



UNIVERSITA' DELLA CALABRIA

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica

Dottorato di Ricerca in

Ingegneria dei Sistemi ed Informatica

Con il contributo del

Ministero dell'Istruzione, dell'Università e della Ricerca

CICLO

XXVII

SOCIAL NETWORKS:

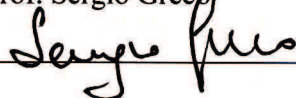
INFLUENCE MAXIMIZATION AND EFFICIENT GRAPH MANAGEMENT

Settore Scientifico Disciplinare ING-INF/05

Coordinatore:

Ch.mo Prof. Sergio Greco

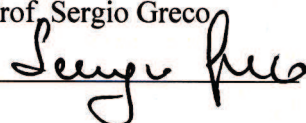
Firma



Supervisore:

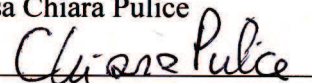
Ch.mo Prof. Sergio Greco

Firma



Dottoranda: Dott.ssa Chiara Pulice

Firma



If you can't explain it simply,
you don't understand it well enough.
Albert Einstein

A mia nonna Angela.
Per il suo volermi bene incondizionato e la felicità
che mi trasmette ogni volta che mi è accanto.

Preface

Social networks became in recent years the most popular platforms for interaction, communication and collaboration between friends. Indeed, they allow people to improve their social connection and to know political and social change almost instantaneously. Moreover, individual entertainment in a wide variety of forms is supported. Social networks also have a great impact in daily life due, in particular, to their effect on the way people are connected each other and can share valuable experiences, disseminate good practices, organize demonstrations and so on. In this respect, significant work on social networks has been done by sociologists and social psychologists since the 1950s, though almost all of this research was carried out on small-scale networks, because of the inherent difficulty of getting reliable, large-scale data from subjects about their friendships.

Nowadays, internet provides a new opportunity to study social interactions on a much larger scale than was previously possible. More in detail, we now have access to large-scale data on interactions and acquaintanceships. Indeed the size of social networks that can be studied has exploded from the hundreds of people to the hundreds of thousands of people or even more. As a result, there are new challenges tied to social networks management such as the efficient computation and maintenance of shortest distances in the network.

In this thesis, we study several questions about social networks from an algorithmic perspective.

In particular, we present two main contributions to the study of social networks:

1. The definition of an effective and efficient technique for shortest distances maintenance;
2. The study of a new and challenging problem such as the non progressive influence maximization.

Our contributions consist both in theoretical discussion on the topics, analysis of related work and finally definition of suitable algorithms. Indeed, we will provide the reader a complete tool for understanding the basic notion

of social networks, shortest paths and social influence, that will guide her through the comprehension of their underlying properties and algorithms.

Organization

This thesis is organized into three parts. Part I is a background on social networks. We begin with discussing some structural properties to move then on to the question of social influence and its effectiveness. In part II we describe the challenging related to the shortest distance evaluation problem on the actual graphs, highlighting limits of the current solutions. We discuss our proposal to efficiently maintain all-pairs shortest distances for graphs stored in relational databases. Finally, part III continues the analysis introduced in part I but at the more fine-grained level of viral marketing. We introduce the influence maximization problem and some of its extensions considered in literature, before concluding with the discussion of a brand new relevant problem and our solution to solve it.

Summary of Publications

The results presented in part II of this thesis will appear in the *Journal of Computer and System Sciences* (JCSS 2015). Results reported in part III have been submitted to a top conference and are currently undergoing the review process.

Ringraziamenti

La tesi è la sintesi di un percorso di studi, ma anche di vita. E scrivere questa sezione mi mette sempre in grossa difficoltà..perché non è facile riuscire a ringraziare in poche parole e senza scadere nelle classiche frasi fatte e di circostanza tutti coloro che mi hanno aiutato e supportato in questi anni. Ringrazio la mia famiglia, che, nonostante gli alti e gli innumerevoli bassi, mi è sempre stata a suo modo vicina. Soprattutto durante i mesi all'estero, con le telefonate giornaliere di ore che facevano tanto sorridere i miei amici canadesi..a pensarci adesso mi stupisco di come ci fosse sempre, da un giorno all'altro, qualcosa di nuovo da raccontare!

Grazie ai miei "soci" Elio e Nunzio per la vostra amicizia e l'enorme pazienza dimostrata, soprattutto negli ultimi tempi.

Ringrazio i miei colleghi Deborah, Roberto e Salvatore perché avete reso questi anni di dottorato più leggeri e pieni di risate. Un ringraziamento particolare a Cristian Molinaro per tutto l'aiuto, i consigli e soprattutto l'incoraggiamento durante le varie prove sostenute.

Il ringraziamento più sincero va al mio supervisore Prof. Sergio Greco. In questi 10 anni qui all'università, sei sempre stato per me un punto di riferimento e un modello da seguire. Perché nonostante il ruolo che rivesti, mi ha

VIII Preface

sempre colpito l'umiltà con cui affronti le cose e il modo in cui riesci a mettermi a mio agio anche nei momenti in cui mi sembrava di non aver fatto abbastanza. Non finirò mai di ringraziarti per la fiducia accordatami e l'opportunità che mi hai dato nel fare questo dottorato proprio con te.

Grazie!

Contents

Part I Social Networks

1	General Keynotes	2
1.1	Structural Properties	2
1.1.1	Scale Free Networks	2
1.1.2	Small World Phenomenon	3
1.2	Information and Influence Diffusion	5
1.2.1	Social Influence	6

Part II Shortest Distances Management in Large Networks

2	Shortest Distances Problem	10
2.1	Definition	11
2.2	Non Incremental Algorithms	11
2.2.1	Limits	11
2.3	Incremental Algorithms	12
3	Shortest Distances Maintenance on DBMS	15
3.1	Preliminaries	15
3.2	Incremental Maintenance of All-Pairs Shortest Distances	17
3.2.1	Single Edge Management	19
3.2.2	Multiple Edges Management	27
3.3	Experimental Evaluation	33
3.3.1	Datasets	33
3.3.2	Experimental setup	35
3.3.3	Results on the DIMACS dataset	36
3.3.4	Results on the DIMES dataset	39
3.3.5	Results on the regular dataset	41
3.4	Discussion	42

Part III Non-Progressive Models for Viral Marketing in Social Networks

4	Influence Maximization Problem	45
4.1	Modeling Cascading Behavior	45
4.1.1	Independent Cascade Model	46
4.1.2	Linear Threshold Model	48
4.2	Problem Statement	50
4.2.1	Greedy Algorithm for Influence Maximization	51
4.3	Extensions	52
4.3.1	Influence Maximization under Competition	53
4.3.2	Continuous-Time Diffusion Process	54
5	Competitive, Continuous Time and Non-Progressive Influence Maximization	56
5.1	Problem Statement and Related Works	57
5.1.1	Non-Progressive Influence Maximization Problem	58
5.2	CT Non-Progressive K-LT Model	59
5.2.1	Non-Progressiveness Property	59
5.2.2	Continuous Time Property	61
5.2.3	Model Definition	62
5.3	NPK-LT Model Properties	63
5.3.1	Reachability under the NP-LE Model	65
5.3.2	Monotonicity and Submodularity	67
5.4	Discussion	71
	Conclusions	72
	References	74

Social Networks

General Keynotes

As social networks are gaining popularity, sociologists and computer scientists have investigated their properties and many results concerning their structure, social influence, social groupings, disease and information propagation, have been obtained. In the following we discuss most relevant social networks properties and introduce some opportunities that markets, online campaigning and viral marketing are exploiting by knowing how people are influenced by the decisions of their social environment. We also describe the potential speed of information diffusion and the dynamics of other kinds of contagion that can spread through these networks.

1.1 Structural Properties

Social network is a collection of people in which some pairs of these people are connected by links. Given their usual large size, it is generally difficult to summarize the whole network succinctly; there are parts that are more or less densely interconnected, sometimes with central “hubs” containing most of the links, and sometimes with natural splits into multiple tightly-linked regions. Participants in the network can be more central or more peripheral; they can straddle the boundaries of different tightly-linked regions or sit squarely in the middle of one. As a result, developing a language for talking about the typical structural features of social networks is an important first step in understanding them.

1.1.1 Scale Free Networks

The structure of a social network is modeled by a graph. A graph is a mathematical way of specifying relationships among a collection of entities. It consists of a set of objects, called *nodes* (or vertices), with some pairs of these objects connected by *links* (or edges). Two nodes connected by edges are named *neighbors*. In social networks, nodes represent people and edges model

their social interaction. More formally, a graph consisting of n nodes and m edges is a pair (V, E) where $V = \{v_1, v_2, \dots, v_n\}$ is the set of nodes and $E \subseteq V \times V$ is the set of edges. A graph is *undirected* if E is a set of unordered pairs; otherwise it is *directed* (i.e. edges are ordered pairs). In the following, we consider directed graphs unless specified otherwise. Relationships within a graph are commonly represented by a $n \times n$ adjacency matrix $g = [g_{ij}]_{i,j \in V}$, where $g_{ij} \in \{0, 1\}$ represents the existence of an edge from node i to node j . Each g_{ij} value can also be non-binary. In this case, it represents the intensity of the interaction among people. Typically, edges of a social network can have a wide range of possible intensity, but for conceptual simplicity, they are classified according to the following categories: *strong ties* (the stronger links, corresponding to friends), and *weak ties* (the weaker links, corresponding to acquaintances) [34]. Simply from their visual appearance, it is clear that social networks structure is quite complex. For more than 40 years, science treated all complex networks as being completely random [22]. In a random network nodes follow a Poisson distribution, and it is extremely rare to find nodes having a number of links that significantly deviates from average. Random networks are also called exponential, because the probability that a node is connected to h other vertices decreases exponentially for large h . However, a variety of complex systems share an important property: some nodes have a tremendous number of connections to other nodes, whereas most nodes have just a handful [5]. The popular nodes, called hubs, can have hundreds, thousands or even millions of links. Networks containing such important nodes are called “scale free”, since these networks appear to have no scale. Social Networks are scale-free [49]. In scale-free networks, the distribution of node linkages follows a power law. Power laws describe systems in which most nodes have just a few connections and some have a tremendous number of links. Power laws are quite different from the bell-shaped distributions that characterize random networks. Indeed, a power law does not have a peak, as a bell curve does, but is instead described by a continuously decreasing function: the probability that a node has degree h is proportional to $h^{-\gamma}$ for large h where $\gamma > 1$ is the power-law coefficient. In Figure 1.1 there is a comparison of these two kinds of network. The upper section (Figure 1.1 (a)) shows a random network and the bell curve distribution of its node linkages, while in the bottom section (Figure 1.1 (b)) there is a scale-free network with a distribution of links resulting in a power law.

1.1.2 Small World Phenomenon

Since social networks are inherently dynamic, it is also useful to think about their evolution over time, i.e. the mechanisms by which nodes are added to or removed from the network and by which edges arise and vanish. One of the most important principles is that if two people in a social network have a friend in common, then there is an increased likelihood that they will become friends themselves at some point in the future. This principle is known as

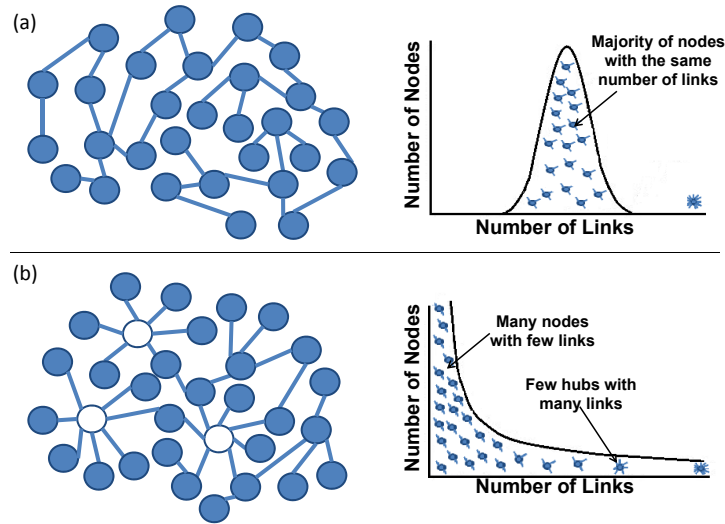


Fig. 1.1. Random versus Scale-Free Networks

triadic closure. The key role of triadic closure in social networks has motivated the formulation of simple social network measures to capture its prevalence.

One of these is the *clustering coefficient* [65]. The clustering coefficient of a node u is defined as the probability that two randomly selected friends of u are friends with each other. In other words, it is the fraction of pairs of u 's friends that are connected to each other by edges. In general, the clustering coefficient of a node ranges from 0 (when none of the node's friends are friends with each other) to 1 (when all of the node's friends are friends with each other). In social networks, this coefficient is significantly high because people tend to have friends who are also friends with each other.

Further significant parameters in understanding social networks properties are the diameter and the average path length. The former is the maximum distance between any pair of nodes in the graph, while the latter is the average number of hops required to get from one node to another over all pairs of nodes in the graph by following the shortest route possible. In 1967 Milgram, a social psychologist at Harvard University, conducted an experiment to find out the average path length between two Americans. He sent hundreds of letters to people in Nebraska, asking them to forward the correspondence to someone they knew in order to eventually reach a designated target person living in Boston as quickly as possible. Milgram found out that the letters that arrived at the final recipient had passed through an average of six individuals, resulting in the phenomenon colloquially known as the *six degrees of separation*. Although Milgram's experiment was hardly conclusive - most of the letters never reached the destination - researchers have recently learned that other networks exhibit this *small-world* property. Indeed, despite social

networks usually have a size comparable to the Web graph, they have average path lengths and diameters remarkably short [49]. This result is quite surprisingly if we consider that social networks abound in triangles, i.e. sets of three people who mutually know each other. As a result, when we think about the nodes you can reach by following edges from your friends, many of these edges go from one friend to another, not to the rest of world. Thus, from the local perspective of an individual, the social network appears to be highly clustered, not the kind of massively branching structure that would more obviously reach many nodes in a few steps. However, the presence of weak ties (the links to acquaintances that connect us to parts of the network that would otherwise be far away), is enough to make the world “small”, with short paths between every pair of nodes. Weak ties can act as bridges, i.e. nodes without which the network will split into two or more subgroups. More in detail, strong ties, representing close and frequent social contacts, tend to be embedded in tightly-linked regions of the network, while weak ties, representing more casual and distinct social contacts, tend to cross between these regions.

One obvious corollary of the small-world phenomenon (there are short paths between most pairs of nodes) is that social networks must have a large connected component containing most of the nodes (there are paths between most pairs of nodes). This giant component typically contains a significant fraction of all the nodes in the network. When a network contains a giant component, it almost always contains only one. Indeed, two co-existing giant components are something really hard to see in real networks because, it’s essentially inconceivable that it does not exist a single edge from someone in the first of these components to someone in the second giant components.

1.2 Information and Influence Diffusion

Diffusion theories have been intensively studied for decades by both epidemiologists and marketing experts. Indeed, there are clear connections between epidemic disease and the diffusion of ideas through social networks. Both diseases and ideas can spread from person to person, across networks connecting people, and in this respect, they exhibit very similar structural mechanisms. For the propagation of any kind of contagion throughout a population, it is common practice to consider the existence of a critical threshold. Any virus, disease or fad that is less infectious than this threshold will inevitably collapse, while those above the threshold will grow exponentially. In the scale-free networks, where hubs are connected to many other nodes, this threshold is zero [53]. Indeed, at least one hub will tend to be infected by any corrupted node. And once a hub has been infected, it will pass the virus to numerous other nodes, eventually compromising other hubs. It means that in social networks, which in many cases appear to be scale-free, all viruses, even those that are weakly contagious, will spread throughout the entire system. Moreover, the

existence of short paths between almost every pair of nodes due to the small world property, has substantial consequences for the potential speed with which information, diseases, and other kinds of contagion can spread through the network. While the ease of propagation can have disruptive effects if we consider virus or diseases, it can also be very beneficial in many business contexts, where companies are interested in starting a viral effect for their products. Viral marketing, for instance, is considerably interested in attracting the attention of the largest possible audience to a brand, a product, or a service. In viral marketing, customers help marketers to promote a product or a service by the so-called “word-of-mouth” advertising. The latter is often a more cost effective type of advertising and, in some cases, more effective in absorbing new customers and making people adopt a new product because people are more affected by their friends or the people they trust. A key question in this category of marketing is finding the best set of people (the most “powerful” or central) so that, targeting them, will speed the adoption of a product to the larger number of people in the network.

1.2.1 Social Influence

All viral marketing strategies are based on the idea that careful targeting a small number of “influential” individuals to use a product, for instance by giving it to them for free or at a discounted price, can have a cascading effect on the adoption of that product. In a network setting, indeed, user actions must not be evaluated as stand alone, but considering that the network will react to them. More in detail, each individual’s actions may be triggered by one of his/her friends recent actions. An example of this scenario is the user purchasing profile. More in detail, it often happens that a user buys a product because one of his/her friends has recently bought the same product. This process has been variously described as *social influence*. Formally, consider a social network represented as a directed graph $G = (V, E)$ and a function $p : E \rightarrow [0, 1]$ assigning a weight or probability $p(u, v)$ (or simply $p_{u,v}$) to every edge $(u, v) \in E$. The latter represents the influence exerted by user u on v . This informally captures the intuition that whenever u performs an action, then v also performs the action after u , with probability $p_{u,v}$. Typically, highest is this probability, larger is the gain that v has in imitating the behavior of u . Indeed, choices made by u can provide indirect information about what it knows and there are immediate payoffs from copying its decisions. As an example, payoffs that arise from using compatible technologies instead of incompatible ones. The success of a viral marketing campaign is strictly related with the identification of situations where social influence between users exists. Indeed, in systems where social influence exists, ideas, modes of behavior, or new technologies can diffuse through the network like an epidemic [3]. Therefore, being able to identify in which cases influence prevails and to detect the most influential nodes are important steps to strategy design.

Influence vs. Homophily

The existence of influence in a network can be difficult to detect because there are other phenomena surrounding users' behavior that are different from influence, but may appear to be as such. One of these is known as *homophily*, the principle that we tend to be similar to our friends, and hence perform similar actions. For example, a person who is overweight tends to have overweight friends. If one of them develops a cardiology disease, followed by one of his/her friends, can we really claim that the health of the first influenced that of the second? Thus, the existence of a social tie does not necessarily cause a certain behavior to propagate. The problem of homophily vs. influence and the introduction of methods for distinguishing them, has been tackled by some researchers [3, 4]. Researchers have also investigated whether influence can really drive substantial viral cascades over real-world social networks. Indeed, as appealing as the viral model marketing seems in theory, its practical implementation is greatly complicated by its low success rate. Even creators of successful viral projects are rarely able to repeat their success with subsequent projects. As a result, there have been studies both supporting the existence of social influence [39, 35] and challenging it [66]. Typically, its effectiveness for applications such as viral marketing depends on the datasets. Thus, before deciding whether to adopt a viral marketing approach, it is recommended a careful analysis of evidence in available datasets.

Influential Nodes

Based on its structural properties, several techniques have been developed to identify key nodes in a social network and a plethora of centrality measures have been defined over the years. Main centrality measures can be summarized in the following three classes:

- **Degree centrality measures:** Number of links a node has with the rest of the networks nodes;
- **Closeness centrality measures:** Average number of “hops” from a given node to all other nodes in the graph;
- **Betweenness centrality measures:** The number of shortest paths that will be affected by a node removal.

Figure 1.2 highlights nodes within a network with the highest values of these centrality measures.

Clearly, degree centrality measures are easy to compute because it is only necessary to count the direct links of the nodes in the network. Nodes with high degree centrality have higher probability of receiving and transmitting information flowing in the network. For this reason, high degree centrality nodes are considered to have great influence over a larger number of nodes and/or are capable of communicating quickly with the nodes in their neighborhood. However, the main disadvantage of the degree centrality measures

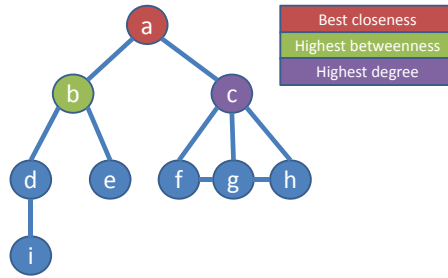


Fig. 1.2. Nodes with different centrality values

is that they only take into account the immediate ties of a node, while indirect contacts are not considered at all. As a consequence, it may happen that a node could be quite central, but only in a local neighborhood. Indeed, it might be tied to a large number of nodes, but those nodes might be rather disconnected from the network as a whole. Closeness centrality approaches, instead, emphasize the distance of a node to all other nodes in the network by focusing on the distance from each node to the others. They are based on the idea that nodes that have a short distance to other nodes, may disseminate information very effectively through the network since they require only few intermediaries for contacting a large number of nodes. Finally, betweenness centrality takes into account the control of the information flow that a node may exert based on its position in the network. This approach assumes implicitly that the communication and interaction between two nodes that are not directly linked depends on the intermediate nodes. Thus, a node is considered to be well connected if it appears in a huge number of shortest paths between pairs of nodes in the network.

**Shortest Distances Management in Large
Networks**

Shortest Distances Problem

The problem of evaluating and maintaining shortest paths between each pair of vertices in a graph has received an increasing attention in recent years due to the several real life scenarios in which information about shortest paths are crucial. Indeed, this problem appears to be critical in many practical applications, such as management of communication or transportation networks, where a prompt reaction to changes (e.g. collapse of a road, crash of a router), through a rapid recalculation of involved shortest paths, is the first form of intervention to ensure high level safety and prevention. There is also a wide range of applications in social network analysis, road networks [67], biological networks [45, 58], keyword search [68], twig-pattern matching [31], graph pattern matching [24], and many others, in which is crucial to know just the shortest distances among vertices. The latter is both an important task in its own right (e.g., if we want to find how close people are in a social network) and an important subroutine for many advanced tasks associated with large graphs (e.g., different measures, such as centrality measures and network diameter, are based on shortest distances). To this end, different variants of this problem have been investigated over the years: *single-pair shortest path* (SPSP) (find a shortest path from a given source vertex to a given destination vertex), *single-source shortest paths* (SSSP) (find a shortest path from a given source vertex to each vertex of the graph), and *all-pairs shortest paths* (APSP) (find a shortest path from u to v for every pair of vertices u and v). Variants of these problems where we are interested only in the shortest distances, rather than the actual paths, have studied as well—we will use SPSD, SSSD, and APSD to refer to the *single-pair shortest distance*, *single-source shortest distances*, and *all-pairs shortest distances* problems, respectively. In this section, we discuss the approaches in the literature to solve the above problems—we first consider non-incremental algorithms and then the incremental ones. The latter have been introduced to deal with graphs subject to frequent updates, since it is impractical to compute shortest paths/distances from scratch every time changes are made to the graph.

2.1 Definition

Let $G = (V, E, \omega)$ be a (directed or undirected) *weighted graph* where $\omega : E \rightarrow \mathbb{R}_0$ is a function assigning a *weight* (or *distance*) to each edge $(u, v) \in E$. A sequence v_0, v_1, \dots, v_l ($l > 0$) of vertices of G is a *path* from v_0 to v_l if $(v_i, v_{i+1}) \in E$ for every $0 \leq i \leq l - 1$. We use $\omega(u, v)$ to denote the weight assigned to the edge (u, v) by ω . Thus, the *weight* (or *distance*) of a path $r = v_0, v_1, \dots, v_l$ can be defined as $\omega(r) = \sum_{i=0}^{l-1} \omega(v_i, v_{i+1})$. Obviously, there can be multiple paths from v_0 to v_l , each having a distance. A path from v_0 to v_l with the lowest distance (over the distances of all paths from v_0 to v_l) is called a *shortest path* from v_0 to v_l and its distance is called the *shortest distance* from v_0 to v_l .

