Mining Summaries of Propagations

Lucrezia Macchia
University "Aldo Moro"
Bari, Italy
lucrezia.macchia@uniba.it

Francesco Bonchi
Yahoo Labs
Barcelona, Spain
bonchi@yahoo-inc.com

Francesco Gullo
Yahoo Labs
Barcelona, Spain
gullo@yahoo-inc.com

Luca Chiarandini
Universitat Pompeu Fabra
Barcelona, Spain
chiarluc@yahoo-inc.com

***Abstract*—Analyzing the traces left by a meme of information propagating through a social network or by a user browsing a website can help to unveil the structure and dynamics of such complex networks. This may in turn open the door to concrete applications, such as finding influential users for a topic in a social network, or detecting the typical structure of a web browsing session that leads to a product purchase.**

In this paper we define the problem of mining summaries of propagations as a constrained pattern-mining problem. A propagation is a DAG where an entity (e.g., information exchanged in a social network, or a user browsing a website) flows following the underlying hierarchical structure of the nodes. A summary is a set of propagations that (i) involve a similar population of nodes, and (ii) exhibit a coherent hierarchical structure when merged altogether to form a single graph. The first constraint is defined based on the Jaccard coefficient, while the definition of the second one relies on the graph-theoretic concept of "agony" of a graph. It turns out that both constraints satisfy the downward-closure property, thus enabling Apriori-like algorithms. However, motivated by the fact that computing agony is much more expensive than computing Jaccard, we devise two algorithms that explore the search space differently. The first algorithm is an Apriori-like, bottom-up method that checks both the constraints level-by-level. The second algorithm consists of a first phase where the search space is pruned as much as possible by exploiting the Jaccard constraint only, while involving the second constraint only afterwards, in a subsequent phase.

We test our algorithms on four real-world datasets. Quantitative results reveal that the choice of the most efficient algorithm depends on the selectivity of the two constraints. Qualitative results show the relevance of the extracted summaries in a number of real-world scenarios.

I. Introduction

We study the novel data-mining problem of extracting informative summaries from a database recording activity sequences on a graph. As better explained next, ours is a general framework that can be instantiated in different contexts, ranging from information propagation in social networks to user browsing activity over the Web.

In our problem, we are given a database $\mathbb{D}$ of *propagations*, where each propagation represents the trace left by a specific entity ϕ that "flows" over an underlying graph $G = (V, E)$. More precisely, a trace of an entity ϕ is a sequence of observations $\langle v, \phi, t\rangle$ representing the fact that the entity ϕ is observed at node v at time t. The trace of ϕ can naturally be represented as a *directed acyclic graph* (DAG) D_ϕ, whose arcs (u, v) express the fact that the same arc exists in the underlying graph G and both u and v activate on ϕ, with u activating strictly before v. In this case, we can think that ϕ flowed from u to v. An example of our input is provided later, in Section II (and Figure 2).

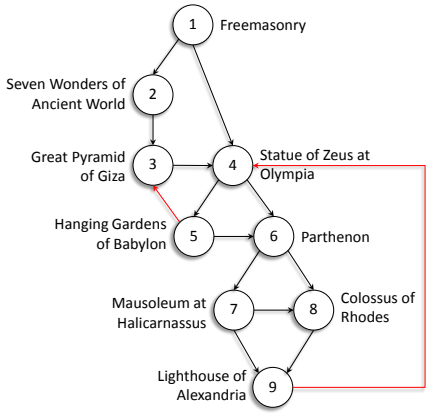


Fig. 1: *An example of summary extracted from a database of browsing sessions in Wikipedia (application scenario #2). The numbers inside the nodes identify one optimal ranking. Links that violate the ranking are depicted in red.*

Our goal is to *extract summaries of propagation traces* from $\mathbb{D}$. A summary consists of two parts: (i) a directed graph $G(S)$ (subgraph of G), and (ii) a ranking r of the nodes in $G(S)$. The graph $G(S)$ results from merging the DAGs corresponding to the propagations in the summary, while the ranking r is the expression of the hierarchical structure underlying $G(S)$. A major requirement that propagations in the same summary should satisfy is to be *structurally similar*. More precisely, we want for the propagations within a summary to involve (more or less) the same population of nodes.

This is however not enough: we also require for the propagations to exhibit a well-defined *hierarchical structure* when merged together. By hierarchical structure, we mean that one can identify a ranking, such that there exist nodes that usually participate early in the propagations inside the summary, and nodes that instead activate later. The ranking of nodes makes our summaries informative and useful in a wide and diverse range of applications, like the ones discussed next.

Application scenario #1. In a social network like Twitter the underlying graph G represents the social connections: the nodes of the graph are the users of the social network and an arc (u, v) exists if u and v are related in some way (e.g., v is a follower of u). Here the entity ϕ is a piece of information (e.g., the URL of an interesting blog, multimedia content, such as photos/videos, etc.) propagating

in the social networks by means of *re-tweets*. Given a database of propagations $\mathbb{D}$, finding summaries corresponds to finding groups of entities that propagate in the social network in a similar way. Nodes of these groups may represent different communities of users interested in different topics: politics-related entities flow trough a different set of nodes than entities related to electronic music. Moreover, a summary also comes with a ranking of nodes which reflects, to some extent, the directionality of the information flow in all the propagations in the summary. This is crucial to distinguish users that are "early adopters" (or "opinion leaders", or "trend setters", or "influencers") from users that are instead simple followers.

Application scenario #2. In another context, the graph G could be a (large) website (e.g., Amazon or Wikipedia), whose nodes and arcs correspond to web pages and hyperlinks between pages, respectively. In this case, the entity ϕ moving in the directed graph corresponds to (a browsing session of) a specific user visiting the website, and multiple users clearly leave different traces. The browsing behavior of a user within the website can be represented as a DAG, where the user moves from the homepage down in the hierarchical structure of the pages of the website. Cycles might arise because of user backtracking or because they are allowed by the website structure. However, as what really matters is the sequence of the pages visited, one can safely ignore cycles and simply consider only the first visit to each page.

Finding summaries, i.e., groups of users (sessions) that navigate the website in a similar fashion is essential for website-usage analysis and re-organization [1]. In fact, a summary indicates how a specific group of users access the website, which are the pages accessed first, and which access page is a good entry point to discover many other pages. The typical browsing activity of a group of users is thus described by its corresponding summary, and different groups of users can be detected and discerned based on their summaries. Moreover, the typical behavior of successful sessions (e.g., the ones ending in a product purchase) can easily be detected and studied in order to improve the website organization.

An example of summary extracted from a database of browsing sessions in Wikipedia is provided in Figure 1 (more details are given in Section IV) . The numbers inside the nodes correspond to one optimal ranking, while red links denote ranking violations. In the next section we will formalize the concept of violating the ranking, and based on that, we will define the constraint to be satisfied in order to guarantee good hierarchical structures.

Paper contributions. Our contributions are as follows:

- We formulate the novel problem of extracting structurally-similar summaries from a set of propagations. We translate the requirement about structural similarity into two constraints that the extracted summaries are required to satisfy. One constraint aims to force all the propagations in a summary to involve more or less the same set of nodes, while the second constraint requires for the nodes in the union graph of a summary to have a well-defined hierarchy.
- We show that both the constraints satisfy the downward-closure property, thus allowing the definition of Apriori-like methods.
- We devise two algorithms to solve our problem, which differ from each other by the way how they visit the lattice of all possible sets of propagations. The first algorithm, called BOTTOM-UP, relies on a bottom-up lattice-traversing strategy, which directly exploits the aforementioned closure properties. Motivated by the fact that checking the hierarchy constraint is more time-consuming, we develop a second algorithm, called UP&DOWN, which consists of two separate phases: a bottom-up phase where the hierarchy constraint is discarded, followed by a top-down phase that partially re-visits the lattice starting from those summaries that violate the hierarchy constraint.
- We extensively evaluate our algorithms on four real-world datasets coming from the application scenarios discussed above, i.e., information propagation on social networks and web browsing. Quantitative results show that the UP&DOWN algorithm generally achieves better efficiency, even though BOTTOM-UP can be faster in some settings. Qualitative results provide evidence about the significance of the summaries extracted and how they can be exploited for practical purposes.

