

Consider a corporate network that consists of different companies as shown in figure 6. Each node represents a company and each edge represents influence. Company *s* wants to reach out to company *d* about working on a project together. The colors of the nodes represent the types of companies. For example, red nodes represent IT companies, blue nodes represent material companies, white nodes represent software companies and green nodes represent hardware companies. The path highlighted in red is the strongest influence path from node *s* to node *d* including at most 15% IT companies. In the highlighted path, node *s* is the only red node,

representing an IT company. The percentage of red nodes on the following SIP is computed by total # of red nodes/total # of nodes = 1/8 = 12.5%.

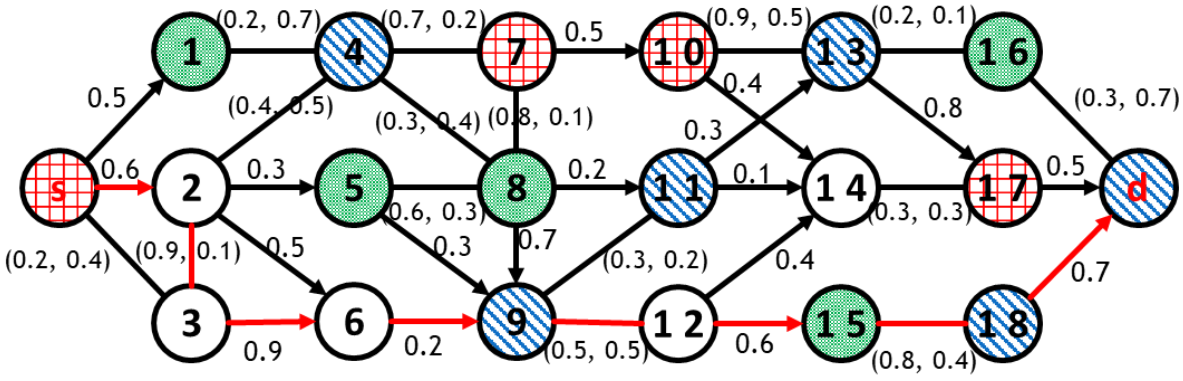


Figure 6. Example *k*-colors SIP-*t%*

7.3. Algorithm of at most k-Colors SIP-t%

Our basic idea is as following. We call the basic SIP function to obtain the SIP from node *s* to node *d*. Then, we check if the SIP satisfies the *t*% constraint. If it does, we return the SIP as the solution. If it does not satisfy the *t*% constraint, we explore alternative SIPs. To compare the SIP values of node *s*'s neighbors, we compute the SIP from *s*'s neighbors to *d* and multiply the influence value from *s* to its neighbors. For example, if *s* had neighboring nodes 1, 2 and 3, we would compare inf(*s*, 1)*SIP(1, *d*), inf(*s*, 2)*SIP(2, *d*) and inf(*s*, 3)*SIP(3, *d*). Out of the paths that meet the *t*% constraint, we pick the one with the maximum influence value. The count function counts the total number of nodes. For instance, count (SIP(*s*, *d*)) counts the total number of nodes on the SIP from node *s* to node *d* and count (SIP(*s*, *d*)$_{\text{green}}$) counts the total number of green nodes on the SIP from node *s* to node *d*.

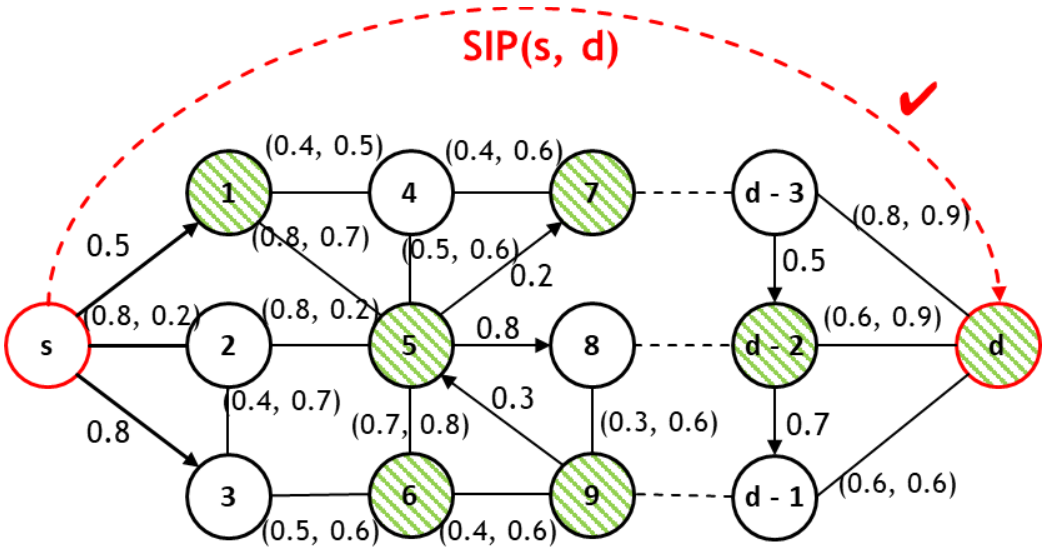


Figure 7(a) Step 1

Consider the example shown in Figure 7(a). First, we find the SIP from *s* to *d* as shown in Figure 5(a). The constraint on SIP(*s*, *d*) is that it has to satisfy $\frac{count(SIP(s,d)_{green})}{count(s,d)} \leq t\ \%$.

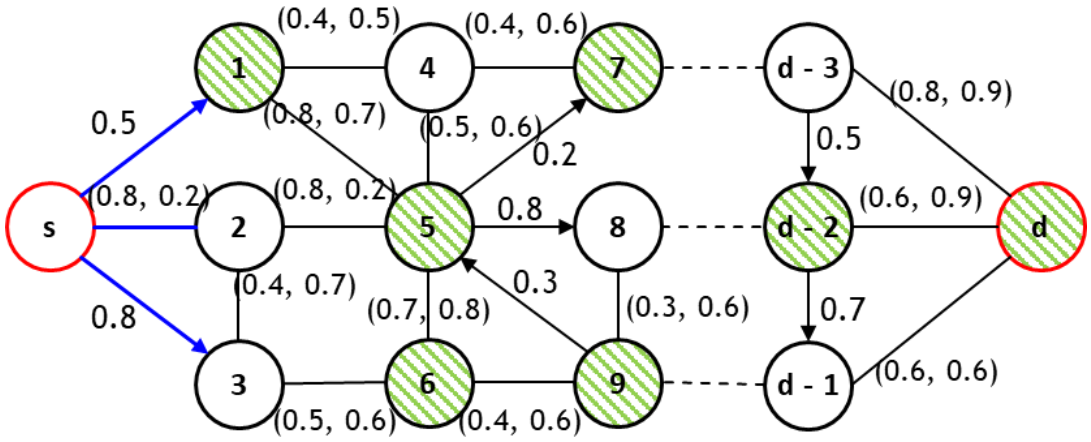


Figure 7(b) Step 2

So we check if SIP(*s*,*d*) meets the green node constraint. If it meets the constraint, SIP(*s*,*d*) is the solution. If it does not, we calculate the influence from the source node (node *s*) to all its neighbors (node 1, node 2, node 3) as shown in blue in Figure 7(b). Now, calculate the influence from node *s* to all its neighbors. The influence value from node *s* to node 1 is 0.5, denoted as *inf*(*s*, 1). So we have *Inf*(*s*,1)=0.5, *Inf*(*s*,2)=0.8 and *Inf*(*s*,3)=0.8.

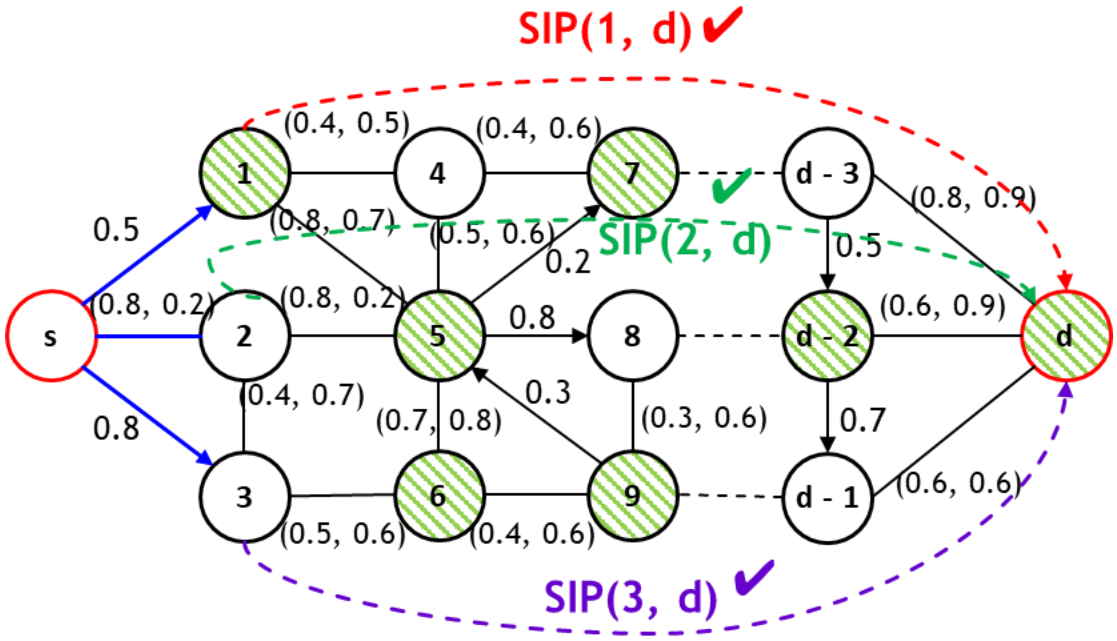


Figure 7(c) Step 3

Then, calculate the SIPs from *s*'s neighbors (node 1, node 2, node 3) to *d* as shown in Figure 7(c) which are SIP(1, d), SIP(2, d) and SIP(3, d). If the total number of green nodes on SIP from node 1 to node d over the total number of green nodes from node 1 to node d plus 1 (node s) is less than or equal to t% ($\frac{count(SIP(1,d)_{green})}{count(SIP(1,d))+1} \leq$ t %), we consider that path as a candidate path. If the conditions $\frac{count(SIP(1,d)_{green})}{count(SIP(1,d))+1} \leq$ t %, $\frac{count(SIP(2,d)_{green})}{count(SIP(2,d))+1} \leq$ t % and $\frac{count(SIP(3,d)_{green})}{count(SIP(3,d))+1} \leq$ t % are satisfied, then the result is the path with the highest influence value out of all the candidate paths: max{inf(s,1) * inf (SIP(1,d)), inf (s,2) * inf (SIP(2,d)), inf (s,3) * inf (SIP(3,d))}.

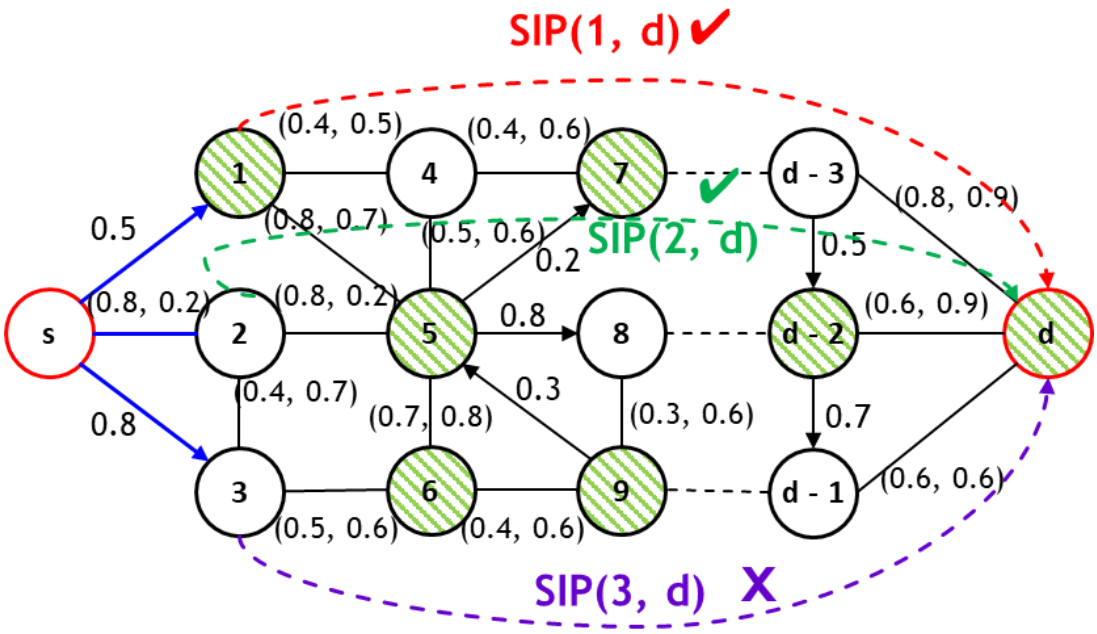


Figure 7(d) Step 4

Among the three candidate paths, we want to pick a path that meets the constraint and has the highest influence value. Because two SIPs, (s→SIP(1, d)) (let us call this path1) and (s→SIP(2, d)) (path2) satisfy the constraint and another (s→SIP(3, d)) (path3) does not satisfy the constraint as shown in Figure 7(d), we compare the influence value of each path. Since *inf* (path1) > *inf* (path2), we do not consider path2 as a solution. If *inf* (path1) > *inf* (path2) and *inf* (path1) > *inf* (path3), then the program ends and the result is path1.

In our example in Figure 7(d), since *inf* (path1) > *inf* (path2) and *inf* (path1) < *inf* (path3), we store the influence value of path1. Next, we consider alternative paths derived from path3 that can meet the constraint. The alternative paths derived from path3 in our example would be s→3→2→SIP(2, d) (path3.1) and s→3→6→SIP(6, d) (path3.2). Since SIP(2, d) or SIP(6, d) could contain more green nodes or less green nodes than path3, these alternate paths may meet the green node constraint. In this example, the constraint is to have no more than 30% green nodes. Assume path3 is s→3→6→5→7→11→15→21→25→d (4 green nodes in the path) and path 3.1 is

s→3→2→5→8→13→19→23→27→*d* (2 green nodes in the path). Path 3.1 meets the constraint while path3 does not. Path 3.2 in this case would be the same path as path3 since path3 is *s*→SIP(3,*d*) which goes through node 2 and path 3.2 is *s*→3→6→SIP(6, *d*). Therefore, they have the same influence values. We then compare the influence value of path 3.1 with the influence value of path1 and choose the path with a higher influence value. We do this since path3 has the highest influence value among the three candidate paths but does not meet the constraint. The alternative paths derived from path3 have a high likelihood of having a higher influence value than path1 with the constraint satisfied.

Note that since path1 meets its green node constraint, we do not need to consider alternate paths of path1 since path1 contains the SIP from node 1 to node *d*. Alternate paths of path1 have weaker influence values than path1. If an alternate path of path1 had a higher influence value than path1, then that path would have been the path1. Path2's influence value is lower than path1's influence value. Path2's alternate paths have weaker influence values than path2 so we do not need to consider alternate paths of path2. The only reason we are considering alternate paths of path3 is because path3 has a stronger influence value than path1 so alternate paths of path3 might still be stronger than path1 while meeting the green node constraint.

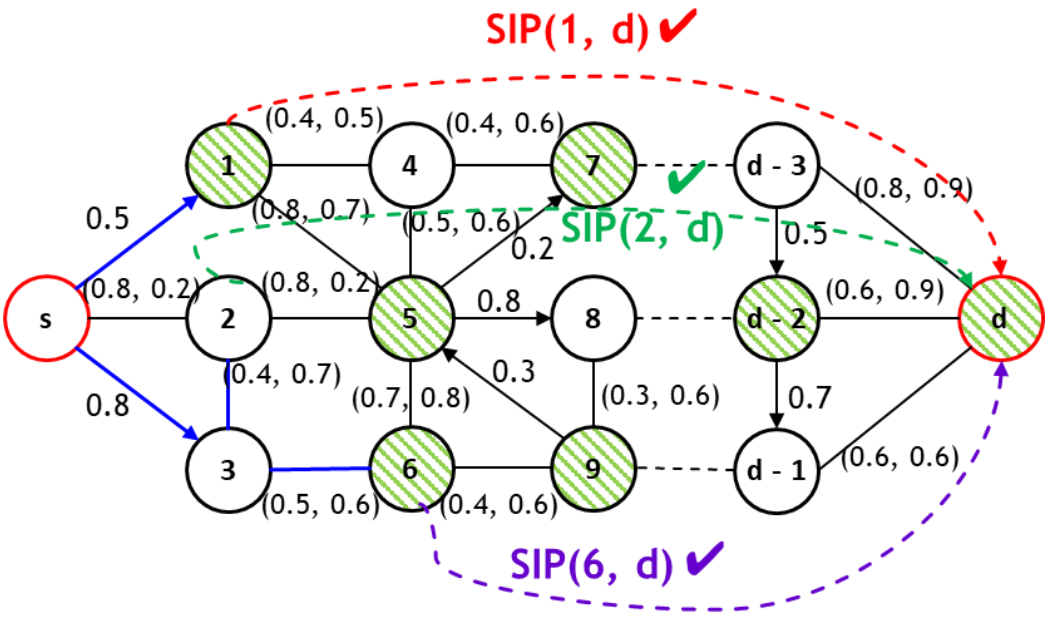


Figure 7 (e) Step 5

Now, we compute the influence values of path3's alternate paths. We calculate the influence from node 3 to all its neighbors which are nodes 2 and 6. *Inf* (3, 2) =0.7 and *inf* (3, 6) = 0.5. Next, we calculate the influence from the source node *s* to node 3 to its neighbors. *Inf* (*s*→ 3→2) = 0.56 and *inf* (*s*→ 3→6) = 0.35. Then, we calculate SIP(2, *d*) and SIP(6, *d*) as shown in Figure 7(e).

