Diffusion Centrality: A Paradigm to Maximize Spread in Social Networks

Chanhyun Kang[a], Sarit Kraus[b], Cristian Molinaro[c], Francesca Spezzano[d], V.S. Subrahmanian[a]

[a]*Department of Computer Science, University of Maryland, USA*
[b]*Department of Computer Science, Bar-Ilan University, Israel*
[c]*DIMES, Università della Calabria, Italy*
[d]*Department of Computer Science, Boise State University, USA*

Abstract

We propose *Diffusion Centrality* (DC) in which semantic aspects of a social network are used to characterize vertices that are influential in diffusing a property p. In contrast to classical centrality measures, diffusion centrality of vertices varies with the property p, and depends on the diffusion model describing how p spreads. We show that DC applies to most known diffusion models including tipping, cascade, and homophilic models. We present a hypergraph-based algorithm (HyperDC) with many optimizations to exactly compute DC. However, HyperDC does not scale well to huge social networks (millions of vertices, tens of millions of edges). For scaling, we develop methods to coarsen a network and propose a heuristic algorithm called "Coarsened Back and Forth" (CBAF) to compute the top-k vertices (having the highest diffusion centrality). We report on experiments comparing DC with classical centrality measures in terms of runtime and the "spread" achieved by the k most central vertices (using 7 real-world social networks and 3 different diffusion models). Our experiments show that DC produces higher quality results and is comparable to several centrality measures in terms of runtime.

Keywords:
Social networks, diffusion model, logic programming, quantitative logic

1. Introduction

An increasingly important problem in social networks (SNs) is that of assigning a "centrality" value to vertices which will reflect their importance within the SN. Well-known measures such as *degree centrality* [21, 46], *betweenness centrality* [8, 20], *PageRank* [9], *closeness centrality* [49, 5], and *eigenvector centrality* [7]

only take the structure of the network into account—they do not differentiate between vertices that are central w.r.t. spreading one topic or meme or sentiment vs. spreading another. A vertex that is important in spreading awareness of a mobile phone program may be very unimportant in spreading information about a restaurant. Likewise, most past work assumes that there is no information about properties of the vertices/edges or edge weights, but in modern social networks, at least some self-declared properties exist and, in many cases, analysis of tweets and posts can provide further information. These omissions can cause different problems as shown in the following toy example.

Example 1 (HIV). *Figure 1 shows four people a, b, c, d, where b has HIV. Solid edges denote sexual relationships, while dashed edges denote friend relationships. Both "friend" and "sexual partner" relationships can play a role in the diffusion of HIV (as friends may, unbeknownst to us, also be sexual partners). Edge weights denote the intensity of the relationships.*

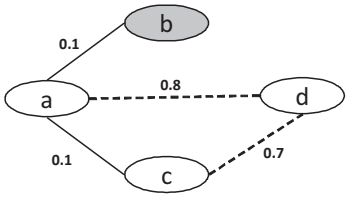

Figure 1: A small HIV social network. Shaded vertices denote people with HIV.

The table below shows the centrality of a, b, c, d w.r.t. various centrality measures. Notice that the nature of the relationships (i.e., friend and sexual partner) are not taken into account by these centrality measures, as they consider only the topological structure of the network.

Centrality Measure	**a**	**b**	**c**	**d**
Degree	1	0.33	0.66	0.66
Betweenness	2	0	0	0
PageRank	0.367	0.141	0.246	0.246
Closeness	0.33	0.2	0.25	0.25
Eigenvector	0.375	0.125	0.25	0.25

The only person in this network capable of spreading HIV is b. However, b has the lowest centrality according to all five centrality measures mentioned above.

Example 2. *Consider again the same network shown in Figure 1 and suppose the vertices denoted users on Twitter. A solid edge (u, v) denotes the fact that u and v have both retweeted at least one of the other's tweets, while a dashed edge*

indicates that they are friends (i.e., u follows v and vice versa). Suppose b was the only person to have tweeted a positive opinion about a political candidate while none of the others have done so. Then, by the same reasoning as in the previous example, and given that Figure 1 is the entire network, we can again infer that any other user (i.e., a, c, d) who tweets positively about the same candidate was either influenced by b or was influenced by some exogenous process. b should clearly get more credit for the other user's positive tweet than anyone else, but has the lowest centrality according to classical centrality measures.

Past centrality measures do not take into account *(i) the property of interest w.r.t. which a vertex's "importance" is measured, (ii) how properties (e.g., HIV in the example above) diffuse through the SN, and (iii) any semantic aspect of the network (properties of vertices and edges)*, solely focusing on the topological structure. Taking all of these aspects into account is crucial in determining the most central vertices. We can readily think of networks (e.g., Twitter) where person A has the highest centrality in terms of spread of support for Republicans, while person B is the central player in terms of spread of support for conserving gorillas. The network in both cases is the same (Twitter), but the most "central" person depends on the diffusive property with respect to which a vertex is considered "influential" or "central". Taking into account diffusion models (e.g., how one person influences another) is another crucial aspect. Different ways of spreading a property (e.g., a disease) may lead to different central vertices. Furthermore, intrinsic properties of vertices (customers, patients) and the nature and strength of the relationships (edges) are important too. For instance, [45] talks about the role of nine different demographic factors in influencing online purchases across 14 product categories, showing that some demographic factors are relevant for some product types, while others are relevant for others. *This paper proposes the novel notion of* diffusion centrality *that takes an SN, a diffusive property p, and a previously learned diffusion model Π for p, and defines centrality of vertices based on this input.* We do not provide algorithms to automatically learn diffusion models—interested readers may find one such algorithm in [10].

The paper's goal is to show how diffusion centrality can be used to achieve higher spread of a diffusive property p by using diffusion models for p rather than classical centrality measures. We further show that this can be done for most diffusion models we have seen in the literature. Last but not least, our methods are shown to scale to social networks with over 2M vertices and 20M edges.

Real-world diffusion models fall into three diverse categories. In *cascade models*, there is some probability that a vertex will spread a diffusive property p to one of its neighbors [34, 36]—these include popular disease spread models such as the SIR model of disease spread [26]. *Tipping models* use other mathematical cal-

culations such as cost-benefit analysis, often involving no probabilities, in order to decide whether a vertex will adopt a certain behavior. Tipping models were introduced by Nobel laureate Tom Schelling [50] to model segregation of neighborhoods, and by Granovetter [24]. In *homophilic models*, similarities between two vertices are considered in order to decide if the two vertices will adopt a similar behavior [42, 12, 3]. Homophilic models use various types of distance measures between attributes of a vertex (e.g., age, occupation, gender) and combine them via non-probabilistic measures to achieve a degree of similarity between two vertices.

Because diffusion models vary dramatically, a paradigm to express them must be capable of: *(i)* expressing semantic properties of vertices and connections between them, *(ii)* representing probabilistic inferences, and *(iii)* expressing inferences based on non-probabilistic, quantitative reasoning. The suite of knowledge representation paradigms offers several starting points. Bayesian nets and causal inference [47] offer an obvious place to start as they can be used to express cascade models. However, it is not clear if they can be used to express generic quantitative information, which is needed to express many other real diffusion models, such as tipping and homophilic models. In contrast, the language of Generalized Annotated (logic) Programs (GAPs) [35] has been well-studied in knowledge representation and is rich enough to capture a wide variety of different forms of reasoning. It can represent both the structure of social networks (with semantics and weight annotations) as well as these diverse types of diffusion models. Indeed, as shown in [51], many existing diffusion models from all three aforementioned categories can be expressed as GAPs, including: the *Susceptible-Infectious-Removed* (SIR) [2] and *Susceptible-Infectious-Susceptible* (SIS) [26] models of disease spread; the *Big Seed* marketing approach [57], which is a strategy of advertising to a large group of individuals who are likely to spread the advertisement further through network effects; models of diffusion of "favorited" pictures in Flickr [13]; tipping models like the Jackson-Yariv model [28]. We also considered richer versions of GAPs, such as hybrid knowledge bases [40], but decided they were far more expressive than needed for reasoning about diffusive processes.

The paper is organized as follows. Related work is discussed in Section 2. Section 3 introduces basic notions used in the rest of the paper. We define *Diffusion Centrality (DC)* in Section 4. Section 5 proposes a general "hypergraph fixed point algorithm" to efficiently compute the likelihood that an arbitrary vertex has certain properties according to the GAP diffusion model. We also define novel classes of GAPs, together with novel optimization methods to develop the HyperDC algorithm for computing DC. In Section 6, we propose the CBAF algorithm for finding the vertices with the top-k highest diffusion centralities in an approximate way. Section 7 describes extensive experiments comparing DC with classical centrality measures in terms of both runtime and the "spread" generated by central vertices.