2.2 Non Incremental Algorithms

Since the introduction of the well-known Dijkstra’s algorithm [20], a plethora of algorithms have been proposed to improve on its performance (see [18, 61] for recent surveys). Some of the techniques addressing the APSP problem, such as [1, 54], take into account specific cases (directed/undirected graphs, non-negative weights, etc.), but, in general, the most efficient solutions require an execution time of $O(n^3)$, where n identifies the number of graph’s vertices [42, 10]. Recently there have been introduced several approaches exploiting on demand distance and path computation. Most of these methods require a pre-processing step for building index structures to support the fast computation of shortest paths and distances [11, 41, 2, 60, 28, 69, 55, 56, 36, 57, 70, 14, 27]. In particular, [11, 41, 2, 60] address the SPSP problem, while a similar approach for the SPSP problem has been proposed in [28]. [69] considers both the SPSP and SPSD problems and also requires the pre-computation of an auxiliary hierarchical index structure. There have been also proposals addressing the approximate computation of SPSD [55, 56, 36, 57]. These methods are based on the selection of a subset of vertices as “landmarks” and the offline computation of distances from each vertex to those landmarks. [70] and [14] propose disk-based index structures for solving the SSSP and SSSD problems, while disk-based index structure for the SPSP and SPSD problems have been proposed in [27].

2.2.1 Limits

Besides the fact that most of the aforementioned techniques assume that graphs, shortest distances, and auxiliary index structures fit in the main memory (which is not realistic for large graphs used in many current applications), the main limit of all the approaches mentioned above is that they need to recompute a solution from scratch every time the graph is modified, even if we make small changes affecting a few shortest paths/distances. Since most of

the real networks are large and dynamic, i.e. they are composed by an high number of vertices and the insertion/deletion of connections between these vertices are very frequent operations, the *recomputation* from scratch of the shortest distances may be unfeasible due to its cost and its limited effectiveness. Indeed, it is unlikely that graph changes cause an update of the majority of shortest distances. In general, a good heuristic is that small variations on graph correspond to little changes to distances among vertices. The following example clarifies this assertion.

Example 2.1. Consider the directed graph represented by solid edges in Figure 2.1(left). Suppose to add the dashed edge $(b, c, 1)$.

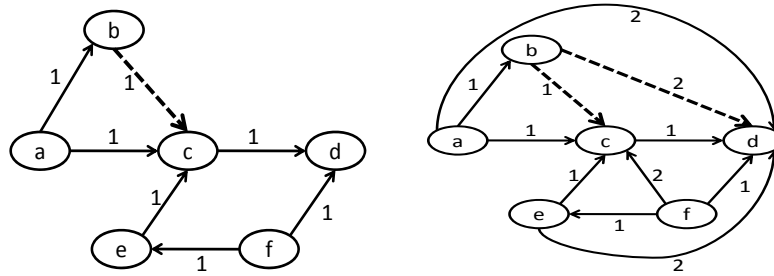


Fig. 2.1. Updated graph (left) and the corresponding shortest distances (right).

In Figure 2.1 (right) we show the resulting shortest distances graph, where the insertion of the edge $(b, c, 1)$ affects only two connections (the dashed edges), while most of the graph remains unchanged.

Therefore, these methods work well if the graph is static and the pre-processing phase, required to build the index structures that are leveraged to answer queries, needs to be done only once; in contrast, if the graph is dynamic, then the expensive pre-processing phase has to be done every time a change is made to the graph and this is impractical for large graphs subject to frequent changes.

2.3 Incremental Algorithms

Recently, several techniques have been proposed to incrementally maintain shortest paths and distances when graph updates occur. The core idea is to apply the recomputation from scratch only when graphs substantially change. In [40] the authors introduce useful data structures to support an arbitrary sequence of delete operations and to easily compute a path between each pair of vertices on a directed acyclic graph. The time complexity of the algorithm for edge deletions management is $O(nm)$ in the worst case, where m is the

number of edges and n the number of vertices. In [19] the dynamic maintenance of APSP is tackled both for edge insertions and edge deletions. The proposed algorithm requires $O(n^2 \log^3 n)$ amortized time per update. In [6] the authors present a hierarchical scheme for efficiently maintaining approximate APSP in undirected unweighted graphs under deletions of edges. The proposed algorithm works in sub-cubic time and needs an upper bound for the path length. In [44] a fully dynamic algorithm for maintaining transitive closure and APSP in digraphs with positive integer weights bounded by a constant b is presented. In [17] the authors propose an algorithm for maintaining nearest neighbor lists in weighted graphs under vertex insertions and decreasing edge weights. This work is tailored for scenarios where queries are a lot more frequent than updates.

All the techniques above share the use of specific (complex) data structures and work in the main memory, which limits their applicability to very large graphs (the number of shortest distances is quadratic in the number of vertices in the worst case).

On the other hand, the growing availability of computer networks led to the development of distributed algorithms, where each vertex is a resource (e.g., a router). In [16] the authors propose a distributed solution to manage the interleaving of insert and delete operations on a network with positive real edge weights. In [15] is addressed the issue of updating shortest paths when multiple edge changes occur simultaneously. The efficiency of these techniques is evaluated w.r.t. the total number of messages sent over the edges and the space required to each vertex. As other distributed solutions known in the literature, these algorithms are based on distance-vector routing protocol and, consequently, suffer of some typical drawbacks, such as a slow convergence to the correct distance and the looping phenomenon, which contribute to restrict their applicability.

Aside to algorithms working on specifically designed data structures, there are many algorithms which exploit database systems, providing structurally simple solutions for the computation of the new content of a database after updates. More in details, when graphs are stored in relational databases, the shortest distance maintenance problem is a special case of the view maintenance problem. The view maintenance problem for both recursive and non-recursive views has been addressed in [37]. Depending on the view type, i.e. recursive or not, authors define two algorithms, respectively *DRed algorithm* and *Counting algorithm*. The latter exploits the number of alternative derivations for a tuple, avoiding not necessary tuple deletions till this number is greater than zero. *DRed algorithm* works on recursive views, thus it can be used for shortest distance maintenance. The main drawback of this technique is that it considers all possible derivations of a tuple only after that this is already deleted. This operation may be expensive on massive graphs because it could remove (and thus causes recomputation) a much larger graph portion than that effectively affected by the modification. A similar approach has been adopted in [52] where the authors propose incremental algorithms to maintain

all pairs of shortest distances for graphs stored in relational DBMSs after edge insertions and deletions. These algorithms improve on [37] by avoiding unnecessary joins and unnecessary tuple deletions in the original graph—however, [52] can deal only with the APSD maintenance problem while [37] works with general views (both in SQL and Datalog).

Shortest Distances Maintenance on DBMS

Although many incremental algorithms to solve the shortest paths/distances maintenance problem have been proposed, they are designed to work in the main memory. This significantly limits their applicability to many current applications where graphs are very large and, consequently, it is prohibitive to keep all shortest distances in the main memory. To the best of our knowledge, [52] is the only disk-based approach for the incremental maintenance of all pairs shortest distances. Specifically, [52] considers graphs stored in relational DBMSs. However, the proposed algorithms can handle only a single insertion or deletion at a time. Moreover, our experimental evaluation revealed that algorithm devoted to edge insertion management performs well on large but sparse graphs, while edge deletion management offers poor performances even on small graphs. To overcome the above mentioned limitations, we propose a novel approach that works on large graphs stored in relational DBMSs and that tackles the shortest distance maintenance problem when graph changes occur. In particular, we introduce two algorithms supporting both edge insertions and deletions on graphs stored in a DBMS. The proposed solution aims to reduce the time needed to deal with the graph updates, avoiding recomputation of shortest distances not affected by the changes made to the graph. Moreover, we designed our algorithms in order to manage multiple insertions (or deletions) in a single step. The validity of the proposed techniques was confirmed by deep and accurate experimental evaluation in which they have been tested considering both real-world and synthetic networks. We also compared our results with the approaches proposed in [52] to assess our superior performances.

3.1 Preliminaries

In this section, we introduce the notation and terminology used in the rest of this chapter. We recall that, for our purposes, graphs and shortest distances are stored in relational databases. Notice that this allows us to take advantage

of full-fledged optimization techniques provided by relational DBMSs. Specifically, the set of edges of a graph is stored in a ternary relation \mathbb{E} containing a tuple (a, b, w) iff there is an edge in the graph from a to b with weight w . We call \mathbb{E} an *edge relation*. Likewise, a ternary relation SD is used to store the shortest distances for all pairs of vertices of the graph, that is, $(x, y, d) \in SD$ iff there is a path from x to y and d is the shortest distance from x to y . We also say that SD is *the shortest distance relation for \mathbb{E}* . Without loss of generality, we assume that graphs do not have self-loops, i.e., edges of the form (a, a) (the reason is that self-loops can be disregarded for the purpose of finding shortest distances). Moreover, we consider directed weighted graphs and call them simply graphs—the extension of the proposed algorithms to undirected graph is trivial.

We deal with the shortest distance maintenance problem defined below.

Problem (All-pairs shortest distances maintenance). Given an edge relation \mathbb{E} , the shortest distance relation SD for \mathbb{E} , and a set of edges $\Delta\mathbb{E}$, compute the shortest distance relation for $\mathbb{E} \cup \Delta\mathbb{E}$ (or $\mathbb{E} - \Delta\mathbb{E}$).

We are interested in solving the problem in an efficient *incremental* fashion, i.e., avoiding to compute the new shortest distance relation from scratch. It is worth noting that the case where we want to compute the shortest distance relation for $\mathbb{E} \cup \Delta\mathbb{E}$ (resp. $\mathbb{E} - \Delta\mathbb{E}$) corresponds to the scenario where the original edge relation \mathbb{E} is modified by adding new edges (resp. deleting edges). In addition to the edge relation for a graph and the corresponding shortest distance relation, our algorithms will use auxiliary relations of arity 3 and 5. Auxiliary relations of arity 3 will be used to store tuples of the form (x, y, d) , called *distance tuples*, whose meaning is that there is an edge from x to y with weight d , or there is a path from x to y with distance d . Relations of arity 5 will store tuples of the form (x, y, d, a, w) , called *extended distance tuples*, whose meaning is that there is a path from x to y with distance d and either the first edge along the path is (x, a) with weight w or the last edge along the path is (a, y) with weight w .

We will use the relational algebra operators π (projection), \bowtie (join), \ltimes (left semi-join), \times (Cartesian product), \cup (union), and $-$ (difference). We will refer to the i -th attribute of a relation as $\$i$. For instance, the projection of a relation R on the first and third attribute is written as $\pi_{\$1, \$3} R$. We will use the generalized projection so that we can write expressions like $\pi_{a, \$1} R$, which is equivalent to $\{a\} \times \pi_{\$1} R$.

Given a tuple $\varphi = (\varphi_1, \dots, \varphi_n)$, the i -th element of φ is denoted as $\varphi[i]$. Given two tuples φ_1 and φ_2 , we say that φ_1 and φ_2 are *similar*, denoted $\varphi_1 \sim \varphi_2$, iff $\varphi_1[1] = \varphi_2[1] \wedge \varphi_1[2] = \varphi_2[2]$. Intuitively, as we are dealing with relations storing distances between vertices, two tuples are similar when they refer to (possibly different) paths between the same pair of vertices.

Below we define the operators min , \oplus , and \ominus , which will be used in the proposed algorithms. Let Q and R be relations containing distance tuples or extended distance tuples (and thus of arity 3 or 5). The min operator is

defined as follows:

$$\min(Q) = \{\varphi \in Q \mid \varphi[1] \neq \varphi[2] \wedge \nexists \varphi' \in Q \text{ s.t. } \varphi' \sim \varphi \wedge \varphi'[3] < \varphi[3]\}$$

Thus, $\min(Q)$ returns all the (extended) distance tuples φ in Q with $\varphi[1] \neq \varphi[2]$ (i.e., φ refers to a path whose endpoints are distinct vertices) and s.t. Q does not contain a similar (extended) distance tuple with a strictly lower distance. The \min operator with two arguments is defined as follows:

$$\min(Q, R) = \{\varphi \in \min(Q) \mid \nexists \varphi' \in R \text{ s.t. } \varphi' \sim \varphi \wedge \varphi'[3] \leq \varphi[3]\}.$$

Thus, $\min(Q, R)$ first applies \min to Q and then returns all the (extended) distance tuples φ in the resulting relation s.t. R does not contain a similar (extended) distance tuple with a lower distance.

The binary operators \oplus and \ominus are defined as follows:

$$\begin{aligned} Q \oplus R &= Q \cup \{\varphi \in R \mid \nexists \varphi' \in Q \text{ s.t. } \varphi' \sim \varphi\} \\ Q \ominus R &= Q - \{\varphi \in Q \mid \exists \varphi' \in R \text{ s.t. } \varphi' \sim \varphi \wedge \varphi'[3] = \varphi[3]\} \end{aligned}$$

Thus, $Q \oplus R$ returns a relation obtained by adding to Q the (extended) distance tuples of R which are not similar to any of the (extended) distance tuples in Q , whereas $Q \ominus R$ returns a relation obtained from Q by deleting every (extended) distance tuple for which there exists a similar (extended) distance tuple in R with the same value on the third attribute (i.e., with the same distance). Notice that all the operators above can be expressed in the relational algebra as well as in SQL.

3.2 Incremental Maintenance of All-Pairs Shortest Distances

In this section, we present novel algorithms for the incremental maintenance of shortest distances. We start by introducing algorithms to handle the insertion and deletion of a *single* edge. After that, we propose algorithms able to deal with the insertion and deletion of *multiple* edges *at once*. As shown in Section 3.3, the latter algorithms outperform the former ones even with a single insertion/deletion. We report the single insertion/deletion algorithms as they ease presentation of the ones for multiple insertions/deletions.

We point out that edge weight updates can also be handled by our algorithms, as changing the weight w of an edge (a, b) into w' can be handled by first deleting (a, b, w) and then inserting (a, b, w') . Furthermore, our algorithms to deal with the insertion and deletion of multiple edges at once can be used to handle an arbitrary sequence of mixed edge insertions/deletions/updates. In fact, given an edge relation \mathbb{E} , the shortest distance relation SD for \mathbb{E} , and a sequence of edge insertions/deletions/updates $\Delta\mathbb{E}$, we can proceed as follows:

- **Step 1.** We apply the sequence of operations in $\Delta\mathbb{E}$ to \mathbb{E} so as to get the final edge relation $\widehat{\mathbb{E}}$.
- **Step 2.** We then apply the algorithm to handle multiple deletions with the following input: \mathbb{E} is the edge relation, SD is the shortest distance relation, and $\mathbb{E} - \widehat{\mathbb{E}}$ are the edges to be deleted. This gives us the shortest distance relation SD' for $\mathbb{E}' = \mathbb{E} - (\mathbb{E} - \widehat{\mathbb{E}})$.
- **Step 3.** Finally, we apply the algorithm to handle multiple insertions with the following input: \mathbb{E}' is the edge relation, SD' is the shortest distance relation, and $\widehat{\mathbb{E}} - \mathbb{E}$ are the edges to be inserted. This gives us the shortest distance relation for $\widehat{\mathbb{E}}$.

The following example illustrates how we can manage a sequence of edge insertions/deletions/updates by following the steps outlined above.

Example 3.1. Consider the graph in Figure 3.1 (left) corresponding to the edge relation $\mathbb{E} = \{(a, b, 1), (b, c, 2), (c, d, 3), (d, b, 2)\}$. The set of changes we want to make to the graph are represented in Figure 3.1 (center) and consists of: (i) deleting $(b, c, 2)$ (red dotted edge), (ii) inserting $(d, a, 1)$ (blue edge), and (iii) decreasing the cost of $(d, b, 2)$ from 2 to 1. Making these changes to the original edge relation yields the new edge relation $\widehat{\mathbb{E}} = \{(a, b, 1), (c, d, 3), (d, a, 1), (d, b, 1)\}$ whose graph is reported in Figure 3.1 (right)—this is the output of step 1.

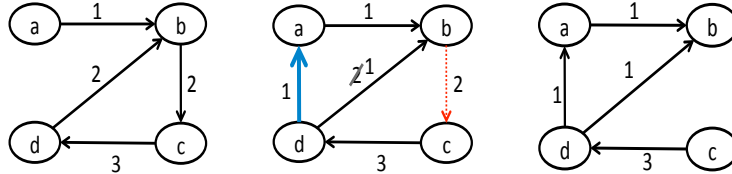


Fig. 3.1. Original graph (left), set of changes (center), and the updated graph (right).

The incremental maintenance of the shortest distance relation involves a multiple edges deletion (step 2) followed by a multiple edges insertion (step 3). More in detail, step 2 consists in the computation of the shortest distance relation SD' for the edge relation \mathbb{E}' obtained from the original one by deleting the set $\mathbb{E} - \widehat{\mathbb{E}} = \{(b, c, 2), (d, b, 2)\}$ —these are the red dotted edges in Figure 3.2 (left). Then, step 3 consists in computing the shortest distance relation for the edge relation obtained from \mathbb{E}' by inserting the new edges $\widehat{\mathbb{E}} - \mathbb{E} = \{(d, a, 1), (d, b, 1)\}$ —these are the blue edges in Figure 3.2 (right).

Also, it is worth noting that insertions and deletions of *vertices* can be straightforwardly reduced to our setting and thus be handled by our algorithms too: vertex insertions (resp. deletions) are handled by inserting (deleting) all edges that are incident from/to the inserted (resp. deleted) vertices.

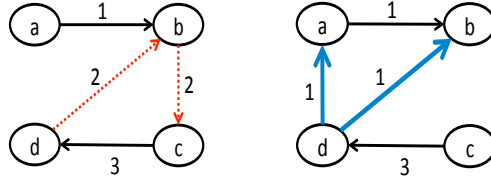


Fig. 3.2. Deletion step (left) and insertion step (right).

As a consequence, our algorithms can easily handle an arbitrary sequence of edge insertions/deletions/updates and vertex insertions/deletions.

3.2.1 Single Edge Management

In this section, we first propose an algorithm to handle the insertion of a single edge and then address the deletion of a single edge. Moreover, we discuss in detail the solution proposed in [52] to better understand the difference between the approaches.

Single edge insertion

Algorithm 1 deals with the insertion of a single edge. We point out that the algorithm (as well as all the other algorithms proposed in this paper) is written in a form that eases presentation without applying optimizations to relational algebra expressions. However, relational DBMSs have full-fledged query optimization techniques to easily optimize the code—indeed, this is one of the advantages of relying on a relational DBMS.

Given the shortest distance relation SD for an edge relation \mathbb{E} , and an edge $e = (a, b, w)$ to be added to \mathbb{E} , Algorithm 1 computes the shortest distance relation for $\mathbb{E} \cup \{e\}$. The precondition $\nexists (a, b, w') \in \mathbb{E}$ with $w' \neq w$ ensures that there does not already exist an edge from a to b . The additional precondition $\nexists (a, b, w') \in SD$ with $w' \leq w$ is imposed just because if it does not hold, then the insertion of e has no effect on the shortest distance relation and thus there is no need to recompute it.

The algorithm works as follows. First, it computes all the distance tuples obtained by concatenating e with shortest paths starting from vertex b (step 1), that is, tuples of the form (a, y, d) s.t. there is a path from a to y with distance d , the first edge along such a path is e , and the path from b to y is a shortest one (w.r.t. the original edge relation). Then, among these distance tuples, the algorithm selects only those that improve on current shortest distances, that is, those tuples (a, y, d) in ΔP_f s.t. either there is no shortest path from a to y in SD or there is one with distance greater than d (step 2).

Next, two analogous steps are performed: first, shortest paths ending in vertex a are concatenated with e (step 3), yielding tuples of the form (x, b, d) s.t. there is a path from x to b with distance d , the last edge along such a

Algorithm 1 Single-Edge-Insertion-Maintenance

Input: Edge relation \mathbb{E}
Shortest distance relation SD for \mathbb{E}
Edge $e = (a, b, w)$ s.t. $\nexists(a, b, w') \in \mathbb{E}$ with $w' \neq w$ and
 $\nexists(a, b, w') \in SD$ with $w' \leq w$

Output: Shortest distance relation for $\mathbb{E} \cup \{e\}$

- 1: $\Delta P_f = \pi_{a, \$2, \$3+w}(\sigma_{\$1=b} SD)$;
- 2: $\Delta SD_f = \min(\Delta P_f, SD)$;
- 3: $\Delta P_\ell = \pi_{\$1, b, \$3+w}(\sigma_{\$2=a} SD)$;
- 4: $\Delta SD_\ell = \min(\Delta P_\ell, SD)$;
- 5: $\Delta P_i = \pi_{\$1, \$5, \$3+\$6-w}(\Delta SD_\ell \times \Delta SD_f)$;
- 6: $\Delta SD = \min(\Delta SD_\ell \cup \Delta P_i \cup \Delta SD_f \cup \{e\}, SD)$;
- 7: $SD^* = \Delta SD \oplus SD$;
- 8: **return** SD^* ;

path is e , and the path from x to a is a shortest one (w.r.t. the original edge relation); then, among the tuples in ΔP_ℓ , the algorithm selects only those that improve on shortest distances in SD (step 4).

After that, the distance tuples obtained at steps 2 and 4 (which correspond to path whose first or last edge is e , respectively) are combined via a Cartesian product (step 5). Specifically, this step computes a relation ΔP_i by combining each tuple (x, b, d_1) in ΔSD_ℓ with each tuple (a, y, d_2) in ΔSD_f so as to get a tuple $(x, y, d_1 + d_2 - w)$. Notice that the distance of tuples in ΔP_i is diminished of w because e is taken into account both in tuples of ΔSD_ℓ and in tuples of ΔSD_f .

Finally, the algorithm selects those tuples in $\Delta SD_\ell \cup \Delta P_i \cup \Delta SD_f \cup \{e\}$ that improve on the shortest distances in SD (step 6) and incorporates them into SD (step 7). The following example illustrates how Algorithm 1 works.

Example 3.2. Consider the directed graph in Figure 3.3 (left) whose shortest distance relation SD is represented in Figure 3.3 (right).

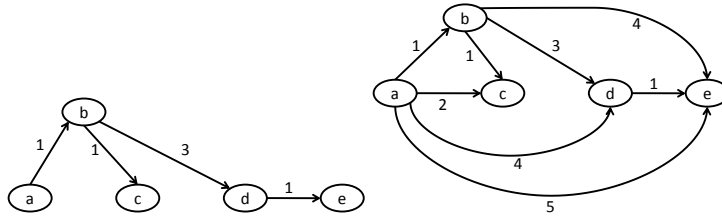


Fig. 3.3. A graph (left) and the corresponding shortest distances (right).

Suppose we add the edge $(c, d, 1)$. Figure 3.4 shows different relations computed by Algorithm 1 in steps 1–5, namely $\Delta SD_f = \{(c, e, 2)\}$ (red edge),

$\Delta SD_\ell = \{(b, d, 2), (a, d, 3)\}$ (green edges), and $\Delta P_i = \{(b, e, 3), (a, e, 4)\}$ (purple edges).

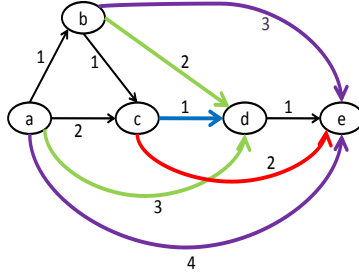


Fig. 3.4. Intermediate steps of Algorithm 1.