Roadmap. The rest of the paper is organized as follows. In Section II we define our problem, while in Section III we describe the proposed algorithms to solve it. Section IV shows experiments. In Section V we briefly review related research, and, finally, Section VI concludes the paper.

II. Problem definition

The input to our problem is (i) a directed graph $G = (V, A)$ representing a network of interconnected objects, (ii) a set $\mathcal{E}$ of *entities*, and (iii) a set $\mathbb{O}$ of *observations* involving the objects of the network and the entities in $\mathcal{E}$. As mentioned in the Introduction, objects can be, e.g., users in a social network or pages of a website, while entities can be, e.g., pieces of information (such as multimedia content) shared by users or web-page visits. Each observation in $\mathbb{O}$ is a triple $\langle v, \phi, t\rangle$, where $v \in V$, $\phi \in \mathcal{E}$, and $t \in \mathbb{N}^+$, denoting that the entity ϕ is observed at node v at time t. We assume that the same entity cannot be observed multiple times at the same node; should this happen, we consider only the first one (in order of time) of such observations.

The set $\mathbb{O}$ of observations can alternatively be viewed as a database $\mathbb{D}$ of *propagation traces* (or simply *propagations*), i.e., traces left by entities that "flow" over G. Formally, a propagation trace of an entity ϕ corresponds to the subset of all observations in $\mathbb{O}$ involving that entity, i.e., $\{\langle v, \phi', t\rangle \in \mathbb{O} \mid \phi' = \phi\}$. Considering the graph G, the database of propagation traces corresponds to a set of *directed acyclic graphs* (DAGs) $\mathbb{D} = \{D_\phi \mid \phi \in \mathcal{E}\}$, where, for each $\phi \in \mathcal{E}$, $D_\phi = (V_\phi, A_\phi)$, $V_\phi = \{v \in V \mid \langle v, \phi, t\rangle \in \mathbb{O}\}$, $A_\phi = \{(u, v) \in A \mid \langle u, \phi, t_u\rangle \in \mathbb{O}, \langle v, \phi, t_v\rangle \in \mathbb{O}, t_u < t_v\}$. Note that each $D_\phi \in \mathbb{D}$ is guaranteed to contain no cycles

$\mathbb{D}$

v	ϕ	t
Ω	ϕ_1	0
v_2	ϕ_1	2
v_3	ϕ_1	4
v_4	ϕ_1	5
v_5	ϕ_1	7
Ω	ϕ_2	0
v_2	ϕ_2	1
v_1	ϕ_2	3
v_5	ϕ_2	6
v_7	ϕ_2	7
v_6	ϕ_2	8
v_3	ϕ_2	9
Ω	ϕ_3	0
v_1	ϕ_3	1
v_2	ϕ_3	3
v_6	ϕ_3	5
v_7	ϕ_3	7
v_4	ϕ_3	8

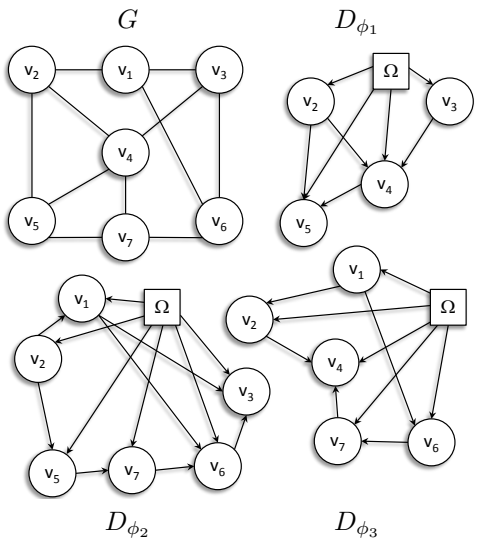


Fig. 2: *An example of the input of our problem: a graph G, and a database of propagation traces $\mathbb{D}$ defined over a set of entities $\mathcal{E} = \{\phi_1, \phi_2, \phi_3\}$. The graph is here represented undirected: each edge corresponds to the two directed arcs. Each propagation is started at time 0 by a dummy node $\Omega \notin V$. Given the graph G, the propagation database $\mathbb{D}$ is equivalent to the set of dags $\{D_{\phi_1}, D_{\phi_2}, D_{\phi_3}\}$.*

due to time irreversibility. Without any loss of generality, we hereinafter refer to $\mathbb{D}$ as a set of DAGs. Moreover, we assume that each propagation is started at time 0 by a dummy node $\Omega \notin V$, representing a source of information external to the network that is implicitly connected to all nodes in V. Thus, each DAG in $\mathbb{D}$ is assumed to contain such a dummy node Ω connected to all its "real" nodes. An example of our input is provided in Figure 2.

A *summary* $S \subseteq \mathbb{D}$ is a set of DAGs. Given a summary S, we denote by $G(S)$ the union graph of all the DAGs in S. The union of two graphs $G_1 = (V_1, A_1)$ and $G_2 = (V_2, A_2)$ is defined as $G_1 \cup G_2 = (V_1 \cup V_2, A_1 \cup A_2)$. An example of graph resulting by the union of two DAGs is given in Figure 3. Note that, even though S is a set of DAGs, $G(S)$ is not necessarily a DAG itself, as the union of multiple DAGs can clearly correspond to a cyclic graph.

As informally anticipated in the previous section, our goal is to extract summaries that (i) are homogeneous in terms of the population of nodes involved, and (ii) exhibit a good hierarchical structure, i.e., they are close as much as possible to a DAG structure when merged together to form a single graph. We next formalize these two concepts in two constraints that we require for our summaries to satisfy.

Homogenous population of nodes. To force our summaries to involve an homogeneous population of nodes, a natural choice is to quantify the amount of nodes common to all DAGs in a summary and require that it is no less than a certain threshold. In this work, we measure the fraction of nodes shared by a set of DAGs by means of the popular *Jaccard similarity coefficient*, which is one of the most used measures of similarity among sets of objects. Moreover, it has

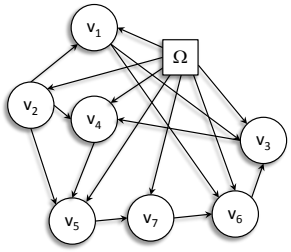


Fig. 3: *The union graph $D_{\phi_1} \cup D_{\phi_2}$ of the* DAGs *D_{ϕ_1} and D_{ϕ_2} depicted in Figure 2.*

the desirable property of having a fixed-range codomain, $[0, 1]$, which makes the threshold-setting task easier. Formally, given a summary S, we define

$$j(S) = \frac{|\bigcap_{D_\phi \in S} V_\phi|}{|\bigcup_{D_\phi \in S} V_\phi|}.$$

Hierarchical structure. Here we borrow the concept of "agony" introduced by Gupte et al. [2] to define a measure of the hierarchy existing in a directed graph. Given a directed graph $G = (V, A)$, consider a ranking function $r : V \to \mathbb{N}$ for the nodes in G, such that $r(u) < r(v)$ expresses the fact that u is "higher" in the hierarchy than v, i.e., the smaller $r(u)$ is, the more u is an "early-adopter". If $r(u) < r(v)$, then the arc $u \to v$ is expected and does not cause any "agony". Instead, if $r(u) \geq r(v)$ the arc $u \to v$ would cause agony because it would mean that u has a follower v (in the social graph terminology) that is higher-ranked than u itself. Therefore, given a graph G and a ranking r, the agony of each arc (u, v) is defined as $\max\{r(u) - r(v) + 1, 0\}$, and the agony $a(G, r)$ of the whole graph given the ranking r is just the sum over all arcs:

$$a(G, r) = \sum_{(u,v) \in A} \max\{r(u) - r(v) + 1, 0\}.$$

In most cases (as in our problem), the ranking r is not explicitly provided. The objective therefore becomes finding a ranking (they might be multiple) that minimizes the total agony of the graph. This way, one can compute the agony of any graph G as

$$a(G) = \min_r a(G, r).$$

As a DAG implicitly induces a partial order over its nodes, it has always zero agony: the nodes of a DAG form a perfect hierarchy. For instance, in the DAGs $D_{\phi_1}, D_{\phi_2}, D_{\phi_3}$ in Figure 2, it is sufficient to take the temporal ordering as ranking, i.e., $r(u) = t_u$ where $\langle u, \phi_i, t_u \rangle \in D_{\phi_i}$, in order to obtain agony equal to zero.