2. Related Work

Several centrality measures have been proposed in graph theory and social network analysis: *degree centrality* [21, 46], *betweenness centrality* [8, 20], *PageRank* [9], *closeness centrality* [49, 5], and *eigenvector centrality* [7]. Variants of such centrality measures have been proposed in [18, 1, 25], while game-theoretic centrality measures have been proposed in [27, 52]. These centrality measures take into account only the network topology for determining vertex centrality, while DC considers additional information: the "semantics" embedded in the network, the diffusive property with respect to which a vertex is considered "influential" or "central", and the model describing how such a property propagates.

There has been extensive work in reasoning about diffusion in social networks. One well-known problem is *influence maximization*, that is, the problem of identifying a subset of vertices in a social network that maximizes the spread of influence. The problem was formulated as an optimization problem in the seminal paper [33], which focuses on two propagation models: the independent cascade and the linear threshold models. Subsequently, several approaches have been proposed to efficiently solve the problem [39, 15, 29, 16, 23, 56, 37]. All the aforementioned approaches consider restricted diffusion models and no vertex/edge properties are taken into account. In contrast, our approach deals with very general diffusion models and takes social network semantic properties into account. Being able to accurately model the diffusion processes and incorporate the network semantics is fundamental for correct analysis of real-world diffusion phenomena.

Another well-studied related problem is the *target set selection* problem [14, 17], which assumes a deterministic tipping model and seeks to find a set of vertices of a certain size that optimizes the final number of adopters. These approaches focus on specific diffusion models and neglect networks' semantics.

The notion of diffusion centrality was first proposed in [31]. This paper extends [31] in different respects. In [31], diffusion models are expressed using simple conditional probability rules, while in this paper we use the more general language of GAPs. The algorithms introduced in this paper are more general and efficient, and exploit several new optimizations introduced in this paper. Moreover, we have addressed the new problem of finding an approximate set of top-k vertices with the highest diffusion centrality and proposed efficient algorithms to solve it.

Recently, after our paper on diffusion centrality [31], a few pieces of work have observed that no individual can be a universal influencer, and influential members of the network tend to be influential only in one or some specific domains of knowledge [4, 11, 53]. This has led to the extension of the classic independent cascade and linear threshold models to be "topic-aware" [4, 11], thus considering propagation with respect to a particular topic. Still, the diffusion models considered by

these works are limited, as well as the characteristics of vertices and edges.

Recently, [48] has addressed the problem of coarsening a social network (that is, finding a more succinct representation of it by grouping vertices together) while preserving its propagation characteristics as much as possible. It works well for the independent cascade model and considers only the structure of the network for merging. The idea of coarsening a network has been extensively used in community detection techniques [32]. However, they use different metrics for coarsening, such as cut-based, flow-based, or heavy-edge matching-based conditions. Methods to compress weighted graphs into smaller ones have been proposed in [54, 55], where nodes and edges are grouped into "supernodes" and "superedges". No semantic properties of vertices/edges or diffusion models are considered. In our CBAF algorithm, when coarsening networks, we explicitly consider a given diffusion model and merge vertices that have the same role in the process, considering the semantic aspects of the SN. Another related problem is graph sparsification, which relies on the notion of "spanners" [22, 41]. The main difference is that graph sparsification removes edges (so the nodes stay the same), while we contract the graph trying to preserve the behavior w.r.t. a given diffusion model. [43] uses probabilistic soft logic to develop graph summarization techniques for grouping similar entities (vertices) and relations (edges) by considering the semantic aspects of networks. The approach does not consider any diffusion process explicitly.

3. Preliminaries

In this section, we define social networks (SNs), illustrate generalized annotated programs (GAPs) from [35], and show how diffusion models can be expressed with GAPs. We refer the reader to [35] for more details on GAPs.

We model SNs as weighted directed graphs where properties can be assigned to vertices and edges. More specifically, properties of vertices and edges are taken from two (disjoint) sets VP and EP, respectively. For instance, a property in VP might be hiv (meaning that a vertex has HIV), while properties in EP might be fr and sp, representing friend and sexual relationships, respectively—e.g., if an edge is assigned property fr, then its endpoints are friends.

A *social network* (SN) is a tuple $(V, E, \mathsf{VL}, \omega)$ where:

1. V is a finite set of *vertices*;
2. $E \subseteq V \times V \times \mathsf{EP}$ is a finite set of *(labeled) edges*;
3. $\mathsf{VL} : V \to 2^{\mathsf{VP}}$ assigns a set of properties to each vertex;
4. $\omega : E \to (0, 1]$ assigns a weight to each edge.

Thus, an SN is a directed graph where VL assigns a set of properties to each vertex, and there can be multiple labeled edges between a given pair of vertices, each of which is associated with a weight and a unique edge property. An SN is depicted in Figure 1, where property hiv is assigned to vertex b, sp is assigned to solid edges, and fr is assigned to dashed edges.

Below we briefly recall GAPs. We start with an example that illustrates a GAP modeling the spread of HIV in SNs like the one in Figure 1.

Example 3. *A GAP Π_{hiv} for the SN of Example 1 might be:*

$$
\begin{array}{ll}
[r_1] & \mathsf{hiv}(V) : 0.9 \times X \times Y \leftarrow \mathsf{sp}(V, V') : Y \wedge \mathsf{hiv}(V') : X \\
[r_2] & \mathsf{hiv}(V) : 0.4 \times X \times Y \times Y' \leftarrow \mathsf{fr}(V, V') : Y \wedge \mathsf{sp}(V', V'') : Y' \wedge \mathsf{hiv}(V'') : X \\
[r_3] & \mathsf{hiv}(V) : 0.6 \times X \times Y \times Y' \leftarrow \mathsf{sp}(V, V') : Y \wedge \mathsf{sp}(V', V'') : Y' \wedge \mathsf{hiv}(V'') : X
\end{array}
$$

The first rule says that the confidence that a vertex V has HIV, given that a partner V' has HIV with confidence X, is $0.9 \times X \times Y$, where Y is the weight of the sexual relationship between the two vertices. The other rules can be similarly read.

We will treat properties in VP as unary predicate symbols, called *vertex predicate symbols*, and properties in EP as binary predicate symbols, called *edge predicate symbols*. Given a vertex (resp. edge) predicate symbol p, then $p(t)$ (resp. $p(t_1, t_2)$) is a *vertex atom* (resp. *edge atom*), where t, t_1, t_2 are variables or constants representing vertices. For instance, in Example 3 above, $\mathsf{hiv}(V)$ and $\mathsf{sp}(V, V')$ are a vertex atom and an edge atom, respectively.

An (edge or vertex) *annotated atom* is of the form $A : \mu$, where A is an (edge or vertex) atom, and μ is an *annotation term*, that is, an expression built from functions, variables (distinct from those used inside atoms), and real numbers in $[0, 1]$. For instance, in Example 3, $\mathsf{hiv}(V) : 0.9 \times X \times Y$ is an annotated atom.

A *GAP-rule* (or simply *rule*) is of the form $A_0 : \mu_0 \leftarrow A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$ where $n \geq 0$ and every $A_i : \mu_i$ is an annotated atom. $A_0 : \mu_0$ is the *head* of the rule, while $A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$ is the *body* of the rule. A *generalized annotated program* (GAP) is a finite set of rules.

We assume that VP contains a distinguished vertex predicate symbol vertex that represents the presence of a vertex in an SN. Every SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$ can be represented by a GAP, denoted $\Pi_{\mathcal{S}}$, as follows:

$$
\begin{array}{rl}
\Pi_{\mathcal{S}} = & \{\mathsf{vertex}(v) : 1 \mid v \in V\} \cup \{p(v) : 1 \mid v \in V \wedge p \in \mathsf{VL}(v)\} \cup \\
& \{ep(v_1, v_2) : \omega(\langle v_1, v_2, ep\rangle) \mid \langle v_1, v_2, ep\rangle \in E\}
\end{array}
$$

When we augment $\Pi_{\mathcal{S}}$ with rules describing how certain properties diffuse through the social network, we get a GAP $\Pi \supseteq \Pi_{\mathcal{S}}$ that captures both the structure of the SN and the diffusion principles. In this paper we consider a restricted class of

GAPs: every rule with a non-empty body has a vertex annotated atom in the head ([35] allows any annotated atom in the head of a rule). Thus, edge atoms can appear only in rule bodies or rules with an empty body. This restriction results from the set of diffusion models we consider in this paper: neither edge weights nor edge labels change as the result of the diffusion. However, most of the techniques developed in this paper can be directly applied to or easily generalized for unrestricted GAPs.