Among these new tuples, Algorithm 1 selects those that improve on current shortest distances (step 6)—intuitively, the algorithm selects the colored edges in Figure 3.4 which are not in Figure 3.3 (right) or are in Figure 3.3 (right) with a greater distance. Finally, the selected tuples are incorporated into the shortest distance relation (step 7). Figure 3.5 shows the updated graph and the new shortest distances.

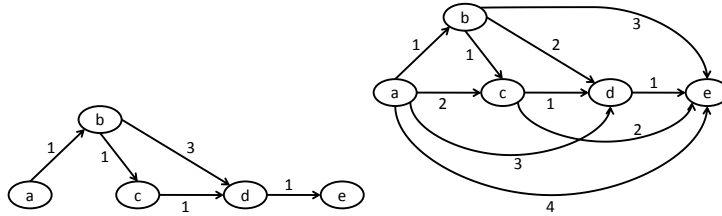


Fig. 3.5. Updated graph (left) and the corresponding shortest distances (right).

Theorem 3.3. Given an edge relation \mathbb{E} , the shortest distance relation SD for \mathbb{E} , and an edge e , Algorithm 1 computes the shortest distance relation for $\mathbb{E} \cup \{e\}$.

Proof. Let $\mathbb{E}_n = \mathbb{E} \cup \{e\}$ and SD_n be the shortest distance relation for \mathbb{E}_n . We start with two observations used in the following. First, if a distance tuple (x, y, d) is in SD^* , then there is a path from x to y in \mathbb{E}_n whose distance is d —indeed, this property holds also for $\Delta P_f, \Delta SD_f, \Delta P_\ell, \Delta SD_\ell, \Delta P_i,$ and ΔSD . Second, SD^* does not contain two distance tuples (x, y, d_1) and (x, y, d_2) s.t. $d_1 \neq d_2$ (to see why, it suffices to look at steps 6–7 and the definitions of min and \oplus).

Soundness ($SD^* \subseteq SD_n$). Let $(x, y, d) \in SD^*$. As noticed above, this means that there is a path from x to y in \mathbb{E}_n ; thus, there must be a shortest one too, which implies that a distance tuple (x, y, d') is in SD_n . We show that $d = d'$. One of the following two cases must occur.

(1) There is a shortest path r' from x to y in \mathbb{E}_n (its distance is d') which does not go through e . Since r' goes only through edges in \mathbb{E} , then $(x, y, d') \in SD$. Since ΔSD contains distance tuples corresponding to paths in \mathbb{E}_n that strictly improve on shortest distances in SD (step 6), then there is no distance tuple (x, y, d') in ΔSD . Since $SD^* = \Delta SD \oplus SD$ (step 7), then $(x, y, d') \in SD^*$. Hence, $d = d'$.

(2) Every shortest path from x to y in \mathbb{E}_n (whose distance is d') goes through e . Let r' be one of such shortest paths. Then, e is either (i) the first edge of r' , or (ii) the last edge of r' , or (iii) an intermediate edge of r' . Notice that in case (i) the subpath of r' that goes from b to y is a shortest path in \mathbb{E}_n and also in \mathbb{E} (because r' does not go through e twice); in case (ii) the subpath of r' that goes from x to a is a shortest path in \mathbb{E}_n and also in \mathbb{E} ; in case (iii) the subpath of r' that goes from x to a and the subpath of r' that goes from b to y are shortest paths in \mathbb{E}_n and also in \mathbb{E} . Now it is easy to see that a distance tuple for r' is computed at step 1 (resp. 3 and 5) when case (i) (resp. (ii) and (iii)) holds. In particular, in case (iii), since every shortest path from x to y in \mathbb{E}_n goes through e , it must be the case that every shortest path from x to b in \mathbb{E}_n has e as last edge, and every shortest path from a to y in \mathbb{E}_n has e as first edge, and thus a distance tuple for r' is computed at step 5. Notice that since every shortest path from x to y in \mathbb{E}_n goes through e , this is the case where the insertion of e strictly improves the shortest distance from x to y . Thus, (x, y, d') belongs to ΔSD (step 6) and SD^* (step 7). Hence, $d = d'$.

Completeness ($SD^* \supseteq SD_n$). Consider a distance tuple (x, y, d) in SD_n . If there is a shortest path r from x to y in \mathbb{E}_n (its distance is d) that does not go through e , then $(x, y, d) \in SD$. This means that ΔSD does not contain a distance tuple (x, y, d) because distance tuples in $\Delta SD_\ell \cup \Delta P_i \cup \Delta SD_f \cup \{e\}$ correspond to paths in SD_n and thus do not strictly improve on p (see step 6). Hence, $(x, y, d) \in SD^*$ (see step 7). If every shortest path from x to y in \mathbb{E}_n goes through e , then it can be verified that $(x, y, d) \in SD^*$ by applying the same reasoning used in part (2) above. \square

Single edge deletion

The algorithm to handle the deletion of a single edge consists of two phases, the first one performed by Algorithm 2 and the second one carried out by Algorithm 3. Roughly speaking, Algorithm 2 deletes from SD shortest distances whose corresponding paths might go through the deleted edge e . Subsequently, Algorithm 3 recomputes the new shortest distances between vertices affected by the deletion of e .

The precondition $e \in (\mathbb{E} \cap SD)$ is to consider only significant cases: if $e \notin \mathbb{E}$ then no edge is effectively deleted from the graph, whereas if $e = (a, b, w) \notin$

SD then there is a path from a to b not using e and with a distance strictly lower than w —in this case, the deletion of e does not affect any shortest distance (recall that we consider non-negative weights).

Algorithm 2 Single-Edge-Deletion-Maintenance

Input: Edge relation \mathbb{E}
 Shortest distance relation SD for \mathbb{E}
 Edge $e=(a, b, w)$ s.t. $e \in (\mathbb{E} \cap SD)$

Output: Shortest distance relation for $\mathbb{E}_n = \mathbb{E} - \{e\}$

- 1: $\Delta P_f = \pi_{a, \$2, \$3+w}(\sigma_{\$1=b}SD)$;
- 2: $\Delta SD_f = (SD \cap \Delta P_f) - (\mathbb{E}_n \cup (\pi_{\$1, \$5, \$3+\$6}(\mathbb{E}_n \bowtie_{\$2=\$1} SD)))$;
- 3: $\Delta P_\ell = \pi_{\$1, b, \$3+w}(\sigma_{\$2=a}SD)$;
- 4: $\Delta SD_\ell = (SD \cap \Delta P_\ell) - (\mathbb{E}_n \cup (\pi_{\$1, \$5, \$3+\$6}(SD \bowtie_{\$2=\$1} \mathbb{E}_n)))$;
- 5: $\Delta P_i = \pi_{\$1, \$5, \$3+\$6-w}(\Delta SD_\ell \times \Delta SD_f)$;
- 6: $\Delta SD_i = SD \cap \Delta P_i$;
- 7: $SD^- = \Delta SD_\ell \cup \Delta SD_f \cup \Delta SD_i \cup \{e\}$;
- 8: $SD = SD - SD^-$;
- 9: $SD^+ = \text{Recalculate}(\mathbb{E}_n, SD^-, SD)$;
- 10: $SD^* = SD \cup SD^+$;
- 11: **return** SD^*

Algorithm 2 works as follows. First, it computes all the distance tuples obtained by concatenating e with shortest paths starting from vertex b (step 1), that is, tuples of the form (a, y, d) s.t. there is a path from a to y with distance d , the first edge along such a path is e , and the path from b to y is a shortest one (w.r.t. the original edge relation). Among these tuples, the algorithm selects those that are shortest distances and for which there does not exist an alternative path having the same (minimum) cost (step 2).

Next, two analogous steps are performed: the algorithm computes all the distance tuples obtained by concatenating shortest paths ending in vertex a with e (step 3) and selects those that are shortest distances and for which there does not exist an alternative path having the same (minimum) cost (step 4).

Then, the tuples computed at step 4 (corresponding to paths where the last edge is e) are combined with the tuples computed at step 2 (corresponding to paths where the first edge is e) via a Cartesian product (step 5). Analogous to step 5 of Algorithm 1, since edge e is taken into account twice, the distance of the tuples obtained from the Cartesian product is diminished of w .

Among the tuples computed at step 5, only those that are shortest distances are kept (step 6). The shortest distances computed in steps 2, 4 and 6, together with edge e , are stored in relation SD^- (step 7) and deleted from SD (step 8).

Finally, Algorithm 3 is called (step 9); this algorithm computes the new shortest distances for paths in SD^- (provided that they still exist) and the new shortest distances are added to SD (step 10).

Algorithm 3 Recalculate

Input: Edge relation \mathbb{E}_n

 Deleted distances SD^-

 Shortest distance relation SD
Output: Relation SD^+ ;

 1: $\Delta P = (\mathbb{E}_n \cup (\pi_{\$1, \$5, \$3+\$6}(\mathbb{E}_n \bowtie_{\$2=\$1} SD))) \times_{\$1=\$1 \wedge \$2=\$2} SD^-$;

 2: $\Delta SD = \min(\Delta P)$

 3: $SD^+ = \Delta SD$;

 4: **while** $\Delta SD \neq \emptyset$ **do**

 5: $\Delta P = \pi_{\$1, \$5, \$3+\$6}(\mathbb{E}_n \bowtie_{\$2=\$1} \Delta SD) \times_{\$1=\$1 \wedge \$2=\$2} SD^-$;

 6: $\Delta SD = \min(\Delta P, SD^+)$;

 7: $SD^+ = \Delta SD \oplus SD^+$;

 8: **end while**

 9: **return** SD^+

Algorithm 3 computes the new shortest distances (when they exist) for the endpoints of the distance tuples in SD^- (recall that SD^- are the shortest distances that have been deleted by Algorithm 2). First, it adds to \mathbb{E}_n all the distance tuples obtained by concatenating edges in \mathbb{E}_n with shortest paths in the updated shortest distance relation SD (step 1). Indeed, only those distance tuples whose endpoints are in a tuple of SD^- are kept and stored in ΔP . Among these distance tuples, the minimum ones are selected (step 2) and added to SD^+ (step 3)— SD^+ is the set of distance tuples that will be eventually returned by the algorithm (and added to the shortest distance relation by Algorithm 2).

Then, SD^+ is iteratively updated as follows (steps 4–8). The algorithm computes all the distance tuples obtained by concatenating edges in \mathbb{E}_n with tuples in ΔSD , and keeps only those whose endpoints are in a tuple of SD^- (step 5). Among these, only distance tuples that improve on shortest distances in SD^+ are kept (step 6) and incorporated into ΔSD^+ (step 7). An example is provided below.

Example 3.4. Consider the graph and the corresponding shortest distances of Figure 3.5. Suppose we delete the edge $(c, d, 1)$ from the graph.

The shortest distances affected by this deletion are marked with dashed links in Figure 3.6 (left) and are obtained at steps 1–7 of Algorithm 2 similar to the intermediate steps of Algorithm 1 depicted in Figure 3.4. Such shortest distances are deleted from the current shortest distance relation. After that, Algorithm 3 tries to recompute such shortest distances by iteratively updating ΔSD . Figure 3.6 (right) shows the new shortest distances computed by Algo-

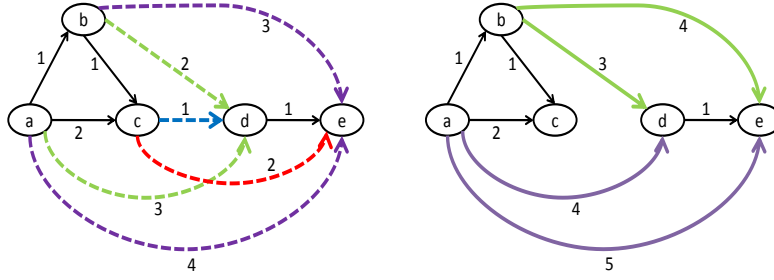


Fig. 3.6. Affected SDs (left) and new SDs (right).

Algorithm 3 highlighting those obtained at the initialization of ΔSD (green edges) and those computed at successive iterations (purple edges). As expected, we obtain the original shortest distances reported in Figure 3.3 (right).

Theorem 3.5. Given an edge relation \mathbb{E} , the shortest distance relation SD for \mathbb{E} , and an edge e , Algorithm 2 computes the shortest distance relation for $\mathbb{E} - \{e\}$.

Proof. Let $\mathbb{E}_n = \mathbb{E} - \{e\}$ and SD_n be the shortest distance relation for \mathbb{E}_n . Notice that each iteration of the loop in Algorithm 3 computes shortest distances that strictly improve on the currently computed ones. As we consider non-negative edge weights, the number of iterations is finite. Thus, Algorithm 3 and consequently Algorithm 2 terminate. Notice that $(x, y, d) \in SD^-$ iff (x, y, d) is in the input relation SD and one of the following conditions holds: (i) there is a shortest path from x to y in \mathbb{E} whose first edge is e and there is no shortest path from x to y in \mathbb{E}_n with distance d (see steps 1–2); (ii) there is a shortest path from x to y in \mathbb{E} whose last edge is e and there is no shortest path from x to y in \mathbb{E}_n with distance d (steps 3–4); (iii) there is a shortest path from x to y in \mathbb{E} where e is an intermediate edge and for the two subpaths starting from and ending in e conditions (i) and (ii) apply, respectively (steps 5–6); (iv) $e = (x, y, d)$ (step 7).

Soundness ($SD^* \subseteq SD_n$). Let $(x, y, d) \in SD^*$. This means that there exists (x, y, d') in the original SD . (1) If there is a shortest path from x to y in \mathbb{E} that does not go through e , then $(x, y, d') \in SD_n$. Conditions (i)–(ii) above do not apply. If none of conditions (iii)–(vi) applies, then (x, y, d') is not in SD^- , remains in SD , and is in SD^* . If one of conditions (iii)–(vi) holds, then $(x, y, d') \in SD^-$ and it is easy to see that (x, y, d') is computed by Algorithm 3 as it tries to compute deleted shortest distances using only edges in \mathbb{E}_n . As SD^* does not contain two distinct distance tuples (x, y, d_1) and (x, y, d_2) , then $d = d'$. (2) If every shortest path from x to y in \mathbb{E} goes through e , then one of conditions (i)–(iv) applies and thus the shortest distance from x to y in \mathbb{E}_n is correctly computed by Algorithm 3 as it computes deleted shortest distances using only edges in \mathbb{E}_n .

Completeness ($SD^* \supseteq SD_n$). Let $(x, y, d) \in SD_n$. This means that there exists (x, y, d') in the original SD . (1) If $d' = d$, then there is a shortest path from x to y in \mathbb{E} that does not go through e . Thus, the same reasoning used in part (1) above can be applied to show that (x, y, d') is not in SD^- , remains in SD , and therefore is in SD^* . (2) If $d' < d$, then every shortest path from x to y in \mathbb{E} goes through e . Thus, as one of conditions (i)–(iv) applies, then $(x, y, d') \in SD^-$. Finally, (x, y, d') is computed by Algorithm 3 and thus is in SD^* . \square

Pang et al.’s algorithms

We briefly discuss the incremental maintenance algorithms proposed in [52] to find the shortest distance between every pair of vertices after single-edge insertion and deletion. The Algorithm 4 for edge insertion is fairly simple and works as follows. When inserting edge (a, b) with distance $w > 0$, the shortest distances formed by any path from node x through (a, b) to y could be affected; such paths are computed via a Cartesian product and stored in ΔP (step 1). Then, among these distance tuples, the algorithm selects only those that improve on current shortest distances, that is, those tuples (x, y, d) in ΔP s.t. either there is no shortest path from x to y in SD or there is one with distance greater than d (step 2). Finally, tuples of SD that no longer express shortest distance are updated with the new value (step 3).

Algorithm 4 Insertion - Pang et al. [52]

Input: Edge relation \mathbb{E} ; for each vertex v , the tuple $(v, v, 0)$ is in \mathbb{E}
Shortest distance relation SD for \mathbb{E}
Edge (a, b, w)

Output: Shortest distance relation for $\mathbb{E} \cup \{(a, b, w)\}$

- 1: $\Delta P = \pi_{\$1, \$5, \$3+\$6+w}(\sigma_{\$2=a}(SD) \times \sigma_{\$1=b}(SD))$
 - 2: $\Delta SD = \min(\Delta P, SD)$;
 - 3: $SD = \Delta SD \oplus SD$;
 - 4: **return** SD ;
-

The Algorithm 5 for edge deletion is clearly more involved. First, it computes all the distance tuples obtained by concatenating (a, b) with shortest paths ending in a and those starting from node b (step 1), that is, tuples of the form (x, y, d) s.t. there is a path from x through (a, b) to y with distance d , and the paths from x to a and from b to y are the shortest ones (w.r.t. the original edge relation). Among these tuples, those that are shortest distances are stored in SD^- (step 2), while the shortest distance tuples that do not use the deleted edge (a, b) are stored in the relation $Trust$ (step 3). Edge (a, b) is then removed from the edge relation (step 4). At this point, Algorithm 5 starts computing the new shortest distances. In ΔP are stored the new tuples (x, y, d) , whose endpoints are in a tuple of SD^- , expressing that there

exists a path between node x and y with distance d , but d may not be the shortest (step 5). Among these distance tuples, the minimum ones are selected (step 6) and added to the tuples in $Trust$, forming eventually the resulting shortest distance relation (step 7).

Algorithm 5 Deletion - Pang et al. [52]

Input: Edge relation \mathbb{E} ; for each vertex v , the tuple $(v, v, 0)$ is in \mathbb{E}
Shortest distance relation SD for \mathbb{E}

Edge (a, b, w)

Output: Shortest distance relation for $\mathbb{E} - \{(a, b, w)\}$

- 1: $\Delta P = \pi_{\$1, \$5, \$3+\$6+w}(\sigma_{\$2=a}(SD) \times \sigma_{\$1=b}(SD))$;
 - 2: $SD^- = \Delta P \cap SD$;
 - 3: $Trust = SD - SD^-$;
 - 4: $\mathbb{E} = \mathbb{E} - \{(a, b, w)\}$;
 - 5: $\Delta P = (\pi_{\$1, \$8, \$3+\$6+\$9}((Trust \bowtie_{\$2=\$1} \mathbb{E}) \bowtie_{\$5=\$1} Trust)) \times_{\$1=\$1 \wedge \$2=\$2} SD^-$;
 - 6: $SD^+ = \min(\Delta P)$
 - 7: $SD = Trust \cup SD^+$;
 - 8: **return** SD
-

3.2.2 Multiple Edges Management

In this section, we provide algorithms to update the shortest distance relation after the insertion or deletion of *multiple* edges. Obviously, the insertion of k edges might be handled by calling k times Algorithm 1, and the deletion of multiple edges might be analogously handled. However, the algorithms we propose in this section leverage the information about the entire set of edges to be inserted or deleted to maintain the shortest distance relation in a more efficient way.

The core idea is to identify paths affected by the insertions/deletions starting from the vertices that are one hop far from the inserted/deleted edges and propagating changes farther only if necessary. If the insertion/deletion of an edge $e = (a, b, w)$ does not influence the shortest distance from a to b , then no updates need to be propagated. On the other hand, if the shortest distance from a to b changes, this may in turn affect the shortest distances of a 's and b 's neighbors (because their shortest paths may use edge e). In this case, our algorithms update the shortest distance from a to b and propagate the variations farther only if a 's and b 's neighbors are really affected. For instance, if there is no path from a to b after the deletion of e , but their neighbors still have the same shortest distances (because of alternative paths), then the algorithms do not look at farther vertices. On the other hand, if changes are propagated, vertices that are two hops from e are considered by applying the same reasoning.

Moreover, proceeding one hop at a time allows the algorithms to combine the effects of the different edges to be inserted/deleted, reducing the computational effort w.r.t. individually handling them one at a time.

In order to deal with the insertion and deletion of multiple edges, we need to extend the algorithms proposed in Section 3.2.1.

One extension regards the schema of the auxiliary relations used in the algorithms. We will use relations ΔP_f , SD_f , ΔSD_f , ΔP_ℓ , SD_ℓ , and ΔSD_ℓ , whose schema has been enriched for algorithmic purposes. Specifically, ΔP_f , SD_f , and ΔSD_f contain extended distance tuples of the form (x, y, d, a, w) meaning that there is an edge from x to a with weight w , a path r from a to y , and $d = w + \omega(r)$. In other words, (x, y, d, a, w) means that there exists a path from x to y , having distance d , and whose first edge is (x, a, w) .

On the other hand, ΔP_ℓ , SD_ℓ , and ΔSD_ℓ contain extended distance tuples of the form (x, y, d, a, w) meaning that there is a path r from x to a , an edge from a to y with weight w , and $d = \omega(r) + w$. In other words, (x, y, d, a, w) means that there is a path from x to y , having distance d , and whose last edge is (a, y, w) .

Multiple edges insertion

Algorithm 6 is an extension of Algorithm 1 that is able to handle the insertion of multiple edges at once. It takes as input an edge relation \mathbb{E} , the shortest distance relation SD for \mathbb{E} , and a set of edges $\Delta\mathbb{E}$ to be added to \mathbb{E} . The output is the shortest distance relation for $\mathbb{E} \cup \Delta\mathbb{E}$.

The algorithm first computes a subset $\Delta\mathbb{E}_s$ of $\Delta\mathbb{E}$ containing those edges (a, b, w) s.t. w is less than the shortest distance from a to b in the input relation SD (step 1); $\Delta\mathbb{E}_s$ is then incorporated into SD (step 2). After this pre-processing phase, the updating phase starts.

More in detail, the algorithm initially considers paths composed by two edges, where the first one belongs to $\Delta\mathbb{E}_s$ (step 3). For each of these paths having a distance lower than the current shortest distance (steps 6–8), the algorithm considers longer paths (in terms of number of edges) obtained by adding one more edge to the right (step 9). The process is repeated until no further lower distances are obtained (steps 5–10).

Then, the algorithm proceeds in a similar way but adding edges to the left. More specifically, it first considers paths with two edges where the last one belongs to $\Delta\mathbb{E}_s$ (step 11) and then iteratively adds further edges to the left as long as paths improving on current shortest distances can be derived (steps 13–18).

After the iterative steps described above, SD_f (resp. SD_ℓ) contains extended distance tuples referring to paths whose first (resp. last) edge belongs to $\Delta\mathbb{E}_s$. In addition, each extended distance tuple belonging to these sets maintains information about the first or last edge being used and its weight. This information is used in the next step (step 19), where an extended distance tuple of SD_ℓ is combined with an extended distance tuple of SD_f iff the

Algorithm 6 Multiple-Edges-Insertion-Maintenance

Input: Edge relation \mathbb{E}
Shortest distance relation SD for \mathbb{E}
Set of edges $\Delta\mathbb{E}$ s.t. for each edge $(a, b, w) \in \Delta\mathbb{E}$,
 $\nexists (a, b, w') \in \mathbb{E}$ with $w' \neq w$

Output: Shortest distance relation for $\mathbb{E}_n = \mathbb{E} \cup \Delta\mathbb{E}$

- 1: $\Delta\mathbb{E}_s = \min(\Delta\mathbb{E}, SD)$;
- 2: $SD = \Delta\mathbb{E}_s \oplus SD$;
- 3: $\Delta P_f = \pi_{\$1, \$5, \$3+\$6, \$2, \$3}(\Delta\mathbb{E}_s \bowtie_{\$2=\$1} \mathbb{E}_n)$;
- 4: $SD_f = \emptyset$;
- 5: **repeat**
- 6: $\Delta SD_f = \min(\Delta P_f, SD)$;
- 7: $SD_f = \Delta SD_f \oplus SD_f$;
- 8: $SD = \pi_{\$1, \$2, \$3}(\Delta SD_f) \oplus SD$;
- 9: $\Delta P_f = \pi_{\$1, \$7, \$3+\$8, \$4, \$5}(\Delta SD_f \bowtie_{\$2=\$1} \mathbb{E}_n)$;
- 10: **until** $\Delta SD_f = \emptyset$
- 11: $\Delta P_\ell = \pi_{\$1, \$5, \$3+\$6, \$4, \$6}(\mathbb{E}_n \bowtie_{\$2=\$1} \Delta\mathbb{E}_s)$;
- 12: $SD_\ell = \emptyset$;
- 13: **repeat**
- 14: $\Delta SD_\ell = \min(\Delta P_\ell, SD)$;
- 15: $SD_\ell = \Delta SD_\ell \oplus SD_\ell$;
- 16: $SD = \pi_{\$1, \$2, \$3}(\Delta SD_\ell) \oplus SD$;
- 17: $\Delta P_\ell = \pi_{\$1, \$5, \$3+\$6, \$7, \$8}(\mathbb{E}_n \bowtie_{\$2=\$1} \Delta SD_\ell)$;
- 18: **until** $\Delta SD_\ell = \emptyset$
- 19: $\Delta P_i = \pi_{\$1, \$7, \$3+\$8-\$5} (SD_\ell \bowtie_{\$4=\$1 \wedge \$2=\$4} SD_f)$;
- 20: $\Delta SD_i = \min(\Delta P_i, SD)$;
- 21: $SD^* = \Delta SD_i \oplus SD$;
- 22: **return** SD^* ;

last edge of the path of the former is equal to the first edge of the path of the latter. Among the obtained extended distance tuples, only those improving on the current shortest distance are kept (step 20) and incorporated into SD (step 21).