However, as already mentioned above, merging several DAGs to form a summary S leads to a union graph $G(S)$ that is not necessarily a DAG, therefore agony can appear. Consider for instance the union graph $D_{\phi_1} \cup D_{\phi_2}$ reported in Figure 3. It is easy to see that the graph $D_{\phi_1} \cup D_{\phi_2}$ is not a DAG as it contains the cycle $v_3 \to v_4 \to v_5 \to v_7 \to v_6 \to v_3$. Due

to this cycle, it is impossible to find a ranking r that provides zero agony. In fact, any directed cycle containing k arcs (and not sharing arcs with any other cycle) always incurs agony equal to k [2]. One ranking r providing the minimum agony for $D_{\phi_1} \cup D_{\phi_2}$ is:
$(\Omega : 0)(v_2 : 1)(v_1 : 2)(v_4 : 2)(v_5 : 3)(v_7 : 4)(v_6 : 5)(v_3 : 6)$. This ranking yields no agony on all the arcs, except on the arc $v_3 \rightarrow v_4$ that incurs agony 6-2+1 = 5, which is indeed the length of that directed cycle.

Although the number of possible rankings of a directed graph is exponential, Gupte et al. provide a polynomial-time algorithm for finding a ranking of minimum agony. They provide a linear-programming formulation and show that (i) the dual problem has an optimal integral solution, and (ii) the optimal value obtained by maximizing the dual problem coincides with the minimum value of the primal. This finding allows to define an algorithm that decomposes the input graph G into a DAG D and a graph H that corresponds to the maximum (in terms of number of arcs) Eulerian subgraph of G (an Eulerian graph is a graph in which the indegree of each node is equal to its outdegree). Let m be the number of arcs and n the number of nodes of G, then the algorithm to compute such a decomposition takes $\mathcal{O}(m^2 n)$ time: it requires at each iteration to find a negative-weight cycle, which can be done by the Bellman-Ford algorithm in $\mathcal{O}(mn)$, while, in the worst case, the number of iterations is m.

Mining maximal summaries under constraints. We have now all the ingredients to define the problem we study in this paper. Informally, given a database of DAGs $\mathbb{D}$, we want to extract sets $S \subseteq \mathbb{D}$ of DAGs that have high Jaccard of nodes (no less than a threshold β), and small agony of the graph $G(S)$ obtained by merging the DAGs in S (no more than a threshold α). Moreover, we want S to be maximal and have non-trivial size (i.e., $|S| \geq 2$). The formal statement of the problem we tackle in this work is reported next.

Problem 1 (Mining summaries of propagations): Given a set of dags $\mathbb{D}$, and two thresholds, $\alpha \in \mathbb{N}$ and $\beta \in [0, 1]$, we want to extract

$$\mathbb{S} = \{S \subseteq \mathbb{D} \mid |S| \geq 2, a(G(S)) \leq \alpha, j(S) \geq \beta, \nexists T \in \mathbb{S} : S \subset T\}.$$

For each summary $S \in \mathbb{S}$ we output the pair $\langle G(S), r^* \rangle$ where r^* is one ranking providing minimal agony for $G(S)$, that is $r^* = \arg\min_r a(G(S), r)$.

III. Algorithms

The search space of Problem 1 corresponds to the whole lattice $2^{\mathbb{D}}$ of all subsets of $\mathbb{D}$. The two constraints that are part of our problem definition hold the downward-closure property, which enables effective pruning of such a large search space, as explained next.

The Jaccard value is monotonically non-increasing as the number of sets to be compared increases. More precisely, given any two sets (of sets) $\mathcal{S}$ and $\mathcal{T}$, with $\mathcal{T} \supset \mathcal{S}$, it holds that $j(\mathcal{S}) \geq j(\mathcal{T})$. For our purposes, this result can easily be translated into the following pruning rule to be exploited during any (bottom-up) traversal of the lattice $2^{\mathbb{D}}$.

Fact 1: Given a set of DAGs $\mathbb{D}$, a subset $S \subseteq \mathbb{D}$, and a threshold $\beta \in [0, 1]$, if $j(S) < \beta$, then $j(T) < \beta$ for all $T \in 2^{\mathbb{D}} : T \supset S$.

The second property we show is that the agony is monotonically non-decreasing with the increasing of the size of the graph. We formally state such a result in the following theorem.

Theorem 1 (Agony is monotone): Given a directed graph $G = (V, A)$ and an arc $(u, v) \notin A$, let $G_{uv} = (V \cup \{u, v\}, A \cup \{(u, v)\})$ denote the graph deriving from adding the arc (u, v) to G. It holds that $a(G) \leq a(G_{uv})$.

Proof: As shown in [2], the problem of minimizing the agony of a graph G can be formulated as a linear program whose dual problem corresponds to finding an Eulerian subgraph of G having the maximum number of arcs. Let $E(G)$ denote the maximum Eulerian subgraph of G and let $|E(G)|$ denote the number of arcs in $E(G)$. Another result reported in [2] is that $|E(G)| = a(G)$. Now, it is easy to see that, although not necessarily optimal, the subgraph $E(G)$ represents at least an admissible solution (thus a lower bound) of the maximum-Eulerian-subgraph problem when the graph in input is G_{uv}. Thus, combining all such results:

$$a(G) = |E(G)| \leq |E(G_{uv})| = a(G_{uv}),$$

which proves the theorem. ∎

An immediate corollary of the above theorem we exploit in our context is the following: if for any subset of DAGs $S \subseteq \mathbb{D}$ it holds that the agony $a(G(S))$ of the corresponding union graph exceeds a certain threshold α, then the agony $a(G(T))$ of (the union graph of) every superset $T \supset S$ is guaranteed to be greater than α as well. An opposite result clearly holds for the subsets of a summary S whose agony is no less than α.

Corollary 1: Given a set of DAGs $\mathbb{D}$, a subset $S \subseteq \mathbb{D}$, and a threshold $\alpha \in \mathbb{N}$, it holds that:

1) *if $a(G(S)) > \alpha$, then $a(G(T)) > \alpha$ for all $T \in 2^{\mathbb{D}} : T \supset S$;*
2) *if $a(G(S)) \leq \alpha$, then $a(G(T)) \leq \alpha$ for all $T \in 2^{\mathbb{D}} : T \subset S$.*

Based on these properties we devise two algorithms, namely BOTTOM-UP and UP&DOWN, which explore the lattice $2^{\mathbb{D}}$ of all subsets of $\mathbb{D}$ in two different ways: the BOTTOM-UP algorithm performs a bottom-up, breadth-first visit, while the UP&DOWN algorithm consists of two phases: a first phase roughly similar to the BOTTOM-UP algorithm where only the Jaccard constraint is considered, followed by a top-down phase where the lattice is partially re-visited to check the agony constraint.

In the remainder of the section, we provide the details of the two algorithms, while also discussing the advantages and disadvantages of both, especially in terms of time/space requirements.