An (annotated) atom (resp. rule, GAP) is *ground* iff it contains no variables (neither inside atoms nor in annotation terms). We use $\mathcal{A}$ to denote the set of all ground atoms. Moreover, $grd(r)$ denotes the *ground instances* of a rule r, i.e. the set of all rules obtained from r by replacing every occurrence of a variable in an annotation term with a real number in $[0, 1]$, and every occurrence of a variable inside an atom with a vertex. Given a GAP Π, we define $grd(\Pi) = \bigcup_{r \in \Pi} grd(r)$.

We now briefly recall the semantics of GAPs. An *interpretation* I is a mapping from the set of all ground atoms $\mathcal{A}$ to $[0, 1]$. We say that I *satisfies* a ground annotated atom $A : \mu$, denoted $I \models A : \mu$, iff $I(A) \geq \mu$. [35] associates an operator $\mathbf{T}_\Pi$ that maps interpretations to interpretations with any GAP Π. Suppose I is an interpretation. Then,

$$\mathbf{T}_\Pi(I)(A) = \max(\{I(A)\} \cup \{\mu \mid A : \mu \leftarrow AA_1 \wedge \ldots \wedge AA_n \text{ is in } grd(\Pi) \text{ and for all } 1 \leq i \leq n,\ I \models AA_i\})$$

Roughly speaking, the semantics of GAPs requires that when there are multiple ground instances of GAP-rules with the same head that "fires", the highest annotation in any of these ground rules is assigned to the head atom. [35] shows that if we start from the interpretation that assigns 0 to every ground atom and iteratively apply $\mathbf{T}_\Pi$, then we reach a least fixed point, denoted $\mathsf{lfp}(\Pi)$, which captures the ground atomic logical consequences of Π.

Example 4. *Consider the social network $\mathcal{S}$ of Example 1 and the GAP Π_{hiv} of Example 3. Let Π be the GAP modeling both the SN and the diffusion model, i.e., $\Pi = \Pi_\mathcal{S} \cup \Pi_{hiv}$. Then, $\mathbf{T}_\Pi$ reaches a least fixed point at the third iteration, as shown in Table 1. Edge atoms are not reported in the table, as their annotations and weights do not change by applying $\mathbf{T}_\Pi$ (because edge atoms do not appear in rule heads). Specifically, 0.1 is assigned to $\mathsf{sp}(a, b)$, $\mathsf{sp}(b, a)$, $\mathsf{sp}(a, c)$, and $\mathsf{sp}(c, a)$, 0.8 is assigned to $\mathsf{fr}(a, d)$ and $\mathsf{fr}(d, a)$, and 0.7 is assigned to $\mathsf{fr}(c, d)$ and $\mathsf{fr}(d, c)$.*

Initially, 0 is assigned to all ground atoms (iteration 0). Next, 1 is assigned to the ground atom $\mathsf{hiv}(b)$, because b has HIV in the original SN (iteration 1). From now on, rules in Π_{hiv} can be used to assign higher values to (vertex) ground atoms. Specifically, at iteration 2, the following rules can "fire":

$[r'_1]$ $\mathsf{hiv}(a) : 0.9 \times 1 \times 0.1 \leftarrow \mathsf{sp}(a, b) : 0.1 \wedge \mathsf{hiv}(b) : 1$
$[r'_2]$ $\mathsf{hiv}(d) : 0.4 \times 1 \times 0.8 \times 0.1 \leftarrow \mathsf{fr}(d, a) : 0.8 \wedge \mathsf{sp}(a, b) : 0.1 \wedge \mathsf{hiv}(b) : 1$
$[r'_3]$ $\mathsf{hiv}(c) : 0.6 \times 1 \times 0.1 \times 0.1 \leftarrow \mathsf{sp}(c, a) : 0.1 \wedge \mathsf{sp}(a, b) : 0.1 \wedge \mathsf{hiv}(b) : 1$

Iteration of $\mathbf{T}_\Pi$	$\mathsf{hiv}(a)$	$\mathsf{hiv}(b)$	$\mathsf{hiv}(c)$	$\mathsf{hiv}(d)$
0	0	0	0	0
1	0	1	0	0
2	0.09	1	0.006	0.032
3	0.09	1	0.0081	0.032
4	0.09	1	0.0081	0.032

Table 1: Iterations of $\mathbf{T}_\Pi$.

and are used to assign 0.09 *to* $\mathsf{hiv}(a)$*, 0.032 to* $\mathsf{hiv}(d)$*, and 0.006 to* $\mathsf{hiv}(c)$*.*

At the subsequent iteration (iteration 3), the following rule can fire and is used to assign a higher value to $\mathsf{hiv}(c)$*, namely 0.0081:*

$$[r_1''] \quad \mathsf{hiv}(c) : 0.9 \times 0.09 \times 0.1 \leftarrow \mathsf{sp}(c, a) : 0.1 \wedge \mathsf{hiv}(a) : 0.09$$

No higher values can be derived from a further application of $\mathbf{T}_\Pi$*, and thus the least fixed point is reached, assigning* 0.09 *to* $\mathsf{hiv}(a)$*, 1 to* $\mathsf{hiv}(b)$*, 0.0081 to* $\mathsf{hiv}(c)$*, and 0.032 to* $\mathsf{hiv}(d)$*.*

Example 5. *Suppose the SN in Figure 1 represents* ***Cell phone users****, all edges have property* fr *representing friendship relations, and vertices have properties like* male*,* female*,* young*,* old*, and* adopter*, telling us if the user adopted a cell phone plan. The phone company wants to identify important users. Suppose* d *is male and everyone else is female; initially nobody is an* adopter*. A cell phone provider may have a diffusion rule learned from past promotions:*

$$\mathsf{adopter}(V') : 0.6 \times X \times Y \leftarrow \mathsf{adopter}(V) : X \wedge \mathsf{male}(V) : Y \wedge \mathsf{fr}(V, V') : 0.1$$

The vertex which has the greatest influence, if given a free mobile phone plan and if the above diffusion model is used, is clearly d *(because this is the only vertex that can "influence" others to adopt the plan). However, we see from the table in Example 1 that* d *is not the most relevant vertex w.r.t. to all centrality measures.*

4. Diffusion Centrality

Diffusion centrality tries to measure how well a vertex v can diffuse a property p (e.g., the hiv property). Given an SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a vertex predicate symbol p, and a vertex $v \in V$, the *insertion* of $p(v)$ into $\mathcal{S}$, denoted $\mathcal{S} \oplus p(v)$, is the SN $(V, E, \mathsf{VL}', \omega)$ where VL' is exactly like VL except that $\mathsf{VL}'(v) = \mathsf{VL}(v) \cup \{p\}$. In other words, inserting $p(v)$ into a social network merely says that vertex v has property p and that everything else about the network stays the same. Likewise, the *removal* of $p(v)$ from $\mathcal{S}$, denoted $\mathcal{S} \ominus p(v)$, is the social network $(V, E, \mathsf{VL}'', \omega)$ which is just like $\mathcal{S}$ except that $\mathsf{VL}''(v) = \mathsf{VL}(v) \setminus \{p\}$.

Definition 1 (Diffusion Centrality). *Let $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$ be an SN, Π a GAP, and p a property. The* diffusion centrality *(DC for short) of a vertex $v \in V$ w.r.t. Π, p, and $\mathcal{S}$, denoted $\mathsf{dc}_{\Pi,p,\mathcal{S}}(v)$, is defined as follows:*

$$\sum_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \oplus p(v)})(p(v')) \; - \sum_{v'' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \ominus p(v)})(p(v''))$$

Whenever Π, p, and $\mathcal{S}$ are clear from the context, we denote the diffusion centrality of a vertex v simply as $\mathsf{dc}(v)$.

This definition says that computing the diffusion centrality of vertex v involves two steps. First, we assume that vertex v has property p and see how much diffusion occurs. This is done by computing the least fixed point of the diffusion model and the SN $\mathcal{S} \oplus p(v)$ (i.e., the original SN where p is assigned to v). Notice that the overall diffusion is quantified by summing up the values that the least fixed point assigns to atoms of the form $p(v')$ across all vertices $v' \neq v$ of $\mathcal{S}$. Then, we assume that vertex v does not have property p and see how much diffusion occurs. This is done by computing the least fixed point of the diffusion model and the SN $\mathcal{S} \ominus p(v)$ (i.e., the original SN where p is not assigned to v). The diffusion centrality of v is the difference between the above two numbers and captures the "impact" that would occur in terms of diffusion of property p if vertex v had property p.[1]

Notice that the least fixed point of Π and $\mathcal{S} \oplus p(v)$ (or $\mathcal{S} \ominus p(v)$) takes into account the initial assignment of p to the vertices of $\mathcal{S}$. Thus, DC depends also on the initial assignment of p in $\mathcal{S}$. For instance, consider two SNs that are identical except that in the first one every vertex has property p, while in the second one no vertex has property p. In the first SN, $\mathsf{dc}(v) = 0$ for every vertex v, because whether p is given to v or not has no impact, since everyone already has p. In contrast, in the second SN, $\mathsf{dc}(v)$ reflects how much overall spread of p we achieve in the SN by assigning p to v.