Theorem 3.6. *Given an edge relation \mathbb{E} , the shortest distance relation SD for \mathbb{E} , and a set of edges $\Delta\mathbb{E}$, Algorithm 6 computes the shortest distance relation for $\mathbb{E} \cup \Delta\mathbb{E}$.*

Proof.

Termination. Each iteration of the two repeat-until loops computes longer (in terms of number of edges) paths that strictly improve on current shortest distances. As we consider non-negative weights, the number of iterations is bounded and thus the algorithm terminates.

Let SD_n be the shortest distance relation for \mathbb{E}_n . We start with some observations used in the following. First, if a distance tuple (x, y, d) is in SD^*

(resp. $\Delta\mathbb{E}_s$, SD , ΔP_i , ΔSD_i) or an extended distance tuple (x, y, d, a, w) is in ΔP_f (resp. ΔSD_f , SD_f , ΔP_ℓ , ΔSD_ℓ , SD_ℓ), then there is a path from x to y in \mathbb{E}_n whose distance is d . Second, SD^* does not contain two distance tuples (x, y, d_1) and (x, y, d_2) s.t. $d_1 \neq d_2$ (to see why, it suffices to look at steps 20–21 and the definitions of min and \oplus).

Soundness ($SD^* \subseteq SD_n$). Consider a distance tuple (x, y, d) in SD^* . As noticed above, this means that there is a path from x to y in \mathbb{E}_n ; thus, there must be a shortest one too, which implies that a distance tuple (x, y, d') is in SD_n . We show that $d = d'$. One of the following two cases must occur. (1) There is a shortest path r' from x to y in \mathbb{E}_n (whose distance is d') which does not go through any of the edges in $\Delta\mathbb{E}$. Since r' goes only through edges in \mathbb{E} , then (x, y, d') belongs to the input relation SD . Since all the auxiliary relations used by the algorithm contain distance tuples or extended distance tuples corresponding to paths in \mathbb{E}_n and none of them strictly improve on (x, y, d') because the latter correspond to a shortest distance in \mathbb{E}_n , then it is easy to check that (x, y, d') remains in SD at all steps and eventually $(x, y, d') \in SD^*$. Hence, $d = d'$. (2) Every shortest path from x to y in \mathbb{E}_n (whose distance is d') goes through at least one edge in $\Delta\mathbb{E}$. This actually implies that every shortest path from x to y in \mathbb{E}_n goes through at least one edge in $\Delta\mathbb{E}_s$. Let r' be one of such shortest paths. There must be an edge $e \in \Delta\mathbb{E}_s$ s.t. one of the following conditions holds: (i) e is the first edge of r' and every subpath of r' starting from x (and ending in any of the vertices in r' following x) improves on the original shortest distances; (ii) e is the last edge of r' and every subpath of r' ending in y (and starting from any of the vertices in r' preceding y) improves on the original shortest distances; (iii) e is an intermediate edge (a, b, w) of r' and every subpath of r' starting from a as well as every subpath of r' ending in b improves on the original shortest distances. In case (i) (resp. (ii)) a distance tuple for r' is computed in the first (resp. second) repeat-until loop starting from e and adding edges from \mathbb{E}_n to the right (resp. left) as long as they improve on shortest distances. In case (iii) a distance tuple for r' is computed at step 19. Notice that since every shortest path from x to y in \mathbb{E}_n goes through at least one edge in $\Delta\mathbb{E}_s$, this is the case where the addition of $\Delta\mathbb{E}$ strictly improves the shortest distance from x to y . Thus, (x, y, d') belongs to SD^* and $d = d'$.

Completeness ($SD^* \supseteq SD_n$). Consider a distance tuple (x, y, d) in SD_n . If there is a shortest path r from x to y in \mathbb{E}_n (its distance is d) that does not go through any of the edges in $\Delta\mathbb{E}$, then $(x, y, d) \in SD$. Notice that distance tuples that are added to SD at steps 2, 8, 16 and to SD^* at step 21 correspond to paths in \mathbb{E}_n that strictly improve on SD (see steps 1, 6, 14, 20). As there is no path from x to y in \mathbb{E}_n with distance strictly lower than d , then (x, y, d) remains in SD at every step and eventually is in SD^* . If every shortest path from x to y in \mathbb{E}_n goes through at least one edge in $\Delta\mathbb{E}$, then it can be verified that $(x, y, d) \in SD^*$ by applying the same reasoning used in part (2) above. \square

Multiple edges deletion

Algorithm 7 is an extension of Algorithm 2 which is able to handle the deletion of multiple edges at once. It updates the shortest distance relation SD for an edge relation \mathbb{E} after the deletion of a set of edges $\Delta\mathbb{E}$ from \mathbb{E} . The precondition $\Delta\mathbb{E} \subseteq (\mathbb{E} \cap SD)$ is imposed for the same reason discussed for the deletion of a single edge (see Section 3.2.1).

Similar to the algorithm for the deletion of a single edge, Algorithm 7 first deletes shortest distances that might have been obtained using edges in $\Delta\mathbb{E}$ and then try to recompute the new shortest distances using Algorithm 3. However, Algorithm 7 performs a more accurate analysis of which shortest distances should be deleted than the algorithm for the deletion of a single edge.

Specifically, the algorithm starts by computing a set $\Delta\mathbb{E}_s$ of edges in $\Delta\mathbb{E}$ for which there does not exist an alternative path having the same cost (step 2) and deletes these edges from SD (step 3).

Analogous to Algorithm 6, the algorithm incrementally computes a set SD_f (resp. SD_ℓ) of extended distance tuples corresponding to paths where a deleted edge is the first (resp. last) one along the path—see steps 4–11 (resp. steps 12–19). However, only extended distance tuples corresponding to shortest distances and for which there does not exist an alternative path having the same cost are considered.

Then, SD_ℓ and SD_f are combined as follows: an extended distance tuple of SD_ℓ is combined with an extended distance tuple of SD_f iff the last edge of the path of the former is equal to the first edge of the path of the latter (step 20). The resulting extended distance tuples belonging to SD are deleted from SD (step 21–22).

Then, the set of shortest distances deleted till that point, defined as SD^- at step 23, is used as input for Algorithm 3 to recompute deleted shortest distances (step 24).

Finally, the shortest distances returned by Algorithm 3 are added to SD (step 25).

Theorem 3.7. *Given an edge relation \mathbb{E} , the shortest distance relation SD for \mathbb{E} , and a set of edges $\Delta\mathbb{E}$, Algorithm 7 computes the shortest distance relation for $\mathbb{E} - \Delta\mathbb{E}$.*

Proof.

Termination. Each iteration of the two repeat-until loops computes strictly longer (in terms of number of edges) paths that are shortest ones. As we consider non-negative weights, the number of iterations is bounded and thus the loops terminate. As already discussed in the proof of Theorem 3.5, Algorithm 3 always terminates too.

Let SD_n be the shortest distance relation for \mathbb{E}_n . Notice that $(x, y, d) \in SD^-$ iff (x, y, d) is in the input relation SD and one of the following conditions holds: (i) there is a shortest path from x to y in \mathbb{E} whose first edge is in $\Delta\mathbb{E}_s$

Algorithm 7 Multiple-Edges-Deletion-Maintenance**Input:** Edge relation \mathbb{E} Shortest distance relation SD for \mathbb{E} Set of edges $\Delta\mathbb{E}$ s.t. $\Delta\mathbb{E} \subseteq (\mathbb{E} \cap SD)$ **Output:** Shortest distance relation for $\mathbb{E}_n = \mathbb{E} - \Delta\mathbb{E}$

- 1: Let θ be the condition $\$1 = \$1 \wedge \$2 = \$2 \wedge \$3 = \3
- 2: $\Delta\mathbb{E}_s = \Delta\mathbb{E} - (\pi_{\$1, \$5, \$3+\$6}(\mathbb{E}_n \bowtie_{\$2=\$1} SD))$;
- 3: $SD = SD - \Delta\mathbb{E}_s$;
- 4: $\Delta P_f = (\pi_{\$1, \$5, \$3+\$6, \$2, \$3}(\Delta\mathbb{E}_s \bowtie_{\$2=\$1} \mathbb{E})) \bowtie_{\theta} SD$;
- 5: $SD_f = \emptyset$;
- 6: **repeat**
- 7: $\Delta SD_f = \Delta P_f \ominus (\mathbb{E}_n \cup (\pi_{\$1, \$5, \$3+\$6}(\mathbb{E}_n \bowtie_{\$2=\$1} SD)))$;
- 8: $SD_f = SD_f \cup \Delta SD_f$;
- 9: $SD = SD - \pi_{\$1, \$2, \$3}(\Delta SD_f)$;
- 10: $\Delta P_f = (\pi_{\$1, \$7, \$3+\$8, \$4, \$5}(\Delta SD_f \bowtie_{\$2=\$1} \mathbb{E})) \bowtie_{\theta} SD$;
- 11: **until** $\Delta SD_f = \emptyset$
- 12: $\Delta P_\ell = (\pi_{\$1, \$5, \$3+\$6, \$4, \$6}(\mathbb{E} \bowtie_{\$2=\$1} \Delta\mathbb{E}_s)) \bowtie_{\theta} SD$;
- 13: $SD_\ell = \emptyset$;
- 14: **repeat**
- 15: $\Delta SD_\ell = \Delta P_\ell \ominus (\mathbb{E}_n \cup (\pi_{\$1, \$5, \$3+\$6}(SD \bowtie_{\$2=\$1} \mathbb{E}_n)))$;
- 16: $SD_\ell = SD_\ell \cup \Delta SD_\ell$;
- 17: $SD = SD - \pi_{\$1, \$2, \$3}(\Delta SD_\ell)$;
- 18: $\Delta P_\ell = (\pi_{\$1, \$5, \$3+\$6, \$7, \$8}(\mathbb{E} \bowtie_{\$2=\$1} \Delta SD_\ell)) \bowtie_{\theta} SD$;
- 19: **until** $\Delta SD_\ell = \emptyset$
- 20: $\Delta P_i = (\pi_{\$1, \$7, \$3+\$8-\$5}(SD_\ell \bowtie_{\$4=\$1 \wedge \$2=\$4} SD_f))$;
- 21: $SD_i = SD \cap \Delta P_i$;
- 22: $SD = SD - SD_i$;
- 23: $SD^- = \Delta\mathbb{E}_s \cup \pi_{\$1, \$2, \$3}(SD_f \cup SD_\ell) \cup SD_i$;
- 24: $SD^+ = \text{Recalculate}(\mathbb{E}_n, SD^-, SD)$;
- 25: $SD^* = SD \cup SD^+$;
- 26: **return** SD^*

and there is no shortest path from x to y in \mathbb{E}_n with distance d (see steps 1–11); (ii) there is a shortest path from x to y in \mathbb{E} whose last edge is in $\Delta\mathbb{E}_s$ and there is no shortest path from x to y in \mathbb{E}_n with distance d (steps 12–19); (iii) there is a shortest path from x to y in \mathbb{E} with an intermediate edge e in $\Delta\mathbb{E}_s$ and for the two subpaths starting from and ending in e conditions (i) and (ii) apply, respectively (steps 20–21); (iv) $(x, y, d) \in \Delta\mathbb{E}_s$ (step 23).

Soundness ($SD^* \subseteq SD_n$). Let $(x, y, d) \in SD^*$. Obviously, this means that there exists a distance tuple (x, y, d') in the original SD . If there is a shortest path from x to y in \mathbb{E} that does not go through any of the edges in $\Delta\mathbb{E}$, then $(x, y, d') \in SD_n$. If none of conditions (i)–(iv) above applies, then (x, y, d') is not in SD^- , remains in SD , and therefore is in SD^* . Otherwise, $(x, y, d') \in SD^-$ but it is then computed by Algorithm 3 as it computes

deleted shortest distances using only edges in \mathbb{E}_n . As SD^* does not contain two distinct distance tuples (x, y, d_1) and (x, y, d_2) , then $d = d'$. If every shortest path from x to y in \mathbb{E} goes through an edge in $\Delta\mathbb{E}$, then the shortest distance from x to y in \mathbb{E}_n is correctly computed by Algorithm 3. *Completeness* ($SD^* \supseteq SD_n$). Let $(x, y, d) \in SD_n$. Obviously, this means that there exists a distance tuple (x, y, d') in the original SD . If $d' = d$, then there is a shortest path from x to y in \mathbb{E} that does not go through any of the edges in $\Delta\mathbb{E}$. Thus, the same reasoning used above for the soundness can be applied to show that $(x, y, d)'$ is not in SD^- , remains in SD , and is in SD^* . If $d' < d$, then every shortest path from x to y in \mathbb{E} goes through an edge in $\Delta\mathbb{E}$. Thus, $(x, y, d') \in SD^-$, (x, y, d) is correctly computed by Algorithm 3, and thus $(x, y, d) \in SD^*$. \square

3.3 Experimental Evaluation

We conducted an extensive experimental evaluation of our algorithms comparing them against the algorithms proposed by Pang et al. [52], which is, to the best of our knowledge, the only disk-based approach for the incremental maintenance of all-pairs shortest distances—in particular, it relies on relational DBMSs too.

Once again, we point out that disk-based incremental algorithms are needed in many current graph applications where graphs are large and frequently updated.

Recall that the algorithms proposed in [52] are able to handle the insertion and deletion of only one edge at a time. In this section, we will consider the following algorithms:

- Single-Edge-Insertion-Maintenance (*SI* for short),
- Single-Edge-Deletion-Maintenance (*SD* for short),
- Multiple-Edges-Insertion-Maintenance (*MI* for short),
- Multiple-Edges-Deletion-Maintenance (*MD* for short),
- the (single) insertion algorithm of [52] (denoted *PI*),
- the (single) deletion algorithm of [52] (denoted *PD*).

The algorithms proposed in [52] are reported in Section 3.2.1.

3.3.1 Datasets

Experiments were carried out on three datasets. Specifically, we used two real-world networks and a synthetic graph with a “regular” structure as described below.

- **9th DIMACS Implementation Challenge – Shortest Paths** (<http://www.dis.uniroma1.it/challenge9/>). This dataset provides a collection of road networks available in two versions depending on the weights assigned

to links. In one version edge weights encode the physical distance between vertices, while in the other version edge weights stand for the transit time between two vertices. We used the network for the city of New York where edge weights stand for distances.

- **DIMES Public data repository** (<http://www.netdimes.org/new/>). This dataset provides monthly snapshots of Autonomous Systems on the Internet. Vertices represent Autonomous Systems while edges represent direct links between Autonomous Systems that were found for a given month. Since the original graphs are unweighted, we assigned unitary weight to every edge. We used the snapshot from October 2011.
- **Regular graph**. This dataset was generated by replicating several times the pattern depicted in Figure 3.7.

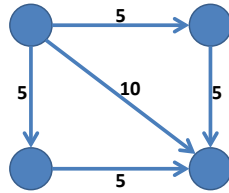


Fig. 3.7. Pattern of the regular graph.

The graphs features are reported in Table 3.1.

Datasets	# Vertices	# Edges	# Shortest distances
DIMACS	264,346	730,100	$\approx 6.98 \times 10^{10}$
DIMES	25,683	81,294	$\approx 1.02 \times 10^8$
Regular	386,100	697,311	$\approx 9.46 \times 10^9$

Table 3.1. Properties of the considered datasets.

While the DIMACS dataset is a strongly connected graph, the DIMES dataset is a much less connected graph: the ratio of the number of shortest distances to the number of all possible pairs of vertices is 0.15 (thus, much smaller than the ratio for the DIMACS dataset, which is 1). The regular graph is even more loosely connected: the ratio of the number of shortest distances to the number of all possible pairs of vertices is 0.06. Using these datasets we were able to run experiments on graphs having different structures and connectivity properties.

We point out that, as we deal with the problem of incrementally maintaining all-pairs shortest distances, the input of all the considered algorithms consists of both a graph and the corresponding shortest distance relation, with the size of the latter being much bigger than the size of the former.

3.3.2 Experimental setup

We considered a strongly connected *induced* subgraph D of the original DIMACS graph. An induced subgraph of a graph $G = (V, E)$ is a graph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$ contains all edges of E connecting two vertices in V' . To assess how the algorithms behave as the dataset size varies, we considered four induced subgraphs of D of increasing size, denoted by $Dimacs_i$ with $i \in [1..4]$, where $Dimacs_4 = D$ and $Dimacs_i \subset Dimacs_{i+1}$, for $i \in [1..3]$. To generate these graphs we started from a vertex with maximum degree in the DIMACS dataset and visited connected vertices in a breadth-first fashion adding to the graph in construction all edges of the original graph involving vertices so far encountered until the desired size was reached. Features of the graphs used in the experiments are reported in Table 3.2—all of them are strongly connected.

Subgraphs	# Vertices	# Edges	Shortest distances
$Dimacs_1$	10,193	24,454	103,887,056
$Dimacs_2$	14,275	34,580	203,761,350
$Dimacs_3$	17,378	42,592	301,977,506
$Dimacs_4$	20,012	49,096	400,460,132

Table 3.2. Properties of the DIMACS datasets.

On the other hand, we considered the entire DIMES dataset. Regarding the regular dataset, as already mentioned before, it was synthetically generated by replicating several times the pattern of Figure 3.7 until the number of nodes and edges in Table 3.1 was reached.

Since one of the aims of the experimental evaluation was to study the algorithms' running times for different kinds of edges, we had to define the "importance" of edges being inserted or deleted. We considered a new measure, called *edge relevance*, which reflects how many shortest distances are affected by the insertion/deletion of a set of edges. Specifically, consider an edge relation \mathbb{E} , the shortest distance relation SD for \mathbb{E} , and a set $\Delta\mathbb{E}$ of edges. The *relevance* of $\Delta\mathbb{E}$ when such edges are to be deleted from or added to \mathbb{E} is defined respectively as follows:

$$\phi_D(\Delta\mathbb{E}) = \frac{U(\Delta\mathbb{E}) + D(\Delta\mathbb{E})}{N}; \quad \phi_I(\Delta\mathbb{E}) = \frac{U(\Delta\mathbb{E}) + I(\Delta\mathbb{E})}{n(n-1)}$$

where $U(\Delta\mathbb{E})$ is the number of shortest distances in SD whose distance changes after deleting or inserting the edges in $\Delta\mathbb{E}$ (i.e., the number of pairs of vertices for which the shortest distance changes), $D(\Delta\mathbb{E})$ (resp. $I(\Delta\mathbb{E})$) is the number of shortest distances removed (resp. created) after the deletion (resp. insertion) of the edges in $\Delta\mathbb{E}$, N is the number of shortest distances in the graph before the deletion or insertion of the edges in $\Delta\mathbb{E}$ (i.e., $N = |SD|$), and n is the number of vertices in the graph.

Notice that when the edges in $\Delta\mathbb{E}$ are deleted, the current shortest distances can possibly be affected in one of two ways: a shortest distance from vertex x to vertex y may not be defined anymore (because there is no path from x to y after the deletions) or its distance increases. Then, $\phi_D(\Delta\mathbb{E})$ is the ratio of the number of shortest distances affected by the deletions (in either of the two ways discussed above), namely $U(\Delta\mathbb{E}) + D(\Delta\mathbb{E})$, to the total number of shortest distances that could possibly be affected, namely N .

When the edges in $\Delta\mathbb{E}$ are inserted, the current shortest distances can possibly be affected by a reduction of their distance; furthermore, new paths between vertices that were not reachable before can be generated. Then, $\phi_I(\Delta\mathbb{E})$ is the ratio of the number of shortest distances affected by the insertion (either shortest distances that have become lower or shortest distances of new paths not existing before), namely $U(\Delta\mathbb{E}) + I(\Delta\mathbb{E})$, to the total number of shortest distances that could possibly be affected, namely $n(n - 1)$.

As a heuristic to identify edges with desired values of relevance, we computed edges' betweenness centrality as this gives good insights on the value of relevance. This is a generalization to edges of the well-known vertex betweenness centrality [26] and it is defined as the number of shortest paths that run along the edge [50]. Thus, the deletion of an edge with high betweenness centrality is likely to have high relevance—even though this might not be the case, and this is why our definition of relevance is more suitable than edge betweenness centrality to measure how many shortest distances are actually affected by the insertion or deletion of edges.

All experiments were run on an Intel i7 3770K 3.5 GHz, 12 GB of memory, running Ubuntu 12.04.

3.3.3 Results on the DIMACS dataset

Single edge management

In this section, we discuss the experimental results for the insertion and deletion of a *single* edge.

We start with the results for the insertion of a single edge. In the first kind of experiments, we evaluated how execution times vary as the size of the input increases. For each dataset $Dimacs_i$ ($i \in [1..4]$), we computed the average execution time after a single edge insertion considering edges with different relevance. Specifically, for each dataset, we considered a set of 20 edges with relevance uniformly distributed in the range $[0.5, 5]\%$, measured the time to handle the insertion of each of them individually, and finally took the average time across all 20 edges. We considered relevance up to 5% because the highest edge relevance we found was approximately 5%. The experimental results are reported in Figure 3.8 and show that the *SI* and *MI* algorithms have similar performances and both outperform the *PI* algorithm. It is worth noticing that the performance gap between our algorithms and *PI* gets bigger as the dataset size increases.

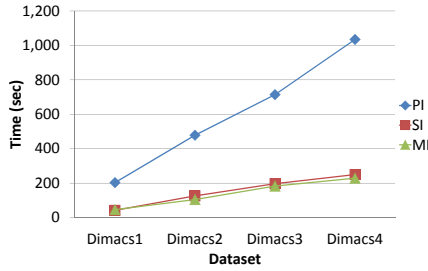


Fig. 3.8. Runtime vs. dataset size (single insertion).

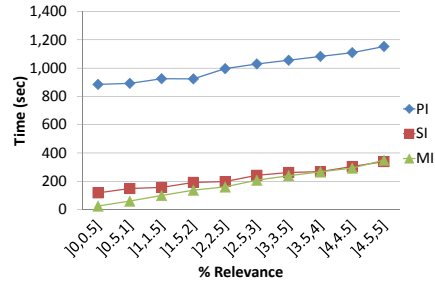


Fig. 3.9. Runtime vs. relevance on *Dimacs*₄ (single insertion).