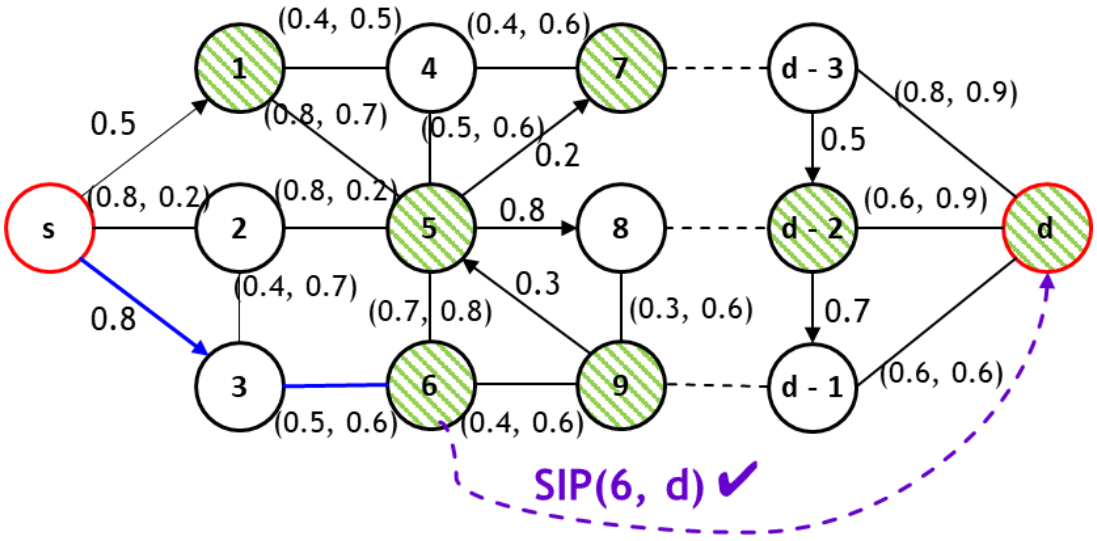


Figure 7 (f) Step 6

Figure 7. At most SIP-*t%*

As shown in Figure 7(f), we check if the alternate paths of path3 meet the green node constraint conditions $\frac{count(SIP(2,d)_{green})}{countSIP((2,d))+2} \leq$ t % and $\frac{count(SIP(6,d)_{green})}{count(SIP(6,d))+2} \leq$ t %. If these conditions are satisfied, then the result is the path with the highest influence value out of all the candidate paths: max{inf (s,1) * inf (SIP(1,d)), inf (s,3) * inf (3,2) * inf (SIP(2,d)), inf (s,3) * inf (3, 6) * inf (SIP(6,d))}. If none of the paths from this step satisfies the t% constraint, repeat the steps until we get the result.

The algorithm is as follows.

SIP-Path t% node of color c constraint Algorithm (Find the strongest influence path from a source node to destination node with exactly t % nodes of color c.)

Input: **Graph *G*, Source Node *s*, Destination Node *d*, percentage *t* of colored node constraint, color *c***

Output: **The strongest influence paths from *s* to *d* containing exactly *t*% nodes of color *c***

```
1.  char findSIP_t_Percent(G, s, d, t, c){
2.  if (checkTP(SIP(s,d), t, c))
3.      return SIP(s, d);
4.  else
5.      char pathShared=[s];
6.      char candidatePaths[];
7.      getAltPaths(pathShared, candidatePaths, t, c);
8.      candidatePaths.sort(key=inf, reverse= True);
9.  return candidatePaths [0];
10. }
11. bool check_t_Percent(path, t, c){
12.     int cNode=0;
13.     bool meetConstraint=false;
14.     for each node p in path:
15.         if p.color == 'c'
16.             cNode++;
17.     if cNode/len(path)*100==t
18.         meetConstraint=true;
19.     return meetConstraint;
20. }
21. void getAltPaths(pathShared, candidatePaths, t, c){
22.     float cMax=0;
23.     u=pathShared[-1];
24.     for every neighbor v of u:
25.         tempPath= pathShared.concat(SIP(v,d))
26.         if (inf(tempPath)>cMax)
27.             if(check_t_Percent(tempPath, t, c)
28.                 cMax=inf(tempPath);
29.                 candidatePaths.append(tempPath);
30.             else
```

```
31.         pathShared.append(v);
32.         getAltPaths(pathShared,        candidatePaths, t, c);
33. }
```

Theorem 2: The *k*-Colors SIP-*t*% algorithm returns the SIP with *t*% nodes of color *c* constraint.

Proof:

That is, findSIP_t_Percent(*s, d, t, c*) returns the specified SIP. Let the original SIP(*s, d*) be SIP_{orig} (lines 2, 3). If SIP_{orig} does not meet *t*% constraint, we explore alternate paths (lines line 5~7). Let *alt_Paths* be the alternate paths of SIP_{orig}. This involves a recursion of *n* iterations until it no candidate alternative path has an influence value exceeding a current maximum influence (lines 21~33). This part refers to "if (inf(*tempPath*) > cMax)" in the algorithm.

This holds the following;

(1) All the candidate alternative paths have been explored.

(2) At the end of the last recursive step, each path in the resulting list of candidate paths starts with *s* and ends at *d*.

Now, sorting the paths in the resulting list and choosing the path with the highest influence value must be the final resulting SIP satisfying the constraint (lines 8, 9). That is, having *t*% of *c* color nodes.

This completes the proof.

■

Theorem 3: The *k*-Colors SIP-*t*% algorithm is an NP-complete problem.

Proof: We first show that the decision version of SIP-*t*% is NP-complete. Then we can reduce the decision version to the optimization version. The decision version of SIP-*t*% is a problem that determines if there exists a path *p* from *s* to *d* that has inf(*p*)≥ *m* and meets the *t*% constraint.

First, we prove that the decision SIP-*t*% is in NP. We shows that given a solution you can verify it in polynomial time. Given a path, it is easy to check if the influence is greater than m and the ratio of nodes of color *c*/all nodes is *t*%. We can multiply the edge weights to check if the path weight is greater than *m* in polynomial time. We can compute the influence and (number of nodes of color *c* in path / number of all nodes in path)*100 to see if it is exactly *t*%. This can be done in polynomial time since counting the number of nodes takes less than edge *e* time. So SIP-*t*% is in NP.

Next, we show the knapsack problem is a special case of SIP-*t*% problem such that *t*=100 in the knapsack. We represent the items in the knapsack in a graph form in which every item is one or more nodes of color *c*, as shown in Figure 7. Each node is color *c* and each weight is 1. The number of nodes of color *c* on the path should be less than the weight constraint *w*. The number of nodes of color *c* on the SIPs could vary from 0 to *w*. Since the weight of each item is different, we link the nodes together to represent different weights. For instance, if the weight of an item is 3kg, we link 3 nodes together with no other outgoing edges. For example, once node 1 is reached, the only outgoing edge is to node 2 and the only outgoing edge from node 2 is to node 3. Each pink bubble in Figure 8 represents an item using a group of nodes. The number of nodes in each bubble represents the weight of the item. Then, we can reduce knapsack to a special case of SIP-*t*%.

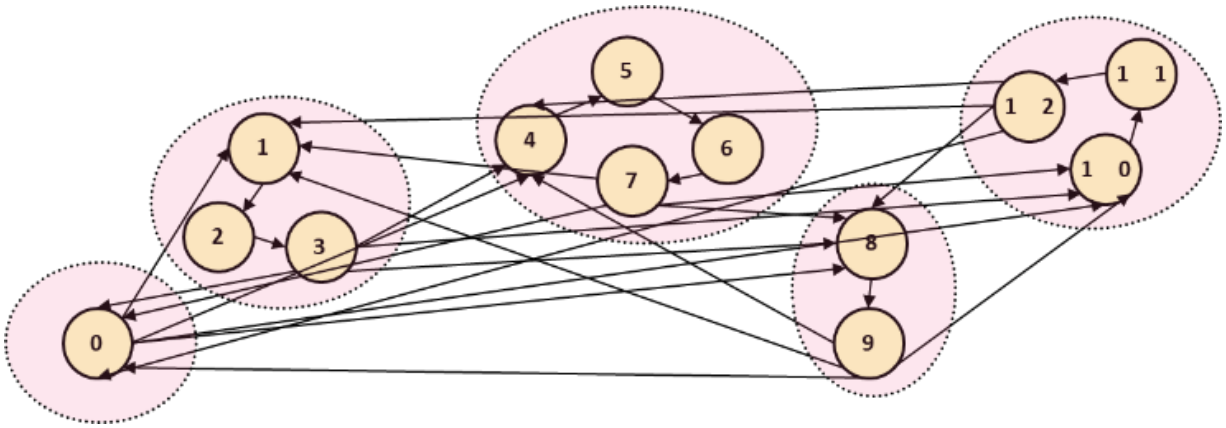


Figure 8. Knapsack Conversion

We create a knapsack problem that has non-negative weights a_1, a_2, $\cdots$, a_n, and profits c_1, c_2, $\cdots$, c_n. The problem is to determine if there a subset of weights with total weight at most B, such that the corresponding profit is at least K. The SIP-t% problem has non-negative nodes n_1, n_2, $\cdots$, n_n and influence weights e_1, e_2, $\cdots$, e_n. The problem here is to determine if there a path consisting of nodes with at most t% nodes of color c such that the influence of the path is at least M ($0<M<1$).

The equivalent parameters between the knapsack problem and the SIP-t% problem, are as follow: $a_i=n_i$, $c_i=e_i$, $B=T$, and $K=M$. The knapsack problem is to solve "Given n items with weights a_1, a_2, $\cdots$, a_n, a combination of these weights will be at most B." This is a special case of SIP-t% where t=100. Since all weights are of color c, t will be 100. The profit of a combination of items c_1, c_2, $\cdots$, c_n ($1<c_i<10$), will be at least K. We convert c_i to $\log(c_i)$ and take the product of each profit value to get the total profit. We want to determine if there exists a subset $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i\in S} a_i \leq B$ and $\prod_{i\in S} \log(c_i) \geq K$? Knapsack is a known NP-complete problem. We show that Knapsack $\leq$ p SIP-t%. The following deduction implies the new problem is equivalent to the original problem: $\sum_{i\in S} a_i \leq B \leftrightarrow \sum_{i\in S} n_i \leq T$ and $\prod_{i\in S} \log(c_i) \geq K \leftrightarrow \prod_{i\in S} e_i \geq M$. Therefore, the decision version of SIP-t% is an NP-complete problem. Since the regular SIP-t% is a more difficult problem than the decision version of SIP-t%, SIP-t% is an NP-complete problem.

■

Chapter 8
A Heuristic Algorithm for *k-Colors* Strongest Influence Problem with *t* %constraint (*k-Colors*/ SIP-*t*%)

In this section, we present our approximation algorithm to find the strongest influence path from a node to another node including exactly/at least/at most *t*% nodes of color *c* in a graph.

8.1. An Heuristic SIP-t% problem

The SIP-*t*% algorithm is an NP-complete algorithm. For large graphs, it would take a significant amount of time. We propose a heuristic algorithm with a significantly better running time of $O(mn^3)$.

8.2. Algorithm of SIP-t%

The following is an example of the SIP-*t*% problem. We want to find the SIP from the source node *s* to the destination node *d*, which includes at most 33% nodes of color *blue*.

Step 1: As shown in Figure 9(a), obtain the original graph *G* and confirm the source node *s* and the destination node *d*.

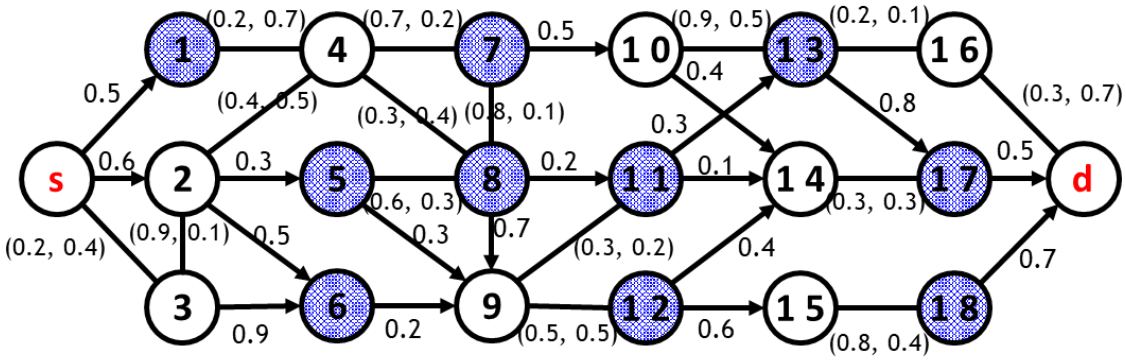


Figure 9 (a) Step 1

Step 2: Apply the algorithm of Top-*m* SIP and calculate the nodes of color *c* %.

As shown in Figure 9(b), we call the Top-*m* SIP algorithm which returns SIP-1, SIP-2, …, SIP-*m*. Then, calculate the ratio of blue nodes on the path to all the nodes on the path. For example, SIP-1 has number of blue nodes on SIP-1/number of all nodes on SIP-1 = 3/8=37.5%. SIP-2 has 3/9=33.3% blue nodes. SIP-m has 4/9 = 44.4%.

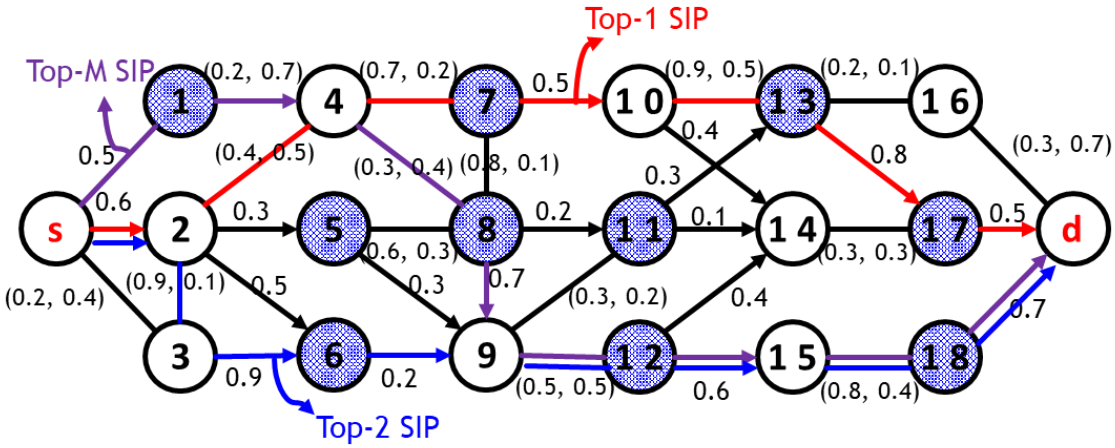


Figure 9(b) Step 2

Figure 9. Heuristic Algorithm SIP-*t%*

Step 3: Starting with SIP-1, check if the color *c* node % constraint is met. If SIP-1 does not satisfy the constraint, check SIP-2, SIP-3 up to SIP-*m*. Let us say Top-3 SIP has 40% blue nodes and SIP-4 has 32.5% blue nodes. We pick SIP-4 to be the solution.

Step 4: However, if we cannot find the result through the Top-*m* algorithm, meaning none of the SIPs found through the top-*m* algorithm meets the blue nodes constraint. We increase the value of *m* and repeat steps 2 and 3. The value of *m* should be increased to higher the chance of finding a solution. Repeat until a solution is found.