Example 6. *Consider again the HIV SN of Example 1 and the GAP of Example 3. Recall that the only vertex with property* hiv *is b. It can be easily verified that the values for the positive and negative summands of Definition 1 for all vertices are as reported in the following table.*

[1] Considering just the first summation of Definition 1 is wrong. Suppose we have an SN and a vertex v s.t. the first summation of $\mathsf{dc}(v)$ is a high number N (i.e., N is the expected number of vertices with property p assuming that v has property p). Suppose that when we assume that v does not have property p, the same value N is determined (i.e., the second summation equals N). Then, intuitively, v should not have a high diffusion centrality as the expected number of vertices with property p is the same regardless of whether v has property p or not (thus, v does not play a central role in diffusing p). In contrast, considering just the first summation would give v a high centrality.

	a	b	c	d
Positive Summand	1.122	0.13	1.122	1.0981
Negative Summand	1.0401	0	1.122	1.0981
Diffusion Centrality	0.0819	0.13	0	0

For instance, consider vertex a. If we assume that hiv *is given to a in the original SN, then we get an overall spread of* hiv *of 1.122 (positive summand of Definition 1). If we assume that* hiv *is not given to a in the original SN, then we get an overall spread of* hiv *of 1.0401 (negative summand of Definition 1). Then, the diffusion centrality of a is the difference of these two values. The same argument can be applied to the remaining vertices. It is worth noting that if* hiv *is not assigned to b in the original SN, then no spread of* hiv *occurs (the negative summand for b is 0), because nobody else has HIV in the SN.*

Thus, b has the highest centrality w.r.t. hiv *and Π_{hiv}—classical centrality measures (Example 1) do not capture this because b is not a "central" vertex from a purely topological perspective. However, b should have the highest centrality because it is the only one with HIV. Vertices c and d do not increase the confidence of any vertex to have HIV. So their diffusion centrality is zero.*

Example 7. *If we return to the cell phone case (Example 5), we see that the DC of d is 1.2, while all other vertices have 0 as their DC. Furthermore, as opposed to classical centrality metrics, c and d do not have the same centrality, because their properties and the diffusion of interest make them important to a different extent.*

Diffusion Centrality Problem (DCP). Given an SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a GAP Π, and a property p, the diffusion centrality problem consists of finding the DC (w.r.t. Π, p, and $\mathcal{S}$) of every vertex of $\mathcal{S}$.

Top-k Diffusion Centrality Problem (kDCP). Given a $0 < k < |V|$, the top-k diffusion centrality problem consists of finding a set T of k vertices of $\mathcal{S}$ having the highest DC, that is, the DC of every vertex in T is greater than or equal to the DC of every vertex in $V \setminus T$.

5. The HyperDC Algorithm for Exact Diffusion Centrality Computation

In this section, we present the *HyperDC* algorithm (Section 5.3), which solves DCP exactly using the *HyperLFP* algorithm (Section 5.2) to compute the least fixed point of a GAP. We start with a set of optimizations that can speed up HyperDC.

5.1. Optimization Steps

We first present optimizations that can be applied to arbitrary GAPs, and then identify two subclasses of GAPs for which DC can be computed even more efficiently. Before that, we introduce notation and terminology used in the following.

The *dependency graph* of a GAP Π is a directed graph $dep(\Pi)$ whose vertices are the predicate symbols in Π. There is an edge from a predicate symbol q to a predicate symbol p iff there is a rule in Π where p occurs in the head and q occurs in the body. We say that p *depends on* q if there exists a path from q to p in $dep(\Pi)$. If p depends on q and vice versa, then we say that p and q are *mutually recursive*. We use $M_{\Pi,p}$ to denote the set of all predicate symbols that are mutually recursive with p. We define $R_{\Pi,p}$ as the set of predicate symbols q such that *(i)* p depends on q, and *(ii)* q appears in the head of some rule of Π. Note that $M_{\Pi,p} \subseteq R_{\Pi,p}$. Given a GAP Π and a property p, we define

$$\Pi_p = \{r \in \Pi \mid \begin{array}{l} p \text{ is the head predicate symbol of } r, \text{ or} \\ p \text{ depends on the head predicate symbol of } r\}. \end{array}$$

We are now ready to introduce our first optimization.

Caching lfp. When computing $\mathsf{dc}(v)$, we note that $\mathcal{S}$, $\mathcal{S} \oplus p(v)$, and $\mathcal{S} \ominus p(v)$ differ only in whether or not vertex v has property p. One way to leverage this is to first compute and cache $\mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}})$ independent of v. We then only need to calculate one summation in order to compute $\mathsf{dc}(v)$.

Proposition 1. *Consider a social network* $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$*, a GAP* Π*, and a property* p*. Let* $\phi = \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}})$ *and* v *be a vertex in* V*. Then,*

$$\mathsf{dc}(v) = \begin{cases} \sum\limits_{v' \in V \setminus \{v\}} \phi(p(v')) - \sum\limits_{v'' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \ominus p(v)})(p(v'')) & \text{if } p \in \mathsf{VL}(v) \\ \sum\limits_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \oplus p(v)})(p(v')) - \sum\limits_{v'' \in V \setminus \{v\}} \phi(p(v'')) & \text{if } p \notin \mathsf{VL}(v) \end{cases}$$

PROOF. Consider a vertex $v \in V$. If v has property p, then, by definition of $\mathcal{S} \oplus p(v)$, we have that $\phi = \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \oplus p(v)})$ and the first equation holds. Likewise, if v does not have property p, then, by definition of $\mathcal{S} \ominus p(v)$, we have that $\phi = \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \ominus p(v)})$, and thus the second equation follows too. □

Network filtering. We now present an optimization, called *network filtering*, which consists of reducing the given SN by removing vertices (and the relative incoming and outgoing edges) that do not play any role in the diffusion process, i.e., those vertices that can never receive or transmit diffusive property p from/to other vertices in the SN via any rule in the considered diffusion model. As shown in the following, network filtering is a sound optimization technique, that is, its aim is to reduce the size of the SN (so as to make the DC computation faster) *without altering the DC values*. Specifically, removed vertices have zero DC in the original SN, while the DC of the remaining vertices in the reduced SN is the same as in the original one (cf. Proposition 2).

Example 8. *Consider the diffusion model Π_{hiv} from Example 3:*

$$\mathsf{hiv}(V) : 0.9 \times X \times W \leftarrow \mathsf{sp}(V, V') : W \wedge \mathsf{hiv}(V') : X$$
$$\mathsf{hiv}(V) : 0.4 \times X \times W \times W' \leftarrow \mathsf{fr}(V, V') : W \wedge \mathsf{sp}(V', V'') : W' \wedge \mathsf{hiv}(V'') : X$$
$$\mathsf{hiv}(V) : 0.6 \times X \times W \times W' \leftarrow \mathsf{sp}(V, V') : W \wedge \mathsf{sp}(V', V'') : W' \wedge \mathsf{hiv}(V'') : X$$

Suppose v is a vertex with no sp *relations (in a given SN). Moreover, suppose none of v's friends have* sp *relations. Then, v is an "unnecessary" vertex (for the purpose of computing DC) because there is no rule $r \in \Pi_{hiv}$ by which vertex v can receive property* hiv *or transmit* hiv *to other vertices. In fact, it is easy to see from the rules above that a vertex cannot transmit* hiv *if it does not have* sp *relations. Moreover, a vertex cannot get* hiv *if it does not have* sp *relations (the first and third rules cannot be applied) and its friends have no* sp *relations (the second rule cannot be applied).*

Thus, roughly speaking, network filtering matches diffusion rules against vertices to identify vertices that do not participate in the activation of any rule, regardless of how many properties are involved in the diffusion rules and in the SN.