Algorithm 1 BOTTOM-UP

Input: set of DAGs $\mathbb{D}$, thresholds $\alpha \in \mathbb{N}$, $\beta \in [0, 1]$
Output: set $\mathbb{S}$ of all maximal subsets of $\mathbb{D}$ such that $|S| \geq 2$, $a(G(S)) \leq \alpha$, and $j(S) \geq \beta$, for all $S \in \mathbb{S}$

```
1: 𝕊 ← ∅,  ℂ ← {{D} | D ∈ 𝔻}
2: while ℂ ≠ ∅ do
3:    ℂ' ← generateCandidates(ℂ)
4:    ℂ ← ∅
5:    for all C ∈ ℂ' do
6:       if j(C) ≥ β then
7:          if a(G(C)) ≤ α then
8:             ℂ ← ℂ ∪ {C}
9:             𝕊 ← 𝕊 \ {S ∈ 𝕊 | S ⊂ C} ∪ {C}
10:         end if
11:      end if
12:   end for
13: end while
```

A. The BOTTOM-UP algorithm

We describe here the first algorithm we devise to solve Problem 1. The pseudocode of the proposed algorithm, called BOTTOM-UP, is summarized in Algorithm 1.

BOTTOM-UP resembles the classic Apriori algorithm for frequent-itemset mining [3]. Given a set of DAGs $\mathbb{D}$ and two thresholds α, β, the proposed algorithm visits the lattice of all possible subsets of $\mathbb{D}$ in a bottom-up fashion (this motivates the name of the algorithm). Particularly, the algorithm performs a breadth-first visit, starting from the subsets of $\mathbb{D}$ of size 2 (level 2), and increasing the level one-by-one until no summaries satisfying the requirements can be extracted. This way, the pruning rules stated in Fact 1 and (the first statement of) Corollary 1 can easily be exploited at each level: whenever a candidate C (i.e., a subset of $\mathbb{D}$) does not satisfy the constraints about either the Jaccard threshold β or the agony threshold α, it is removed from the candidate set so to skip the visit of all its supersets. Particularly, as computing the agony of a candidate C is much more time-consuming than computing Jaccard (i.e., $\mathcal{O}(m^2n)$ vs. $\mathcal{O}(n)$, where n and m are the number of nodes and arcs in the union graph $G(C)$, respectively), the first constraint checked by the algorithm is the one concerning Jaccard (Line 6). Only if this constraint is not violated, then the algorithm proceeds with computing the agony and checking whether its value is within the corresponding threshold (Line 7).

The $generateCandidates$ procedure invoked in Line 3 aims to derive the set of candidates to be processed in the next level $i + 1$ from the set of potential candidates $\mathbb{C}'$ that have passed the tests about the threshold α and β at level i. The procedure essentially performs a classic Apriori-like join step. Each pair of potential candidates C_1, C_2 in $\mathbb{C}'$ sharing a common prefix of length $i-1$ (i.e., for which $|C_1 \cap C_2| = i-1$) is merged to form the set $C_{12} = C_1 \cup C_2$ of size $i + 1$; such a set C_{12} will be then included in the being formed candidate set $\mathbb{C}'$ only if all its subsets of size i are present in $\mathbb{C}$.

In order to further speed-up the execution, the $generateCandidates$ procedure exploits the following simple result about Jaccard: given any two sets S_1, S_2, with $|S_1| \leq |S_2|$, the Jaccard value $j(\{S_1, S_2\})$ between such sets is upper-bounded by the size of the smaller-sized set divided by the size of the larger-sized one, i.e., $j(\{S_1, S_2\}) \leq \frac{|S_1|}{|S_2|}$. Hence, a further test is performed on a candidate $C_{12} = C_1 \cup C_2$ that has passed all the tests required by the Apriori-like join phase: C_{12} is actually included in the final set $\mathbb{C}'$ only if the above constant-time-computable upper bound on Jaccard is no less than the threshold β.

B. The UP&DOWN algorithm

A key advantage of the BOTTOM-UP algorithm is the capability of profitably exploiting the pruning rules stated in Fact 1 and Corollary 1, which allow a smart yet efficient visit of the lattice. Moreover, due to its closeness to the classic frequent-itemset-mining Apriori algorithm, it is rather simple and easy-to-implement. Nevertheless, BOTTOM-UP still suffers from a couple of major limitations:

1) The efficiency bottleneck of the algorithm is the computation of the agony, which, as said above, takes $\mathcal{O}(m^2n)$ time. Even though the BOTTOM-UP algorithm tries to minimize the number of agony computations by first checking the less expensive Jaccard constraint, unneeded agony computations may still arise. Indeed, each maximal summary S satisfies the property that all its subsets do not exceed the agony threshold (Corollary 1, Statement 2)); this means that, for each of such maximal summaries S, all subsets of S have necessarily been involved into an agony computation in previous iterations that resulted in no pruning. For a bottom-up visit there is no way to avoid that. An idea could be to compute agony incrementally: however this is not even a viable solution, as even adding a single new arc might force the algorithm for agony computation to perform a number of operations comparable to re-doing the whole computation from scratch [2].
2) The space complexity of BOTTOM-UP can be high, as the algorithm needs to keep in memory the union graph of all candidates of the current level to allow agony computations. On the other hand, the solution of loading at runtime the union graph of the various candidates would slow down the algorithm too much.

Within this view, we devise here a second algorithm to overcome the above issues. We call it UP&DOWN, as it performs a two-step traversal of the lattice $2^{\mathbb{D}}$ in which a bottom-up phase is followed by a top-down phase. The outline of UP&DOWN is reported as Algorithm 2.

The first step of UP&DOWN (Lines 2-11) is a bottom-up visit similar to the BOTTOM-UP algorithm, but here performed considering only the Jaccard constraint (thus completely discarding agony). The aim of this step is to find the set $\mathbb{S}_j$ of all maximal summaries that satisfy Jaccard. Among these summaries, there will clearly be some that do not satisfy agony; such summaries are collected into the set $\mathbb{S}_{\neg a}$ (Line 12). The second step of the algorithm (Lines 15-24) restarts from these summaries that violate the agony constraint and performs a top-down visit of the lattice aimed at discovering

Algorithm 2 UP&DOWN

Input: set of DAGs $\mathbb{D}$, thresholds $\alpha \in \mathbb{N}$, $\beta \in [0,1]$
Output: set $\mathbb{S}$ of all maximal subsets of $\mathbb{D}$ such that $|S| \geq 2$, $a(G(S)) \leq \alpha$, and $j(S) \geq \beta$, for all $S \in \mathbb{S}$

1: $\mathbb{S}_j \leftarrow \emptyset$, $\mathbb{C} \leftarrow \{\{D\} \mid D \in \mathbb{D}\}$
2: **while** $\mathbb{C} \neq \emptyset$ **do**
3: $\quad \mathbb{C}' \leftarrow generateCandidates(\mathbb{C})$
4: $\quad \mathbb{C} \leftarrow \emptyset$
5: $\quad$ **for all** $C \in \mathbb{C}'$ **do**
6: $\quad\quad$ **if** $j(C) \geq \beta$ **then**
7: $\quad\quad\quad \mathbb{C} \leftarrow \mathbb{C} \cup \{C\}$
8: $\quad\quad\quad \mathbb{S}_j \leftarrow \mathbb{S}_j \setminus \{S \in \mathbb{S}_j \mid S \subset C\} \cup \{C\}$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: **end while**
12: $\mathbb{S}_{\neg a} \leftarrow \{S \in \mathbb{S}_j \mid a(G(S)) > \alpha\}$
13: $\mathbb{S} \leftarrow \mathbb{S}_j \setminus \mathbb{S}_{\neg a}$
14: $M \leftarrow \max_{S \in \mathbb{S}_{\neg a}} |S|$
15: **for** $i = M, M-1, \ldots, 2$ **do**
16: $\quad \mathbb{S}_{\neg a}^{(i)} \leftarrow \{C \in \mathbb{S}_{\neg a} \mid |C| = i\}$
17: $\quad$ **for all** $C \in \mathbb{S}_{\neg a}^{(i)}$ **do**
18: $\quad\quad$ **if** $a(G(C)) \leq \alpha$ **then**
19: $\quad\quad\quad \mathbb{S} \leftarrow \mathbb{S} \cup \{C\}$
20: $\quad\quad$ **else**
21: $\quad\quad\quad \mathbb{S}_{\neg a} \leftarrow \mathbb{S}_{\neg a} \cup \{C' \in \mathbb{S}_j \mid C' \subset C, |C'| = i-1, \nexists S \in \mathbb{S} : S \supset C'\}$
22: $\quad\quad$ **end if**
23: $\quad$ **end for**
24: **end for**