The algorithm is as follows.

SIP-Path t% node of color c constraint Approximation Algorithm (Find all strongest influence paths from a node to any other node with at most t% nodes of color c.)

Input: Graph *G*, Source Node *s*, Destination Node *d*, *m* for top-*m*, percentage of color *c* constraint *t*, color *c*,

Output: The strongest influence path from *s* to *d* containing exactly *t*% nodes of color *c*

1. Store paths to result[]=topmSIP(*s, d, m*)
2. For each path *p* in result:
3. if count(*p.c*)/count(*p*)*100 ==*t*
4. return *p*
5. *m*=exponentialBackoff(*s, d*)
6. SIPTPercent(*G, s, d, m, t, c*)

The time complexity of SIP-*t*% heuristics is the same as that of the top-m SIP algorithm, $O(mn^3)$. The complexity of the top-m SIP is mn^3. After the top-m SIP algorithm is called, the *m* SIP nodes of color *c* constraint % is calculated and checked, which takes *m* times. Even though steps 2 and 3 could be repeated, it is minor so the total time complexity is $mn^3 + m$ *which is* $O(mn^3)$.

We may use exponential backoff [86] to find an acceptable number for the value of *m*. Exponential backoff is an algorithm that uses feedback to multiplicatively decrease the rate of some process, in order to gradually find an acceptable rate. $m=n^c - 1$ and the initial value of *m* can be set to the total number of nodes in the graph-1 and c is the number of times the algorithm is running the SIP-*m*. If the initial run of SIP-*m* does not yield a result, re-run the SIP-*m* with an updated value of *m* where *c* =2 since it is the second time running SIP-*m*. If it still does not work, re-run it with an updated value of *m* with *c*=3. As the number of re-run attempts increases, the number of possibilities for finding the solution increases exponentially.

Chapter 9
Influence Maximization Problem based on SIP (IM-SIP)

In this section, we present our approaches to finding the influence maximization problem based on the SIP model in a graph.

9.1. Definition of IM-SIP

Given a graph $G(V,E)$, find s nodes which can influence G maximally. With inputs G and s and output S that maximizes $\{\sum \mathrm{SIP}(S \rightarrow V)\}$, where S is the set of all the seed nodes and V is the set of all the nodes in the graph G.

9.2. Assumptions of IM-SIP

The IM-SIP problem is very complex. In order to simplify our problem, we make three assumptions:

Assumption 1: We only consider the influence value of SIP as the influence from one node to another node.

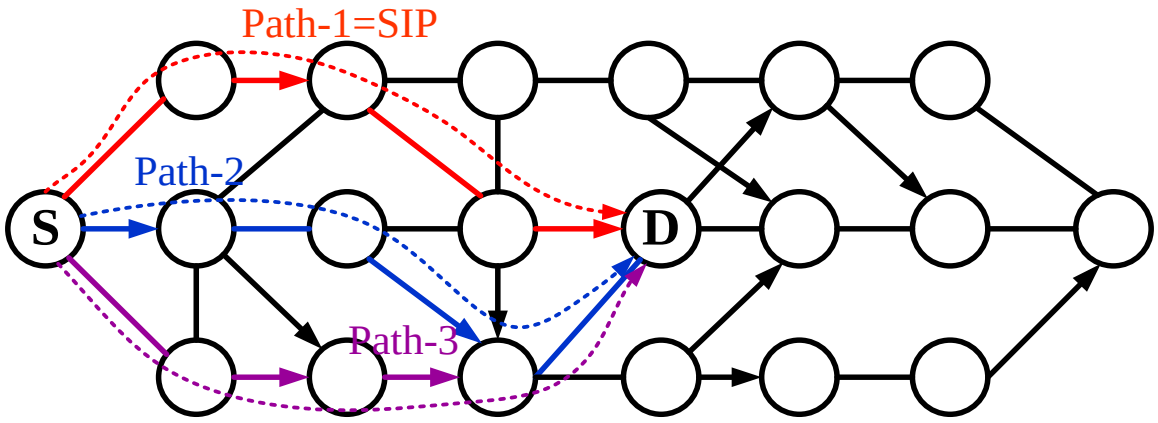


Figure 10. Assumption 1

For example, consider the network shown in Figure 10, from node *S* to node *D* there are many paths (more than 3), we only consider the influence value of the SIP as the influence value from node *S* to node *D*. In Figure 9, path 1 is the SIP from node *S* to node *D* so the influence value of path 1 is the influence from node *S* to node *D*. Without this assumption, there would be multiple influence values from one node to another.

Assumption 2: Assume that when computing multiple nodes' combined influence on a single node, only the SIP that has the strongest influence value influences that node.

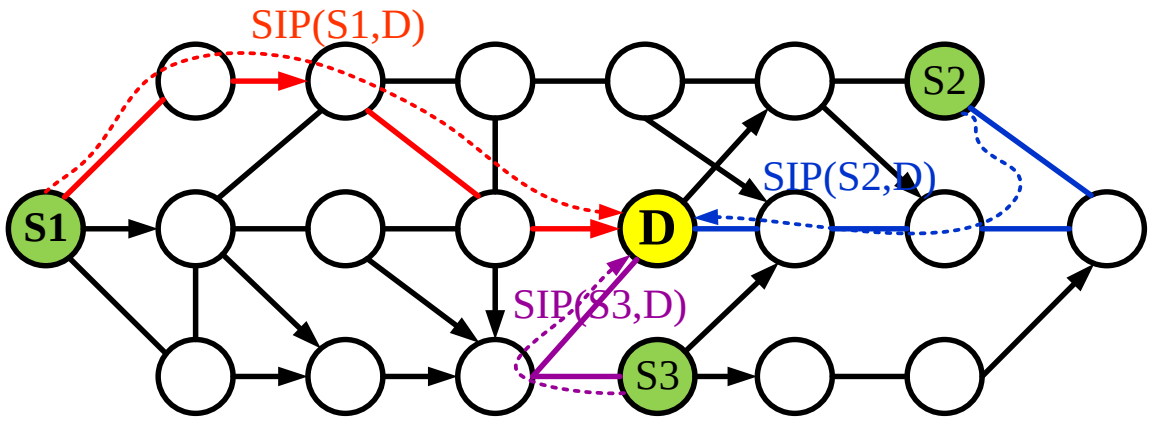


Figure 11. Assumption 2

That is the combined SIP: cSIP(S, y)= max $x \in S$ {SIP(x, y)}, where S is the given set of nodes. For example, as shown in Figure 11, the combined influence of *S1*, *S*2 and *S*3 on *D* is the strongest influence value among the influence values of SIP(*S*1, *D*), SIP(*S*2, *D*) and SIP(*S*3, *D*).

Without this assumption, we would add up the influence of the 3 paths to the destination node. This is because based on the nature of our SIP model, the influence from a node to another node is the strongest influence path (assumption 1). Let us say SIP (*S1*, *D*) is *S1*→ 2→ 3→5→7→*S2*→14→11→9→*D*, SIP(*S2*, *D*) is *S2*→14→11→9→*D* and SIP(*S3*, *D*) is *S3* →11→9→*D*. If we were to add up all the influences, the influence value of 11→9→*D* would be counted 3 times and the influence value of 14→11→9→*D* would be counted twice. This is

redundant in counting the same influence more than once. Therefore, we use the maximum function for our model.

Assumption 3: The influence from one node to itself is equal to 1 (as shown in Figure 12).

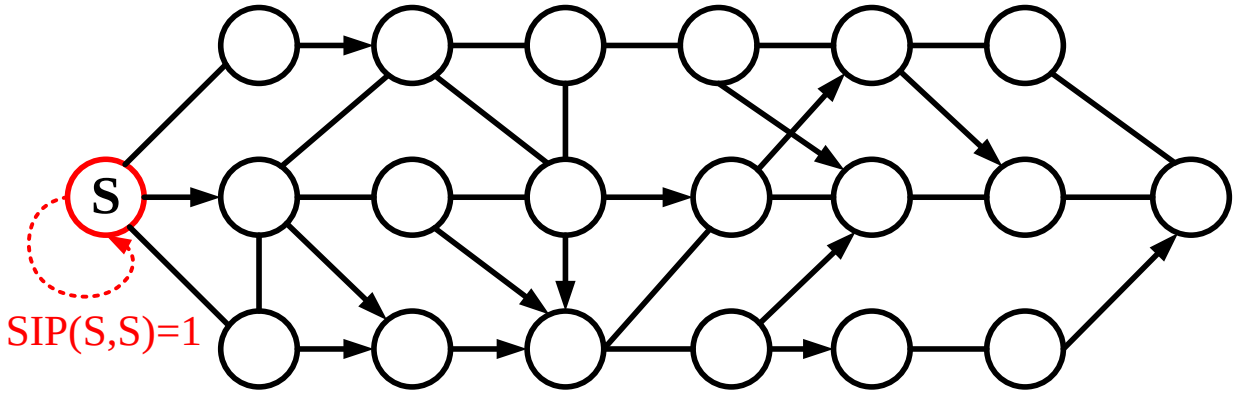


Figure 12. Assumption 3

9.3. Algorithm of IM-SIP

The brute force IM-SIP can be described as the following. We want to find *s* seed nodes which influence the entire graph maximally.

First, we obtain all possible combinations of s seed nodes. Let us say *s*=3 in our example. In Figure 13(a), the possible *s* seed sets would include {0, 1, 2}, {0, 1, 3}, {0, 1, 4}, …, {8, 9, 10}.

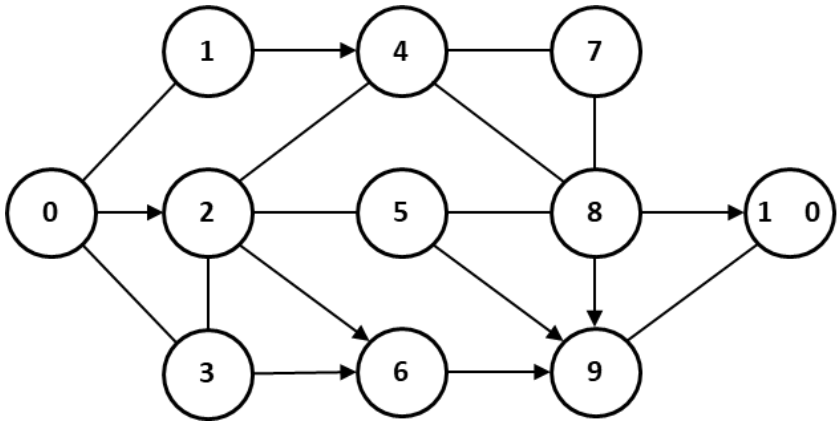


Figure 13 (a) Step 1

Then, we calculate the combined influence of each seed set. For example, in order to compute the combined influence of a seed set {1, 2, 8}, we use the maximum function. In Figure 13 (b), the

non-seed nodes are 0, 3, 4, 5, 6, 7, 9 and 10. As stated in assumption 2, we choose the seed node that has the strongest influence on each of these non-seed nodes. If SIP(1, 0)= 0.7, SIP (2, 0) = 0.56, and SIP(8, 0)= 0.252, cSIP({1, 2, 8}, 0)= max (SIP(1, 0), SIP (2, 0), SIP (8, 0))=max(0.7, 0.56, 0.252) = 0.7. Then we repeat the same steps for the next non-seed node, which is node 3. cSIP({1, 2, 8}, 3)= max (SIP(1, 3), SIP (2, 3), SIP (8, 3))=max(0.64, 0.8, 0.28) = 0.8. We sum up the cSIPs from the seed nodes to nodes in the entire graph to compute the combined influence and repeat this step for each combination of the *s* seed nodes.

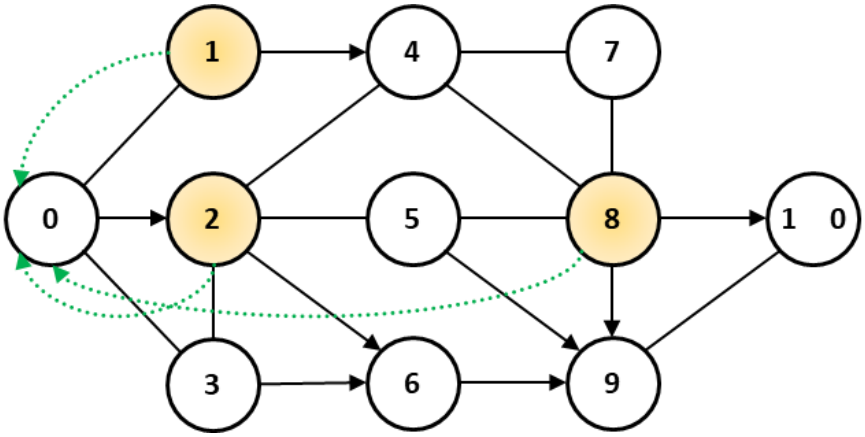


Figure 13(b) Step 2

Figure 13. IM-SIP

Next, we choose the *s* seed nodes with the highest combined influence. It takes *s***n* to compute the combined influence from a seed set to a node. We repeat these steps for $\frac{n!}{s!(n-s)!}$ times. IM-SIP is NP-complete, as the time complexity of SIP is very large: $O(\frac{n!}{s!(n-s)!} * s * n)$.

Table 1 shows the number of different combinations of *s* seed nodes that can be created from a given number of nodes. As one can see, as the value of *s* increases, the number of combinations consisting of *s* nodes increases exponentially. As the size of the graph increases and the value of *s* increases, it could take weeks, months and even years.

Table 1. Number of s node combinations

# of nodes	s	Combination
100	1	100
100	2	4,950
100	3	161,700
100	4	3,921,225
100	5	75,287,520
100	6	1,190,052,400
100	7	16,007,560,800
100	8	186, 087,894,300
100	9	1,902,231,808,400
100	10	17,310,309,456,440

9.4. Definition of IM-SIP-T

Given a graph $G(V,E)$, find s nodes which can influence G maximally. With inputs G, s, t and c output S with exactly t nodes of color c that maximizes $\{\sum \mathrm{SIP}(S \rightarrow V)\}$, where S is the set of all the seed nodes and V is the set of all the nodes in the graph G.

9.5. Use case of IM-SIP-T

Consider a citation network as an example, shown in Figure 14, where each node represents a publication and each edge represents the direction of the information flow (e.g., edge (2, 5) shows node 5 cited node 2).

The node colors represent the publication year. For example, the white nodes are publications between 1996 and 2000, and the blue nodes are publications between 2001 and 2005, the red nodes are publications between 2006 and 2010, and the yellow nodes are publications between 2011 and 2015.

The edge weights represent the strength of the influence. For example, an edge weight of 0.9 would indicate that the publication citing the corresponding publication in a positive way. An edge weight of 0.1 would indicate that the publication was cited in a negative way. We want to

find five most influential publications in the network including at least 3 that were published between year 2011 and 2015 (recent).

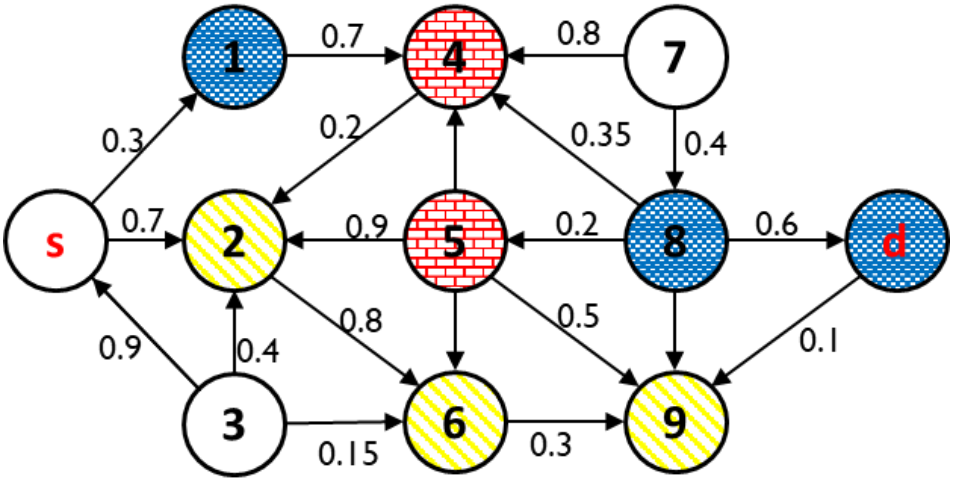


Figure 14. Citation network

9.6. Algorithm of IM-SIP-T

The brute force IM-SIP can be described as the following. We want to find *s* seed nodes including exactly *t* seed nodes of color *c* which influence the entire graph maximally.