Definition 2 (Rule activation). *Given a GAP Π, a property p, and a rule $r \in \Pi$, we define $relbody(r) = \{AA \mid AA$ is an annotated atom in the body of r and its predicate symbol is not in $R_{\Pi,p}\}$. Vertex v of an SN $\mathcal{S}$* activates *a rule $r \in \Pi$ iff there exists a ground rule $r' \in grd(r)$ such that:*

1. *for every $AA \in relbody(r')$, $\Pi_{\mathcal{S}} \models AA$;*
2. *if $A : \mu$ is the head of r', then $\mu > 0$; and*
3. *v appears in an annotated atom of the body of r'.*

Definition 3 (Necessary and unnecessary vertices). *Let $\mathcal{S}$ be an SN, Π be a GAP, and p a property. A vertex of $\mathcal{S}$ is* necessary *if it activates a rule of Π_p, otherwise it is* unnecessary.

Roughly speaking, a necessary vertex is one that might "trigger" some rule during the least fixed point computation, while unnecessary vertices have no chance of being involved in the least fixed point computation. Identifying necessary vertices is similar in spirt to the identification of relevant rules in probabilistic logic programs, where the aim is to find ground rules that are relevant for the given query [19]. In our case, we are interested in the values assigned to ground atoms of the form $p(v)$, which are in a sense the query atoms—here p is the diffusive property. However, rather than getting rid of ground rules, we get rid of vertices

(i.e., constants) that are "irrelevant". This is important in our applications where the SN may have millions of vertices and disregarding the irrelevant ones can yield significantly better performances. Of course, disregarding some vertices lead to disregarding some ground rules as well (i.e, those ground rules containing at least one irrelevant vertex). The filtering of a social network eliminates all unnecessary vertices (along with their incoming/outgoing edges).

Definition 4 (Network Filtering). *Let $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$ be an SN, Π a GAP, p a property, and U the set of unnecessary vertices. The* filtering *of $\mathcal{S}$ (w.r.t. Π and p) is the SN $\mathcal{S}' = (V', E', \mathsf{VL}', \omega')$ where:*

1. $V' = V \setminus U$;
2. $E' = E \setminus \{\langle u, v, q\rangle \in E \mid u \in U \vee v \in U\}$;
3. $\mathsf{VL}'(v) = \mathsf{VL}(v)$ *for all* $v \in V'$;
4. $\omega'(e) = \omega(e)$ *for all* $e \in E'$.

The filtering $\mathcal{S}'$ of an SN $\mathcal{S}$ is useful because unnecessary vertices have a DC of zero in $\mathcal{S}$, while the DC of necessary vertices can be computed on the (smaller) SN $\mathcal{S}'$ in a sound way, that is, their DC in $\mathcal{S}'$ is the same as in $\mathcal{S}$.

Proposition 2. *Let $\mathcal{S}$ be an SN, Π be a GAP, p a property, and $\mathcal{S}'$ the filtering of $\mathcal{S}$. For every vertex v of $\mathcal{S}$, if v is unnecessary, then $\mathsf{dc}_{\Pi,p,\mathcal{S}}(v) = 0$, otherwise ($v$ is necessary) $\mathsf{dc}_{\Pi,p,\mathcal{S}}(v) = \mathsf{dc}_{\Pi,p,\mathcal{S}'}(v)$.*

PROOF. If a vertex v is unnecessary, it cannot activate any rule in Π, thus, independent of whether or not it has the diffusion property p, it does contribute to diffusing p to other vertices (see item 2 of Definition 2). It follows that
$\sum_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \oplus p(v)})(p(v')) = \sum_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \ominus p(v)})(p(v')) =$
$\sum_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}})(p(v'))$, and then $\mathsf{dc}_{\Pi,p,\mathcal{S}}(v) = 0$.
If a vertex v is necessary, all the rules it activates contain necessary vertices, and then we have that
$\sum_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \oplus p(v)})(p(v')) = \sum_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}' \oplus p(v)})(p(v'))$,
$\sum_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S} \ominus p(v)})(p(v')) = \sum_{v' \in V \setminus \{v\}} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}' \ominus p(v)})(p(v'))$.
It follows that $\mathsf{dc}_{\Pi,p,\mathcal{S}}(v) = \mathsf{dc}_{\Pi,p,\mathcal{S}'}(v)$. □

Subclasses of GAPs. We now introduce *p-monotonic* GAPs, a class of GAPs to which we can apply further optimizations (in addition to those discussed so far).

Definition 5 (p-monotonic GAP). *We say that a rule $A_0 : \mu_0 \leftarrow A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$ is* monotonic *iff μ_0 is a monotonic function*[2]*. Given a GAP Π and a property p, we say that Π is* p-monotonic *iff, for every rule r in Π, when the head predicate symbol is in $R_{\Pi,p}$ then r is monotonic.*

Example 9. *Consider the GAP Π of Example 3, for which $R_{\Pi,\mathsf{hiv}} = \{\mathsf{hiv}\}$. To see if Π is* hiv*-monotonic we need to check if for every rule in Π having an annotated atom of the form $\mathsf{hiv}(V) : \mu$ in the head, we have that μ is a monotonic function. Since this is the case, then Π is* hiv*-monotonic. The following GAP*

$$p(V) : 0.5 \times X \times W \leftarrow \mathsf{fr}(V, V') : W \wedge p(V') : X$$
$$q(V) : (1 - X) \times W \leftarrow \mathsf{fr}(V, V') : W \wedge p(V') : X$$
$$p(V) : 0.9 \times X \times W \leftarrow \mathsf{fr}(V, V') : W \wedge q(V') : X$$

is not p-monotonic *because $R_{\Pi,p} = \{p, q\}$ and the annotation of the head atom of the second rule is a non-monotonic function.*

Given a GAP Π and a property p, we define the *p-interfered predicate set* as follows:

$$I_{\Pi,p} = \begin{cases} M_{\Pi,p} & \text{if } \Pi \text{ is } p\text{-monotonic} \\ R_{\Pi,p} & \text{if } \Pi \text{ is not } p\text{-monotonic} \end{cases}$$

Thus, the p-interfered predicate set contains all the predicate symbols that are mutually recursive with p if the GAP Π is p-monotonic, while, if Π is not p-monotonic, it is the set of predicate symbols q such that p depends on q and q appears in the head of some rule in Π. For instance, $I_{\Pi,\mathsf{hiv}} = \{\mathsf{hiv}\}$ for the GAP of Example 3, while $I_{\Pi,p} = \{p, q\}$ for the GAP of Example 9. Notice that when $I_{\Pi,p} = R_{\Pi,p}$, then $I_{\Pi,p}$ is bigger as $M_{\Pi,p} \subseteq R_{\Pi,p}$.

Recall that, given a GAP Π and a property p, then Π_p is the set of rules r in Π such that either p depends on the head predicate symbol of r or p is the head predicate symbol. We also define

$$\Pi_p^* = \{r \in \Pi_p \mid \text{every predicate symbol } q \text{ in the body of } r \text{ is s.t. } q \notin I_{\Pi,p}\}.$$

Roughly speaking, rules in Π_p are the only ones that may affect the values assigned to ground atoms of the form $p(v)$ in the least fixed point—so, we can ignore rules in $\Pi \setminus \Pi_p$. Moreover, Π_p can be partitioned into two sets: Π_p^* and $\Pi_p \setminus \Pi_p^*$. We can first evaluate Π_p^* over the SN, as it does not depend on the

[2]If μ_0 is a constant, it is considered to be monotonic. Moreover, if $\mu_0 = f(\ldots)$, then μ_0 is monotonic if f is monotonic, i.e., if $x_i \leq y_i$ for all $1 \leq i \leq n$, then $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$, where n is the number of arguments of f.

remaining rules in Π_p. Then, $\Pi_p \setminus \Pi_p^*$ can be evaluated. This somehow reminds of the stratification of logic programs, where a program is partitioned into different "strata", which are evaluated one at a time according to an order dictated by their dependencies. In our case, we first evaluate a "base stratum" (namely Π_p^*), and then use its result as a starting point for the evaluation of a stratum consisting of "mutually recursive" rules (namely $\Pi_p \setminus \Pi_p^*$). Thus, in our setting, we have two strata, as we are interested in just one "special" predicate p modeling the diffusive property. The following proposition states precisely how Π_p and Π_p^* are exploited (recall that $\mathcal{A}$ denotes the set of all ground atoms).