We also evaluated running times w.r.t. the edge relevance. For each of the four DIMACS datasets, we considered a set of 40 edges with relevance in the range $(0, 5]\%$. We report in Figure 3.9 the results obtained for the largest graph *Dimacs*₄; the other three datasets showed similar trends. Not surprisingly, runtime increases as the edge relevance increases (because more modifications need to be made to the shortest distance relation). However, we found out that most of the edges have a low relevance and only few edges have high relevance. Once again, *SI* and *MI* have similar runtimes and both outperform *PI*.

Regarding the deletion of a single edge, we were not able to compare our algorithms against *PD* on the four DIMACS datasets because the execution of *PD* did not terminate within 5 days. Thus, in order to compare our algorithms against *PD*, we considered much smaller induced graphs with up to 2,000 vertices. The results are reported in Figure 3.10 and show that both *SD* and *MD* have significantly lower running times than *PD*—as an example, for the graph with 2,000 vertices, our *MD* algorithm updates the shortest distances in 5 seconds while *PD* takes 7 hours. It is also quite evident that the execution time of *PD* rapidly grows as the number of vertices increases.

The same running times are reported in Figure 3.11 with the y -axis on a logarithmic scale and the x -axis reporting the number of shortest distances. Our algorithms are at least three orders of magnitude faster than *PD*.

For the sake of completeness, we report in Figure 3.12 the performances of our algorithms on all DIMACS subgraphs, thus showing how the execution time varies as the dataset size increases. Figure 3.13 shows the running time for edges with different relevance—here we used the largest DIMACS subgraph, namely *Dimacs*₄.

Multiple edges management

In all the experiments of this section we used the largest DIMACS subgraph, namely *Dimacs*₄.

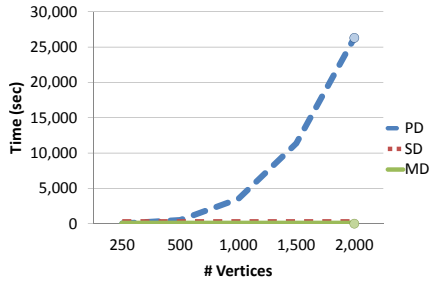


Fig. 3.10. Runtime vs. dataset size on small graphs (single deletion).

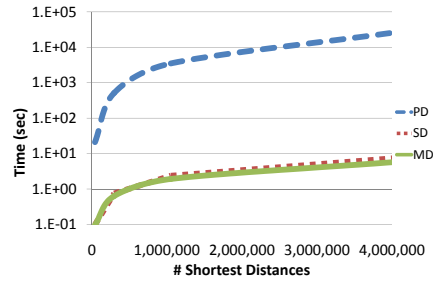


Fig. 3.11. Runtime vs. dataset size on small graphs (single deletion).

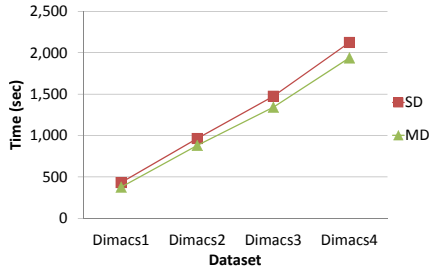


Fig. 3.12. Runtime vs. dataset size (single deletion).

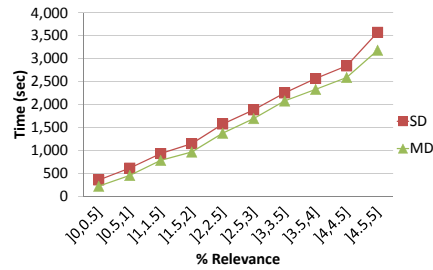


Fig. 3.13. Runtime vs. relevance on *Dimacs4* (single deletion).

We first discuss the results for the insertion of multiple edges. Figure 3.14 shows how running times vary as the number of inserted edges goes from 1 to 8. Each running time is the average over forty sets of edges (of the same cardinality) with overall relevance uniformly distributed in $]1, 5.5]$ %. Recall that both *SI* and *PI* can manage only one edge insertion at a time and thus, in order to handle k edge insertions, we need to run *SI* and *PI* k times. Figure 3.14 shows that the execution time of *PI* linearly grows w.r.t. the number of inserted edges. Both *SI* and *MI* notably outperform *PI* and the gap gets bigger as the number of inserted edges augments. The *MI* algorithm is the fastest one with an execution time that is almost constant. This is because *MI* has been designed to deal with the insertion of multiple edges at once and minimize the computational effort, while *SI* and *PI* must be run once for each inserted edge.

Figure 3.15 reports the execution times for the insertion of 8 edges whose overall relevance varies from 1% to 5.5%. Each time is the average across 4 different sets (of 8 edges). These results confirm that *MI* is the fastest algorithm, *SI* comes second, and *PI* has the worst performance.

Regarding edge deletions, we point out again that we were not able to compare our algorithms against *PD* as its execution did not halt within 5 days even with the deletion of a single edge (recall also that managing the deletion

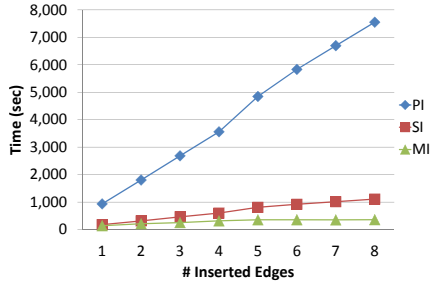


Fig. 3.14. Runtime vs. # of inserted edges.

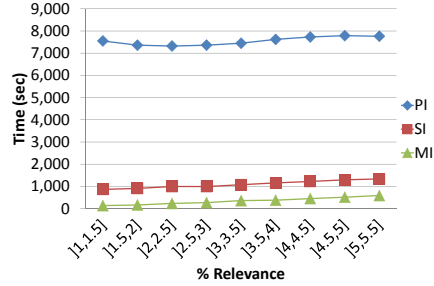


Fig. 3.15. Runtime vs. relevance (multiple insertions).

of multiple edges with *PD* and *SD* requires multiple runs of the algorithms). Figure 3.16 shows the execution times of *SD* and *MD* for different numbers of deleted edges. Once again, each time reported in the figure is the average across 40 sets of edges with overall relevance uniformly distributed in the range]1, 5.5]%. Similarly to the case of multiple insertions, our algorithm for handling multiple deletions at once (i.e., *MD*) outperforms the counterpart able to deal with one deletion at a time (i.e., *SD*).

Figure 3.17 shows the results for the deletion of several sets of edges having different overall relevance. Each time is the average across 4 sets, each consisting of 8 edges. Also in this case, the *MD* algorithm is the fastest one.

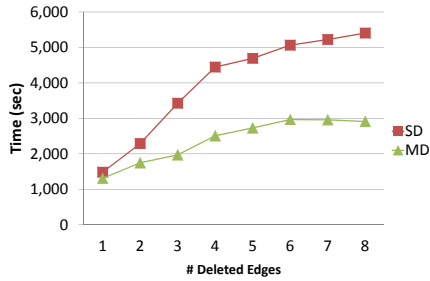


Fig. 3.16. Runtime vs. # of deleted edges.

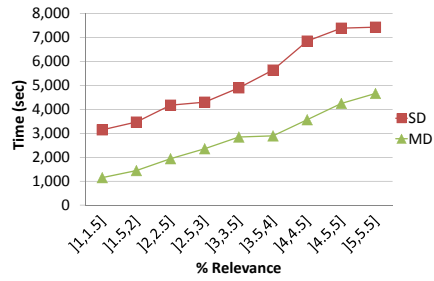


Fig. 3.17. Runtime vs. relevance (multiple deletions).

3.3.4 Results on the DIMES dataset

Single edge management

For the DIMES dataset it is quite hard to select edges with high relevance because of the graph structure (recall that this dataset is much more less connected than the DIMACS dataset, which is strongly connected). For the

insertion of a single edge, the highest edge relevance was approximately 0.15% while for the deletion of a single edge the highest relevance was approximately 1.00%.

Figure 3.18 reports the execution time for the insertion of a single edge. We considered 100 edges with different relevance ranging in the interval $[0, 0.15]\%$. The figure shows that most of the edges have a relevance below 0.05%. Also for this dataset, *MI* outperforms both *SI* and *PI*. We point out that *PI* exhibits poor performances even if the considered edges affect a negligible number of shortest distances.

Regarding the deletion of a single edge, we compared only *SD* and *MD* because also with this dataset *PD* did not terminate within 5 days for different edges (for which the *MD* algorithm took less than one second). In this case, we considered 100 edges and were able to identify edges with relevance up to 1%, even though most of them have a relevance below 0.2%. Figure 3.19 shows the execution times and confirms the better performances of *MD* over *SD*.

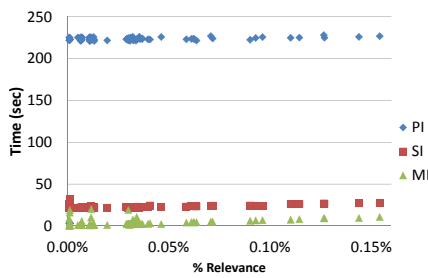


Fig. 3.18. Runtime vs. relevance (single insertion).

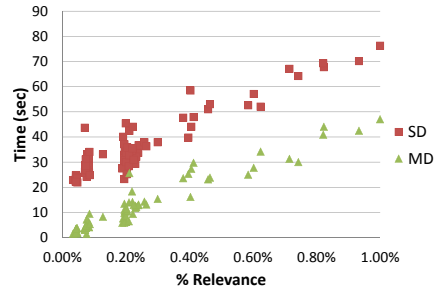


Fig. 3.19. Runtime vs. relevance (single deletion).

Multiple edges management

To assess the performance of the algorithms with the insertion and deletion of multiple edges, we varied the number of edges from 1 to 16. Each time is obtained as the average of 8 different sets having the same number of edges. Figure 3.20 reports the running times for the insertion case. Once again, both *SI* and *MI* are faster than *PI* and the gap gets bigger as we insert more edges. Notably, the growth of the running time of *MI* is negligible as the number of inserted edges increases.

Finally, Figure 3.21 reports the performance of *MD* and *SD* for the deletion of multiple edges. Also in this case, *MD* shows better performances than *SD*.

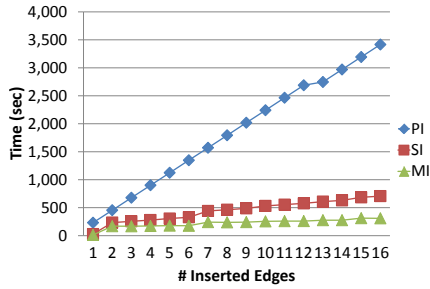


Fig. 3.20. Runtime vs. # of inserted edges.

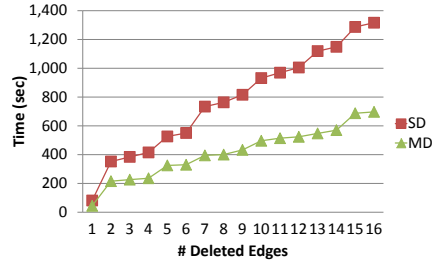


Fig. 3.21. Runtime vs. # of deleted edges.

3.3.5 Results on the regular dataset

The last dataset we considered is the regular one, which is bigger than the two datasets previously considered.

As for the approach in [52], neither the insertion nor the deletion algorithm was able to handle a single edge within 15 hours. Thus, in the following, we analyze the performance of our algorithms only.

We varied the number of edges to be inserted/deleted from 1 to 8. Each time reported in the figures discussed below is the average over three groups of 1, 2, ..., 8 edges.

The comparison of algorithms *SI* and *MI* (resp. *SD* and *MD*) is reported in Figure 3.22 (resp. Figure 3.24). It is evident that the algorithms for handling multiple insertions and deletions notably outperform the single edge counterpart—the gap gets bigger and bigger as we augment the number of edges to be handled. Figure 3.23 (resp. Figure 3.25) zooms in on the running time of *MI* (resp. *MD*) reported in Figure 3.22 (resp. Figure 3.24).

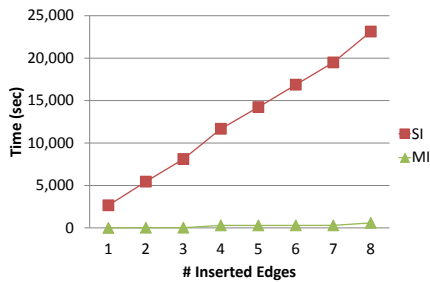


Fig. 3.22. Runtime vs. # of inserted edges.

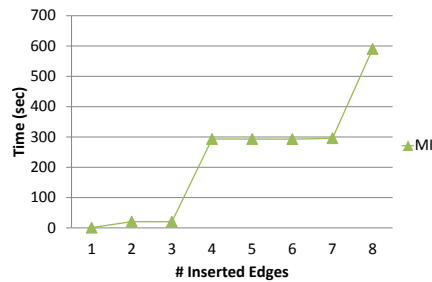


Fig. 3.23. Runtime vs. # of inserted edges.

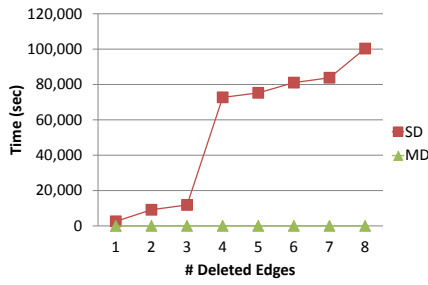


Fig. 3.24. Runtime vs. # of deleted edges.

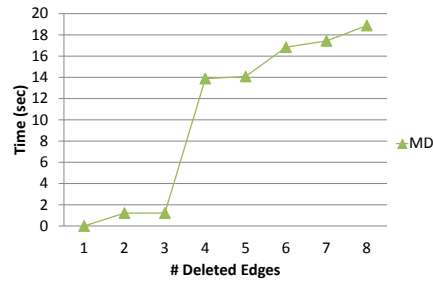


Fig. 3.25. Runtime vs. # of deleted edges.

3.4 Discussion

In this chapter, we have introduced our proposal for the incremental maintenance of all-pairs shortest distances for graphs stored in relational databases. The proposed algorithms significantly outperform the state-of-the-art techniques designed for the same setting. This is confirmed by the experimental evaluation where we considered three datasets (both real-world and synthetic) with different properties in terms of connectivity of the graph. Indeed, from the experimental results reported in Section 3.3, we can draw the following conclusions.

- Our algorithms outperform the algorithms proposed in [52] in all cases. Recall that, to the best of our knowledge, [52] is the state-of-the-art approach for the all-pairs shortest distances maintenance problem working on the secondary memory (in particular, graphs are stored in relational DBMSs as in our approach).
- [52] can handle edge deletions only over very small graphs; experiments over the regular graph have shown that also edge insertion becomes infeasible. Overall, [52] is impractical for the datasets we considered in our experimental evaluation. These limitations are overcome by our algorithms which are able to handle both edge insertions and deletions over significantly larger graphs.
- Algorithms *MI* and *MD* (which, as opposed to all other algorithms, are able to handle the insertion and deletion of *multiple* edges *at once*) are consistently the fastest ones.

Not surprisingly, their running time increases as the size of the dataset or the edge relevance augments. Moreover, the execution time grows slowly as the number of edges to be inserted or deleted increases. The difference between the multiple edge algorithms and the single edge counterpart gets bigger and bigger as the number of edges to be inserted/deleted augments (such a difference was particularly high on the regular graph), which supports the value of our multiple edge insertion/deletion algorithms and

confirms their ability of leveraging the information about a batch of edges to reduce the time needed to update the shortest distance relation.

**Non-Progressive Models for Viral Marketing in
Social Networks**

Influence Maximization Problem

As mentioned in Part I, deciding whether to adopt an innovation (such as a political idea or product), individuals are frequently influenced, explicitly or implicitly, by their social contacts (friends, acquaintances, or colleagues). Indeed, the way in which new practices spread through a population depends mainly on the fact that people influence each other's behavior. In short, as a person see more and more people doing something, s/he generally become more likely to do it as well because, typically, the benefits gained by adopting a new behavior increase as more and more of your neighbors in the social network adopt it. As discussed in Chapter 1, it is essential for companies to target “opinion leaders”, as influencing them will lead to a large cascade of further recommendations. This is the goal of each viral marketing campaign, and corresponds in solving the influence maximization problem. To this end, it is mandatory to accomplish two tasks:

1. a detailed modeling of the influence diffusion process in the network;
2. efficient identification of the target nodes given a diffusion model.

In the following, we focus on this problem, introducing its complexity and the solutions proposed in literature. We discuss two of the most important and widely-studied influence propagation models, i.e. the *Independent Cascade Model* and the *Linear Threshold Model*. Then, we summarize some results regarding several extensions both for the influence diffusion models and the influence maximization problem.

4.1 Modeling Cascading Behavior

Consider a social network, a new technology *tech* and let v be an inactive node, i.e. v has still not adopted *tech*. From v 's perspective, the process of influence diffusion will be (roughly) as follows: as time unfolds, more and more of v 's neighbors will use *tech* (i.e. they become active). It can happen that at a given point, due to its neighbors behaviors, v becomes active as well by adopting

tech. As a consequence, v 's decision may in turn trigger further decisions by nodes to which v is connected. Many diffusion models were proposed to describe such a process. We consider the two basic ones: *Independent Cascade Model* and *Linear Threshold Model*. For both these models the diffusion of information (or influence) begins with an initial set of active nodes, denoted as S_0 , and proceeds in discrete time steps, generating the active sets S_t for all $t = 1, 2, \dots$. The process runs until no more activations are possible. It implies that once $S_t = S_{t-1}$, then the set of active nodes will no longer change, and the diffusion process halts returning the final active set S_t . Moreover, both models are progressive. It means that once a node becomes active (or is activated), it *stays* active, i.e. for all $t \geq 0$, $S_t \subseteq S_{t+1}$. In the following, we will describe in details the distinguishing features of the above mentioned models.

4.1.1 Independent Cascade Model

In the *Independent Cascade Model* (denoted as *IC*) first described in [43], each edge $(u, v) \in E$ has an associated influence probability $p_{u,v} \in [0, 1]$, corresponding to the influence exerted by node u to node v . More in detail, given an input social graph $G = (V, E)$, the influence probability of all arcs, and the initial set of active nodes S_0 , the process unfolds in discrete steps according to the following randomized rules:

- When node u first becomes active in step t , it is given a single chance to activate each currently inactive neighbor v ; it succeeds with probability $p_{u,v}$, independently of the history thus far;
- if u succeeds, then v will become active at step $t + 1$; but whether or not u succeeds, it cannot make any further attempts to activate v in subsequent rounds. Clearly, if v has multiple newly activated neighbors, their attempts are sequenced in an arbitrary order.

Figure 4.1 shows an example of this diffusion process. Blue numbers next to the edges are the influence probabilities, while red nodes denote those ones activated during the process. Solid red edges from an active node u to its inactive out-neighbors mean that u is trying to activate its neighbors through these edges, while a dotted blue edge from u to v denotes an edge that cannot be used to propagate the influence because either u has got its chance to activate v or v is already active.

Initially, at time $t = 0$, the seed node a is active (Figure 4.1 (a)). At step $t = 1$, a successfully activates b and f , but fails to activate e (Figure 4.1 (b)). At step $t = 2$, f fails to activate d , e and g , while b successfully activates c (Figure 4.1 (c)). At this point, the diffusion stops because the last activated node, i.e. c , does not have any outgoing edges to inactive nodes.

Live-Edge Graph with independent edge selection

The *IC* Model can be also defined in an equivalent way by using the *live-edge* model with independent arc selection and assuming that all random choices

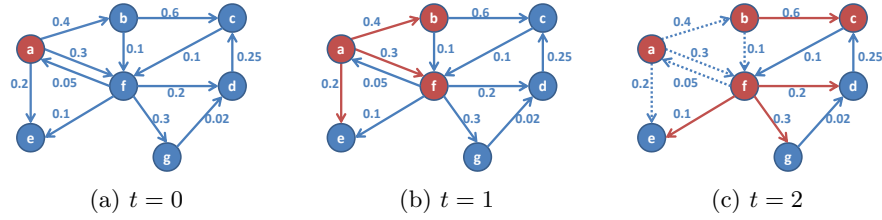


Fig. 4.1. An example of the diffusion process of the IC model.

involving the diffusion process have been made in advance [43]. More in detail, for each edge $(u, v) \in E$, a coin of bias $p_{u,v}$ is flipped at the very beginning of the process. The edges in the original graph G for which the coin flip indicated that an activation has been successful are marked as “live” while the remaining edges are marked as “blocked”. Once fixed the outcomes of the coin flips, the set of active nodes at the end of the diffusion process can be deterministically computed. Indeed, starting from the seed set S_0 , the active nodes will be those connected to at least one node in S_0 by a path consisting only of live edges.

Let X be the *live edge* graph obtained by considering only the edges marked as live, then the final set of active nodes corresponds to the set of nodes in V that are reachable from S_0 in X . Clearly, we can have a live edge graph X for each possible set of outcomes for all the coin flips on the edges. Thus, each X can be seen as a possible world. We denote with $R_t^X(S_0)$ the set of nodes that are reachable in X from S_0 within i steps, with $i = 0, 1, 2, \dots$

To show the equivalence between the *IC* model and the *live-edge* model, the following two distributions need to be equivalent:

1. the distribution over active sets obtained by running the Independent Cascade model starting from S_0 , and
2. the distribution over sets of nodes reachable from S_0 , when live edges are selected as previously described.

More in detail, for any $t \geq 1$, consider any sequence of subsets $A_1, A_2, \dots, A_t \subseteq V$ such that $A_1 \subseteq A_2 \subseteq \dots \subseteq A_t$, and once two consecutive subsets are equal, all the subsequent subsets are also equal. Then, the two models are equivalent if

$$Pr(S_t = A_t \mid S_1 = A_1, \dots, S_{t-1} = A_{t-1}) = Pr(R_t^X(S_0) = A_t \mid R_1^X(S_0) = A_1, \dots, R_{t-1}^X(S_0) = A_{t-1})$$

for any such sequence. [12] shown that these probabilities are exactly the same and are equal to:

$$\prod_{v \in A_t \setminus A_{t-1}} \left(1 - \prod_{u \in A_{t-1} \setminus A_{t-2}} (1 - p_{u,v}) \right) \prod_{v \in V \setminus A_t} \prod_{u \in A_{t-1} \setminus A_{t-2}} (1 - p_{u,v}) \quad (4.1)$$

Indeed, under the independent cascade model, the set S_t of nodes active at step t corresponds to A_t given $S_0, S_1 = A_1, \dots, S_{t-1} = A_{t-1}$ if and only if:

1. each node $v \in A_t \setminus A_{t-1}$ is also activated at step t , i.e. $v \in S_t \setminus S_{t-1}$, thanks to the influence exerted from its in-neighbors in $A_{t-1} \setminus A_{t-2}$;
2. no node in $V \setminus A_t$ is activated by any node in $A_{t-1} \setminus A_{t-2}$.

Analogously, the set $R_t^X(S_0)$ of nodes reachable within t step given $R_0^X(S_0), R_1^X(S_0) = A_1, \dots, R_{t-1}^X(S_0) = A_{t-1}$ is A_t if and only if:

1. each node $v \in A_t \setminus A_{t-1}$ is reached in one step by at least one node in $A_{t-1} \setminus A_{t-2}$;
2. no node in $V \setminus A_t$ is reachable in one step by any node in $A_{t-1} \setminus A_{t-2}$.

Since all activation events in the *IC* model are independent and in the live edge graph each edge (u, v) is independently selected with probability $p_{u,v}$, we have that Equation 4.1 holds.