the summaries whose agony is instead within the threshold α. The top-down visit is performed in a breadth-first fashion (like the bottom-up counterpart), starting from level $i = M$, where M denotes the maximum size of a summary in $\mathbb{S}_{\neg a}$ (in general $\mathbb{S}_{\neg a}$ may in fact contain summaries of different size). For each level i, the algorithm computes the agony of all candidate summaries in $\mathbb{S}_{\neg a}^{(i)}$, which denotes the set of all summaries in $\mathbb{S}_{\neg a}$ of size i (Line 18). Each candidate $C \in \mathbb{S}_{\neg a}^{(i)}$ satisfying the agony constraint is added to the solution set $\mathbb{S}$ (Line 19): indeed, according to (the second statement of) Corollary 1, all subsets of C are guaranteed to satisfy agony as well, then no backtracking is further needed from C. If the agony constraint is violated, the candidate C in $\mathbb{S}_{\neg a}^{(i)}$ is processed so to add to the candidate set to be considered in the next iteration all $(i-1)$-sized subsets of C that do not have any superset in the current solution set $\mathbb{S}$ (Line 21).

C. Comparing BOTTOM-UP and UP&DOWN

The main advantage of the two-step lattice traversal of the UP&DOWN algorithm is that, in most cases, it significantly reduces both the total number of agony computations and the space complexity, thus offering a valid solution to the issues of the BOTTOM-UP algorithm discussed at the beginning of Section III-B. Indeed, the bottom-up phase of UP&DOWN completely ignores the agony constraint, whose computation, as said, constitutes a bottleneck in terms of both time and space. The agony constraint is considered only in the subsequent top-down phase, where, however, the number of agony computations is expected to be less than the agony computations performed by a purely bottom-up strategy. For a better explanation, let us consider the following example.

Example 1: Let $\mathbb{D} = \{A, B, C, D, E, F, G, H, I, J, K, L\}$ *and assume that the bottom-up phase of the* UP&DOWN *algorithm produces the set* $\mathbb{S}_j = \{ABCD, EFGH, IJKL\}$ *(where* $ABCD$ *is a shorthand for* $\{A, B, C, D\}$*). Assume also that, among the summaries in* $\mathbb{S}_j$*,* $IJKL$ *violates the agony constraint, while* $ABCD$ *and* $EFGH$ *do not, i.e., assume that* $\mathbb{S}_{\neg a} = \{IJKL\}$*.*

For the summaries in $\mathbb{S}_j \setminus \mathbb{S}_{\neg a}$ *(i.e.,* $ABCD$ *and* $EFGH$*), the speed-up achieved by the* UP&DOWN *algorithm with respect to* BOTTOM-UP *is guaranteed and quite evident: for each of such summaries,* UP&DOWN *computes agony only once, while* BOTTOM-UP *would perform a number of agony computations proportional to the number of subsets of that summary (i.e., 11 agony computations for a* 4*-sized summary like the ones in our* $\mathbb{S}_j \setminus \mathbb{S}_{\neg a}$*). Also the space requirements of* UP&DOWN *are significantly less, as, for the bottom-up phase,* UP&DOWN *needs to keep in memory only the set of nodes of each candidate summary (because only the node set is needed to compute Jaccard), unlike the* BOTTOM-UP *algorithm that requires in memory the entire graph* $G(C)$ *for each candidate summary* C *(needed for computing agony).*

Concerning the summaries in $\mathbb{S}_{\neg a}$*, instead, the speed-up and the memory saving achieved by* UP&DOWN *depend on how further the top-down phase needs to go. For instance, assume that the actual summaries that satisfy agony are* $\{JKL, IKL, IJK\}$*. This way, the top-down phase of* UP&DOWN *would last only one level, thus still guaranteeing a speed-up and memory saving with respect to* BOTTOM-UP*. But if, as another example, the actual set of summaries satisfying agony is instead the singleton* $\{IJ\}$*, the* UP&DOWN *algorithm would need to visit a large portion of the lattice under* $IJKL$ *before encountering the set* IJ*, whereas* BOTTOM-UP *would have processed* IJ *quite soon.*

According to the above reasoning, the main conclusion that may be drawn here is that UP&DOWN guarantees better efficiency and smaller space complexity than BOTTOM-UP in most of the cases. Nevertheless, the BOTTOM-UP algorithm still remains preferable in cases where the time (and space) spent by UP&DOWN in the top-down phase is predominant. This may happen, e.g., when small agony thresholds α and/or large Jaccard thresholds β are involved. As we will show in Section IV, this conclusion is also confirmed by experimental evidence.

IV. EXPERIMENTAL EVALUATION

We provide here experimental evidence of the performance of our BOTTOM-UP and UP&DOWN algorithms. We experiment with four real-world datasets, whose main characteristics are summarized in Table I. Three datasets (i.e., Twitter, Last.fm, and Flixster) come from the domain of information propagation in a social network (application scenario #1 in the Introduction), while the remaining one (i.e., Wikipedia) concerns the web-browsing domain (application scenario #2 in the Introduction).

TABLE I: *Characteristics of the datasets used in the experiments: number of observations ($|\mathbb{O}|$); number of propagations/*DAGs *($|\mathbb{D}|$); nodes ($|V|$) and arcs ($|A|$) in the input graph G; min, max, and avg number of nodes in a* DAG *in $\mathbb{D}$ (n_{min}, n_{max}, n_{avg}); min, max, and avg number of arcs in a* DAG *in $\mathbb{D}$ (m_{min}, m_{max}, m_{avg}).*

	$\lvert\mathbb{O}\rvert$	$\lvert\mathbb{D}\rvert$	$\lvert V\rvert$	$\lvert A\rvert$	n_{min}	n_{max}	n_{avg}	m_{min}	m_{max}	m_{avg}
Twitter	580 141	8 888	28 185	1 636 451	12	13 547	66	11	240 153	347
Last.fm	1 208 640	51 495	1 372	14 708	6	472	24	5	2 704	39
Flixster	6 529 012	11 659	29 357	425 228	14	16 129	561	13	85 165	1 561
Wikipedia	50 092	5 477	39 756	46 610	4	125	9	4	131	9

Twitter (twitter.com). We obtained the dataset by crawling the public timeline of the popular online microblogging service. The nodes of the graph G are the Twitter users, while each arc (u, v) expresses the fact that v is a follower of u. The entities in $\mathcal{E}$ correspond to URLs, while an observation $\langle u, \phi, t\rangle \in \mathbb{O}$ means that the user u (re-)tweets (for the first time) the URL ϕ at time t.

Last.fm (www.last.fm). Last.fm is a music website, where users listen to their favorite tracks and communicate with each other. The dataset was created starting from the *HetRec 2011 Workshop* dataset available at www.grouplens.org/node/462, and enriching it by crawling. The graph G corresponds to the friendship network of the service. The entities in $\mathcal{E}$ are the songs listened by the users. An observation $\langle u, \phi, t\rangle \in \mathbb{O}$ means that the first time that the user u listens to the song ϕ happens at time t.