Step 1: Compute every possible combination of *s* seed nodes from graph.

Step 2: From the candidate seed set list from step 1, remove the ones that do not meet the *t* colored nodes constraint. (Remove the ones that have less than or equal to *t* nodes of color *c*.

Step 3: From the remaining list of step 2, compute the combined average influence for each candidate seed set.

Step 4: Sort based on the combined average influence value.

Step 5: Return the top set from step 4.

The following heuristic algorithm is for the IM-SIP problem which is NP-complete. The heuristic algorithm is greedy – It builds the set of seed nodes incrementally – in every iteration it

brings in a new node that combined with the nodes in the current set of seed nodes has the highest combined influence on the remaining nodes.

Chapter 10
A Greedy Algorithm for k-Colors IM-SIP-t

As shown in the previous section, the brute force algorithm of IM-SIP is NP-complete. In order to improve the running time, we propose a greedy algorithm to solve the problem of finding *s* seed nodes from graph *G*, of which there are exactly *t* nodes of the same color, which influence the whole graph maximally.

10.1. Definition of k-Colors IM-SIP-T

Find *s* seed nodes from graph *G*, of which there are exactly/ at most/ at least *t* nodes of the same color, which influence the whole graph maximally.

Given a graph *G*(*V*,*E*), find s nodes which can influence the graph *G* maximally. With inputs *G*, *s* and *t*, and output arg max{$\sum$ SIP(<*S*,*t*>$\rightarrow$*V*)}, *s* is seed nodes, *t* is the number of nodes of a chosen color, and *V* is all nodes in the graph *G*.

10.2. Algorithm of k-Colors t-IM-SIP

The following heuristic algorithm is for a k-color network and we look for exactly t nodes of the same color that influence the entire network the most. Because the IM-SIP problem is NP complete, it is obviously that the k-Colors t-IM-SIP problem is NP-complete also (The IM-SIP problem is a special case of the k-Colors t-IM-SIP problem where k is 1 .)

Step 1: Check if the values of *s*, *t* and *c* are validate inputs. Check for *s* being larger than the size of the graph, *t* being larger than *s*, the number of nodes of color *c* in graph *G* being too large (the number of nodes not of color *c* in graph G is less than *s*-*t*) or too small (graph *G* contains less than *t* nodes of color *c*).

Step 2: Compute the influence of each node on the entire graph and store these values.

Step 3: Sort the nodes based on their average influence values.

Step 4: From the sorted list, choose the node from the top (node with the highest average influence value) to be a seed node.

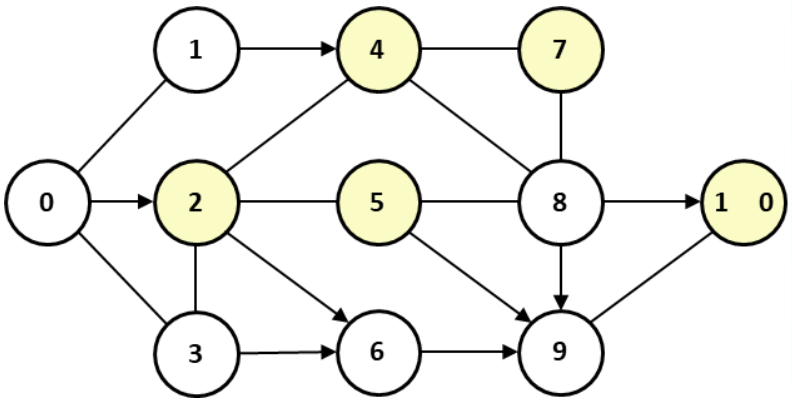
Step 5: Among the nodes that are not in the set of seed nodes, for each one of them compute the combined influence of the set of seed nodes on the entire graph and sort them based on the combined average influence values.

Step 6: With the node with the highest combined influence value with the existing seed nodes, check for the color nodes constraint. Keep going down the sorted list until a node that doesn't violate the color nodes constraint is found and add it to the seed set.

Step 7: Repeat steps 5 and 6 until all *s* seeds are selected.

Step 8: Return *s* seed nodes selected from the list.

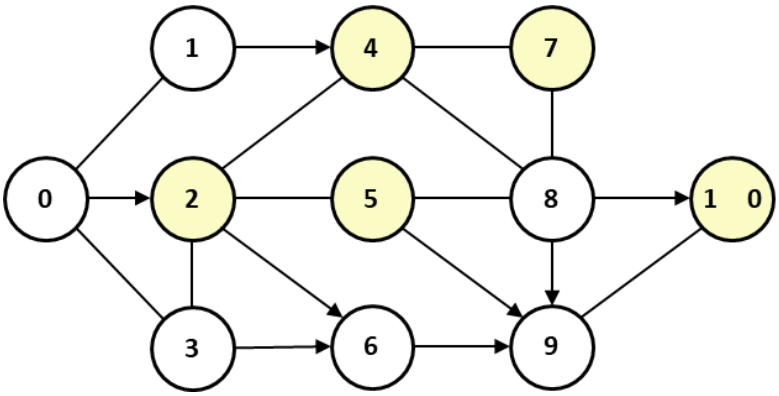
In our example, let us say we want to find 4 seed nodes including exactly 2 yellow nodes. First, compute the influence of each node to the entire graph as shown in Figure 15 (a).



Node	Influence
0	3.93
1	2.88
2	6.25
3	4.17
4	5.54
5	4.69
6	3.21
7	3.89
8	7.72
9	0.7
10	0.8

Figure 15 (a) Step 1

Next, sort the nodes based on their influence value as shown in Figure 15(b).



Node	Influence
8	7.72
2	6.25
4	5.54
5	4.69
3	4.17
0	3.93
7	3.89
6	3.21
1	2.88
10	0.8
9	0.7

Figure 15 (b) Step 2

Then, from the sorted list, we choose the node from the top (node with the highest influence) as the initial seed node. We keep track of the current count of yellow nodes and the current count of non-yellow nodes of the seed set. In Figure 15(c), we initialize the current yellow node count and the non-yellow node count in the seed set to be 0s. If the current non-yellow node count is *s-t*, and the yellow node count is *t*, we end the program and return the seed set. If the node color is yellow, check if the current yellow node count is less than *t*. If it is, increase the current yellow node count and append the node to the seed set. If the node color is not yellow, check if the current non-yellow node count is less than *s-t*. If it is, increase the current non-yellow node and append the node to the seed set.

Node	Influence
8	7.72
2	6.25
4	5.54
5	4.69
3	4.17
0	3.93
7	3.89
6	3.21
1	2.88
10	0.8
9	0.7

yellow	non-yellow	Seed set
0	0	

Figure 15 (c) Step 3

We pick the first node from the sorted list. As in Figure 15(d), the first node is 8. As it is a non-yellow node, we check to see if the current count of non-yellow nodes is less than *s*-*t*, which is 4-2=2. Since 0<2, we increase the non-yellow node count and add node 8 to the seed set.

Node	Influence
8	7.72
2	6.25
4	5.54
5	4.69
3	4.17
0	3.93
7	3.89
6	3.21
1	2.88
10	0.8
9	0.7

yellow	non-yellow	Seed set
0	0	
0	1	8

Figure 15 (d) Step 4

Next, we calculate the combined influence value of each node with the existing seed node. For example, the combined influence of node 2 would be the combined influence of node 2 and existing seed node 8 to the entire graph. Shown in Figure 15(e), we see the combined influence of each node with node 8. The node with the strongest combined influence with the existing seed node 8 is 2. As it is a yellow node, we check to see if the current count of yellow nodes is less than *t*, which is 2. Since 0<2, we increase the yellow node count and add node 2 to the seed set.

Nodes	Combined Influence
8, 2	8.94
8, 4	8.88
8, 3	8.71
8, 5	8.58
8, 0	8.42
8, 7	8.34
8, 6	8.28
8, 1	8.11
8, 10	7.92
8, 9	7.82

yellow	non-yellow	Seed set
0	0	
0	1	8
1	1	8, 2

Figure 15 (e) Step 5

We need 4 seed nodes and have 2 seed nodes so far. We repeat the previous step and recalculate the combined influence value of each node with the existing seed nodes 8 and 2. As shown in Figure 14(f), node 0's combined influence with nodes 8 and 2 is 9.64. Since node 0 has the strongest combined influence and it is a non-yellow node, we check to see if the current count of non-yellow nodes is less than *t*. Since 1<2, we increase the non-yellow node count and add node 0 to the seed set.

Nodes	Combined Influence
8, 2, 0	9.64
8, 2, 3	9.61
8, 2, 1	9.43
8, 2, 7	9.38
8, 2, 4	9.35
8, 2, 6	9.21
8, 2, 5	9.18
8, 2, 10	9.14
8, 2, 9	9.04

yellow	non-yellow	Seed set
0	0	
0	1	8
1	1	8, 2
1	2	8, 2, 0

Figure 15 (f) Step 6

We need 4 seed nodes and have 3 seed nodes so far. We repeat the previous step. Shown in Figure 15(g), the node with the strongest combined influence with the existing see nodes 8, 2 and 0 is node 1. As it is a non-yellow node, we check to see if the current count of non-yellow nodes is less than *s*-*t*, which is 2. Since 2=2, we skip this node and go to the next node.

Nodes	Combined Influence
8, 2, 0, 1	10.04
8, 2, 0, 7	10.03
8, 2, 0, 4	10.01
8, 2, 0, 3	9.96
8, 2, 0, 5	9.91
8, 2, 0, 6	9.85
8, 2, 0, 10	9.84
8, 2, 0, 9	9.74

yellow	non-yellow	Seed set
0	0	
0	1	8
1	1	8, 2
1	2	8, 2, 0

Figure 15(g) Step 7

As shown in Figure 15(h), the next node is 7. As it is a yellow node, we check to see if the current count of yellow nodes is less than *t*, which is 2. Since 1<2, we increase the yellow node count and add node 7 to the seed set. The yellow node count 1is equal to the value of *t* which is 2, and the non-yellow node count is equal to the value of *s*-*t* which is 2, we terminate the program. Finally, we return the seed set.

Nodes	Combined Influence
8, 2, 0, 1	10.04
8, 2, 0, 7	10.03
8, 2, 0, 4	10.01
8, 2, 0, 3	9.96
8, 2, 0, 5	9.91
8, 2, 0, 6	9.85
8, 2, 0, 10	9.84
8, 2, 0, 9	9.74

yellow	non-yellow	Seed set
0	0	
0	1	8
1	1	8, 2
1	2	8, 2, 0
2	2	8, 2, 0, 7

Figure 15 (h) Step 8
Figure 15. IM-SIP-T Greedy

The complexity of this greedy algorithm is O($en^2\log n$). It takes O($e\log n$) to run the SIP algorithm. We need to calculate the SIP from each node in the graph to every node in the graph. This takes O($en^2\log n$). When we calculate each node's combined influence, it takes O(n) since we don't need to calculate the SIP value from one node to another. We simply look it up from the stored values done in the first step. Recalculating the combined influence is repeated *s*-1 times so together, it is O(sn). The value of *s* can be at most *n* so the worst case is O(n^2). O($en^2\log n$) + O(n^2) = O($en^2\log n$).

Theorem 4: δSIP(S) is monotonic. i.e. **δ**SIP ($S \cup \{v\}$) ≥**δ**SIP (S)

Proof:

We prove that when we add node v to the seed set S, the overall influence to the entire graph goes up. The SIP algorithm calculates the overall influence of a node by the following:

```
for each node i in S:
   for each node j in G:
       overall_inf +=max(inf(i, j))
```

Let us say the seed set $S=\{s_1, s_2,\ldots, s_k\}$. When we compute the seed set's influence to node v ($j=v$), $\inf(S, v)=\max(\inf(i,v))= \max(\inf(s_1, v), \inf(s_2, v), \ldots, \inf(s_k, v)) < 1$, since the influence from one node to another node ranges between 0 and 1. When v is added to S, $S=\{s_1, s_2,\ldots, s_k, v\}$. At $j=v$, $\max(\inf(s_1, v), \inf(s_2, v),\ldots, \inf(s_k, v), \inf(v,v)) = \inf(v,v)=1$ because the influence value from a node to itself is always 1, so $\inf(v, v)=1$. When $j=v$, $\inf((S \cup\{v\}),v)) =1$ while $\inf(S, v)<1$. Therefore, $\inf(v, v)>\inf(S, v)$. So even if node v does not have the maximum influence among other seed nodes to all the other nodes in G, the overall influence will still go up. Therefore, δ SIP ($S \cup\{v\}$) > δ SIP (S).

■

Theorem 5: **δ**SIP (S) is sub modular for any $S1$ and $S2$: $S1 \subseteq S2$, and **δ**SIP ($S2 \cup\{v\}$)-**δ**SIP ($S2$) ≤**δ**SIP ($S1 \cup\{v\}$)-**δ**SIP ($S1$)

Proof:

Let us say graph $G=\{g_0, g_1,\ldots, g_n\}$, seed set $S1=\{v_0, v_1,\ldots, v_i\}$ and seed set $S2=S1\cup\{ v_{i+1}, v_{i+2},\cdots, v_j\}$. Since $S1$ is a subset of $S2$, $i<j$. **δ**SIP(S) denotes the total influence value from the seed node set S to each node in the graph G. That is, **δ**SIP(S)=max($\{v_0, v_1, v_2,\ldots, v_s\}\rightarrow g_0$)+ max($\{v_0, v_1, v_2,\ldots, v_s\} \rightarrow g_1$)+…+ max($\{v_0, v_1, v_2,\ldots, v_s\} \rightarrow g_n$).

The overall influence of set $S1$ and node v to G is $\delta\text{SIP}(S1 \cup\{v\}) = \max(\delta\text{SIP}(S1, g_0), \delta\text{ SIP}(v, g_0))+ \max(\delta\text{SIP}(S1, g_1), \delta\text{ SIP}(v, g_1))+\ldots+ \max(\delta\text{ SIP}(S, g_n), \delta\text{ SIP}(v, g_n))$. The influence from $S1$

and node v to each node G_i in graph G is max($\{v_0, v_1, v_{2,\dots}, v_i, v\}$,$G_i$). The overall influence of $S2$ and node v to G is δSIP($S2 \cup \{v\}$) = max(δ SIP($S1$, g_0), δ SIP(v, g_0), δ SIP($\{v_{i+1}, v_{i+2,\dots}, v_j\}$, g_0))+ max(δ SIP($S1$, g_1), δ SIP(v, g_1), δ SIP($\{v_{i+1}, v_{i+2,\dots}, v_j\}$, g_1))+…+ max(δ SIP(S, g_n), δ SIP(v, g_n), δ SIP($\{v_{i+1}, v_{i+2,\dots}, v_j\}$, g_n)). The influence from $S2$ and node v to each node G_i in graph G is max($\{v_0, v_1, v_{2,\dots}, v_i, v, v_{i+1}, v_{i+2},\dots, v_j\}$,$G_i$).

The seed set $S2$ has more nodes ($v_0, v_1, v_{2,\dots}, v_i, v_{i+1}, v_{i+2},\dots, v_j$) than $S1$($v_0, v_{1,\dots}, v_i$) that could have the highest influence on each node in G. When node v joins $S2$ as opposed to $S1$, v has a lower chance of being the node to have the highest influence on each node in G, since $S2$ has a more nodes that could have the highest influence on each node in G. The worst case is λSIP ($S2 \cup \{v\}$)-λSIP ($S2$) = δ SIP ($S1 \cup \{v\}$)- δ SIP ($S1$) since $S1$ is a subset of $S2$. If one node in G is influenced by v (because v has the highest influence to this node) in ($S1 \cup \{v\}$) but this node is influenced by one of ($v_{i+1}, v_{i+2},\dots, v_j$) in ($S2 \cup \{v\}$), then λSIP ($S2 \cup \{v\}$)-λSIP ($S2$) > δ SIP ($S1 \cup \{v\}$)- δ SIP ($S1$). Therefore, λSIP ($S2 \cup \{v\}$)-λSIP ($S2$) ≤ δ SIP ($S1 \cup \{v\}$)- δ SIP ($S1$).

■

Theorem 6: Let S be the solution returned by Greedy, then $\delta_{SIP}(S) \geq (1 - \frac{1}{e})\delta_{SIP}(OPT)$. Let $OPT \subseteq \{1, 2, \dots, n\}$ denote an optimal solution to maximum influence.

Proof:

When s=1, the solution returned by the greedy algorithm is 100% accurate. The greedy algorithm obtains the first seed node by calculating each node's influence on the entire graph and choosing the node with the highest overall influence value. This method is the same as the optimal solution given by exhaustive search. Let x_i denote the added value influenced by the choice in iteration i. $x_i = \delta_{SIP}(S \cup \{v\}) - \delta_{SIP}(S)$. Let y_i denote the total value influenced by the choice in

iterations 1, 2, …, i. Let $z_i = \delta_{SIP}(OPT) - y_i$, which is the difference in the total influence at i^{th} iteration.