4.1.2 Linear Threshold Model

The independent cascade model is suitable for modeling the diffusion of simple contagions, such as the adoption of technologies or viruses, where activations may be triggered from a single source. However, there are many situations in which exposure to multiple independent sources are needed for an individual before taking a decision. The *Linear Threshold* (or *LT*) Model has been introduced in [43] to reflect this kind of behavior.

In this model, a node v can be influenced by each neighbor u according to a weight $b_{u,v}$ such that $\sum_{u \in N^{in}(v)} b_{u,v} \leq 1$, where $N^{in}(v)$ is the in-neighbors set of node v . The dynamics of the process proceed as follows. Each node v chooses a threshold ϑ_v uniformly at random from the interval $[0, 1]$. This represents the weighted fraction of v 's neighbors that must become active in order for v to become active. Given a random choice of thresholds, and an initial set of active nodes S_0 (with all other nodes inactive), the diffusion process unfolds deterministically in discrete steps: in step t , all nodes that were active at step $t-1$ stay active, and any node v for which $\sum_{u \in N^{in}(v) \cap S_{t-1}} b_{u,v} \geq \vartheta_v$ is activated.

Thus, v becomes active if the total weight of its active neighbors is at least ϑ_v .

An example of the diffusion process of the linear threshold model is reported in Figure 4.2. As in Figure 4.1, red nodes denote those ones activated during the process. Blue numbers next to the edges are the influence weights, while the grey ones next to the nodes are the thresholds each node has independently selected uniformly at random at the very beginning of the process. Solid red edges from an active node u to its inactive out-neighbors mean that u is trying to activate its neighbors through these edges, while a solid blue edge from u to v denotes that u has already tried to activate v , but its influence does not suffice to get v active. Differently from the dotted ones (that

cannot be used to propagate the influence), a solid blue edge from u to v denotes that the influence exerted from u to v may be take into account when another v 's neighbor will have a chance to activate v .

Initially, at time $t = 0$, only node a is active (Figure 4.2 (a)). At step $t = 1$, a successfully activates b , but its influence is not enough to activate e and f (Figure 4.2 (b)). At step $t = 2$, b succeeds in activating c , and, together with the influence exerted by a , f is activated as well (Figure 4.2 (c)). Then f tries to activate d , e and g , but only the latter is successfully activated (Figure 4.2 (d)). At time $t = 4$, g fails in activating d and the diffusion process stops (Figure 4.2 (e)).

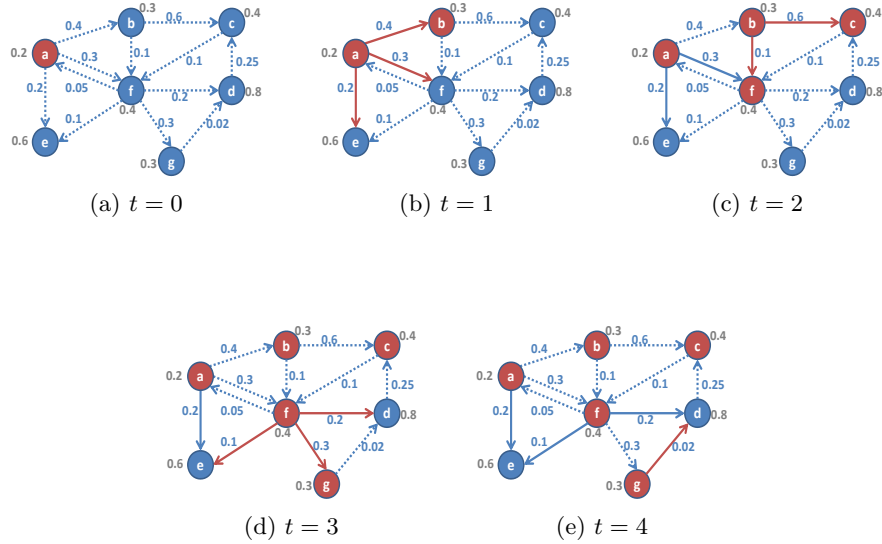


Fig. 4.2. An example of the diffusion process of the LT model.

Live-Edge Graph with proportional edge selection

Analogously to the independent cascade model, the linear threshold model also has an equivalent live-edge graph model. More in detail, we build a random live-edge graph X by selecting, for each node $v \in V$, at most one incoming edge of v with probability proportional to the weight of the edge. More specifically, the node v selects the edge from u with probability $b_{u,v}$, and no edge with probability $1 - \sum_{u \in N^{in}(v)} b_{u,v}$. The selected edges are those “live”, while the

remaining ones are “blocked”. As before, $R_i^X(S_0)$ is the set of nodes that are reachable in X from S_0 within i steps, with $i = 0, 1, 2, \dots$. To show that reachability under the live-edge graph defines a process equivalent to that

of the LT Model, we report the original proof given in [43] that works by induction over the iteration of the LT model. More in detail, let $A_i \in V$ be the set of nodes active after the i -th iteration of the LT process, for $i = 0, 1, \dots$. An inactive node v becomes active at time t under the LT model if the influence weights in $A_{t-1} \setminus A_{t-2}$ push it over its threshold, given that its threshold was not exceeded already. This probability is $\frac{\sum_{u \in A_{t-1} \setminus A_{t-2}} b_{u,v}}{1 - \sum_{u \in A_{t-2}} b_{u,v}}$. Similarly, we can

compute one step at a time the set of nodes reachable from the seed set S_0 , obtaining the set A_i of nodes reachable within i steps, for $i = 0, 1, \dots$. If node v has not be reachable within $t-1$ step, then the probability it will be reachable at stage t is equal to the chance its live-edge comes from $A_{t-1} \setminus A_{t-2}$, given that its live edge has not come from any of the earlier sets. This probability corresponds to the activation probability under the LT process, that, as a result, is equivalent to the reachability under the live-edge graph.

4.2 Problem Statement

The influence maximization problem can be formally defined as follows:

Problem (Influence Maximization). Given a social network $G = (V, E)$, a diffusion model on G , and a budget k , find a seed-set $S \subseteq V$ with $|S| \leq k$ of influential people such that by targeting them to adopt a product, we maximize the spread of viral propagation $\sigma(S)$, i.e. the number of other people in the network who adopt the same product.

This problem can be split into two subtasks. We first need a way to compute the influence spread $\sigma(S_0)$ given a set S_0 , and then searching for the seed set that maximizes the influence spread. In any case, influence maximization in classic diffusion models including both the independent cascade and the linear threshold models is computationally intractable, more precisely NP-hard [43]. There are two sources of complexity:

1. Modeling in detail the influence diffusion process in the network and computing the expected number of active nodes, given a seed set of initial adopters;
2. Identifying efficiently the seed set, i.e. the nodes that can maximize the spread under the above diffusion model.

To this end, several approximation algorithms and scalable heuristics have been proposed in the literature. It is worth noting that, the problem of finding an optimal set of k influential nodes is different from the problem of selecting k individuals that are each, individually optimal. Intuitively enough, if two top influential nodes both have strong influence on the same group of nodes, the influence exerted by the ensemble combination will be quite a bit less than the sum of the influence of each. As a result, applying naive centrality measures to find the optimal set of influential nodes will likely fail.

4.2.1 Greedy Algorithm for Influence Maximization

The first provable approximation guarantees for efficient algorithms has been introduced in [43] and it is based on a natural diminishing property of the problem, submodularity.

Definition 4.1 (Submodular and Monotone function). *Given a set Ω , a submodular function is a set function $f : 2^\Omega \rightarrow \mathbb{R}$ satisfying, $\forall X, Y \subseteq \Omega$ with $X \subseteq Y$ and $\forall i \in \Omega \setminus Y$, the following properties:*

$$f(X \cup \{i\}) - f(X) \leq f(Y \cup \{i\}) - f(Y)$$

A submodular function f is monotone if $\forall X, Y \subseteq \Omega$ with $X \subseteq Y$

$$f(X) \leq f(Y).$$

In [43] it has been shown that if the function $\sigma(\cdot)$ is monotone and submodular, then a simple greedy algorithm able to select a good set of seed nodes can be exploited. Such an algorithm gives a $(1-1/e)$ -approximation to the optimum. It implies that the resulting set S_0 activates at least $(1 - 1/e) > 63\%$ of the number of nodes that any other size- k set could activate.

Algorithm 8 Greedy Algorithm for Influence Maximization

Input: Size of seed set k
 Social Network $G = (V, E)$
Output: Seed set S_0
 1: $S_0 = \emptyset$;
 2: **while** $|S_0| < k$ **do**
 3: $u \leftarrow \operatorname{argmax}_{w \in V \setminus S_0} (\sigma(S_0 \cup \{w\}) - \sigma(S_0))$;
 4: $S_0 = S_0 \cup \{u\}$;
 5: **end while**
 6: **return** S_0 ;

The greedy algorithm, as depicted in Algorithm 8, works by adding to the candidate set S_0 the node u providing the largest marginal gain $\sigma(S_0 \cup \{u\}) - \sigma(S_0)$. This selection process is repeated until k elements are selected. For many diffusion models, including *IC* and *LT*, σ satisfies monotonicity and submodularity. The latter, in both the independent cascade model and the linear threshold model, is proved in [43] by utilizing the equivalent live-edge graph models. Indeed, we reported before the proof that both the IC and LT models correspond to live-edge graphs with different probabilistic edge selections. Despite the way in which the live-edge graphs are built is different, for both models we have:

$$\sigma(S_0) = \sum_{\forall X} Pr(X) |R^X(S_0)|$$

where $Pr(X)$ denotes the probability that X is selected from among all possible live-edge graphs. It is well known that a non-negative linear combination of monotone (resp. submodular) functions is also monotone (resp. submodular). Therefore, to prove that $\sigma(\cdot)$ is monotone and submodular, it is sufficient to show that, for any live-edge graph X , $|R^X(\cdot)|$ is monotone and submodular. Clearly, $|R^X(\cdot)|$ is monotone. To see that it is also submodular, we have to verify that, for any two sets of nodes, S and T , and a node $v \in V \setminus T$, $R^X(T \cup \{v\}) \setminus R^X(T) \subseteq R^X(S \cup \{v\}) \setminus R^X(S)$. It implies that the number of elements in $R^X(\{v\})$ that are not already in $R^X(S)$ has to be equal or bigger than the number of elements in $R^X(\{v\})$ that are not already in $R^X(T)$. Since $S \subseteq T$, the set of nodes reachable from S can not be larger than those reachable from the bigger set T . Thus, $|R^X(S \cup \{v\})| - |R^X(S)| \geq |R^X(T \cup \{v\})| - |R^X(T)|$, which is the definition of submodularity. It means that the greedy approach can be applied for these diffusion models as $\sigma(\cdot)$ is submodular and monotone. However, the greedy algorithm requires repeated evaluation of $\sigma(\cdot)$ (see line 3 Algorithm 8), which is NP-hard. To this end, in [43] the authors propose a greedy solution based on the Monte Carlo simulations of the diffusion process to estimate the influence spread. Given a seed set S_0 , we can simulate the randomized diffusion process with seed set S_0 for l times. Each time, we count the number of active nodes after the diffusion ends, and then take the average of these counts over the l times. Clearly, the bigger is l , the more accurate it will be the estimate of $\sigma(S_0)$. It is worth noting that, since Monte Carlo simulations only return an estimate of influence spread and not the exact value, the greedy algorithm gives a $(1 - 1/e - \varepsilon)$ -approximation to the optimum (for any $\varepsilon > 0$).

Despite in literature there are not other algorithms able to outperform the greedy approach, the naive Monte Carlo greedy algorithm has a serious drawback, i.e. its inefficiency. Indeed, it results unfeasible to run even for graphs with a moderate size. To deal with this problem, a number of optimization techniques and heuristics have been proposed with the intent of developing more efficient and scalable influence maximization algorithms [46, 13, 33, 64, 62]. The main idea behind most of these works is to avoid Monte Carlo simulations by exploiting specific aspects of the graph structure and the diffusion model to significantly speed up the influence computations.

4.3 Extensions

In this section, we summarize some of the results appeared in the literature where several extensions both for the influence diffusion models and the influence maximization problem have been proposed. All the diffusion models proposed satisfy a property called progressiveness, i.e. they assume that, once a node is activated, it will not change its state any more. More specifically, a node may change from the inactive state to an active state, but it does not change from an active state back to the inactive state.

4.3.1 Influence Maximization under Competition

All of the above research works only consider the diffusion of a single product/service in the social networks. However, in a more real scenario, we may have that different products propagate concurrently in the same network and, typically, these diffusions interfere with each other. For example, if a user has been influenced by friends and, as a result, he has bought a product of a given brand, it is unlikely he will buy a similar product of a competing brand in the short period, even under a strong influence from other people. To this end, extensive research has been done concerning various maximization problems under competitive influence diffusions [7, 8, 48, 9, 38]. In the following we deal with only two competing products, despite the rationale can be easily extended to more than two competitors. Under this assumption, each node has two possible states: inactive and active, but, in the latter case, a node is either positively or negatively activated (according if it is active for the target company or not). Let S_0^+ (resp. S_0^-) be the seed set for the target company (resp. competitor), with $S_0^+ \cap S_0^- = \emptyset$. At every time step $t \geq 1$, there will be the positive and negative activation attempts of the inactive nodes. Clearly, since in most of the models proposed the influence propagates in discrete time, it may happen that both activation attempts are successful on the same node. It implies that a tie-breaking rule is necessary to determine the node's choice.

According to the objective function, there are many categories of maximization problems under competitive influence diffusions.

A natural extension to the traditional problem of influence maximization is to maximize the spread of one technology in the presence of one or more competitors [7, 8]. This problem corresponds to a situation where a company knows the current status of diffusion of the competing product, and it needs to select its own seed set to maximize its own coverage, considering the influence over users by its competitor. In [7] the authors propose a generalization of the independent cascade model. Despite they only provide a polynomial approximation algorithm for trees, an interesting feature introduced in this work is the use of continuous time in node activation. The latter allows to avoid the explicit definition of a tie-breaking rule because the probability of two nodes activating a neighbor at the same time is zero. In [8], the authors suggest several natural extensions to the linear-threshold model. However, submodularity does not hold for any of these models, thus the classic greedy approach for selecting the seed set S_0^+ that maximizes the target company diffusion cannot be used. Extensions of the LT model to the competitive influence scenario, usually, do not satisfy submodularity. One exception is the *K-LT model* proposed in [48], where the authors extend the LT model to a competitive setting while retaining submodularity. They consider the competitive influence maximization problem from the perspective of the social network owner, instead of from the perspective of one of the competing players. Each company that intends to run a campaign, specifies a budget, a fee to pay to the network

owner, and the latter selects and allocates the seeds to the various companies in the fairest possible manner.

Another possible objective function is to minimize the coverage of the competing product or opinion. This is especially the case when a negative opinion or a rumor about a company or political party is propagating, and the company or party tries to spread positive and true information to reduce the spread of the rumor as much as possible [9, 38]. This problem is known as *influence-blocking maximization* (IBM) and it has been also investigated in the epidemiology literature. Indeed, the negative opinion can also represent a disease, thus it is relevant to find a set of k nodes to immunize so that the spread of the disease is limited as much as possible. [9] considers two competing campaigns having different acceptance rates in the network and one of these starts with a delay, in response to the other. The authors shown that the general extension of the IC model, in which positive and negative influences have separate set of parameters, is not submodular. To achieve submodularity, they assume that positive propagation probability is 1 or is the same as negative propagation probability, which limits the expressiveness of the model. [38] shows that the objective function of the IBM problem is submodular under the competitive LT model and design an efficient algorithm to overcome the slowness of the greedy approach.

4.3.2 Continuous-Time Diffusion Process.

There are many real-world examples in which interactions between nodes occur at different rates (e.g. in viral marketing, some user can be quicker than others to recommend a service). Moreover, in most cases, influence must be estimated or maximized up to a given time. Maximizing influence spread within a certain time constraint is relevant in practice. For example, a marketer would like to have her advertisement viewed by a million people in one month, rather than in one hundred years. The goal is to find the seed set that maximizes the average number of nodes infected in a time period \mathbb{T} and under the assumption that influence can spread in continuous time [21, 30]. The continuous time influence maximization problem is NP-hard under the continuous time diffusion model proposed in [21, 30]. Indeed, when $\mathbb{T} \rightarrow \infty$, the independent cascade model is a particular case of that model. However, the objective function is monotone and submodular, thus it is possible to use the greedy algorithm to find a near-optimal solution. In contrast to previous discrete-time models which associate each edge with a fixed infection probability, the models proposed in [21, 30] associate to each edge (u, v) a transmission function, $f_{uv}(t_v|t_u)$, that is the conditional density of node v getting infected at time t_v given that node u was infected at time t_u . More specifically, in [30] the authors show that the influence estimation problem can be solved exactly when the transmission functions are exponential densities, by using continuous time Markov processes theory. However, the computational complexity of such approach, in general, scales exponentially with the size and density of the

network. This drawback is solved in [21], where the proposed algorithm can easily scale up to networks of millions of nodes. In particular, they introduce a scalable randomized algorithm for influence estimation in a continuous-time diffusion network with heterogeneous edge transmission functions.

In any case, modeling information diffusion using continuous-time diffusion networks can provide significantly more accurate models than discrete time models.

Competitive, Continuous Time and Non-Progressive Influence Maximization

Recent studies on modeling influence propagation focus on progressive models, i.e., once a node is influenced (active), it stays in that state and cannot become inactive. However, in many real life applications, this assumption is unrealistic as nodes can get activated and deactivated many times. Consider as an example the following scenarios:

- Customers subscribing to one of several competing service providers and have the opportunity and incentive to switch back;
- The spread of diseases among populations (Epidemic models) [51];
- Modeling the interactions and diffusion of conflicting ideas (e.g., opinions on political candidates) in the network, where people’s opinion may change due to the interaction with their friends (Voter model) [25, 23].

Extensive research has been done to define several extensions both for the influence diffusion models and the influence maximization problem. The goal is to catch those behaviors closer to the real world. To this end, some topics have been extensively investigated, e.g. as competitive viral marketing [8, 9, 48], in which two or more players compete with similar products on the same network, continuous time diffusion networks [21], where the diffusion process evolves in continuous time and interactions between nodes occur at different rates. However, non progressive models for viral marketing are still in their infancy. In particular, to the best of our knowledge, there is lack of work considering the competitive viral marketing problem in a non progressive context when diffusion evolves in continuous time. In the following, we define the *non-progressive influence maximization* problem and propose a new diffusion model, named *NPK-LT Model*, obtained by extending the K-LT Model defined in [48]. The proposed model captures the non-progressive features in viral marketing and satisfies the desired properties of monotonicity and submodularity.

5.1 Problem Statement and Related Works

Non-progressive models for viral marketing are catching increasing attention recently. Their importance has been already dealt in the seminal work [43], where the authors show that the non-progressive influence maximization problem can be reduced to the progressive case in a different graph. Given a graph $G = (V, E)$ and a time limit \mathbb{T} , the new graph is obtained by replicating the nodes in G for each time stamp and connecting each node in this graph with its neighbors in G indexed by the previous time step. However, since in a non-progressive process a node can switch between states indefinitely, the classical target function of influence maximization under progressive models, where the goal is to maximize the expected number of influenced nodes, can not be exploited in this context. To this end, [47] suggest a new objective function, i.e. maximizing the total time in which nodes are using a product. They propose a discrete-time non-progressive model obtained by exploiting the idea in [43] and, to overcome the scalability bounds associated with this solution, they introduce an efficient implementation of a continuous-time non-progressive model. This is the work most closely related to ours. However, they do not consider competitors that is the focus of our research instead. Indeed, non-progressive models are more suitable for modeling diffusion of competing products over the same network. This has been confirmed in [63], where a data-driven model for predicting the adoption of competing products has been proposed. Differently from our works, they model how frequently users adopt a product, but they are not interested in detecting influential nodes and maximizing the overall adoption.

Epidemic models and voter models are intrinsically non-progressive models. The former are used originally to study the spread of diseases. Various epidemic models have been proposed [51]. A frequently used model is the *SIS* model, where a node in the *I* state (infected) may transition back to state *S* (susceptible) with a given probability in every time unit, so that the node becomes susceptible to disease again. Another more sophisticated model is the *SIRS* model, where infected nodes may recover from a disease but not get lifelong immunity. As a result, they may still be susceptible to the disease and become infected again.

The voter model is a non-progressive model as well and it has been adopted as the basis for influence diffusion in several studies [23, 25]. Indeed, in the most basic form of the model, each node in the graph has state 0 or 1 and may change back and forth between 0 and 1. In each time step, a node v randomly selects one of its neighbors u and mimics the state that u had in the previous step. In [23] the authors consider the classic influence maximization problem under the voter model, which allows to deactivate nodes. In particular, they show that selecting k nodes with the largest degrees and setting them to state 1 is the best strategy in this model to maximize the probability of nodes in state 1 in the steady state. In [25], the authors deal with the problem of finding a perfect seed set of nodes in a competitive setting such that, activating them

at the beginning of the diffusion process, the whole network ends up being infected for the target company. They introduce a simple discrete-time non-progressive model where, at each time step t , a node results infected if the strict majority of its neighbors is active at time $t - 1$. Their proposal has clear limits: first of all the objective function is not realistic; then there are no scalable algorithms able to model this diffusion process.

5.1.1 Non-Progressive Influence Maximization Problem

Assume that K players, with $K \geq 2$, compete with similar products on the same network. We are interested in maximizing the total expected time during which nodes stay active for a target company up to \mathbb{T} , under the assumptions that switch of state are possible, and influence between nodes can spread at different rates. Maximizing influence spread within a certain time constraint is relevant in practice. For example, a marketer would like to have her advertisement viewed by a million people in one month, rather than in one hundred years. Moreover, there are many real-world examples in which interactions between nodes occur at different rates (e.g. some user can be quicker than others to recommend a service). We denote with S_t^j the set of nodes active for the company C^j at time t and with $\mathbf{S}_t = \{S_t^1, \dots, S_t^K\}$ the set of adapters for each company after t . Clearly, since at a fixed time a node can be active at most for a company, at every time $t \geq 0$ the sets in \mathbf{S}_t are disjoint. In the following, when the time is omitted, we are implicitly referring to time $t = 0$. More specifically, we define $\mathbf{S} = \{S^1, \dots, S^K\}$ as the *seed set allocation*, i.e. the set of active nodes at time $t = 0$. Moreover, similarly to [48], we use \mathbf{S}^{-i} to denote the set of seed sets for all companies $j \in \{1, 2, \dots, K\}$, with $j \neq i$, i.e. $\mathbf{S}^{-i} = \{S^1, \dots, S^{i-1}, S^{i+1}, \dots, S^K\}$.

The problem can be formalized as follow.

Problem (Non-Progressive Influence Maximization). Given a social network G , a target company C^i , a budget k , a time horizon \mathbb{T} and the set \mathbf{S}^{-i} of initial adopters for each company $C^j \neq C^i$, find a seed-set $S^i \subseteq V$ with $|S^i| \leq k$ of influential people such that initially targeting them to adopt C^i , we maximize the total Expected Active Time $\sigma_{np}^i(S^i, \mathbb{T})$ (informally EAT^i) during which nodes are active for C^i up to time \mathbb{T} .

Let $\sigma^i(S^i, \mathbf{S}^{-i}, t)$ be the expected number of nodes active for C^i at time t given a seed set allocation, then EAT^i is defined as $\sigma_{np}^i(S^i, \mathbb{T}) = \int_{t=0}^{\mathbb{T}} \sigma^i(S^i, \mathbf{S}^{-i}, t) dt$.