Proposition 3. *Consider an SN $\mathcal{S}$, a property p, and a GAP Π. Let ψ be the interpretation $\mathsf{lfp}(\Pi_\mathcal{S} \cup \Pi_p^*)$. Then, $\mathsf{lfp}(\Pi \cup \Pi_\mathcal{S})(p(v)) = \mathsf{lfp}((\Pi_p \setminus \Pi_p^*) \cup \{A : \psi(A) \mid A \in \mathcal{A}\})(p(v))$ for every vertex v of $\mathcal{S}$.*

While the previous proposition can be applied to arbitrary GAPs, it becomes more effective for p-monotonic ones, because $I_{\Pi,p}$ is smaller and thus Π_p^* (which is "precomputed" at an initial stage) is larger. In general, the smaller $I_{\Pi,p}$ is, the more effective the proposition above will be.

Example 10. *Consider the following GAP Π:*

$$
\begin{array}{ll}
[r_1] & s(V) : 0.5 \times X \leftarrow p(V) : X \\
[r_2] & p(V) : W \times X \times Y \leftarrow \mathsf{fr}(V, V') : W \wedge p(V) : X \wedge s(V') : Y \\
[r_3] & s(V) : 0.6 \times X \times W \leftarrow \mathsf{fr}(V, V') : W \wedge q(V') : X \\
[r_4] & q(V) : 0.9 \times X \times W \leftarrow \mathsf{fr}(V, V') : W \wedge \mathsf{male}(V') : X
\end{array}
$$

Suppose p is the diffusive property. Clearly, Π is p-monotonic (all rule heads contain monotonic functions), $\Pi_p = \Pi$, and $\Pi_p^* = \{r_3, r_4\}$, as all predicate symbols in the body of r_3 (resp. r_4) are not mutually recursive with p. Proposition 3 says that, for every SN $\mathcal{S}$, we can first compute the interpretation ψ defined as $\mathsf{lfp}(\Pi_\mathcal{S} \cup \Pi_p^*)$. Then, starting from ψ, the GAP consisting only of r_1 and r_2 is evaluated.

Below we present another class of GAPs, called *p-dwindling*. As discussed in the next section, for p-dwindling GAPs our HyperLFP algorithm has a faster convergence to the least fixed point.

Definition 6 (p-dwindling GAP). *Suppose p is a property. A GAP Π is* p-dwindling *iff for every ground rule $A_0 : \mu_0 \leftarrow A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$ in $grd(\Pi)$ s.t. q is the predicate symbol of A_0 and $q \in I_{\Pi,p}$, it is the case that $\mu_0 \leq \mu_i$ for every $1 \leq i \leq n$ s.t. the predicate symbol of A_i is in $I_{\Pi,p}$.*

For the Flickr, Jackson-Yariv, and SIR models (see Appendix A) we consider in our experimental evaluation, we can state the following properties.

Proposition 4. *The Flickr model is p-monotonic and p-dwindling. The Jackson-Yariv model is p-monotonic but not p-dwindling. The SIR model is neither p-monotonic nor p-dwindling.*

5.2. The HyperLFP Algorithm

In this section, we propose an efficient hypergraph-based algorithm, *HyperLFP*, to compute the least fixed point used for diffusion centrality computation.

A *directed hypergraph* is a pair $\langle V, H\rangle$ where V is a finite set of vertices and H is a finite set of directed hyperedges. A hyperedge is a pair $\langle S, t\rangle$ where S is a set of vertices, called *source set*, and t is a vertex, called *target vertex*. Given a hyperedge $h \in H$, $S(h)$ denotes its source set and $t(h)$ denotes its target vertex.

We now define a hypergraph that captures how a property p diffuses through an SN $\mathcal{S}$ according to a GAP Π. The hypergraph does not depend on which vertices have property p in the original SN, but depends only on Π and the structure of $\mathcal{S}$ in terms of edges and vertex properties other than p. *Therefore, given a GAP Π and an SN $\mathcal{S}$, the diffusion hypergraph has to be computed only once and can be used with different assignments of a property p to the vertices of $\mathcal{S}$.* This allows us to save time in computing diffusion centrality which requires computing the least fixed point for different initial assignments of p. However, if the SN changes, the diffusion hypergraph needs to be recomputed. In addition, the hypergraph allows us to eliminate diffusion rules that are useless for computing the least fixed point.

Definition 7 (Enabled Rule). *Consider an SN $\mathcal{S}$, a GAP Π, and a property p. Let $\varphi = \mathsf{lfp}(\Pi_{\mathcal{S}} \cup \Pi^*_p)$. A rule $r \in grd(\Pi - \Pi^*_p)$ is* enabled *iff $\varphi(A) \geq \mu$ for every annotated atom $A : \mu$ in the body of r whose predicate symbol is not in $I_{\Pi,p}$.*

Intuitively, enabled rules are the ground rules that can affect the diffusion of p (directly or indirectly) in the least fixed point computation.

Example 11. *Consider the GAP Π_{hiv} of Example 3 and the SN of Example 1 (cf. Figure 1). In this case, we have $\Pi^*_p = \emptyset$, $\varphi = \mathsf{lfp}(\Pi_{\mathcal{S}})$, and the following ground instance of the second rule belongs to $grd(\Pi - \Pi^*_p)$:*

$$[r] \quad \mathsf{hiv}(d) : 0.4 \times 0.8 \times 0.1 \times 1 \leftarrow \mathsf{fr}(d, a) : 0.8 \wedge \mathsf{sp}(a, c) : 0.1 \wedge \mathsf{hiv}(c) : 1$$

The rule above is enabled as $\varphi(\mathsf{fr}(d, a)) = 0.8$ and $\varphi(\mathsf{sp}(a, c)) = 0.1$. Notice that the atom $\mathsf{hiv}(c)$ does not play any role in determining whether or not the rule is enabled because $\mathsf{hiv} \in I_{\Pi,\mathsf{hiv}}$ (recall that $I_{\Pi,\mathsf{hiv}} = \{\mathsf{hiv}\}$).

Definition 8 (Diffusion Hypergraph). *Consider an SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a GAP Π, and a property p. The hyperedge associated with a ground rule $r \in grd(\Pi)$*

whose head annotated atom is of the form $p'(v) : \mu$ is defined as $\langle \{p''(v_i) \mid p''(v_i) : \mu_i$ is in the body of r and $p'' \in I_{\Pi,p}\}, p'(v)\rangle$ and is denoted by $hedge(r)$. The diffusion hypergraph $\mathcal{H}(\mathcal{S}, \Pi, p)$ *is a triple $\langle N, H, W\rangle$ such that:*

1. *$\langle N, H\rangle$ is a directed hypergraph with $N = \{q(v) \mid q \in I_{\Pi,p}$ and $v \in V\}$, and $H = \{hedge(r) \mid r$ is an enabled ground rule of Π whose head predicate symbol is in $I_{\Pi,p}\}$,*

2. *W is a function such that for each $h \in H$ and matrix $M[1...|I_{\Pi,p}|][1..|V|]$ of real values in $[0, 1]$, $W(h, M)$ is the head annotation of the ground rule r satisfying the following two properties: (i) $hedge(r) = h$, and (ii) for every atom $q(v)$ appearing in S(h), $M[q][v]$ is equal to the annotation of $q(v)$ in r.*

Example 12. *Figure 2 shows the diffusion hypergraph for the GAP Π_{hiv} of Example 3 and the social network of Example 1. Since $I_{\Pi,\mathsf{hiv}} = \{\mathsf{hiv}\}$ and $V = \{a, b, c, d\}$, the nodes of the diffusion hypergraph are $N = \{\mathsf{hiv}(a), \mathsf{hiv}(b), \mathsf{hiv}(c), \mathsf{hiv}(d)\}$. The hyperedges are derived from enabled ground rules. Consider for instance the enabled ground rule r from Example 11. This rule corresponds to the hyperedge $hedge(r) = \langle\{\mathsf{hiv}(c)\}, \mathsf{hiv}(d)\rangle$ because $\{\mathsf{hiv}(c)\}$ is the set of atoms in the body of r s.t. the predicate symbols belongs to $I_{\Pi,\mathsf{hiv}}$ and $\mathsf{hiv}(d)$ is the atom in the head of the rule. The hyperedge labels represent the function W, i.e. the label on a hyperedge is the function in the annotation of the head atom of the corresponding rule. For instance, for the rule r from Example 11, the label on the hyperedge $hedge(r)$ is $0.4 \times (0.8 \times 0.1) \times \mathsf{hiv}(c)$ (where 0.4 is a constant in the function and the values 0.8 and 0.1 are precomputed) and represents how we have to update the value for $\mathsf{hiv}(d)$ once that the value of $\mathsf{hiv}(c)$ has changed.*

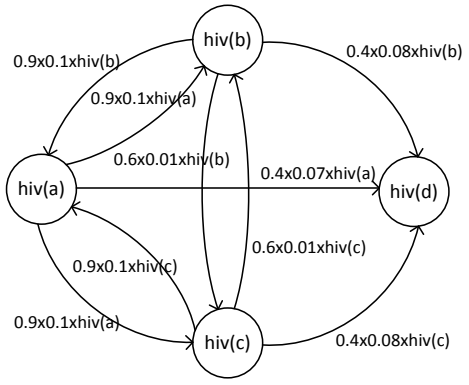

Figure 2: A diffusion hypergraph.