Flixster (www.flixster.com). Flixster is a social movie site where people can meet each other based on tastes in movies. The graph G corresponds to the social network underlying the site. The entities in $\mathcal{E}$ are movies, and an observation $\langle u, \phi, t\rangle$ is included in $\mathbb{O}$ when the user u rates for the first time the movie ϕ with the rating happening at time t.

Wikipedia (www.wikipedia.com). We created this dataset by looking at the browsing sessions of the popular online encyclopedia. Any node of the graph G corresponds to a Wikipedia page, while an arc (u, v) is present in G if there exists a browsing session where the page v has been reached from the page u, even making use of intermediate pages external to the website. An entity $\phi \in \mathcal{E}$ corresponds to a browsing session. An observation $\langle u, \phi, t\rangle \in \mathbb{O}$ means that the page u is visited during the session ϕ at time t. For each page, we consider only the first visit among the multiple ones possibly performed within the same session.

In the following we discuss the results achieved by the proposed algorithms from both a quantitative (Section IV-A) and a qualitative (Section IV-B) viewpoint. We use Twitter and Last.fm mainly for quantitative evaluation, while we resort to Wikipedia and Flixster for qualitative evaluation.

All algorithms are implemented in JAVA, and all experiments are performed on a single machine with Intel Xeon CPU at 2.20GHz and 48GB RAM.

A. Quantitative evaluation

We report here quantitative results achieved by our BOTTOM-UP and UP&DOWN algorithms on Twitter and Last.fm. We test our algorithms with Jaccard thresholds $\beta \in [0.7, 0.9]$ and agony thresholds $\alpha \in [10, 60]$ (Twitter) or $\alpha \in [3, 10]$ (Last.fm).

TABLE II: *Number of maximal summaries extracted from Twitter (left) and Last.fm (right).*

	α					
β	10	20	30	40	50	60
0.7	515	646	537	416	410	411
0.8	121	128	128	120	116	116
0.9	44	51	51	47	45	45

	α			
	3	5	7	10
	12 208	12 292	12 306	12 293
	5 126	5 136	5 128	5 132
	1 895	1 903	1 900	1 902

TABLE III: *Max and avg size (i.e., number of* DAG*s) of the maximal summaries extracted from Twitter (top) and Last.fm (bottom).*

	α					
β	10	20	30	40	50	60
0.7	8 (3.3)	9 (4.4)	11 (4.6)	12 (3.9)	14 (4.1)	14 (4.1)
0.8	5 (2.4)	9 (2.7)	10 (2.8)	12 (2.9)	13 (2.7)	13 (2.7)
0.9	5 (2.4)	8 (2.7)	8 (2.8)	10 (2.7)	11 (2.7)	11 (2.7)

	α			
β	3	5	7	10
0.7	13 (3.3)	13 (3.3)	13 (3.3)	13 (3.3)
0.8	13 (2.9)	13 (2.9)	13 (3.4)	13 (2.9)
0.9	11 (2.5)	11 (2.5)	11 (2.5)	11 (2.5)

General characterization. Tables II–IV show general statistics about the summaries extracted. In particular, Table II reports on the number of maximal summaries, while the remaining tables show statistics about the size of the summaries: maximum and average number of DAGs in a summary (Table III) and maximum and average number of nodes and arcs in the union graph of a summary (Table IV).

As expected, the size of the summaries increases as the agony threshold α increases and/or the Jaccard threshold β decreases (Tables III–IV), because this corresponds to less selective constraints. As far as the number of summaries (Table II), this is not necessarily true, because it may happen that, for a less restrictive constraint, the number of maximal summaries is less but the summaries include a larger number of DAGs.

Efficiency evaluation. The running times of our algorithms are shown in Figures 4 and 5. Particularly, in Figure 4 we show the results with varying the Jaccard threshold β while keeping fixed the agony threshold α; in Figure 5, instead, we keep β fixed and show the times with varying α.

Figure 4 clearly shows that the running times of both BOTTOM-UP and UP&DOWN are decreasing as the Jaccard threshold β increases: this is expected as the larger β, the smaller the number of candidates to be processed (larger β denotes a more restrictive constraint). The two algorithms instead differ from each other when looking at the behavior with

TABLE IV: *Max/avg number of nodes/arcs of the union graph of the maximal summaries extracted from* Twitter *(top) and* Last.fm *(bottom).*

	nodes						*arcs*					
	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$	$\alpha = 40$	$\alpha = 50$	$\alpha = 60$	$\alpha = 10$	$\alpha = 20$	$\alpha = 30$	$\alpha = 40$	$\alpha = 50$	$\alpha = 60$
$\beta = 0.7$	2 252(34)	2 252(33)	2 252(35)	2 252(37)	2 252(37)	2 252(37)	13 003(129)	13 003(137)	13 003(141)	13 003(142)	13 003(142)	13 003(142)
$\beta = 0.8$	2 252(55)	2 252(52)	2 252(53)	2 252(55)	2 252(57)	2 252(57)	13 003(221)	13 003(220)	13 003(224)	13 003(233)	13 003(237)	13 003(237)
$\beta = 0.9$	2 185(73)	2 185(65)	2 185(65)	2 185(70)	2 185(72)	2 185(72)	6 771(214)	6 771(201)	6 771(199)	6 771(208)	6 771(212)	6 771(212)

	nodes				*arcs*			
	$\alpha = 3$	$\alpha = 5$	$\alpha = 7$	$\alpha = 10$	$\alpha = 3$	$\alpha = 5$	$\alpha = 7$	$\alpha = 10$
$\beta = 0.7$	483 (22)	493 (22)	493 (22)	496 (22)	2 904 (35)	2 960 (36)	2 990 (37)	3 019 (38)
$\beta = 0.8$	376 (20)	376 (20)	394 (19)	407 (19)	1 998 (32)	1 998 (32)	2 126 (31)	2 221 (30)
$\beta = 0.9$	333 (17)	333 (17)	333 (17)	333 (17)	1 621 (22)	1 621 (22)	1 621 (22)	1 621 (22)