Figure 16 shows a diagram of x_i, y_i, z_i, and $\delta_{SIP}(OPT)$. The image on the top left is when i=0 and $z_0 = \delta_{SIP}(OPT) - y_0 = \delta_{SIP}(OPT)$. The image below is when i=1 and $z_1 = \delta_{SIP}(OPT) - y_1$. Since y_1 is the total value influenced by the choice in iterations 1, $x_1 = y_1$. The image on the right is when i=2 and $z_2 = \delta_{SIP}(OPT) - y_2$. Since y_2 is the total value influenced by the choice in iterations 1 and 2, $y_2 = x_1 + x_2$.

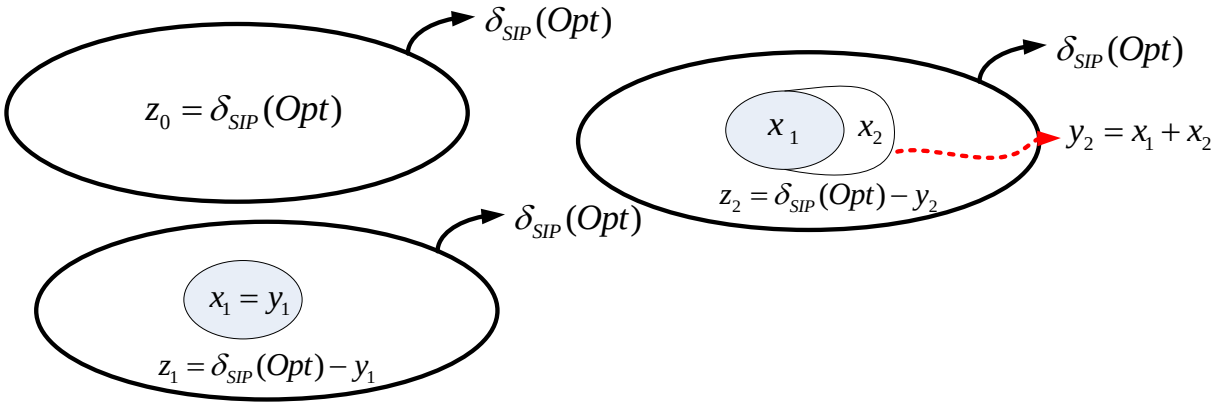


Figure 16. Greedy and Optimal solution

Our claim is $z_i \leq \left(1 - \frac{1}{s}\right)^i \delta_{SIP}(OPT)$. Suppose this were true, then:$y_s = \delta_{SIP}(OPT) - z_s \geq \delta_{SIP}(OPT) - \left(1 - \frac{1}{s}\right)^s \delta_{SIP}(OPT)$. Recall that $e^x \geq 1 + x$ for all real number x. We assume that $x = -\frac{1}{s}$. Then we get $y_s \geq \delta_{SIP}(OPT) - \left(e^{-\frac{1}{s}}\right)^s \delta_{SIP}(OPT)$, which is $y_s \geq \left(1 - \frac{1}{e}\right) \delta_{SIP}(OPT)$.

We prove the correctness of the following by proof of induction: $z_{i-1} \leq \left(1 - \frac{1}{s}\right)^{i-1} \delta_{SIP}(OPT)$.

The base case is when i-1=0, which is clearly true. Our inductive hypothesis is that $z_{i-1} \leq \left(1 - \frac{1}{s}\right)^{i-1} \delta_{SIP}(OPT)$ holds. Using inductive hypothesis, we prove $z_i \leq \left(1 - \frac{1}{s}\right)^i \delta_{SIP}(OPT)$

holds. We define $\Delta(i|A) = \delta SIP(A \cup \{i\}) - \delta SIP(A)$, which is the added value of i to set A. $A_i = (v_1, v_2, \ldots, v_i)$ denotes the nodes selected by the greedy algorithm and $A^* = (v_1^*, v_2^*, \ldots, v_s^*)$ denotes the optimal solution.

We want to prove: $x_i \geq \frac{z_{i-1}}{s}$. By monotonicity, $\delta SIP(A^*) \leq \delta SIP(A^* + A_i)$. By performing more greedy selection, $= \delta SIP(A_i) + \sum_{j=1}^{s} \Delta(v_j^* | A_i \cup \{v_1^*, v_2^*, \ldots, v_{j-1}^*\})$. By sub modularity, $\leq \delta SIP(A_i) + \sum_{k \in A^*} \Delta(k|A_i)$. By the property of greedy, $\leq \delta SIP(A_i) + \sum_{k \in A^*} \Delta(v_{i+1}|Ai)$. Then, $= \delta SIP(A_i) + s^* \Delta(v_{i+1}|Ai)$, which is $\delta SIP(A^*) \leq \delta SIP(A_i) + s^* \Delta(v_{i+1}|A_i)$. By simple math, $s^* x_{i+1} \geq (\delta SIP(A^*) - \delta SIP(A_i))$, which is $x_{i+1} \geq \frac{1}{s} (\delta SIP(A^*) - \delta SIP(A_i)) = \frac{1}{s}^* z_i$. Finally we get $x_i \geq \frac{z_{i-1}}{s}$. By the definition of z_i, $z_i = z_{i-1} - x_i$.

Combining the previous two, we get $z_i \leq z_{i-1} - \frac{z_{i-1}}{s} = \left(1 - \frac{1}{s}\right) z_{i-1} \leq \left(1 - \frac{1}{s}\right)^i \delta_{SIP}(OPT)$. Therefore, $\delta_{SIP}(S) \geq \left(1 - \frac{1}{e}\right) \delta_{SIP}(OPT)$ holds.

■

Chapter 11
k-Colors Influence Maximization on Biomedical Domain [1]

In this section, we apply the influence maximization on biomedical domain.

11.1.k-Colors IM on Biomedical Domain

Gastrointestinal (GI) cancers are the most common human tumors encountered worldwide [88]. These include colorectal cancer, gastric cancer, pancreatic cancer, and cancers of the liver and of the biliary tract. Although early-stage GI cancers are amenable to surgical resection with curative intent, the overall 5-year relapse rate remains high. The addition of neoadjuvant or adjuvant chemotherapy and radiation therapy only modestly improves the overall long-term survival [89]. Approximately 25% of GI cancers are diagnosed in an advanced stage, whereas another 25 to 50% of patients will develop metastases during the course of the disease [90]. GI cancers are still a leading cause of cancer death [91]. Therefore, it is imperative to explore potential effective influential genes to increase the number of patients qualified for curative treatments. Both text mining and network analysis have been applied to find the hidden knowledge and rules behind the huge amount of information [92].

11.2.Biomedical Datasets

In this section, we use data from four cancer types that belong to The Cancer Genome Atlas (TCGA) [93]. We employ text mining to search for the genes related to four gastrointestinal cancers that are scattered in PubMed, using the query “colon adenocarcinoma”, “liver hepatocellular carcinoma”,

[1] This chapter is extracted from “Identification of Most Influential Co-Occurring Gene Suites for GI Cancers using Biomedical Literature Mining and Graph-based Influence Maximization,” by Charles C.N. Wang, Jennifer Jin, Jan-Gowth Chang, Masahiro Hayakawa, Atsushi Kitazawa, Jeffrey J.P. Tsai and Phillip C.-Y. Sheu, 2019, Future Generation Computer Systems Special Issue on Data Exploration in the Web 3.0 Age, submitted on Feb 10, 2019.

“pancreatic adenocarcinoma”, “stomach adenocarcinoma”, “stomach cancer”, “colorectal cancer”, “gallbladder cancer”, “liver cancer”, and “pancreatic cancer”. The biomedical literature metadata such as PMID, title, abstract, journal name, and its ISSN and publication date are extracted.

Stomach cancer

Search terms including “("stomach neoplasms"[MeSH Terms] OR ("stomach"[All Fields] AND "neoplasms"[All Fields]) OR "stomach neoplasms"[All Fields] OR ("stomach"[All Fields] AND "cancer"[All Fields]) OR "stomach cancer"[All Fields])” are used in our search strategies. The publication date is limited to the last 10 years and a total of 35,097 articles are retrieved.

Pancreatic cancer

Search terms including “("pancreatic neoplasms"[MeSH Terms] OR ("pancreatic"[All Fields] AND "neoplasms"[All Fields]) OR "pancreatic neoplasms"[All Fields] OR ("pancreatic"[All Fields] AND "cancer"[All Fields]) OR "pancreatic cancer"[All Fields])” are used in the search strategies. The publication date is limited to the last 10 years and a total of 42,397 articles are retrieved.

Liver cancer

Search terms including “("liver neoplasms"[MeSH Terms] OR ("liver"[All Fields] AND "neoplasms"[All Fields]) OR "liver neoplasms"[All Fields] OR ("liver"[All Fields] AND "cancer"[All Fields]) OR "liver cancer"[All Fields])” are used in the search strategies. The publication date is limited to the last 10 years and a total of 99,061 articles are retrieved.

Colorectal cancer

Search terms including “("colorectal neoplasms"[MeSH Terms] OR ("colorectal"[All Fields] AND "neoplasms"[All Fields]) OR "colorectal neoplasms"[All Fields] OR ("colorectal"[All Fields] AND "cancer"[All Fields]) OR "colorectal cancer"[All Fields])” are used in the search strategies. The publication date is limited to the last 10 years and a total of 95,800 articles are retrieved.

We use the abstracts as the training set to train the literature ranking tool MedlineRanker [94], which ranks the biomedical literature according to the relevance of a topic learned from the training set. The trained MedlineRanker is used to rank the PubMed publications published in the last 10 years, and the top 1000 publications are selected to conduct the following biomedical text mining procedures.

We use text mining on articles available in PubMed to generate a list of gene/protein co-occurrences related to gastrointestinal cancer interactions. The method accounts for the position of the co-occurring terms within sentences or abstracts. According to the semantic structure of each sentence and the whole abstract, the genes co-occurring with the customized concepts are likely to be related to gastrointestinal cancers reported in the biomedical literature.

In the co-occurrences analysis, the gene pairs are categorized into four types [95]:

(1) Two entities co-occur in an abstract (type 4)

(2) Two entities co-occur in a sentence (type 3)

(3) Two entities co-occur in a sentence with an interaction term (e.g., activates, induces, inhibits) anywhere in the sentence (type 2)

(4) Two entities co-occur in a sentence with an interaction term in between the entity names (type 1).

11.3. Representing Biomedical data in Graph form

In this representation, a node represents a gene, and an edge represents a relevance occurrence between two genes for a given disease. The color of the edge represents a certain type of disease. Hence, every edge is of a certain disease and can be treated as a colored edge as shown in Figure 17(a). Because edges are colored, there can be multiple edges between a pair of genes as in Figure 17(b).

Figure 17. Relevance between Genes for Diseases

The value of an edge represents the relevance strength of two genes for the given disease. That is, the value specifies the probability that two genes together are relevant to the given disease. Therefore, the relevance on an edge is bi-directional, and it is not like an influence relationship. The relevance value is given between 0 and 1 inclusively. The value near '1' implies that two genes are strongly relevant to the disease, and the value near '0' implies that two genes are not relevant to the given disease. An example of the graph representation of gene disease relevance is shown in Figure 18.

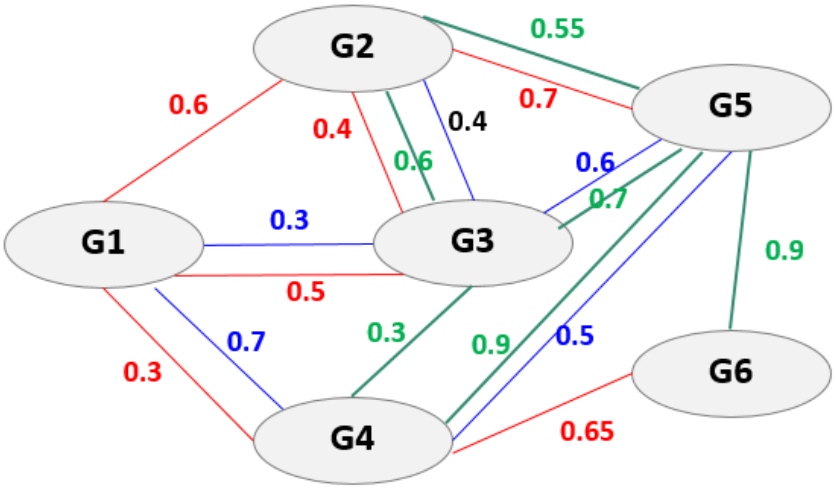


Figure 18. Example of 'Gene-as-Node' Relevance Graph

The example shows a graph representation of 6 genes, 3 types of diseases, and a number of relevance edges. Each edge is associated with a relevance value. Note that the genes G1 and G2 are relevant for just one disease, and the genes G2 and G3 are relevant for all three diseases. The gene G6 has only two edges, meaning that the gene is weakly associated with diseases. In contrast,

the gene G3 has a total of 9 relevance edges, and therefore this gene is vital in analyzing disease-related analytics.

Another observation regarding the example is that the relevance edges for a disease are not necessarily reachable among them. The relevance edge between genes G4 and G6 for a disease colored ‘red’ is not reachable from other edges of the same disease.

11.4.Applying IM on Biomedical Dataset

In the past, some studies apply network analysis methods to gene regulatory networks to solve the influence maximization problem. Hashimoto et al. [96] develop an algorithm to grow small genetic-regulatory subnetworks from a smaller number of genes of interest, or a seed set of genes. Their algorithm is based on the strength of the connection between prospective genes to be added and the subnetwork at the current stage of the algorithm. Hecker et al. [97] propose an algorithm for starting with a seed network of genes and expending it to query the composite correlational network in a way that allows users to rank other genes for possible inclusion in an extended seed network. Gibbs and Shmulevich [98] apply IM methods to biological networks to discover the set of regulatory genes with the greatest influence on network dynamics. An influence ranking on genes is produced by solving the IM problem over different numbers of source nodes and is compared to other metrics of network centrality. Nalluri et al. [99] apply information diffusion theory to quantify the influence diffusion in a miRNA (MicroRNA)-miRNA regulation network across many disease classes. Their method regulates the specific miRNAs critical diseases which perform an underlying part in their signing cascade and therefore may regulate disease progression. Although some use IM methods on gene regulatory networks, no work has the flexibility to customize it to specific set of genes or diseases. For example, our approach supports queries such as “Find 2 disease that are the most closely related to a

gene set [P53, CD44, CDC6, STAT3].” and “Find 5 genes that are the most closely related to a disease set [“colorectal cancer”, “liver cancer”]” that could not be solved using existing approaches.

We apply an influence maximization method to a biological co-occurrence network, aiming to discover each set of regulatory genes that together have the greatest influence on the network dynamics. In applying the above algorithm, we use a graph of 487 nodes (genes) and 1626 edges (co-occurrences). We set the edge weights to 1 (type 1), 0.75 (type 2), 0.5 (type 3), and 0.25 (type 4), respectively.

11.5.Results

The following sections discuss the results obtained in the study.

11.5.1. Identification of influential genes

In this study, a total of 487 genes related to gastrointestinal cancers are extracted from 272,355 PubMed articles. With the influence maximization algorithm, an influence ranking on gene suites is produced. We remove those gene suites with a relevance rate less than or equal to 5 and find seven sets of regulatory influence genes with the greatest influence on the GI cancer network with four cancers, as shown in Figure 19.