Algorithm 1 HyperLFP

Input: For a social network $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a GAP Π, and a property p,
two matrices $U[1...|I_{\Pi,p}|][1..|V|]$ and $M[1...|I_{\Pi,p}|][1..|V|]$,
a max-heap $Heap$, the diffusion hypergraph $\mathcal{H}(\mathcal{S}, \Pi, p) = \langle N, H, W \rangle$
Output: $\mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}})(q(v))$ for all $v \in V$ and $q \in I_{\Pi,p}$

```
 1: C ← copy of M;
 2: M' ← copy of M;
 3: while Heap ≠ ∅ do
 4:     Heap' ← ∅;
 5:     while Heap ≠ ∅ do
 6:         ⟨h, w⟩ ← deleteMax(Heap);
 7:         Let q(v) = t(h);
 8:         if M'[q][v] < w then
 9:             M'[q][v] ← w;
10:             for each h' ∈ U[q][v] do
11:                 w' ← W(h', M');
12:                 Let q'(v') = t(h');
13:                 if w' > C[q'][v'] then
14:                     C[q'][v'] ← w';
15:                     if Π is p-monotonic then
16:                         Add ⟨h', w'⟩ to Heap;
17:                     else
18:                         Add ⟨h', w'⟩ to Heap';
19:     Heap ← Heap';
20: return M';
```

The rough idea of the HyperLFP algorithm (Algorithm 1) is that hyperedges that propagate a value greater than zero are kept in a max-heap and those propagating higher values are visited first; the max-heap is updated as propagation unfolds.

For all $q \in I_{\Pi,p}$ and $v \in V$, $M[q][v]$ is the initial value of the ground atom $q(v)$, $U[q][v]$ keeps track of the hyperedges having $q(v)$ in their source set, and $M'[q][v]$ is the current assignment to $q(v)$. Specifically, $M'[q][v]$ is initially set to M and then iteratively updated by the algorithm. C keeps track of the highest values propagated by hyperedges that were added to $Heap$. At each iteration of the **while** loop on lines 5–18, a pair $\langle h, w \rangle$ with maximum w is retrieved from $Heap$. If $M'[q][v]$ is less than w, it is set to w, otherwise another hyperedge is retrieved from $Heap$. If w is assigned to $M'[q][v]$, then hyperedges that can be affected by this are inspected (**for each** loop on lines 10–18). Only the hyperedges having $t(h)$ in the source set are inspected. For each of them, if no hyperedge added to $Heap$ propagated a higher value (line 13), then if Π is p-monotonic the hyperedge is added to $Heap$ (along with the value it propagates), otherwise it is added to $Heap'$. The reason for this is that when Π is p-monotonic we can retrieve hyperedges in descending order of their weights even if their values were derived from different

iterations of $\mathbf{T}_\Pi$. However, if Π is not p-monotonic, the iterations of $\mathbf{T}_\Pi$ must be executed one after the other without mixing the values derived at each of them. So, when Π is not p-monotonic, hyperedges are retrieved in descending order of their weight for each iteration of $\mathbf{T}_\Pi$. When $Heap$ is empty, $Heap'$ is assigned to $Heap$. M' is returned if both heaps are empty.

If a GAP Π is p-dwindling and p-monotonic, then HyperLFP ensures that when a value w is assigned to a ground atom $q(v)$, w is the final value for $q(v)$ in the least fixed point; hence the hyperedge that propagated w as well as any other hyperedge having $q(v)$ as a target atom no longer needs to be considered in order to see if a new higher value can be assigned to $q(v)$.

Proposition 5. *The worst-case time complexity of Algorithm HyperLFP is* $O(|N|+\frac{1}{\alpha}\cdot|H|\cdot(\log|H|+U_{max}\cdot(S_{max}+\log|H|)))$*, where* $U_{max}=\max_{v\in V,q\in I_{\Pi,p}}\{|\{h\mid h\in H\wedge q(v)\in S(h)\}|\}$*,* $S_{max}=\max_{h\in H}\{|S(h)|\}$*, and* α *is the minimum value obtained at line 9 for* $(w-M'[q][v])$*.*

5.3. The HyperDC Algorithm

The HyperDC algorithm (Algorithm 2) initializes M_{in}, U, and $Heap_{in}$ by calling Algorithm 3 (line 1), and then uses them to compute $\mathsf{lfp}(\Pi\cup\Pi_{\mathcal{S}})$ (lines 2–4). After that, the diffusion centrality of each vertex is computed (lines 6–25). Specifically, the value of atom $p(v)$ is incorporated into M_{in}, and $Heap$ is updated accordingly (lines 7–16). Then, the least fixed point is computed using the updated M_{in} and $Heap$ (lines 17–18). Finally, the diffusion centrality of vertex v is computed and the value of $p(v)$ in M_{in} is restored (lines 19–25). In this last step, Proposition 1 is leveraged. In fact, if $p\in\mathsf{VL}(v')$, the positive summand of the definition of DC has already been computed in lines 2–4 and what is being computed is the negative summand. In this case, the positive summand is equal to $sum_p-F[p][v]$ and the negative summand is equal to sum'_p, so the diffusion centrality of vertex v is $(sum_p-F[p][v])-sum'_p$. If $p\notin\mathsf{VL}(v')$, the negative summand has been computed already in lines 2–4. In this case, the diffusion centrality of vertex v is $sum'_p-(sum_p-F[p][v])$.

Proposition 6. *The worst-case time complexity of Algorithm HyperDC is* $O(|V|\cdot(|N|+\frac{1}{\alpha}\cdot|H|\cdot(\log|H|+U_{max}\cdot(S_{max}+\log|H|))))$*, where* $U_{max}=\max_{v\in V}\{|\{h\mid h\in H\wedge v\in S(h)\}|\}$*,* $S_{max}=\max_{h\in H}\{|S(h)|\}$*, and* α *is defined as in Proposition 5.*

A brief note is in order about how the techniques in this section yield better scalability. Compared to our past work [31], HyperDC is approximately 100 times faster (this is the average speedup obtained in our experimental evaluation). In particular, the U and $Heap_{init}$ structures used in HyperDC are built just once. This yields a speedup of approximately 5x. In addition, the filtering based on Proposition 3 yields a further speedup of 20x, leading to a total speedup of 100x.

Algorithm 2 HyperDC

Input: An SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a GAP Π, a property p, the diffusion hypergraph $\mathcal{D} = \mathcal{H}(\mathcal{S}, \Pi, p) = \langle N, H, W \rangle$
Output: $\{\langle v, \mathsf{dc}(v) \rangle \mid v \in V\}$

$\langle M_{in}, U, Heap_{in} \rangle \leftarrow \mathsf{Init}(\mathcal{S}, I_{\Pi,p}, \mathcal{D})$;
$Heap \leftarrow$ copy of $Heap_{in}$;
$F \leftarrow \mathsf{HyperLFP}(U, M_{in}, Heap, \mathcal{D})$;
$sum_p \leftarrow \sum_{v \in V} F[p][v]$;
$Result \leftarrow \emptyset$;
for each $v \in V$ **do**
  $Heap \leftarrow$ copy of $Heap_{in}$;
  if $p \in \mathsf{VL}(v)$ **then**
    $M_{in}[p][v] \leftarrow 0$;
  else
    $M_{in}[p][v] \leftarrow 1$;
  for each $h \in U[p][v]$ **do**
    remove $\langle h, w \rangle$ from $Heap$;
    Let $p'(v') = t(h)$;
    if $W(h, M_{in}) > M_{in}[p'][v']$ **then**
      Add $\langle h, W(h, M_{in}) \rangle$ to $Heap$;
  $M \leftarrow \mathsf{HyperLFP}(U, M_{in}, Heap, \mathcal{D})$;
  $sum'_p \leftarrow \sum_{v' \in V, v' \neq v} M[p][v']$;
  if $p \in \mathsf{VL}(v)$ **then**
    $\mathsf{dc}(v) \leftarrow (sum_p - F[p][v]) - sum'_p$;
    $M_{in}[p][v] \leftarrow 1$;
  else
    $\mathsf{dc}(v) \leftarrow sum'_p - (sum_p - F[p][v])$;
    $M_{in}[p][v] \leftarrow 0$;
  Add $\langle v, \mathsf{dc}(v) \rangle$ to $Result$;
return $Result$;