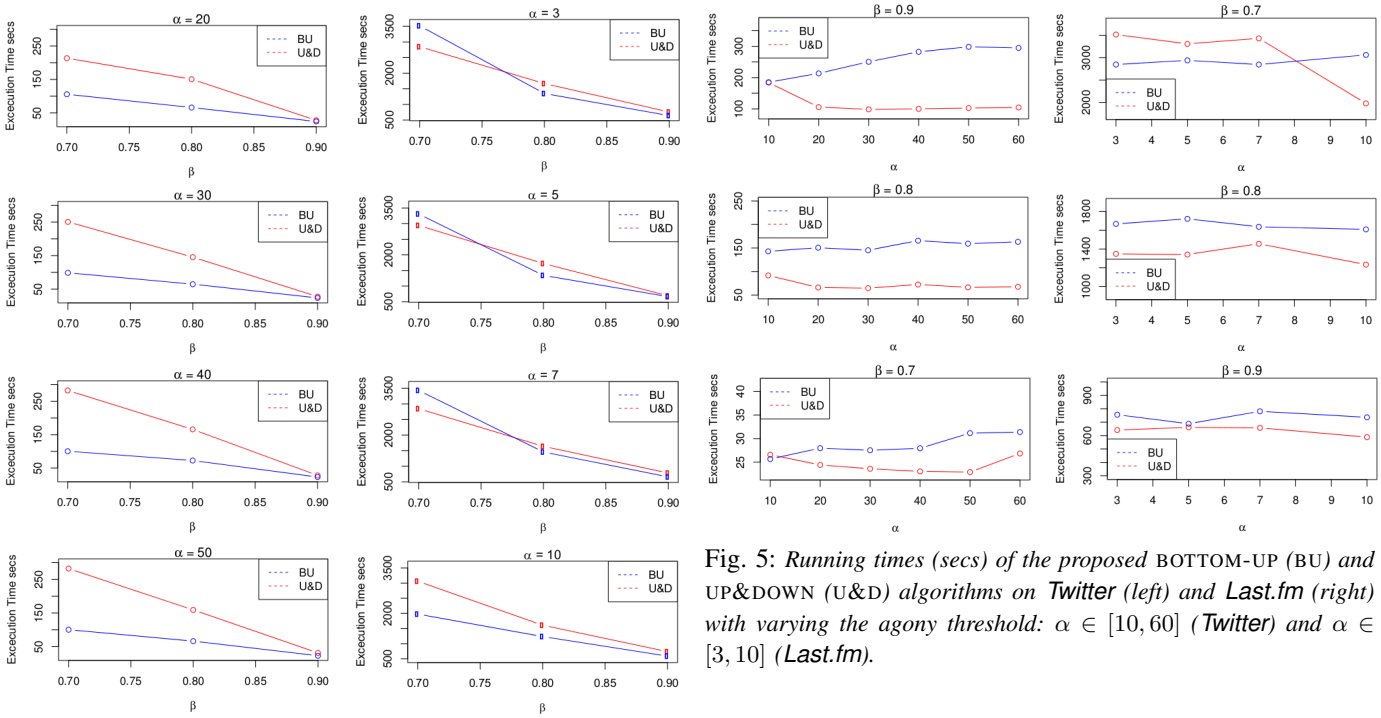


Fig. 5: *Running times (secs) of the proposed* BOTTOM-UP *(*BU*) and* UP&DOWN *(*U&D*) algorithms on* Twitter *(left) and* Last.fm *(right) with varying the agony threshold:* $\alpha \in [10, 60]$ *(*Twitter*) and* $\alpha \in [3, 10]$ *(*Last.fm*).*

Fig. 4: *Running times (secs) of the proposed* BOTTOM-UP *(*BU*) and* UP&DOWN *(*U&D*) algorithms on* Twitter *(left) and* Last.fm *(right) with varying the Jaccard threshold:* $\beta \in [0.7, 0.9]$.

varying the agony threshold α (while keeping β fixed). Indeed, as Figure 5 shows, the BOTTOM-UP times are increasing as α increases, while the opposite happens for UP&DOWN (there are some fluctuations, but mainly due to bookkeeping). This is again expected, because larger values of α imply more candidates to be processed at each level of the BOTTOM-UP algorithm. On the other hand, the larger α is, the more likely is that the summaries found at the end of the first phase of the UP&DOWN algorithm satisfy the agony constraint, thus leading to a more lightweight (and faster) top-down phase.

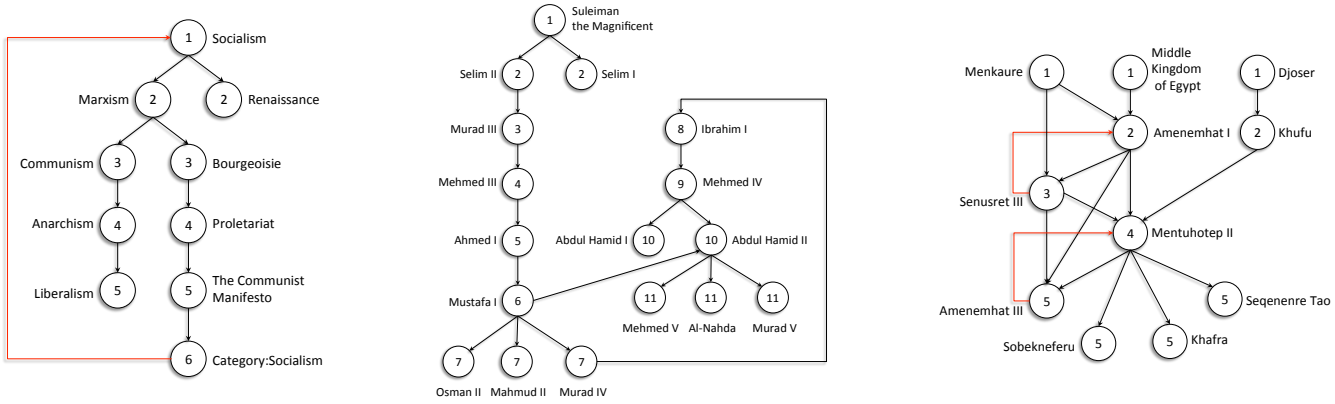
As far as the comparison of the two algorithms with each other, the results confirm what discussed in Section III-C. Indeed, we can observe here that the UP&DOWN algorithm is more efficient than BOTTOM-UP in most settings, with gains up to 65% (Twitter) and 35% (Last.fm). Nevertheless, in some cases BOTTOM-UP outperforms UP&DOWN. This happens, for instance, for small agony thresholds α and/or large Jaccard thresholds β (indeed, the gain achieved by UP&DOWN over BOTTOM-UP is overall decreasing as α decreases and/or β increases). The reason is that smaller α and/or larger β imply a smaller number of candidates to be processed in the bottom-up lattice-traversing phase, which is an advantage for BOTTOM-UP (it reduces the number of agony computations), but a disadvantage for UP&DOWN, as it increases the time spent in the top-down phase.

In conclusion, hence, although in most cases the fastest algorithm is UP&DOWN, the efficiency of the two algorithms depends on the selectivity of the constraints used, thus leading to cases where BOTTOM-UP is instead preferable.

B. Qualitative evaluation

We analyze here the summaries extracted by our algorithms from a qualitative viewpoint.

Wikipedia. Figure 6 shows examples of summaries from Wikipedia. Summary (a) consists of two main parts: the first part is a loop going through *Socialism* and *Proletariat*; the second is a branch that ends up in *Liberalism*. One could imagine that the first part is due to users who would like to explore



(a) *(b)* *(c)*

Fig. 6: *Three example summaries extracted from Wikipedia. The numbers inside the nodes correspond to one optimal ranking. Links that violate the ranking are represented in red.*

the topic of socialism, while the second one is browsing to a different one. Summary (b) is a chronological sequence of the sultans from the Ottoman Empire. This summary has no agony. The nodes of Summary (c) are pharaohs of Ancient Egypt. The ranking in the summary is almost the same as the chronological order. The exceptions are *Mentuhotep II*, who reigned before *Amenemhat I*, and *Khafra*, who reigned between *Khufu* and *Menkaure*.

We can observe that each summary is related to a train of thought. The hierarchy of the summaries defines a temporal sequence of concepts that are presented one after another. For example, Summary (a) consists in the description of the socialism, passing through Karl Marx, and the connections to other political philosophies, while Summary (b) goes through the history of the Ottoman Empire. Applications might exploit this to organize knowledge and present it to people. A summary could also inspire a lecture since it condenses the way how many people move through concepts. Moreover, as we can see in the examples, nodes in a summary are topically similar to each other. Summaries could hence help categorizing the knowledge space. Finally, summaries can be used for online contextual recommendation of sequences of pages. Given the current user session, one can fit a summary and predict not only the next page the user will visit but the whole future browsing path. This recommendation is contextualized since it leverages the browsing history of the user.

Flixster. In Table V we report two examples of summaries extracted from Flixster. For each of such summaries, we show some information (i.e., title, genre, and director) of the movies corresponding to the DAGs in that summary. We can observe that movies in the same summary exhibit some homogeneity of genre and type of audience. In this context, one might exploit summaries to discover early adopters for a topic (group of movies), by looking at the ranking that is coupled with each summary. Applications may leverage this information to target recommendations of new movies sharing similar topics with the summary, so to guarantee the desired spread over the network.