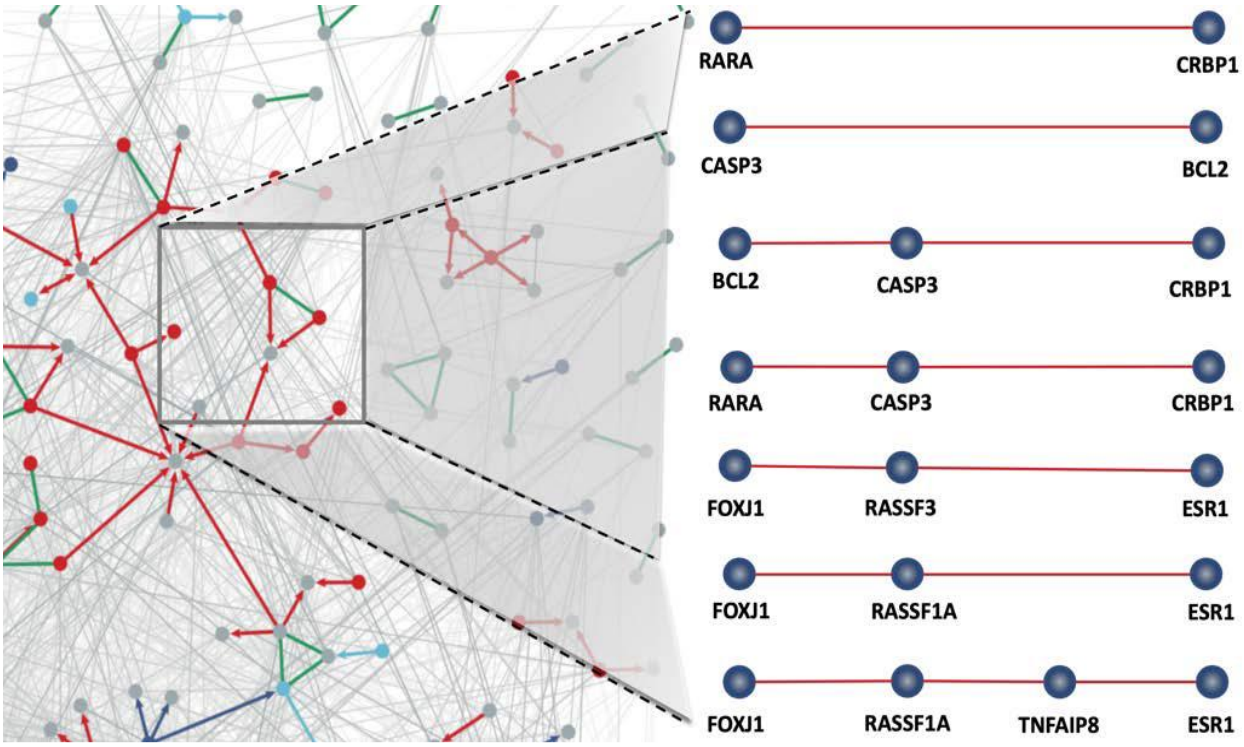


Figure 19.A GI cancer network derived from abstracts that are stored in PubMed, using co-occurrence and text mining

11.5.2. Validation of influential genes

We first conduct literature review to validate their importance and potential as clinical genes.

Retinoic Acid Receptor Alpha

The RARA gene represents a nuclear retinoic acid receptor. It has been implicated in the regulation of development, differentiation, apoptosis, and transcription of clock genes. In a recent study, Xiang *et al*. [100] find that RARA is a drug sensitive biomarker of ERBB2-targeted treatment. ERBB2-related pathways can help us finding sensitive molecules and potential combined therapeutic targets of ERBB2-targeted therapy for gastric cancers.

Cellular Retinol-Binding Protein 1

The CRBP1 gene is encoded in the carrier protein involved in the transport of retinol (vitamin A alcohol) from the liver storage site to the peripheral tissue. In a previous study, colorectal and

gastric adenomas frequently display methylation of the CRBP1 promoter region. The percentage found in the invasive colorectal and gastric tumors suggests that methylation-associated inactivation of CRBP1 is an early event in human tumorigenesis. Also, aberrant methylation of CRBP1 has predictive value [101].

Caspase 3

CASP3 is a cysteine-aspartic acid protease that plays a central role in the execution-phase of cell apoptosis. Qiang *et al*. [102] discover Caspase-3 protein levels are upregulated in colorectal cancer tissues. Furthermore, high expressions of Caspase-3 are correlated with decreased overall survival and unfavorable clinicopathologic characteristics. Cox regression analysis shows that high Caspase-8 and Caspase-3 expressions are independent negative markers of overall survival. The result suggests that Caspase-3 expressions in tumor tissues are novel candidate prognostic markers for colorectal cancer patients.

BCL2, Apoptosis Regulator

BCL2 is a key regulator of apoptosis whose dysregulation can cause various pathological consequences including the development of cancer [103]. In a meta-analysis study, BCL2 high expression is significantly correlated with favorable overall survival, better disease-free survival, and recurrent free survival. Hence, BCL2 may be a valuable prognostic-predictive marker in colorectal cancer [104].

Forkhead Box J1

FOXJ1 is a forkhead transcription factor that has been previously studied mostly as a ciliary transcription factor. In a recent report, an increased expression of FOXJ1 associated with the clinical stage, metastasis of lymph node, and invasion depth in colon cancer suggest FOXJ1 is a tumor promoter in colorectal cancer. The results suggest that increased FOXJ1 contributes to the

progression of colorectal cancer, which might be associated with the promotion effect of β-catenin nuclear translocation, and it may be a novel therapeutic target in colorectal cancer [105].

Ras Association Domain Family Member 3/ Ras Association Domain Family Member 1

RASSF is a family of 10 members (RASSF1-10) implicated in a variety of key biological processes, including cell cycle regulation, apoptosis and microtubule stability. RASSFs have been implicated in colorectal cancer and several family members are now thought to be tumor suppressors. In particular, RASSF1A and RASSF3, methylation have been associated with colorectal cancer development, although the mechanisms of action remain poorly understood. RASSF1 and RASSF3 have been considered as potential biomarkers and for the development of new targeted therapies for colorectal cancer [106].

Estrogen Receptor 1

Caiazza et al. [107] analyzes the estrogen pathway as a possible therapeutic avenue in colorectal cancer. The experimental evidence explains the cellular and molecular mechanisms of estrogen-mediated protection against colorectal tumorigenesis and suggests that ESR1 future challenges and potential avenues for colorectal cancer targeted therapy.

TNF Alpha Induced Protein 8

In a recent study, TNFAIP8 has been associated with the tumorigenicity of gastric cancer. The decreased expression of TNFAIP8 inhibits the growth, invasion and migration of gastric cancer. It is a meaningful approach for treating human gastric cancer. In addition, the expression levels of TNFAIP8 may be considered as a biomarker of gastric cancer [108].

The influence genes can be categorized into three functional groups: Biological Process (BP), Cellular Component (CC), and Molecular Function (MF). The influence genes in the BP group are mainly enriched in signal transduction, apoptosis, and regulation of nucleobase; the influence

genes in the MF group are mainly enriched in transcription factor activities; the influence genes in the CC group are significantly enriched in cytoplasm, nucleus, and mitochondrion. Figure 20 shows associations of genes with GI cancers based on the literature, gene ontology, pathway, and transcription factor enrichment analysis. For the literature category, the number in each box indicates the number of times the particular gene appeared in each type of TCGA archive- TCGA-COAD (COAD, Colon adenocarcinoma), TCGA-LIHC (Liver Hepatocellular Carcinoma), TCGA-PAAD (Pancreatic adenocarcinoma) and TCGA-STAD (Stomach adenocarcinoma). Each filled box indicates that the particular gene was involved in that pathway. According to Kyoto Encyclopedia of Genes and Genomes (KEGG) pathway analysis, our results demonstrate that these genes are mainly involved in PAR1-mediated thrombin signaling events, VEGF and VEGFR signaling networks, retinoic acid receptors-mediated signaling, BMP receptor signaling, Apoptosis, p53 pathway, ATR signaling pathway, and the regulation of RAC1 activities. The transcriptional analysis results suggest that these genes are highly associated with transcription factors such as SP1, SP4, IRF1, RREB1, EGR1, HENMT1, and NHLH1 as shown in Figure 20. Some of these gene terms, pathways, and transcription factors are well known to be associated with GI cancers. In addition, at least 487 instances of the seven sets of regulatory influence genes are associated with the biomedical literature on one or more cancer types.

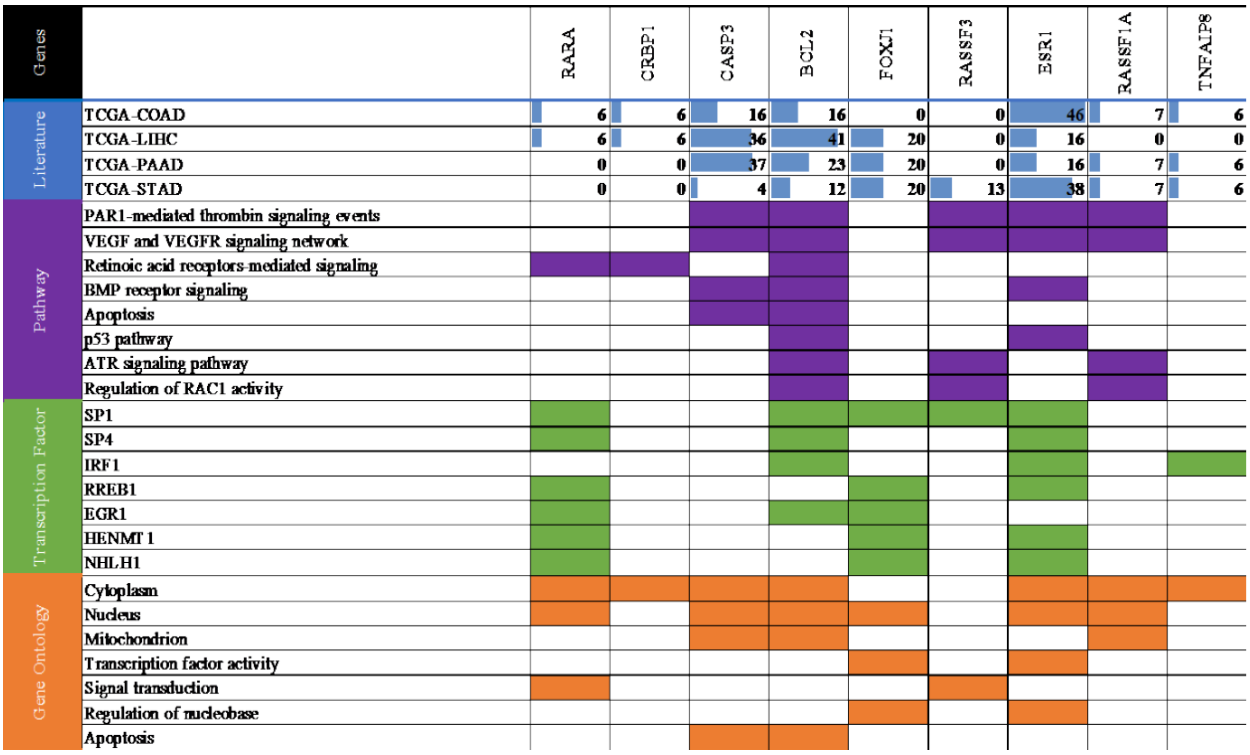


Figure 20. Associations of genes with GI cancers based on the literature, gene ontology, pathway, and transcription factor enrichment analysis

11.5.3. Validation of influential co-occurring gene suites

To determine the significance of the concordance index values, we generate a null distribution composed of 1517 random models of 50 genes for the TCGA datasets used. To assess the concordance index prediction of the influential genes in datasets other than TCGA, we use SurvExpress [109] which provides evaluations of gene lists across cancer types. For this, we use normalized datasets that include overall survival times (without considering recurrence, metastases, or relapse) and only those studies containing more than 26 samples. We use MSigDB and DAVID to determine which gene suites reveal important biological associations across pathways, transcriptional control, gene ontologies, and other biological terms associated with the set of co-occurring genes. We also compare the concordance index values of the co-occurring influence

gene suites discovered by us with those of other multi-cancer sets of influence genes reported in the biomedical literature.

The identified seven sets of regulatory influence genes with the highest influence on the GI cancer network include RARA - CRBP1, CASP3 - BCL2, BCL2 - CASP3 – CRBP1, RARA - CASP3 – CRBP1, FOXJ1 - RASSF3 - ESR1, FOXJ1 - RASSF1A - ESR1, FOXJ1 - RASSF1A - TNFAIP8 - ESR1 with 9 constituent genes RARA, CRBP1, CASP3, BCL2, FOXJ1, RASSF3, ESR1, RASSF1A and TNFAIP8. These influence gene suites are able to discriminate low- and high-risk groups efficiently in the GI cancers through statistical association, i.e., those cases with a mutation in each of the 7 suites statistically have a lower survival rate compared to those cases without a mutation.

Chapter 12
Graph Reduction based on SIP

As shown in the previous sections, brute force algorithms have high time complexity. Given the size of most social networks, the computing time could take days, weeks, or months. In order to improve the running time, we propose graph reduction algorithms to reduce the size of the graph while not compromising the accuracy on our SIP model.

12.1. Graph Reduction based on SIP

Given a graph *G*, assume an application is only interested in finding the strongest influence paths between any pair of two nodes. Can we reduce *G* to a smaller graph *G'* which preserves all the strongest influence paths between any pair of nodes? We call this problem the GR-SIP (Graph Reduction - Strongest Influence Path) Problem.

Some query-based applications are only interested in finding the SIP for each pair of nodes. So, if we preserve all SIP of *G*, and we can obtain a reduced graph *G'*.

SIP(*G*,*a*,*b*)==SIP(*G'*,*a*,*b*).

12.2. GR-SIP Algorithm

Wang *et al.* describe a set of graph reduction algorithms [82]. The main idea of their algorithm is that if the influence of an edge between two nodes is smaller than the SIP path between them, then they can remove the edge. It consists of the following two steps.

Step 1: Find the SIP between any pair of nodes a and b, and calculate the SIP $\delta G(a,b)$.

Step 2: For any edge (*a*,*b*) that connects *a* and *b*, if the inf*(a,b*) is smaller than $\delta G(a,b)$, remove the edge (*a*,*b*).

GR-SIP algorithm (Find a minimal reduced graph that preserves all the SIPs between any pair of nodes from the original graph.)

Input: Graph *G*
Output: A reduced graph *G*'

```
1. G'←G;
2. N←Number of Nodes(G)
3. For i←1 to N do
4.   Inf[i] ←SIP(G,i); //use SIP algorithm to compute the SIP from source i to other
     nodes.
5.      For j←1 to N do
6.         If edge E[i,j] exists and E[i,j]<SIP[i][j] then
7.              Remove E[i,j] from G';
8.         End
9.       End
10. End
11. Return G'
```

The time complexity of the GR-SP algorithm is $n*T(\text{SIP})$, where n is the number of nodes and $T(\text{SIP})$ is the time complexity of the SIP algorithm.

For most implementations, $T(\text{SIP})=O(n^2)$. With a min-priority queue implemented by a Fibonacci heap, the time complexity of SIP algorithm can be improved to $O(E+n\log n)$, where E is the number of edges.

12.3.GR-SIP2- Graph reduction based on SIP for colored graphs

Given a graph *G* with nodes in two colors, assume an application is only interested in finding the SIPs between any pair of nodes having the same color. Can we reduce *G* to a smaller (and smallest) graph *G*' which preserves all the SIP paths between any pair of nodes with the same color? We call this problem the GR-SIP2 Problem.

Our idea is to find a reduced graph G_r for the nodes in one color (say red), and a simplified graph G_b for the nodes in the other color (say black), and then merge them together.

Method to find G_r and G_b is as following.

Step 1: Check if there is a direct edge between any two red nodes in *G*.

Step 2: For any pair of two red nodes, find the SIP between them.

Step 3: Use a similar method as GR-SIP to remove redundant edges.

Step 4: Retain all other red nodes and all edges that connect them and call it G_r.

Step 5: Remove any edge that does not belong to any SIP.

Step 6: Repeat steps 1 through 5 for any two black nodes and call it G_b.

Step 7: Merge G_r and G_b.

The GR-SIP2 algorithm is as the follows.

GR-SIP2 algorithm (Find a minimal reduced graph that preserves all the SIPs between any pair of same color nodes from the 2-Colors original graph.)

Input: A 2-Colors Graph *G*
Output: A reduced graph *G*'

1. Find all edges which connect color-1 nodes directly
2. Compute the SIP for each edge
3. if the influence between any pair of color-1is more than their SIP
4. keep the edge
5. else
6. delete the edge
7. Keep all other color-1 nodes and their edges which are connected to them
8. Delete all other edges that not belong to the SIPs and previous line
9. Return G_r
10. Find all edges which connect color-2 nodes directly
11. Compute the SIP for each of edge
12. if the influence between any pair of color-1is more than their SIP
13. keep the edge
14. else
15. delete the edge
16. Keep all other color-2 nodes and their edges which are connected to them
17. Delete all other edges that not belong to the SIPs and and previous line
18. Return G_b
19. Merge(G_r,G_b)
20. **Return** *G'*

The time complexity of the GR-SIP2 algorithm is the same as that of the GR-SIP algorithm.

12.4.GR-SIPk

The *GR-SIP*2 algorithm can be easily extended to graphs that consist of nodes in *k* colors.

Step 1: Check if there is a direct edge between any two red nodes in *G*.

Step 2: For any pair of two red nodes, find the SIP between them.

Step 3: Use a similar method as GR-SIP to remove redundant edges.