The above problem formulation assumes that we know the set \mathbf{S}^{-i} when trying to find S^i . That is, company C^i knows the allocations of its competing companies. Since the influence maximization is an intractable problem even considering the simplest and progressive diffusion models, we decided to work under this assumption. However, we plan to propose a different formulation in the future to better model real case scenarios. It is also worth noting that

there are plenty of papers dealing with the same assumption [7, 8], i.e. situations where a company knows the current status of diffusion of the competing product, and it needs to select its own seed set to maximize its own coverage, considering the influence over users by its competitor.

5.2 CT Non-Progressive K-LT Model

In the following, we propose an extension of the K-LT Model defined in [48] that works under the continuous time assumption when switches of company are allowed. In the classical K-LT Model, K companies compete with similar products on the same network. The latter is a directed graph $G = (V, E)$ and, as for the LT Model, each edge $(u, v) \in E$ has an associated influence weight $b_{u,v}$ such that $\sum_{u \in N^{in}(v)} b_{u,v} \leq 1$, where $N^{in}(v)$ is the in-neighbors set of node

v . The propagation process starts by selecting a seed set for each company and unfolds at discrete steps. Initially, each node $v \in V$ randomly picks a threshold ϑ_v in the range $[0, 1]$. For each time step $t \geq 1$, if the total weight of the active neighbors for an inactive node v is greater than its threshold ϑ_v , node v becomes *influenced*. Then, at the same time t , v becomes *active* for one of the companies of its in-neighbors that activated at $t - 1$. The basic assumption is that as a node is activated it never deactivates. However, as we are interested in modeling a non-progressive phenomena, a node can switch between active and inactive states many times. Moreover, the diffusion process evolves in continuous time. In the following, we report a detailed analysis of these new features and the way we deal with.

5.2.1 Non-Progressiveness Property

Since we are considering a non-progressive scenario, as a node v has picked a company C^i , all its neighbors active for companies $j \neq i$, could try to influence v in adopting C^j . Indeed, a node active for a company i is assumed to be inactive for all the other companies $j \neq i$, with $i, j \in \{1, 2, \dots, K\}$. It means that, once a node v is influenced, i.e. its threshold ϑ_v is exceeded, its following state will be active for the neighbors that choose the same company, while inactive for the others. As a result, we need to keep track of the time intervals during which each node stays active for the company we want to maximize the spread. More in detail, for each company C^i with $i \in \{1, 2, \dots, K\}$ and for each node $u \in V$, we can define the *activation window*, $W^{u,i}[l]$, as the l -th time interval during which node u is first activated and then deactivated for C^i . Every activation window $W^{u,i}[l]$ is composed by an *activation time*, $t_A^{u,i}[l]$, and a *deactivation time*, $t_D^{u,i}[l]$. Intuitively enough, $t_A^{u,i}$ (resp. $t_D^{u,i}$) corresponds to the time in which node u is activated (resp. deactivated) in the considered time interval for C^i . In the following, if it is not specified otherwise, for each node $u \in V$ and a given time t , we are going to consider

the *current* u 's activation window, i.e. the activation window $W^{u,i}[l]$ such that $t \in [t_A^{u,i}[l], t_D^{u,i}[l])$. Since a node u can not be active for more than one company at the same time, there is no overlapping of activation windows belonging to different companies. This implies that at every time t , u 's current activation window is unique and well specified. Figure 5.1 shows the activation windows of a node u and depicts the correlation among these new variables. In this example, node u has been activated for three companies and the bold window is the current one.

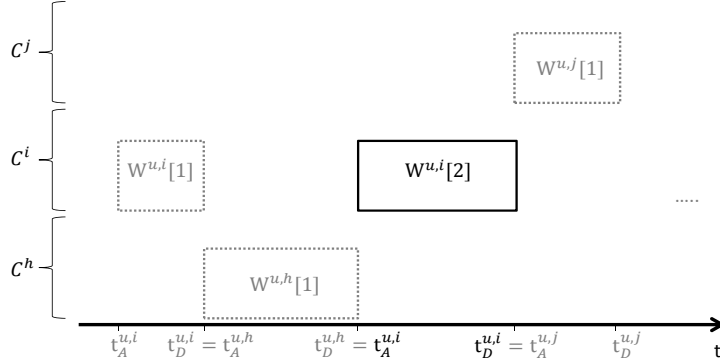


Fig. 5.1. An example of activation windows of a node

The main feature of this non-progressive process is that, once a node is active, it can change company anytime in the future. However, company switches are not only based on the influence weight of v 's neighbors, but they are strictly related to the last time node v has been involved in state switching.

More specifically, we assume that v is as less motivated in changing company as recently it has been activated for its current one. The rationale behind this choice is based on the consideration that when becoming active for a company has a cost (for example, purchasing a product or service), customers are more likely to exploit the service until its cost has been amortized by its use. To this end, we introduce a function that denotes the willingness of v to change company at a given time t and that allows to avoid the continuous (and unrealistic) switches of company every time a neighbor $u \in N^{in}(v)$ tries to encourage v to adopt its same company. More in detail, let t_A^v be the activation time of the current v 's activation window. We define a *switching function* $\rho(v, t) = 1 - e^{-\psi \Delta t}$, with $\Delta t = t - t_A^v$ and ψ constant, as the probability that at time t node v is motivated to leave its current company for another. Clearly, if node v has never been involved in an activation before the considered time, this function should not to affect the probability of its first activation. To this end, when t_A^v is undefined, we assume that $\rho(v, t) = 1$.

5.2.2 Continuous Time Property

The diffusion process evolves in continuous time. We model this feature by considering that information (or influence) spreads from an active node to its neighbors at different rates across different edges. To this end, for each outgoing edge $(u, v) \in E$, we associate a random *spreading time*, τ_{uv} , sampled from a density over time. The spreading time of an edge (u, v) is defined as the time interval after which the active node u can propagate the influence to an inactive node v , i.e. if t_A^u (resp. t_D^u) is the current activation (resp. deactivation) time of node u , then at time $t_A^u + \tau_{uv}$ the influence would reach its neighbor v . As a result, influence propagation does not rely *only* on the state of a node (active or inactive). We say that a node u is *influential* for its neighbor v at time t if u is still active for the same company at that time, i.e. $t < t_D^u$, and u stays active enough to allow its influence to span over node v , i.e. $t \geq t_A^u + \tau_{uv}$. It is worth noting that, each time a node u is activated, new values for u 's outgoing spreading times have to be considered. To this end, a list of pre-sampled spreading times is associated to each edge $(u, v) \in E$. Thus, we can further refine the notion of influential node as follows. Given an edge (u, v) , we say that node u , active for the company C^i , is also influential for v at time t , if there exists an activation window $W^{u,i}[l]$ such that $t \in [t_A^{u,i}[l] + \tau_{uv}, t_D^{u,i}[l])$.

We denote with $\mathbb{I}^i(v, t)$ the set of influential neighbors of v active for the company C^i at time t and with $\mathbb{I}(v, t) = \bigcup_{i \in \{1, 2, \dots, K\}} \mathbb{I}^i(v, t)$ the overall set of

its influential neighbors. Among the nodes belonging to the set $\mathbb{I}(v, t)$, it may exist a neighbor u for which $t = t_A^u + \tau_{uv}$. We say that this node is *actively influential* for v to denote that, differently from the other influential nodes that have already had their chance, u is the only one that can try to activate v at time t .

In the K-LT Model, the probability that an influenced node v gets active at time t for company C^i is equal to:

$$\sum_{u \in A_{t-1}^i \setminus A_{t-2}^i} b_{u,v} / \sum_{u \in A_{t-1} \setminus A_{t-2}} b_{u,v} \quad (5.1)$$

with A_{t-1}^i denoting the set of nodes active for company C^i at time $t-1$, and $A_{t-1} = \bigcup_{i \in \{1, 2, \dots, K\}} A_{t-1}^i$ the overall set of nodes active at the end of time $t-1$. Thus, v picks a company by considering the influence exerted by its in-neighbors that activated at time $t-1$. In our model, the rationale is quite similar, but, clearly, we need to take into account the influential nodes. Hence, it seems natural to compute the correspondent activation probability by just replacing the sets of active nodes in Equation 5.1 with the influential ones, as follows:

$$\sum_{u \in \mathbb{I}^i(v, t) \setminus \mathbb{I}^i(v, t^-)} b_{u,v} / \sum_{u \in \mathbb{I}(v, t) \setminus \mathbb{I}(v, t^-)} b_{u,v} \quad (5.2)$$

However, differently from the K-LT Model, it is worth noting that we operate in continuous time. Introducing continuous time in influence propagation has

a positive side effect, i.e., at each time $t > 0$, at most one event can occur and it involves only one node (e.g. v). It means that the probability of two nodes activating one node at the same time is zero. More in detail, at every time $t > 0$, just the event in which a node becomes actively influential for one of its neighbors and, consequently, tries to activate it for its same company, can occur. Occurrence of events induces a well-defined total order on time. For any given time instant t , we denote by t^- and $t^=$ the instants of time, respectively, of the last two events that occurred before t . Thus, if at time t , node u is actively influential for v , then $\mathbb{I}(v, t) \setminus \mathbb{I}(v, t^-) = \{u\}$ and $\mathbb{I}^i(v, t) \setminus \mathbb{I}^i(v, t^-)$ is not empty only if the company promoted by node u is exactly C^i . As a result, Equation 5.2 does not define a probability, but a boolean value, that is one if u is active for C^i , zero otherwise.

5.2.3 Model Definition

Our Non-Progressive model can be formally defined as follows:

Definition 5.1 (Continuous Time and Non-Progressive K-LT Model (NPK-LT Model)). *Given the seed set allocation $\mathbf{S} = \{S^1, \dots, S^K\}$. The diffusion process unfolds in continuous time according to the state switches of the nodes in the network that are defined as follow:*

- **Influenced Event.** *An inactive node v is influenced at time t if the cumulative influence of all its influential neighbors is such that $\sum_{u \in \mathbb{I}(v, t)} b_{u, v} \geq \vartheta_v$. It means that node v is influenced if the total weight of the in-neighbors of v still active and whose influence has reached v , is at least ϑ_v . We denote with $\alpha(v, t)$ the boolean storing the outcome of this event, i.e.*

$$\alpha(v, t) = \begin{cases} 1 & \text{if } \sum_{u \in \mathbb{I}(v, t)} b_{u, v} \geq \vartheta_v \\ 0 & \text{otherwise} \end{cases}$$

- **Activation Event.** *A node v influenced at time t is activated for the company C^i if there exists a v 's in-neighbor u , active for the company C^i , that became actively influential for v at time t , i.e. $u \in \mathbb{I}^i(v, t) \setminus \mathbb{I}^i(v, t^-)$ where $t^- < t$ is the timestamp of the last event; We denote with $\beta^i(v, t)$ the boolean storing the outcome of this event, i.e.*

$$\beta^i(v, t) = \begin{cases} 1 & \text{if } \exists u \in N^{in}(v) \text{ s.t. } u \in \mathbb{I}^i(v, t) \setminus \mathbb{I}^i(v, t^-) \\ 0 & \text{otherwise} \end{cases}$$

- **Switching Event.** *A node v active for the company C^i switches to company C^j at time t if one of its neighbor is become actively influential at that time, i.e. $\beta^j(v, t) = 1$, and v wants to change company, whose interest is modeled by the switching function $\rho(v, t)$. We denote with $\gamma^{ij}(v, t)$ the probability of this event, i.e.*

$$\gamma^{ij}(v, t) = \rho(v, t) * \beta^j(v, t)$$

Figure 5.2 shows a possible state evolution of a node under the assumptions of this extended K-LT Model with $K = 2$.

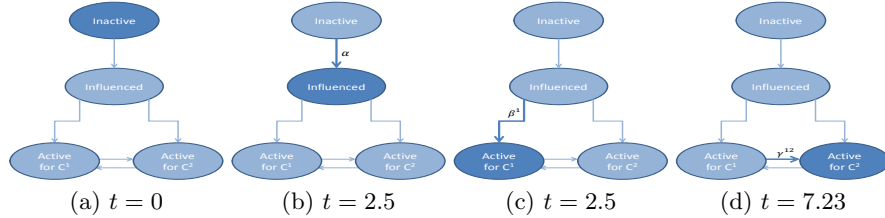


Fig. 5.2. State dynamics under the NPK-LT Model.

The node stays inactive until time $t = 2.5$ (Figure 5.2 (a)), after which it is first influenced (Figure 5.2 (b)) and then activated for C^1 (Figure 5.2 (c)). At time $t = 7.23$, it switches company, getting active for C^2 (Figure 5.2 (d)).

It is worth noting that seed nodes can be involved only in switching actions, since their initial state is active for the company they are promoting.

5.3 NPK-LT Model Properties

In order to prove that, given a seed set allocation and a target company C^i , EAT^i is monotone and submodular under the NPK-LT Model, we show that the latter is equivalent to an extended version of the live-edge model introduced in [43]. We recall that, in the considered social network G , each edge has an associated influence weight and a list of spreading times. Moreover, when the propagation process starts, each node $u \in V$ randomly chooses a threshold ϑ_u in the range $[0, 1]$.

Definition 5.2 (Continuous Time and Non-Progressive Live-Edge Model (NP-LE Model)). Let $G = (V, E)$ be a directed graph. A possible universe U can be computed by selecting $\forall v \in V$ at most one of v incoming edges. In particular, v picks an edge $(u, v) \in E$ with probability $b_{u,v}$, and no edge with probability $1 - \sum_{u \in N^{in}(v)} b_{u,v}$.

The above definition of NP-LE Model is equivalent to the classical LT live edge model introduced in [43], but it is worth noting that, since we are in a non-progressive scenario, edges incoming to seed nodes are not discarded. Indeed, also nodes in the seed sets can switch company, thus being influenced from the neighbors. A further difference w.r.t. the traditional definition of live

edge model is that we introduce the notion of *possible universe*, that seems replacing the classical *possible world* concept. The difference between these nomenclatures will be clear soon.

In the LT live-edge model discussed in [43], the selected edges (that build a possible world) are referred as “live” and the discarded ones are said “blocked”. In our model, instead, the selected edges build a possible universe U , while a possible world \mathbf{X}_t is a collection of K disjoint subset of V storing, for each $u \in V$, the time $t_A^u \leq t$ starting from which u belongs to the set $X_t^i \in \mathbf{X}_t$. Before defining how a possible world can be obtained, we introduce the notion of “ghost” edges. At every time $t \geq 0$, we have to distinguish, among the edges in U , the real live edges and those ones are ghosts. An edge (u, v) that is ghost at time t acts as a blocked edge. The rationale behind this choice is that such an edge cannot be used to propagate the influence to v , thus, ignoring it, will not affect the result of the diffusion process.

Definition 5.3 (Ghost Edges). *Let $E_s \subseteq E$ be the set of edges in the possible universe U and $\mathbf{X}_{t^-} = \{X_{t^-}^1, \dots, X_{t^-}^K\}$ a possible world. Then, the edge $(u, v) \in E_s$ is ghost at a given time $t > t^-$ if one of the following conditions holds:*

1. $t \neq t_A^u + \tau_{uv}$, or
2. v is not motivated in leaving its current set $X_{t^-}^i \in \mathbf{X}_{t^-}$.

The ghost edges are our instruments to introduce the influence propagation rule of the NPK-LT Model into the NP-LE Model. Indeed, the first condition in the definition above takes into account that influence can be propagated from u to v only if $t = t_A^u + \tau_{uv}$, i.e. u is actively influential for v at time t . The second condition models the feature that the less is motivated v in changing company, the highest is the probability that every attempt to influence it will fail. We recall that $\rho(v, t)$ represents the interest (eq. motivation) of node v in changing company at time t and takes into account the last activation time of node v . Thus, assuming that the possible world \mathbf{X}_{t^-} corresponds to the sets of nodes active at time t^- , with probability $1 - \rho(v, t)$, (u, v) is a ghost edge despite u is actively influential for v at time t . Clearly, since the temporal parameters associated to the graph play a role also in the live-edge model, at every time $t \geq 0$ at most one node can be actively influential and, as a result, at each time t we can have at most one live edge.

In order to better understand the notion of Universe and possible World, we can exploit the notion of quantum measurement. More in detail, the state of a system can be determined only by performing a measure on the system that gives us its actual state. In our case the system is represented by the universe and, as time flows, different possible worlds will arise. Thus, the measurement of the universe state at a given time t , will select the actual possible world induced by the evolution of the universe due to the events causing edges to become live. As will be clear in the following, this measure corresponds to the notion of reachability that in turn is function of the sequence of events occurring in the universe.

5.3.1 Reachability under the NP-LE Model

The notion of reachability is also peculiar to our model. We are in a competitive scenario, thus we have to consider that K companies C^j , with $j \in \{1, 2, \dots, K\}$, compete on the same network and each of them has a set of initial adopter S^j associated. In the following, since at most a single edge can be live at every time $t \geq 0$, we introduce first the notion of *one-step reachability*.

Definition 5.4 (One-Step Reachability). *Given a node v , a possible universe U , a possible world \mathbf{X}_{t^-} and a candidate set $X_{t^-}^i \in \mathbf{X}_{t^-}$, we say that v is one-step reachable at time t from $X_{t^-}^i$ if (u, v) is the current live edge in U given \mathbf{X}_{t^-} and $u \in X_{t^-}^i$.*

Hence, according to the definition above, a node v is one-step reachable at time t from $u \in X_{t^-}^i$ if the edge (u, v) is live at time t . In other words, v is one-step reachable if, starting from the time in which u is lately activated, the elapsed time t suffices to allow the influence to propagate to the node v across the edge connecting these two nodes. We recall that at every time $t \geq 0$ at most one edge can be live, thus the same target node can not be reachable from two sets $X_{t^-}^i, X_{t^-}^j \in \mathbf{X}_{t^-}$, with $j \neq i$.

Definition 5.5 (Reachable nodes). *Given a possible world \mathbf{X}_{t^-} and a possible universe U , the set of nodes reachable at time t from a set $X_{t^-}^i \in \mathbf{X}_{t^-}$ denoted as $R_t^U(X_{t^-}^i, \mathbf{X}_{t^-})$, includes:*

- each node $u \in X_{t^-}^i$ and the node $v \in V$ such that v is one step reachable at time t from $X_{t^-}^i$, or
- each node $u \in X_{t^-}^i$ such that u is not one step reachable at time t from any set $X_{t^-}^j \in \mathbf{X}_{t^-}$, with $j \neq i$.

We denote with $R_t^U(\mathbf{X}_{t^-})$ the overall set of subsets $R_t^U(X_{t^-}^i, \mathbf{X}_{t^-})$ of nodes reachable in U from any $X_{t^-}^i \in \mathbf{X}_{t^-}$ at time t , i.e.

$$R_t^U(\mathbf{X}_{t^-}) = \bigcup_{X_{t^-}^i \in \mathbf{X}_{t^-}} R_t^U(X_{t^-}^i, \mathbf{X}_{t^-})$$

The set $R_t^U(\mathbf{X}_{t^-})$ corresponds to a possible world at time t . Indeed, given a seed set allocation \mathbf{S} and a universe U , a possible world \mathbf{X}_t can be recursively constructed as follows:

$$\mathbf{X}_t = \begin{cases} \mathbf{S} & \text{if } t = 0 \\ R_t^U(\mathbf{X}_{t^-}) & \text{otherwise} \end{cases}$$

where $t^- < t$ represents the last time stamp in which a couple $(u, v) \in E_s$ became live. Thus, since the event in which an edge becomes live is not completely deterministic, we may have different possible worlds at time t .

Example 5.6. Figure 5.3 shows how we can emulate the diffusion process on a possible universe applying iteratively this notion of reachability. Initially, given the graph G (Figure 5.3 (a)), we can generate a set of possible universe, according to the NP-LE Model definition and the weight influence (not reported for sake of readability in the plot) associated to each edge in G . In (Figure 5.3 (b)) we consider one of these possible universe, named U , and two disjoint subsets of nodes, $A^1 = \{a\}$ and $A^2 = \{f\}$ (highlighted in red and green and representing the possible world $\mathbf{X}_{t=0}$). Dotted (resp. solid) edges denote *ghost* (*live*) edges at the specified time. The blue colored numbers associated to edges denote their spreading times, while nodes reached at least once during the process, are labeled with the last time in which this event occurred (the orange colored numbers).

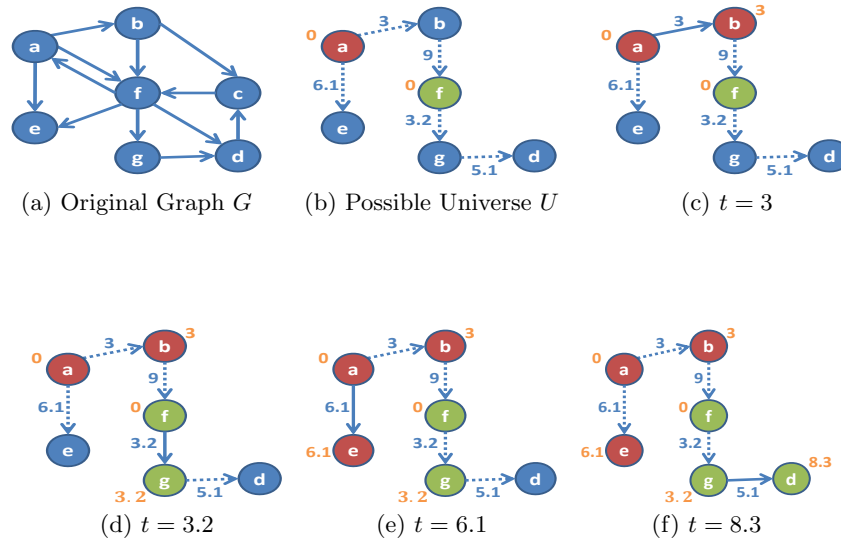


Fig. 5.3. An example of diffusion process under the NP-LE Model.

Starting from a possible world \mathbf{X}_{t^-} and assuming that at time $t = t^- + \delta$ an edge becomes live, we obtain a new possible world \mathbf{X}_t by computing the reachable nodes from \mathbf{X}_{t^-} .

At time $t = 3$, edge (a, b) becomes live with probability 1 and b is added to A^1 , thus we get the possible world $\mathbf{X}_{t=3} = \{A^1 = \{a, b\}; A^2 = \{f\}\}$ (Figure 5.3 (c)). Analogously, node g is added to A^2 at time $t = 3.2$ as (f, g) gets live (Figure 5.3 (d)), while at time $t = 6.1$, A^1 includes e thanks to the in-going live edge (a, e) (Figure 5.3 (e)). At time $t = 8.3$, d is added to A^2 (Figure 5.3 (f)). At this point, each node has been reached at least once, obtaining the possible world $\mathbf{X}_{t=8.3} = \{A^1 = \{a, b, e\}; A^2 = \{f, g, d\}\}$. The process may continue by considering, at time $t = 12$, the edge (b, f) .

At that time, node b becomes actively influential for f but, in this case, the edge (b, f) will be actually live with probability $\rho(f, t)$. We recall indeed that the switching probability of a node that has already been reached once, is not equal to 1. In this example, we assume that (b, f) remains ghost, i.e. we consider the possible world $\mathbf{X}_{t=12} = \mathbf{X}_{t=8.3}$.

From the previous example, it is clear that given a universe and starting from the seed set allocation \mathbf{S} , we can build a tree of possible worlds by computing iteratively the reachable nodes and, assuming of flipping a coin with bias $\rho(v, t)$ each time an edge $(u, v) \in E_s$ may become live, considering each possible outcome of the coin flip. In Figure 5.4 we depict this tree for the universe and the seed sets reported in Figure 5.3 (b). The value reported besides each node in this tree is the probability of that possible world. As expected, the possible worlds tree has a single branch till time $t = 8.3$, i.e. until each node in the graph has not been reached more than once.