TABLE V: *Two summaries extracted from Flixster: title, genre, and director of the movies corresponding to the* DAG*s of each of the two summaries.*

Title	*Genre*	*Director*
Man of the Year	comedy/drama/romance	B. Levinson
Conversations With Other Women	comedy/drama/romance	H. Canosa
Step Up 2 the Streets	drama/music/romance	J. M. Chu
Brubaker	drama	S. Rosenberg

Title	*Classification*	*Director*
Prick Up Your Ears	biography/drama	S. Frears
Crooklyn	comedy/drama	S. Lee
Boy Culture	drama/comedy	Q. A. Brocka
One Eyed Jacks	western/drama/action adventure	M. Brando
Cop and a Half	family/comedy	H. Winkler
Operation Pacific	drama/war/action adventure/comedy	G. Waggner

V. Related work

The problem addressed in this paper represents an original type of structured pattern-mining problem, for which not much related prior research exists. We however briefly discuss the work in some neighborhood areas.

Graph pattern mining. The problem of graph pattern mining is to extract graph patterns (e.g., trees or subgraphs) that appear frequently in a graph database, i.e., a database composed by a large set of graphs. This research area has been quite active in the last decade and a lot of algorithms have been defined, such as AGM [4], FSG [5], gSpan [6], FFSM [7], SPIN [8], and Gaston [9]. Moreover, Yan et al. [10] propose a general framework to mine different graph patterns with possibly non-monotonic objective functions. The problem we study in this paper departs from graph pattern mining, as we do not mine frequent sub-structures from a set of graphs, rather we look for sets of graphs (DAGs) that satisfy certain requirements when merged together.

Graph summarization. The problem of graph summarization is to create a coarser-grained version of a graph such that the most important features of the graph are retained. The typical approach is based on identifying and aggregating sets of similar nodes so that the error deriving from the aggregation is minimized. Navlakha et al. [11] exploit the MDL principle

to summarize a graph with accuracy guarantees. Tian et al. [12] define a graph-summarization method which allows the user to specify the granularity level of the summarization in real time. Our problem is evidently different from graph summarization: the output of graph summarization is a reduced version of a single graph, whereas our summaries are sets of graphs (DAGs) structurally similar.

Applications. The study of the spread of information and influence through a social network has a long history in the social sciences. The first investigations focused on the adoption of medical [13] and agricultural innovations [14]. Later marketing researchers have investigated the "word-of-mouth" diffusion process for *viral marketing* applications [15], [16], which has then attracted most of the attention of the data-mining community, fueled by the seminal work by Domingos and Richardson [17] and Kempe et al. [18]. The main computational problems in this area are: (i) distinguishing genuine social influence from "homophily" and other factors of correlation [19], [20], [21]; (ii) measuring the strength of social influence over each social link [22], [23]; (iii) discovering a set of influential users [17], [18], [24]. Finally, a wide literature exists on the analysis of social influence in specific domains: for instance, studying person-to-person recommendation for purchasing books and videos [25], telecommunication services [26], or studying information cascades driven by social influence in Twitter [27], [28].

Our framework can also find applications in the field of *website usage analysis and re-organization* [1]. Typical browsing patterns can be exploited for reorganizing a website, creating *quicklinks* [29], and in general, making the navigation in the website more efficient for the users.

VI. Conclusions

In this paper we studied for the first time the problem of extracting informative summaries from a database of propagations. We defined the summaries of interest using two constraints: the first constraint is defined based on the Jaccard coefficient, while the definition of the second one relies on the graph-theoretic concept of "agony" of a graph. We showed that both constraints satisfy the downward-closure property, thus enabling Apriori-like algorithms. We developed two algorithms that visit the search space in different ways and apply to various real-world datasets.

Our work can be extended from various perspectives. First, taking inspiration from the wide literature on pattern mining, more efficient algorithms can be devised: for instance by depth-first visit of the search space or by taking advantage of the fact that we only look for the maximal patterns. Second, we can devise methods that adaptively decide which constraint to check first as the computation progresses [30]. Finally, we can study how to avoid setting rigid thresholds and make the constraints soft [31].

Acknowledgments. This work was supported by MULTISENSOR project, partially funded by the European Commission, under the contract number FP7-610411.

References

[1] R. Srikant and Y. Yang, "Mining web logs to improve website organization," in *WWW*, 2001, pp. 430–437.
[2] M. Gupte, P. Shankar, J. Li, S. Muthukrishnan, and L. Iftode, "Finding hierarchy in directed online social networks," in *WWW*, 2011, pp. 557–566.
[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB*, 1994, pp. 487–499.
[4] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *PKDD*, 2000, pp. 13–23.
[5] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *IEEE ICDM*, 2001, pp. 313–320.
[6] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *IEEE ICDM*, 2002, pp. 721–724.
[7] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *IEEE ICDM*, 2003, pp. 549–552.
[8] J. Huan, W. Wang, J. Prins, and J. Yang, "Spin: mining maximal frequent subgraphs from graph databases," in *KDD*, 2004, pp. 581–586.
[9] S. Nijssen and J. N. Kok, "A quickstart in frequent structure mining can make a difference," in *KDD*, 2004, pp. 647–652.
[10] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining significant graph patterns by leap search," in *SIGMOD*, 2008, pp. 433–444.
[11] S. Navlakha, R. Rastogi, and N. Shrivastava, "Graph summarization with bounded error," in *SIGMOD*, 2008, pp. 419–432.
[12] Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *SIGMOD*, 2008, pp. 567–580.
[13] J. Coleman, H. Menzel, and E. Katz, *Medical Innovations: A Diffusion Study*. Bobbs Merrill, 1966.
[14] T. Valente, *Network Models of the Diffusion of Innovations*. Hampton Press, 1955.
[15] F. Bass, "A new product growth model for consumer durables," *Management Science*, vol. 15, pp. 215–227, 1969.
[16] J. Goldenberg, B. Libai, and E. Muller, "Talk of the network: A complex systems look at the underlying process of word-of-mouth," *Marketing Letters*, vol. 12, no. 3, pp. 211–223, 2001.
[17] P. Domingos and M. Richardson, "Mining the network value of customers," in *KDD*, 2001, pp. 57–66.
[18] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003.
[19] A. Anagnostopoulos, R. Kumar, and M. Mahdian, "Influence and correlation in social networks," in *KDD*, 2008, pp. 7–15.
[20] D. J. Crandall, D. Cosley, D. P. Huttenlocher, J. M. Kleinberg, and S. Suri, "Feedback effects between similarity and social influence in online communities," in *KDD*, 2008, pp. 160–168.
[21] T. L. Fond and J. Neville, "Randomization tests for distinguishing social influence and homophily effects," in *WWW*, 2010, pp. 601–610.
[22] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "Learning influence probabilities in social networks," in *WSDM*, 2010, pp. 241–250.
[23] K. Saito, R. Nakano, and M. Kimura, "Prediction of information diffusion probabilities for independent cascade model," in *KES*, 2008, pp. 67–75.
[24] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "A data-based approach to social influence maximization," *PVLDB*, vol. 5, no. 1, pp. 73–84, 2011.
[25] J. Leskovec, A. Singh, and J. M. Kleinberg, "Patterns of influence in a recommendation network," in *PAKDD*, 2006, pp. 380–389.
[26] S. Hill, F. Provost, and C. Volinsky, "Network-based marketing: Identifying likely adopters via consumer networks," *Statistical Science*, vol. 21, no. 2, pp. 256–276, 2006.
[27] E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts, "Everyone's an influencer: quantifying influence on twitter," in *WSDM*, 2011, pp. 65–74.
[28] D. M. Romero, B. Meeder, and J. M. Kleinberg, "Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on Twitter," in *WWW*, 2011, pp. 695–704.
[29] D. Chakrabarti, R. Kumar, and K. Punera, "Quicklink selection for navigational query results," in *WWW*, 2009, pp. 391–400.
[30] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi, "Adaptive constraint pushing in frequent pattern mining," in *PKDD*, 2003.
[31] S. Bistarelli and F. Bonchi, "Interestingness is not a dichotomy: Introducing softness in constrained pattern mining," in *PKDD*, 2005.