Step 4: Retain all other red nodes and all edges that connect them and call it G_r.

Step 5: Remove any edge that does not belong to any SIP.

Step 6: Repeat steps 1 through 5 for each color.

Step 7: Merge all the subgraphs.

The time complexity of the GR-SIPk algorithm is the same as that of the GR-SIP algorithm.

The proof of correctness for GR-SIP, GR-SIP2 and GR-SIPk can be found in [78].

Chapter 13
Experiments

In order to evaluate the efficiency of the heuristic algorithms, we experiment some algorithms with open source data.

13.1. Experiment Environment

The database we use for the experiment is MySQL. Python 2.7 is used to implement the algorithms and the IDE is Spyder on Anaconda 2.

13.2. Evaluation Function

With our heuristic algorithms, we reduce the runtime at the expense of accuracy. The brute force algorithms explore all possible options to find the most optimal solution, so it will always return the best possible solution.

Since we know brute force will always return the best possible result (the path with the highest influence value), we compare the influence value returned from the brute force algorithm and the influence value returned from the heuristic algorithm.

$$\text{Accuracy(Heuristic)} = \frac{InfluenceValue(\text{Heuristic})}{InfluenceValue(Brute\ Force)}$$

13.3. Experiment with Social Network Data

The dataset we use for the social network is Stanford's SNAP- Facebook dataset. It consists of 4,038 nodes and 88,234 edges.

After downloading data from https://snap.stanford.edu/data/, we assign edge weight (influence probability) to each edge where every edge is generated uniformly at random in the range [0, 1].

The average value of the influence values is 0.499 and standard deviation of influence values is 0.283. The edge weights represent the influence from one person to another.

Since the original dataset is too large to run the analytics, we define a reduced and normalized dataset using the original dataset. The reduced dataset is obtained by taking the first 100 nodes and their relationships from the original dataset.

The number of edges at each node on average is 5.74 with a standard deviation of 5.41. We randomly assign 4 colors to each node.

We experiment the following Influence maximization with the *t* colored nodes constraint algorithms. We run both the brute-force and heuristic algorithm to compare the runtime and accuracy. Although we have an algorithm that returns an accurate result for IM-SIP-t colored nodes constraint, it is NP-complete. Our heuristic algorithm has a better run time. We compare the brute force and the heuristic algorithm in terms of time elapsed time and accuracy.

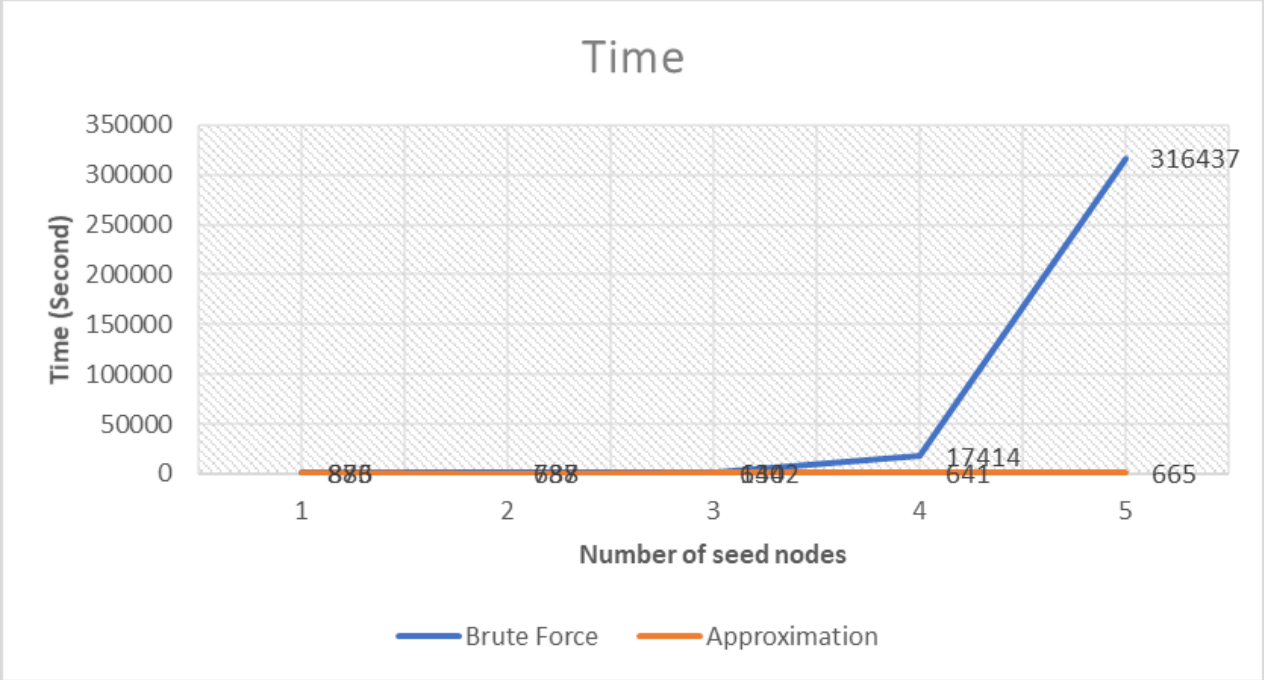


Figure 21.Time elapse for IM-SIP-*t* colored nodes constraint

Figure 22 shows the time elapsed to run the brute force algorithm for IM-SIP-*t* algorithm and the heuristic algorithm for IM-SIP-*t*. As the value of *t* increases, the brute force is shown to explode. Running the brute-force algorithm beyond 5 for the value of *t* took an unreasonable time to run. On the other hand, the beauty of the heuristic algorithm is that the value of *t* doesn't affect the running time as drastically. With the heuristic algorithm, when the value of *t* was 50, the running time was 683 seconds and when *t* was 100, the running time was 731 seconds.

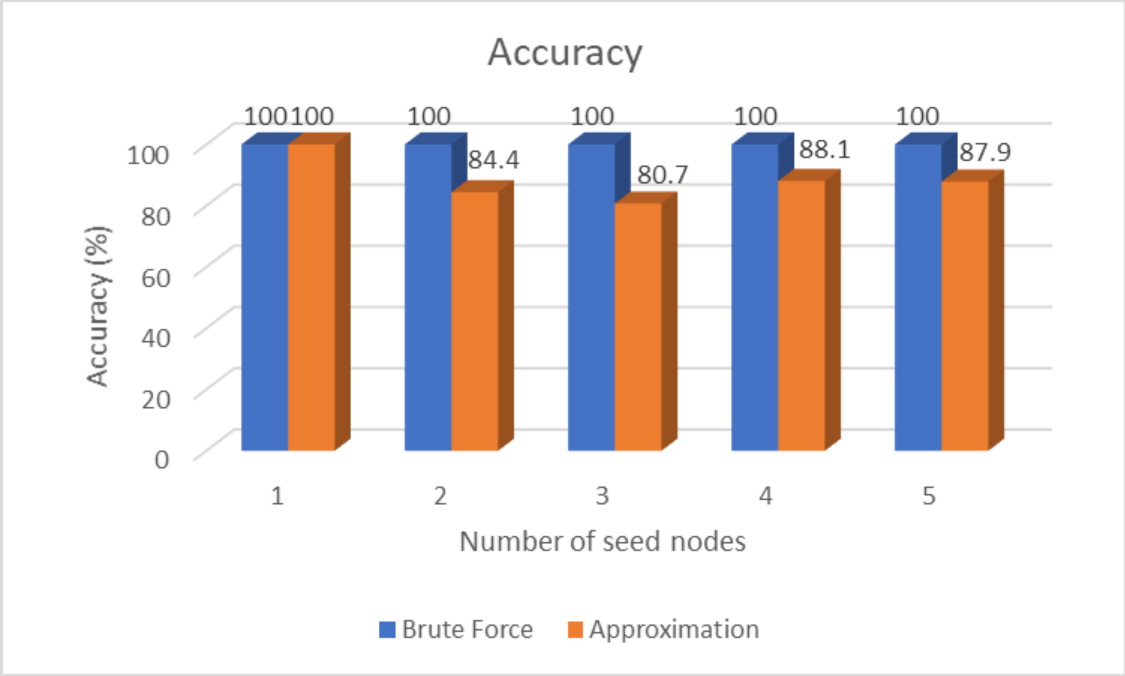


Figure 22. Accuracy for IM-SIP-*t* colored nodes constraint

Figure 23 displays the accuracy of the brute force algorithm for the IM-SIP-*t* algorithm and the heuristic algorithm for IM-SIP-*t*. Since the brute force algorithm checks all possibilities, the accuracy will always be 100%. As shown, other than when *t*=1, the accuracy stays within the 80~90% range, regardless of the value of *t*. The accuracy will always be 100% for *t*=1 since the heuristic chooses the node with the highest overall influence on the entire graph to choose its first seed node. Theorem 6 shows the error bound to be $\delta_{SIP}(S) \geq (1 - \frac{1}{e})\delta_{SIP}(OPT)$. The bound $1 -$

$\frac{1}{e}$ = 1- 0.3678 = 0.632. The accuracies shown in Figure 17, ranging between 100% and 80.7%, validate the theorem.

Chapter 14
Conclusions and Future Work

Influence is the key conceptual foundation for making various decisions on our daily activities. The complexity of handing various types of analytics in an influence network is high.

In this paper we present a comprehensive framework for representing various analytics networks using the GOLAP analytics methods and propose effective schemes to handle the runtime complexity. Our contribution can be summarized in three folds; the *k*-colors SIP model with various constraints, *k*-colors influence maximization methods, and performance optimization using heuristic and reduction schemes. In addition to providing formal proofs on the proposed methods, we also present the results of experiments.

One observation we made through this research is that most social networks in practice are too large and hence running analytics on them could take up to days and weeks. Hence, we plan to explore further advanced ways of optimizing the runtime efficiency on large social networks.

In addition, we plan to extend our work with the following:

- Network analysis on graph with dynamic edges,
- Analyzing influence networks with colored edges,
- Devising methods to estimate the processing time for the analytics, and
- Conducting extensive experiments on diverse domains including gene-disease networks, publication networks, and e-commerce networks.

Acknowledgment

The research is supported in part by NEC Solution Innovators, Ltd., Japan and in part by the Ministry of Science and Technology of Taiwan [MOST 106-2632-E-468-003], Asia University.

Bibliography

[1] Z. Lu, Y. Long and V. O. K. Li, "Cascade with varying activation probability model for influence maximization in social networks," 2015 International Conference on Computing, Networking and Communications (ICNC), Garden Grove, CA, 2015, pp. 869-873.

[2] S. Zhou, K. Yue, Q. Fang, Y. Zhu and Weiyi Liu, "An efficient algorithm for influence maximization under linear threshold model," The 26th Chinese Control and Decision Conference (2014 CCDC), Changsha, 2014, pp. 5352-5357.

[3] B. Chou and M. Kim, "Graph Online Analytical Processing," Encyclopedia with Semantic Computing. (2018)

[4] A. Anagnostopoulos, R. Kumar, M. Mahdian, "Influence and correlation in social networks," In KDD'08, pages 7-15, 2008.

[5] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," In WSDM'10, pages 207–217, 2010.

[6] S. Milgram, "The Small World Problem," Psychology Today, 1967, Vol. 2, 60–67

[7] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," Nature, pages 440–442, Jun 1998.

[8] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang, "Social Influence Analysis in Large-scale Networks," In KDD'09, pages 807-816, 2009.

[9] J.Sun and J. Tang, "A Survey of Models and Algorithms for Social Influence Analysis. Social Network Data Analytics, " Aggarwal, C. C. (Ed.), Kluwer Academic Publishers, pages 177–214, 2011.

[10] J. Tang and J. Sun, "Models and Algorithms for Social Influence Analysis," In WWW'14. (Tutorial).

[11] C. Tan, L. Lee, J. Tang, L. Jiang, M. Zhou, and P. Li, "User-level sentiment analysis incorporating social networks," In KDD'11, pages 1397–1405, 2011.

[12] T. Lou, J. Tang, J. Hopcroft, Z. Fang and X. Ding, "Learning to Predict Reciprocity and Triadic Closure in Social Networks," In TKDD, Vol 7(2), 2013.

[13] N. Agarwal, H. Liu, L. Tang, and P. S. Yu, "Identifying the influential bloggers in a community," In WSDM'08, pages 207–217, 2008.

[14] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Discovering leaders from community actions," In CIKM'08, pages 499–508, 2008.

[15] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Technical Report SIDL-WP-1999-0120, Stanford University, 1999.

[16] M. E. J. Newman, "A measure of betweenness centrality based on random walks," Social Networks, 2005.

[17] J.C. Miller, G. Rae, F. Schaefer, L.A. Ward, T. LoFaro and A. Farahat, "Modifications of Kleinberg's HITS algorithm using matrix exponentiation and web log records," In Proc. of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, Systems Engineering-Theory & Practice. 2001.

[18] Y. Liu, M. Tang, T. Zhou and Y. Do, “Improving the accuracy of the k-shell method by removing redundant links: From a perspective of spreading dynamics,” Scientific report 5: 13172. August 2015

[19] F. Wang and D.P. Landau, “Determining the density of states for classical statistical models: A random walk algorithm to produce a flat histogram,” Physical Review 64: 056101. Oct 2001.

[20] J. Wang, X. Wu, B. Yan and J. Guo, “Improved Method of Node Importance Evaluation Based on Node Contraction in Complex Networks,” In proced of CEIS 15 pp.1600-1604. 2011.

[21] J. Zhang, B. Liu, J. Tang, T. Chen, and J. Li, “Social Influence Locality for Modeling Retweeting Behaviors,” In IJCAI'13, pages 2761-2767, 2013.

[22] V. Tejaswi, P. V. Bindu and P. S. Thilagam, "Diffusion models and approaches for influence maximization in social networks," 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, 2016, pp. 1345-1351.

[23] J. Ok, Y. Jin, Jaeyoung Choi, J. Shin and Y. Yi, "Influence maximization over strategic diffusion in social networks," 2014 48th Annual Conference on Information Sciences and Systems (CISS), Princeton, NJ, 2014, pp. 1-5.

[24] J. Ok, Y. Jin, J. Shin and Y. Yi, "On Maximizing Diffusion Speed Over Social Networks With Strategic Users," in IEEE/ACM Transactions on Networking, vol. 24, no. 6, pp. 3798-3811, December 2016.

[25] G. Zhang, S. Li, J. Wang, P. Liu, Y. Chen and Y. Luo, “New Influence Maximization Algorithm Research in Big Graph,” 14th Web Information Systems and Applications Conference (WISA). 2017.

[26] D. Król, “On Modelling Social Propagation Phenomenon,” Asian Conference on Intelligent Information and Database Systems pp. 227-236. 2014.

[27] V. Tejaswi, P. V. Bindu and P. S. Thilagam, "Target specific influence maximization: An approach to maximize adoption in labeled social networks," 2017 9th International Conference on Communication Systems and Networks (COMSNETS), Bengaluru, India, 2017, pp. 542-547.

[28] F. H. Li, C. T. Li and M. K. Shan, "Labeled Influence Maximization in Social Networks for Target Marketing," 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, Boston, MA, 2011, pp. 560-563.

[29] H. Qiang and G. Yan, "A method of personalized recommendation based on multi-label propagation for overlapping community detection," 2012 3rd International Conference on System Science, Engineering Design and Manufacturing Informatization, Chengdu, 2012, pp. 360-364.

[30] H. Zhuang, Y. Sun, J. Tang, J. Zhang and X. Sun, "Influence Maximization in Dynamic Social Networks," 2013 IEEE 13th International Conference on Data Mining, Dallas, TX, 2013, pp. 1313-1318.

[31] A. Kumari and S. N. Singh, "Online influence maximization using rapid continuous time independent cascade model," 2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence, Noida, India, 2017, pp. 356-361.

[32] G. Tong, W. Wu, S. Tang and D. Z. Du, "Adaptive Influence Maximization in Dynamic Social Networks," in IEEE/ACM Transactions on Networking, vol. 25, no. 1, pp. 112-125, Feb. 2017.

[33] F. Hao, C. Zhu, M. Chen, L. T. Yang and Z. Pei, "Influence Strength Aware Diffusion Models for Dynamic Influence Maximization in Social Networks," 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing, Dalian, 2011, pp. 317-322.

[34] M. Maghami and G. Sukthankar, "Hierarchical influence maximization for advertising in multi-agent markets," 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), Niagara Falls, ON, 2013, pp. 21-27.

[35] X. Weng, Z. Liu and Z. Li, "An Efficient Influence Maximization Algorithm Considering Both Positive and Negative Relationships," 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, 2016, pp. 1931-1936.