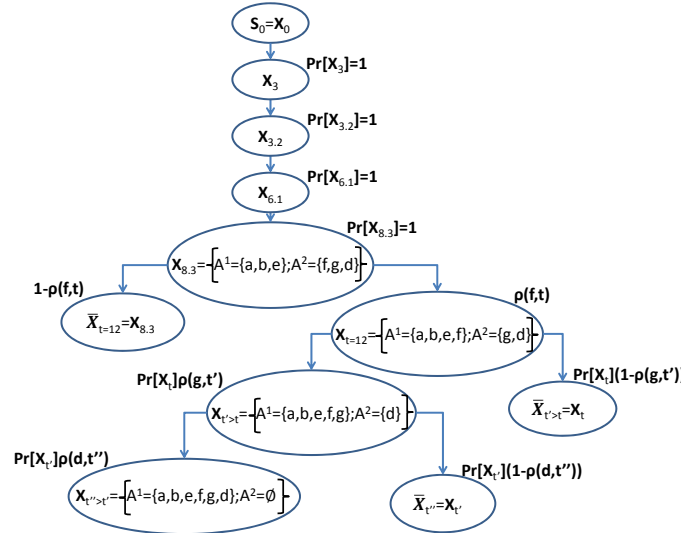


Fig. 5.4. Tree of Possible Worlds

We can build new possible worlds till time t'' , corresponding to the time in which all the nodes belong to the same set. We recall that, each time a node switch set, new values of its outgoing spreading times are picked.

5.3.2 Monotonicity and Submodularity

We need to show that reachability under the NP-LE Model is a monotone and submodular function. More in detail, given a universe U and a possible world \mathbf{X} , we prove that the function $R_t^U(X^i, \mathbf{X}^{-i})$ is monotone and submodular in

X^i , with \mathbf{X}^{-i} fixed. We recall that the formalism \mathbf{X}^{-i} denotes a collection of disjoint subsets of nodes, one for each company $j \in \{1, 2, \dots, K\}$, with $j \neq i$. To this end, given two sets of nodes S and T such that:

1. $S \subseteq T$, and
2. $T \cap X^j = \emptyset$ for each $X^j \in \mathbf{X}^{-i}$.

we prove that the following conditions

1. monotonicity

$$|R_t^U(S, \mathbf{X}^{-i})| \leq |R_t^U(T, \mathbf{X}^{-i})|$$

2. submodularity

$$|R_t^U(S \cup \{v\}, \mathbf{X}^{-i})| - |R_t^U(S, \mathbf{X}^{-i})| \geq |R_t^U(T \cup \{v\}, \mathbf{X}^{-i})| - |R_t^U(T, \mathbf{X}^{-i})|$$

are satisfied.

Since the time is fixed and at most one node can be reached every time, the first condition holds. As concern the submodularity, the proof straightforwardly follows from the one given in [43]. The quantity $|R_t^U(S \cup \{v\}, \mathbf{X}^{-i})| - |R_t^U(S, \mathbf{X}^{-i})|$ (resp. $|R_t^U(T \cup \{v\}, \mathbf{X}^{-i})| - |R_t^U(T, \mathbf{X}^{-i})|$) is the number of elements in $R_t^U(\{v\}, \mathbf{X}^{-i})$ that were not included in $R_t^U(S, \mathbf{X}^{-i})$ (resp. $R_t^U(T, \mathbf{X}^{-i})$). As the number of nodes in $R_t^U(T, \mathbf{X}^{-i})$ is greater or equal to those in $R_t^U(S, \mathbf{X}^{-i})$, the submodularity condition holds.

Theorem 5.7. *The expected spread $\sigma^i(S^i, \mathbf{S}^{-i}, t)$ of a company C^i under the NPK-LT Model is monotone and submodular.*

Proof. In order to prove that $\sigma^i(S^i, \mathbf{S}^{-i}, t)$ under NPK-LT Model is monotone and submodular, we need to show the equivalence between the diffusion process defined by our model and the reachability under NP-LE Model.

More in details, consider, at any time $t \geq 0$, any set $\mathbf{A}_t = \{A_t^1, \dots, A_t^K\}$ of K disjoint subsets of V . Given the set \mathbf{S}_t of active sets at time t for each company C^i , with $i \in \{1, 2, \dots, K\}$, we have to show that

$$Pr(\mathbf{S}_t = \mathbf{A}_t \mid \mathbf{S}_0 = \mathbf{A}_0, \dots, \mathbf{S}_{t-} = \mathbf{A}_{t-}) =$$

$$Pr(R_t^U(\mathbf{S}_{t-}) = \mathbf{A}_t \mid R_0^U(\mathbf{S}_0) = \mathbf{A}_0, \dots, R_{t-}^U(\mathbf{S}_{t-}) = \mathbf{A}_{t-})$$

Claim. The diffusion process defined by NPK-LT Model is equivalent to the reachability under NP-LE Model.

The rationale for the claim relies on the fact that the probability of reaching at time t a new node in the NP-LE Model (i.e. the probability of a given possible world) corresponds to the probability that an activation (or, equivalently, a switching of company) occurs at time t in the NPK-LT Model. This follows from the proof of equivalence between the LT diffusion process and the LT live-edge reachability given in [43]. However, there are some differences

we have to take into account. First of all, we need to consider the propagation delay of the influence. This feature is shared by both models. Indeed, the spreading times are sampled (thus known) every time a node is activated (eq. reachable). Moreover, a node can get deactivated for a company (thus activated for another). To this end, we will consider, separately, the events of activation of a node for a company and its deactivation. It is worth noting that, since we are assuming continuous time evolution, at most one activation can occur at a given time $t > 0$ and at most two companies will be affected by this change (the chosen one and the left one).

- **Activation Event.**

In the following, we will show that, given K disjoint sets of nodes, one for each company C^i , the probability of activation of a node for C^i at time t in the NPK-LT Model is equivalent to the probability that this node is added to the set of C^i reachable nodes at time t according to the NP-LE Model. More in detail, let t^- be the time in which the last event occurs prior to t . Then, the probability that a node v is activated for company C^i at time $t > t^-$ under the NPK-LT Model is similar to the probability defined in [48] where, instead of directly considering the set of active nodes at the last iteration, i.e. S_{t^-} , we consider only the node $u \in \mathbb{I}(v, t) \subseteq S_{t^-}$ actively influential for v , i.e. the active node that is able to propagate the influence at time t . Moreover, since we are in a non progressive scenario, a node v can become adopter at time t of a given company C^i regardless of whether it was active for another company at time $t' < t$. It implies that v 's activation probability for a given company C^i at time t under the NPK-LT Model is only related to the motivation of v in adopting a company at that time and to the probability that it will actually adopt C^i , i.e.

$$Pr [S_t^i = S_{t^-}^i \cup \{v\} \mid \mathbf{S}_{t^-}] = \begin{cases} \rho(v, t) \cdot b_{u,v} & \text{if } \exists u \in N^{in}(v) \text{ s.t. } u \in \mathbb{I}^i(v, t) \setminus \mathbb{I}(v, t^-) \\ 0 & \text{otherwise} \end{cases}$$

In this case, we have to consider the switching probability of node v that takes into account how interested is v in changing company at the specified time. We recall that the latter is equal to 1 when the node v has never been activated.

Regarding the NP-LE Model, the probability that a node v is reachable at time t from the set of C^i reachable nodes in the universe U corresponds to the existence of a live edge induced from $\mathbb{I}^i(v, t) \setminus \mathbb{I}(v, t^-)$, i.e.

$$Pr [R_t^X(S_{t^-}^i, \mathbf{S}_{t^-}) = S_{t^-}^i \cup \{v\}] = \begin{cases} \rho(v, t) \cdot b_{u,v} & \text{if } \exists u \in N^{in}(v) \text{ s.t. } u \in \mathbb{I}^i(v, t) \setminus \mathbb{I}(v, t^-) \\ 0 & \text{otherwise} \end{cases}$$

Due to the NP-LE Model construction, this probability is exactly the value obtained for the NPK-LT Model. Thus, by induction over the iterations of the NPK-LT Model and the live-edge process, we proved that the distributions produced are the same.

- **Deactivation or Switching Event.**

The probability of a node's deactivation follows the same distribution as well. In the NPK-LT Model, the deactivation event for a company C^i corresponds to the activation for another company C^j , with $j \neq i$. However, from the perspective of C^i , it is not relevant to know for which company the deactivated node will become active. Thus, the probability of this event is the following:

$$Pr [S_t^i = S_{t^-}^i \setminus \{v\} \mid \mathbf{S}_{t^-}] = \sum_{S_t^j \in \mathbf{S}_{t^-}^{-i}} Pr [S_t^j = S_{t^-}^j \cup \{v\}]$$

Equivalently, in the NP-LE Model, the deactivation event of a node v at time t for a company C^i corresponds to the reachability of v from nodes of another company C^j , with $j \neq i$. Thus, we obtain:

$$Pr [R_t^X(S_{t^-}^i, \mathbf{S}_{t^-}) = S_{t^-}^i \setminus \{v\}] = \sum_{S_t^j \in \mathbf{S}_{t^-}^{-i}} Pr [R_t^X(S_{t^-}^j, \mathbf{S}_{t^-}) = S_{t^-}^j \cup \{v\}]$$

As equivalence has been proven, we are going to show that, under the NPK-LT Model, the expected number of active nodes at time t for a company C^i , $\sigma^i(S^i, \mathbf{S}^{-i}, t)$, is monotone and submodular.

To this end, we express the expected number of active nodes $\sigma^i(S^i, \mathbf{S}^{-i}, t)$ at a specific time t as:

$$\sigma^i(S^i, \mathbf{S}^{-i}, t) = \sum_{\forall U} Pr(U) \cdot \left(\sum_{\forall \mathbf{X}_{t^-}} Pr(\mathbf{X}_{t^-}) \cdot |R_t^U(X_{t^-}^i, \mathbf{X}_{t^-})| \right)$$

where

- \mathbf{X}_{t^-} is a leaf node of the possible worlds tree pruned in correspondence of value of $t > t^-$, and
- $X_{t^-}^i \in \mathbf{X}_{t^-}$ represents the set of nodes that result active for the company C^i at time t^- for the possible world considered.

Thus $\sigma^i(S^i, \mathbf{S}^{-i}, t)$ is monotonic and submodular as it corresponds to a non-negative linear combination of monotonic and submodular functions. \square

Based on the above results, it is straightforward to prove the following corollary.

Corollary 5.8. *Function EAT^i is monotone and submodular.*

As definition, $\sigma_{np}^i(S^i, \mathbb{T}) = \int_{t=0}^{\mathbb{T}} \sigma^i(S^i, \mathbf{S}^{-i}, t) dt$. Thus EAT^i is also submodular as it corresponds to a non-negative linear combination of submodular functions.

5.4 Discussion

In this chapter, we presented an extension to the K-LT Model to accommodate continuous time diffusion in a non-progressive setting. Despite the adoption of a service/product is typically a non-progressive phenomena, extensive research has been done to define several progressive diffusion models, that rarely fit the real user behaviours. To overcome this limitation, we provide a non-progressive propagation model and we have shown that the spread under the proposed model satisfies the desired properties of monotonicity and submodularity. The implication of this result is that all the techniques developed for seed selection under the classical LT model, including greedy approximation using Monte Carlo simulation as well as various efficient high quality heuristics are now available for solving the non-progressive influence maximization problem. Indeed, it is worth noting that, while non-submodular diffusions behaviors often exist in reality, it is still widely open on how to deal with optimization problems beyond submodular function maximization techniques. Therefore, if submodularity does not hold, the above problem would be still quite open and we could only rely on heuristics without theoretical guarantee.

Clearly, there are still some challenges to solve. First of all, we described how influence propagates over the network, more precisely, how influence diffuses to a node from its neighbors thanks to some parameters (such as spreading times and influence weights associated to the edges). However, we cannot assume that the latter are given as input. Indeed, since real world social network data do not come with such informations, we have to learn the parameters of our influence propagation model from past observations, i.e. from any evidence of past influence among nodes. Despite there are many algorithms devoted to this task [32, 29], the real technical challenge is dealing with large-scale graphs. This is also the main drawbacks of the current solutions for the seed selection. Indeed, despite dramatic advances have been made in making the process of seed selection efficient, with heuristics offering a quality close to the theoretical approximation guarantee, scaling approximation algorithms up to a billion node network and learning efficiently model parameters are quite open problems that we are dealing with.

Conclusion and Future Directions

Recent years have witnessed a proliferation of applications dealing with a peculiar form of graph data as social networks. This huge amount of application lead to the design of ad hoc algorithms devoted to an efficient and effective management of these data. In this thesis, we focused on two categories of such algorithms, namely shortest distances maintenance and influence diffusion in a non progressive environment. Computing shortest distances is one of the most fundamental problems when managing social networks as the availability of accurate information on paths being created or canceled due to the network evolution is crucial in many real life scenarios. Although many algorithms have been proposed for this purpose, they are designed to work in main memory assuming static graphs. This assumption, significantly limits their applicability to many current applications where graphs are very large and frequently updated (e.g. Facebook graph). Indeed, it is prohibitive to keep all shortest distances in main memory and compute them from scratch every time as (even) small changes will cause a great overload for their calculation.

To overcome these limitations, we have proposed efficient algorithms for the incremental maintenance of all-pairs shortest distances for graphs stored in relational databases. Our algorithms significantly outperform the state-of-the-art algorithms designed for the same setting and can handle the insertion and deletion of multiple edges at once. The proposed algorithms might be easily extended to solve the APSP problem by keeping track of the edges used during the shortest distance computation (this might be achieved by using extended distance tuples). Another interesting direction for future work is to apply the ideas underlying our approach to the development of algorithms working on graph database systems [59].

As regards our extension to the K-LT Model for dealing with continuous time diffusion in a non-progressive setting, we point out the innovative nature of our work as, despite the adoption of a service/product is typically a non-progressive phenomena, extensive research has been done to define several progressive diffusion models, that rarely fit the real user behaviors. Thus, we defined a non-progressive propagation model and we showed that the spread

under the proposed model satisfies the desired properties of monotonicity and submodularity. This result is clearly a strong one, because all the techniques developed for seed selection under the classical LT model, including greedy approximation using Monte Carlo simulation as well as various efficient high quality heuristics are still adequate for solving the non-progressive influence maximization problem.

The intriguing features of this problem leave room for interesting follow up of this work. First of all, a big technical challenge is dealing with large-scale graphs. As a matter of fact, the basic greedy algorithm for selecting the seed set has a very high running time, that turns to be unaffordable for actual graphs. Moreover, we cannot assume that the parameters of our influence propagation model are given as an input (e.g. the influence weight), because social network data do not come with such information. In this respect, despite dramatic advances have been made in making the process of seed selection efficient by adopting heuristics whose theoretical approximation guarantee a satisfactory accuracy, scaling approximation algorithms up to a billion node network and learning efficiently model parameters are quite open problems that we plan to deal with.

Finally, it will be quite interesting to extend our proposal in order to consider the influence diffusion from the point of view of the owner of the social network, instead from the perspective of one of the competing companies. The latter could be exploited from the owner of the network to guarantee a fair seed set allocation to the clients running a campaign on it, offering viral marketing as a service, for a price.

References

- [1] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert Endre Tarjan. Faster algorithms for the shortest path problem. *J. ACM*, 37(2):213–223, 1990.
- [2] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD Conference*, pages 349–360, 2013.
- [3] Aris Anagnostopoulos, Ravi Kumar, and Mohammad Mahdian. Influence and correlation in social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 7–15. ACM, 2008.
- [4] Sinan Aral, Lev Muchnik, and Arun Sundararajan. Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. *Proceedings of the National Academy of Sciences*, 106(51):21544–21549, 2009.
- [5] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
- [6] Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Maintaining all-pairs approximate shortest paths under deletion of edges. In *SODA*, pages 394–403, 2003.
- [7] Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In *Internet and Network Economics, Third International Workshop, WINE 2007, San Diego, CA, USA, December 12-14, 2007, Proceedings*, pages 306–311, 2007.
- [8] Allan Borodin, Yuval Filmus, and Joel Oren. Threshold models for competitive influence in social networks. In *WINE*, pages 539–550, 2010.
- [9] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *WWW*, pages 665–674, 2011.
- [10] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *STOC*, pages 590–598, 2007.

- [11] Lijun Chang, Jeffrey Xu Yu, Lu Qin, Hong Cheng, and Miao Qiao. The exact distance to destination in undirected world. *VLDB J.*, 21(6):869–888, 2012.
- [12] Wei Chen, Laks V. S. Lakshmanan, and Carlos Castillo. *Information and Influence Propagation in Social Networks*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.
- [13] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 199–208, 2009.
- [14] James Cheng, Yiping Ke, Shumo Chu, and Carter Cheng. Efficient processing of distance queries in large graphs: a vertex cover approach. In *SIGMOD Conference*, pages 457–468, 2012.
- [15] Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, and Daniele Frigioni. Partially dynamic efficient algorithms for distributed shortest paths. *Theor. Comput. Sci.*, 411(7-9):1013–1037, 2010.
- [16] Serafino Cicerone, Gabriele Di Stefano, Daniele Frigioni, and Umberto Nanni. A fully dynamic algorithm for distributed shortest paths. *Theor. Comput. Sci.*, 297(1-3):83–102, 2003.
- [17] Tom Crecelius and Ralf Schenkel. Pay-as-you-go maintenance of pre-computed nearest neighbors in large graphs. In *CIKM*, pages 952–961, 2012.
- [18] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In *Algorithmics of Large and Complex Networks*, pages 117–139, 2009.
- [19] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004.
- [20] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
- [21] Nan Du, Le Song, Manuel Gomez-Rodriguez, and Hongyuan Zha. Scalable influence estimation in continuous-time diffusion networks. In *NIPS*, pages 3147–3155, 2013.
- [22] P. Erdős and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.
- [23] Eyal Even-Dar and Asaf Shapira. A note on maximizing the spread of influence in social networks. In *WINE*, pages 281–286, 2007.
- [24] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1):264–275, 2010.
- [25] MohammadAmin Fazli, Mohammad Ghodsi, Jafar Habibi, Pooya Jalaly Khalilabadi, Vahab S. Mirrokni, and Sina Sadeghian Sadeghabad. On the non-progressive spread of influence through social networks. In *LATIN*, pages 315–326, 2012.
- [26] L.C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.

- [27] Ada Wai-Chee Fu, Huanhuan Wu, James Cheng, and Raymond Chi-Wing Wong. Is-label: an independent-set based labeling scheme for point-to-point distance querying. *PVLDB*, 6(6):457–468, 2013.
- [28] Andrew V. Goldberg and Renato Fonseca F. Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.
- [29] Manuel Gomez-Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 561–568, 2011.
- [30] Manuel Gomez-Rodriguez and Bernhard Schölkopf. Influence maximization in continuous time diffusion networks. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.
- [31] Gang Gou and Rada Chirkova. Efficient algorithms for exact ranked twig-pattern matching over graphs. In *SIGMOD Conference*, pages 581–594, 2008.
- [32] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4-6, 2010*, pages 241–250, 2010.
- [33] Amit Goyal, Wei Lu, and Laks V. S. Lakshmanan. SIMPATH: an efficient algorithm for influence maximization under the linear threshold model. In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pages 211–220, 2011.
- [34] M.S. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973.
- [35] Daniel Gruhl, R. Guha, David Liben-Nowell, and Andrew Tomkins. Information diffusion through blogspace. In *Proceedings of the 13th International Conference on World Wide Web, WWW '04*, pages 491–501, 2004.
- [36] Andrey Gubichev, Srikanta J. Bedathur, Stephan Seufert, and Gerhard Weikum. Fast and accurate estimation of shortest paths in large graphs. In *CIKM*, pages 499–508, 2010.
- [37] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *SIGMOD Conference*, pages 157–166, 1993.
- [38] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012.*, pages 463–474, 2012.
- [39] Junming Huang, Xue-Qi Cheng, Hua-Wei Shen, Tao Zhou, and Xiaolong Jin. Exploring social influence via posterior effect of word-of-mouth recommendations. In *Proceedings of the Fifth ACM International Con-*

- ference on Web Search and Data Mining, WSDM '12, pages 573–582, 2012.
- [40] Giuseppe F. Italiano. Finding paths and deleting edges in directed acyclic graphs. *Inf. Proc. Lett.*, 28(1):5–11, 1988.
 - [41] Ruoming Jin, Ning Ruan, Yang Xiang, and Victor E. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *SIGMOD Conference*, pages 445–456, 2012.
 - [42] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977.
 - [43] David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.
 - [44] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *FOCS*, pages 81–91, 1999.
 - [45] Jan Küntzer, Christina Backes, Torsten Blum, Andreas Gerasch, Michael Kaufmann, Oliver Kohlbacher, and Hans-Peter Lenhof. Bndb - the biochemical network database. *BMC Bioinformatics*, 8, 2007.
 - [46] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 420–429, 2007.
 - [47] Vincent Yun Lou, Smriti Bhagat, Laks V. S. Lakshmanan, and Sharan Vaswani. Modeling non-progressive phenomena for influence propagation. *CoRR*, 2014.
 - [48] Wei Lu, Francesco Bonchi, Amit Goyal, and Laks V. S. Lakshmanan. The bang for the buck: fair competitive viral marketing from the host perspective. In *KDD*, pages 928–936, 2013.
 - [49] Alan Mislove, Massimiliano Marcon, P. Krishna Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 29–42, 2007.
 - [50] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review*, E 69, 2004.
 - [51] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010.
 - [52] Chaoyi Pang, Guozhu Dong, and Kotagiri Ramamohanarao. Incremental maintenance of shortest distance and transitive closure in first-order logic and SQL. *ACM Trans. Database Syst.*, 30(3):698–721, 2005.
 - [53] Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Physical Review Letters*, 86(14):3200–3203, 2001.
 - [54] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004.

- [55] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, pages 867–876, 2009.
- [56] Zichao Qi, Yanghua Xiao, Bin Shao, and Haixun Wang. Toward a distance oracle for billion-node graphs. *PVLDB*, 7(1):61–72, 2013.
- [57] Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. In *ICDE*, pages 462–473, 2012.
- [58] Syed Asad Rahman, P. Advani, R. Schunk, Rainer Schrader, and Dietmar Schomburg. Metabolic pathway analysis web service (pathway hunter tool at cubic). *Bioinformatics*, 21(7):1189–1193, 2005.
- [59] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases*. O’Reilly Media, Incorporated, 2013.
- [60] Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, and Rina Panigrahy. A sketch-based distance oracle for web-scale graphs. In *WSDM*, pages 401–410, 2010.
- [61] Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46:45:1–45:31, 2014.
- [62] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: near-optimal time complexity meets practical efficiency. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 75–86, 2014.
- [63] Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P. Gummadi. Modeling diffusion of competing products and conventions in social media. *CoRR*, 2014.
- [64] Chi Wang, Wei Chen, and Yajun Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Min. Knowl. Discov.*, 25(3):545–576, 2012.
- [65] D. J. Watts and S. H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393(6684):409–10, 1998.
- [66] Duncan J. Watts and Jonah Peretti. Viral Marketing for the Real World. *Harvard Business Review*, 2007.
- [67] Lingkun Wu, Xiaokui Xiao, Dingxiong Deng, Gao Cong, Andy Diwen Zhu, and Shuigeng Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *PVLDB*, 5(5):406–417, 2012.
- [68] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. *Keyword Search in Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.
- [69] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. Shortest path and distance queries on road networks: towards bridging theory and practice. In *SIGMOD Conference*, pages 857–868, 2013.
- [70] Andy Diwen Zhu, Xiaokui Xiao, Sibowang Wang, and Wenqing Lin. Efficient single-source shortest path and distance queries on large graphs. In *KDD*, pages 998–1006, 2013.