[36] A. Bruckman. Influence Maximization in Social Networks: Considering both positive and negative relationships," 2015 International Conference on Collaboration Technologies and Systems (CTS), Atlanta, GA, 2015, pp. 479-480.

[37] Y. Zhang, Y. Bai, L. Chen, K. Bian and X. Li, "Influence Maximization in Messenger-Based Social Networks," 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, 2016, pp. 1-6.

[38] M. Yu, W. Yang, W. Wang, G. Shen, G. Dong and L. Gong, "UGGreedy: Influence Maximization for User Group in Microblogging," in Chinese Journal of Electronics, vol. 25, no. 2, pp. 241-248, 3 2016.

[39] J. Zhao, Q. Liu, L. Wang and X. Wang, "Relative influence maximization in competitive dynamics on complex networks," 2015 54th IEEE Conference on Decision and Control (CDC), Osaka, 2015, pp. 6583-6588.

[40] H. Zhang, D. T. Nguyen, H. Zhang and M. T. Thai, "Least Cost Influence Maximization Across Multiple Social Networks," in IEEE/ACM Transactions on Networking, vol. 24, no. 2, pp. 929-939, April 2016.

[41] S. Mihara, S. Tsugawa and H. Ohsaki, "Influence maximization problem for unknown social networks," 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), Paris, 2015, pp. 1539-1546.

[42] C. W. Chang, M. Y. Yeh and K. T. Chuang, "On influence maximization to target users in the presence of multiple acceptances," 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), Paris, 2015, pp. 1592-1593.

[43] J. Lu, L. Wei and M. Y. Yeh, "Influence maximization in a social network in the presence of multiple influences and acceptances," 2014 International Conference on Data Science and Advanced Analytics (DSAA), Shanghai, 2014, pp. 230-236.

[44] D. T. Nguyen, S. Das and M. T. Thai, "Influence maximization in multiple online social networks," 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA, 2013, pp. 3060-3065.

[45] I. Rahaman and P. Hosein, "On the multi-stage influence maximization problem," 2016 IEEE Latin American Conference on Computational Intelligence (LA-CCI), Cartagena, 2016, pp. 1-6.

[46] J. Pan, F. Jiang and J. Xu, "Influence Maximization in Social Networks Based on Non-backtracking Random Walk," 2016 IEEE First International Conference on Data Science in Cyberspace (DSC), Changsha, 2016, pp. 260-267.

[47] Pei Li, Zhixu Li, J. He, Xiaoyong Du and Hongyan Liu, "Assessing the influence probability between objects: A random walker approach," 2009 IEEE Symposium on Computational Intelligence and Data Mining, Nashville, TN, 2009, pp. 25-32.

[48] E. Cohen, "Greedy Maximization Framework for Graph-Based Influence Functions," 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), Washington, DC, 2016, pp. 29-35.

[49] Z. Lu, W. Zhang, W. Wu, B. Fu and D. Du, "Approximation and Inapproximation for the Influence Maximization Problem in Social Networks under Deterministic Linear Threshold Model," 2011 31st International Conference on Distributed Computing Systems Workshops, Minneapolis, MN, 2011, pp. 160-165.

[50] Purnawansyah and Haviluddin, "K-Means clustering implementation in network traffic activities," 2016 International Conference on Computational Intelligence and Cybernetics, Makassar, 2016, pp. 51-54.

[51] L. Gu, "A novel locality sensitive k-means clustering algorithm based on subtractive clustering," 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, 2016, pp. 836-839.

[52] M. Yesilbudak, "Clustering analysis of multidimensional wind speed data using k-means approach," 2016 IEEE International Conference on Renewable Energy Research and Applications (ICRERA), Birmingham, 2016, pp. 961-965.

[53] T. S. Xu, H. D. Chiang, G. Y. Liu and C. W. Tan, "Hierarchical K-means Method for Clustering Large-Scale Advanced Metering Infrastructure Data," in IEEE Transactions on Power Delivery, vol. 32, no. 2, pp. 609-616, April 2017.

[54] X. Dong, L. Qian and L. Huang, "Short-term load forecasting in smart grid: A combined CNN and K-means clustering approach," 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), Jeju, 2017, pp. 119-125.

[55] Z. Fan and Y. Sun, "Clustering of College Students Based on Improved K-Means Algorithm," 2016 International Computer Symposium (ICS), Chiayi, 2016, pp. 676-679.

[56] J. R. Lee and C. W. Chung, "A Query Approach for Influence Maximization on Specific Users in Social Networks," in IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 2, pp. 340-353, Feb. 1 2015.

[57] F. Jiang, S. Jin, Y. Wu and J. Xu, "A uniform framework for community detection via influence maximization in social networks," 2014 IEEE/ACM International Conference on

Advances in Social Networks Analysis and Mining (ASONAM 2014), Beijing, 2014, pp. 27-32.

[58] M. Gong, C. Song, C. Duan, L. Ma and B. Shen, "An Efficient Memetic Algorithm for Influence Maximization in Social Networks," in IEEE Computational Intelligence Magazine, vol. 11, no. 3, pp. 22-33, Aug. 2016.

[59] A. Goyal, W. Lu and L. V. S. Lakshmanan, "SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model," 2011 IEEE 11th International Conference on Data Mining, Vancouver,BC, 2011, pp. 211-220.

[60] J. Li and Y. Yu, "Scalable Influence Maximization in Social Networks Using the Community Discovery Algorithm," 2012 Sixth International Conference on Genetic and Evolutionary Computing, Kitakushu, 2012, pp. 284-287.

[61] X. Wang, Y. Zhang, W. Zhang, X. Lin and C. Chen, "Bring Order into the Samples: A Novel Scalable Method for Influence Maximization (Extended Abstract)," 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, 2017, pp. 55-56.

[62] W. Chen, Y. Yuan and L. Zhang, "Scalable Influence Maximization in Social Networks under the Linear Threshold Model," 2010 IEEE International Conference on Data Mining, Sydney, NSW, 2010, pp. 88-97.

[63] A. Mohan, S. Kunnakadan, B. Neelakantan, A. Jayakumar and H. Salim, "A scalable model for efficient information diffusion in large real world networks," 2016 International Conference on Next Generation Intelligent Systems (ICNGIS), Kottayam, 2016, pp. 1-6.

[64] W. Chen, C. Wang and Y. Wang, “Scalable influence maximization for prevalent viral marketing in large-scale social networks,” Proc. of international conference on Knowledge discovery and data mining, pp. 1029-1038, 2010.

[65] M. Han, J. Li, Z. Cai and Q. Han, "Privacy Reserved Influence Maximization in GPS-Enabled Cyber-Physical and Online Social Networks," 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), Atlanta, GA, 2016, pp. 284-292.

[66] Y. Bao, X. Wang, Z. Wang, C. Wu and F. C. M. Lau, "Online influence maximization in non-stationary Social Networks," 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), Beijing, 2016, pp. 1-6.

[67] B. E. Yellakuor, Qin Zhen, Xiong Hu and Qin Zhiguang, "Exploring online social networks for influence maximization," 2015 International Conference and Workshop on Computing and Communication (IEMCON), Vancouver, BC, 2015, pp. 1-7.

[68] F. Wang, K. Xu and H. Wang, "Discovering Shared Interests in Online Social Networks," 2012 32nd International Conference on Distributed Computing Systems Workshops, Macau, China, 2012, pp. 163-168.

[69] J. L. Z. Cai, M. Yan and Y. Li, "Using crowdsourced data in location-based social networks to explore influence maximization," IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, 2016, pp. 1-9.

[70] S. Li, Y. Zhu, D. Li, D. Kim, H. Ma and H. Huang, "Influence maximization in social networks with user attitude modification," 2014 IEEE International Conference on Communications (ICC), Sydney, NSW, 2014, pp. 3913-3918.

[71] B. Liu, G. Cong, Y. Zeng, D. Xu and Y. M. Chee, "Influence Spreading Path and Its Application to the Time Constrained Social Influence Maximization Problem and Beyond," in IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 8, pp. 1904-1917, Aug. 2014.

[72] B. Liu, G. Cong, D. Xu and Y. Zeng, "Time Constrained Influence Maximization in Social Networks," 2012 IEEE 12th International Conference on Data Mining, Brussels, 2012, pp. 439-448.

[73] B. V. Srinivasan, N. Anandhavelu, A. Dalal, M. Yenugula, P. Srikanthan and A. Layek, "Topic-based targeted influence maximization," 2014 Sixth International Conference on Communication Systems and Networks (COMSNETS), Bangalore, 2014, pp. 1-6.

[74] X. Deng, Y. Pan, Y. Wu and J. Gui, "Credit Distribution and influence maximization in online social networks using node features," 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Zhangjiajie, 2015, pp. 2093-2100.

[75] H. Nguyen and R. Zheng, "On Budgeted Influence Maximization in Social Networks," in IEEE Journal on Selected Areas in Communications, vol. 31, no. 6, pp. 1084-1094, June 2013.

[76] H. Hu; Y. Wen; S. Feng, "Budget-Efficient Viral Video Distribution Over Online Social Networks: Mining Topic-Aware Influential Users," in IEEE Transactions on Circuits and Systems for Video Technology , vol.PP, no.99, pp.1-1

[77] H. Zhang, H. Zhang, A. Kuhnle and M. T. Thai, "Profit maximization for multiple products in online social networks," IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, 2016, pp. 1-9.

[78] D.B. Johnson, “A note on Dijkstra’s shortest path algorithm,” Journal of the ACM. 20: 3. 1973.

[79] A. Aini and A. Salehipour, “Speeding up the Floyd–Warshall algorithm for the cycled shortest path problem,” Applied Mathematics Letters 25: 1. January 2012.

[80] I.S. Alshawi, L.Yan, W. Pan and B. Luo, “Lifetime enhancement in wireless sensor networks using fuzzy approach and A-star algorithm,” IET Conference on Wireless Sensor Systems. 2012.

[81] C. Cheng, R. Riley, S.P.R. Kumar and J.J. Garcia-Luna-Aceves, “A loop-free extended Bellman-Ford routing protocol without bouncing effect,” In Proc. of Symposium proceedings on Communications architectures & protocols Pp. 224-236. 1989.

[82] Details omitted due to double-blind reviewing.

[83] S.Navlakha, R. Rastogi and N.Shrivastava, “Graph Summarization with Bounded Error,” In SIGMOD, 2008.

[84] H.Toivonen, F. Zhou, A. Hartikainen and A. Hinkka, “Compression of weighted graphs,” In KDD, 2011.

[85] S.Hangal, D.Maclean, M.S. Lam, and J. Heer, "All Friends are Not Equal: Using Weights in Social Graphs to Improve Search," Proc. of the 4th SNA-KDD Workshop '10 (SNA-KDD'10). Washington: ACM (2010).

[86] J. Y. Yen, "Finding the K shortest loopless paths in a network," Management Science, 17:712–716, 1971.

[87] B. Kwak, N. Song and L.E. Miller, "Performance analysis of exponential backoff," In Proc. of 2005 IEEE/ACM Transactions on Networking vol 13, Isue 2, April 2005.

[88] P.G. Toomey, N.A. Vohra, T. Ghansah, A.A. Sarnaik, S.A.J.C.C. Pilon-Thomas, Immunotherapy for gastrointestinal malignancies, 20 (2013) 32-42.

[89] C. Pöttgen, M.J.C.t.r. Stuschke, Radiotherapy versus surgery within multimodality protocols for esophageal cancer–a meta-analysis of the randomized trials, 38 (2012) 599-604.

[90] M.D. Vesely, R.D.J.A.o.t.N.Y.A.o.S. Schreiber, Cancer immunoediting: antigens, mechanisms, and implications to cancer immunotherapy, 1284 (2013) 1-5.

[91] T.J. Zumwalt, A.J.C.c.c.r. Goel, Immunotherapy of metastatic colorectal cancer: prevailing challenges and new perspectives, 11 (2015) 125-140.

[92] N. Ali, E. Amer, H. Zayed, Understanding Medical Text Related to Breast Cancer: A Review, in: International Conference on Advanced Intelligent Systems and Informatics, Springer, 2017, pp. 280-288.

[93] N. Cancer Genome Atlas Research, J.N. Weinstein, E.A. Collisson, G.B. Mills, K.R. Shaw, B.A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, J.M. Stuart, The Cancer Genome Atlas Pan-Cancer analysis project, Nat Genet, 45 (2013) 1113-1120.

[94] J.F. Fontaine, A. Barbosa-Silva, M. Schaefer, M.R. Huska, E.M. Muro, M.A. Andrade-Navarro, MedlineRanker: flexible ranking of biomedical literature, Nucleic Acids Res, 37 (2009) W141-146.

[95] A. Barbosa-Silva, J.F. Fontaine, E.R. Donnard, F. Stussi, J.M. Ortega, M.A. Andrade-Navarro, PESCADOR, a web-based tool to assist text-mining of biointeractions extracted from PubMed queries, BMC Bioinformatics, 12 (2011) 435.

[96] R.F. Hashimoto, S. Kim, I. Shmulevich, W. Zhang, M.L. Bittner, E.R.J.B. Dougherty, Growing genetic regulatory networks from seed genes, 20 (2004) 1241-1247.

[97] M.H.W. Greenlee, V.G. Honavar, L.A. Hecker, T.A.J.B. Alcon, B. Insights, Using a seed-network to query multiple large-scale gene expression datasets from the developing retina in order to identify and prioritize experimental targets, 2 (2008) 91-102.

[98] D.L. Gibbs, I.J.P.c.b. Shmulevich, Solving the influence maximization problem reveals regulatory organization of the yeast cell cycle, 13 (2017) e1005591.

[99] J.J. Nalluri, P. Rana, D. Barh, V. Azevedo, T.N. Dinh, V. Vladimirov, P.J.S.r. Ghosh, Determining causal miRNAs and their signaling cascade in diseases using an influence diffusion model, 7 (2017) 8133.

[100] Z. Xiang, X. Huang, J. Wang, J. Zhang, J. Ji, R. Yan, Z. Zhu, W. Cai, Y.J.F.i.p. Yu, Cross-database analysis reveals sensitive biomarkers for combined therapy for ERBB2+ gastric cancer, 9 (2018).

[101] M. Esteller, M. Guo, V. Moreno, M.A. Peinado, G. Capella, O. Galm, S.B. Baylin, J.G.J.C.r. Herman, Hypermethylation-associated inactivation of the cellular retinol-binding-protein 1 gene in human cancer, 62 (2002) 5902-5905.

[102] Q. Yao, W. Wang, J. Jin, K. Min, J. Yang, Y. Zhong, C. Xu, J. Deng, Y.J.C.B. Zhou, Synergistic role of Caspase-8 and Caspase-3 expressions: Prognostic and predictive biomarkers in colorectal cancer, (2018) 1-10.

[103] P.E. Czabotar, G. Lessene, A. Strasser, J.M.J.N.r.M.c.b. Adams, Control of apoptosis by the BCL-2 protein family: implications for physiology and therapy, 15 (2014) 49.

[104] Q. Huang, S. Li, P. Cheng, M. Deng, X. He, Z. Wang, C.-H. Yang, X.-Y. Zhao, J.J.W.j.o.g. Huang, High expression of anti-apoptotic protein Bcl-2 is a good prognostic factor in colorectal cancer: Result of a meta-analysis, 23 (2017) 5018.

[105] K. Liu, J. Fan, J.J.M.s.m.i.m.j.o.e. Wu, c. research, Forkhead box protein J1 (FOXJ1) is overexpressed in colorectal cancer and promotes nuclear translocation of β-catenin in SW620 cells, 23 (2017) 856.

[106] M.S. Fernandes, F. Carneiro, C. Oliveira, R.J.I.j.o.c. Seruca, Colorectal cancer and RASSF family—a special emphasis on RASSF1A, 132 (2013) 251-258.

[107] F. Caiazza, E.J. Ryan, G. Doherty, D.C. Winter, K.J.F.i.o. Sheahan, Estrogen receptors and their implications in colorectal carcinogenesis, 5 (2015) 19.

[108] Y. Li, C. Jing, Y. Chen, J. Wang, M. Zhou, X. Liu, D. Sun, L. Mu, L. Li, X.J.M.m.r. Guo, Expression of tumor necrosis factor α-induced protein 8 is upregulated in human gastric cancer and regulates cell proliferation, invasion and migration, 12 (2015) 2636-2642.

[109] R. Aguirre-Gamboa, H. Gomez-Rueda, E. Martínez-Ledesma, A. Martínez-Torteya, R. Chacolla-Huaringa, A. Rodriguez-Barrientos, J.G. Tamez-Pena, V.J.P.o. Trevino, SurvExpress: an online biomarker validation tool and database for cancer gene expression data using survival analysis, 8 (2013) e74250.