Algorithm 3 Init

Input: A social network $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a p-interfered predicate set $I_{\Pi,p}$, a diffusion hypergraph $\mathcal{H} = \langle N, H, W \rangle$.
Output: $M[1..|I_{\Pi,p}|][1..|V|]$, $U[1..|I_{\Pi,p}|][1..|V|]$, $Heap$

$n = |I_{\Pi,p}|$; $m = |V|$;
$M[1..n][1..m]$; $U[1..n][1..m]$;
$Heap \leftarrow \emptyset$;
for each $v \in V$, $q \in I_{\Pi,p}$ **do**
  $M[q][v] \leftarrow 0$, $U[q][v] \leftarrow \emptyset$;
for each $v \in V$, $q \in \mathsf{VL}(v) \cap I_{\Pi,p}$ **do**
  $M[q][v] \leftarrow 1$;
for each $h \in H$ **do**
  Add $\langle h, W(h, M) \rangle$ to $Heap$;
  for each $q(v) \in S(h)$ **do**
    $U[q][v] \leftarrow U[q][v] \cup \{h\}$;
return $\langle M, U, Heap \rangle$;

6. CBAF Algorithm for Approximating kDCP

The goal of this section is to develop our Coarsened Back and Forth (CBAF) algorithm to approximately compute the top-k diffusion centrality vertices in huge social networks where it is not feasible to compute DC for all vertices. The basic idea is to coarsen the original social network $\mathcal{S}$ into a smaller network $\mathcal{S}'$ which tries to preserve the "diffusive behavior" of $\mathcal{S}$. The top-k vertices are then computed over $\mathcal{S}'$ and the solution is mapped back to a subgraph of $\mathcal{S}$ on which we again compute the top-k vertices. Given a social network $\mathcal{S}$, a GAP Π, and a property p, CBAF performs the following steps: *(1)* Compute the filtering $\mathcal{S}'$ of $\mathcal{S}$; *(2)* Coarsen $\mathcal{S}'$ by merging its vertices so as to obtain a new social network $\mathcal{S}_C$ and a mapping from the vertices of $\mathcal{S}'$ to the vertices of $\mathcal{S}_C$; *(3)* Compute a set T_C of

top-k vertices of $\mathcal{S}_C$; *(4)* Use T_C to compute an approximate set of top-k vertices of $\mathcal{S}$. The first step has already been described in Section 5.1, while the other steps will be detailed in the rest of this section.

6.1. Social Network Coarsening

This section proposes a new semantical coarsening technique that reduces network size by merging together vertices that are similar, while trying to preserve the structural and semantic properties of the network w.r.t. a property p. The coarsening process involves the following issues: *(i)* how to select "similar" vertices to be merged together, *(ii)* how to assign properties to a merged vertex and to its edges after merging, and *(iii)* how to compute edge weights between merged vertices. These issues are addressed in the following.

Vertex Similarity. CBAF can work with any function to determine whether two vertices are similar. Throughout this paper, we assume that given any vertex v, there is a set $SIM(v) \subseteq V$ of vertices that are similar to it. Function SIM can obviously be defined in many ways. We provide one such way that takes the diffusion model for p into account (as well as the social network structure).

Consider an SN $\mathcal{S}$, a GAP Π, and a property p. Given two vertices u and v of $\mathcal{S}$, we write $u \sim v$ iff u and v activate the same set of rules of Π (cf. Definition 2 for the the notion of "activation"). Using this equivalence relation, we define $SIM_\Pi(v) = \{u \in V \mid u \sim v\}$. Obviously, many other definitions are also possible, but this is the one used in our experiments.

Vertex Merging. We now define how to merge similar vertices. When a set of vertices is merged into a new vertex v, we have to specify vertex properties of v, as well as associated edge properties/weights.

Definition 9 (Vertex properties merging). *Let $\mathcal{S}$ be an SN and $\{v_1, \ldots, v_n\}$ be a subset of the vertices of $\mathcal{S}$ (to be merged). For each $p \in \mathsf{VP}$, let $g_p : \{0,1\}^n \to \{0,1\}$ be any associative and commutative function. Then, we define:*

$$mergeVP(\{v_1, \ldots, v_n\}, \mathsf{VL}) = \{p \in \mathsf{VP} \mid g_p(\vartheta(v_1, p), \ldots, \vartheta(v_n, p)) = 1\}$$

where $\vartheta(v_i, p) = 1$ if $p \in \mathsf{VL}(v_i)$, otherwise $\vartheta(v_i, p) = 0$.

The above definition assumes the existence of a function g_p that takes the values (0 or 1) of the p-property of the vertices being merged and combines them into a single 0 or 1 value denoting whether the merged vertex has property p or not.

Some examples for the function g_p can be computing the property intersection or union (i.e., taking the minimum or maximum value across the x_i's). We can also use a "majority" function where the new vertex has the property p if the majority of the vertices being merged have the property p.

We now address the problem of assigning edges to the new merged vertex. Let $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$ be an SN and $V' = \{v_1, \ldots, v_n\}$ be a subset of V to be merged into a new vertex v'. For each edge $(v, u, ep) \in E$ such that either $v \in V'$ and $u \in V \setminus V'$ or $u \in V'$ and $v \in V \setminus V'$, the new vertex v' will have the outgoing edge (v', u, ep) if $v \in V'$, or the incoming edge (v, v', ep) if $u \in V'$.

Edge weighting. We now define how to assign a weight to an arbitrary set of edges (having the same label) in $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$.

Definition 10 (Edge weighting). *Let $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$ be an SN and $ep \in \mathsf{EP}$. Moreover, let $\{e_1, \ldots, e_m\} \subseteq (V \times V \times \{ep\})$ be an arbitrary set of edges between vertices in V labeled with ep, and $g_{ep} : (0, 1]^m \to (0, 1]$ be any associative and commutative function. Then, we define*

$$weight(\{e_1, \ldots, e_m\}) = g_{ep}(\omega^*(e_1), \ldots, \omega^*(e_m))$$

where $\omega^*(e) = \omega(e)$ if $e \in E$, otherwise $\omega^*(e) = 0$.

Given two sets of vertices V' and V'' and an edge property $ep \in \mathsf{EP}$, we define the set $possEdges(V', V'', ep)$ as the set of all possible edges having property ep that can exist from vertices in V' to vertices in V'', i.e. $possEdges(V', V'', ep) = \{(v', v'', ep) \mid v' \in V' \wedge v'' \in V''\}$. Finally, if v' is a new vertex obtained by merging a set V' of vertices and $e' = (v, v', ep)$ is a new incoming edge, then its weight is computed as $weight(possEdges(\{v\}, V', ep))$, while, if $e'' = (v', v, ep)$ is a new outgoing edge, then its weight is computed as $weight(possEdges(V', \{v\}, ep))$.

Social Network Coarsening. We are now ready to give the definition of social network coarsening.

Definition 11 (Social network coarsening). *Let $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$ be an SN and θ be a real number in $(0, 1]$ called the* contraction factor. *A* coarsening *of $\mathcal{S}$ is an SN $\mathcal{S}' = (V', E', \mathsf{VL}', \omega')$ together with an onto mapping $\pi : V \to V'$ s.t.:*

- $|V'| \leq \theta \cdot |V|$;
- $E' = \{\langle \pi(v_1), \pi(v_2), ep\rangle \mid \langle v_1, v_2, ep\rangle \in E \wedge \pi(v_1) \neq \pi(v_2)\}$;
- $\mathsf{VL}'(v') = mergeVP(\{v \in V \mid \pi(v) = v'\}, \mathsf{VL})$, *for* $v' \in V'$;
- $\omega'(e') = weight(possEdges(\pi^{-1}(v_1), \pi^{-1}(v_2), ep))$, *for* $e' = \langle v_1, v_2, ep\rangle \in E'$.

Coarsening an SN $\mathcal{S}$ yields a new SN $\mathcal{S}'$ together with a mapping π from the vertices of $\mathcal{S}$ to the vertices of $\mathcal{S}'$ such that *(i)* the number of vertices of $\mathcal{S}'$ is smaller than that of $\mathcal{S}$ by a factor θ; *(ii)* if there was an edge between two vertices u and v

in $\mathcal{S}$, and u has been merged into a new vertex u' in $\mathcal{S}'$ while v has been merged into a new vertex $v' \neq u'$ in $\mathcal{S}'$, then there is an edge between u' and v' in $\mathcal{S}'$; *(iii)* the vertex properties of each merged vertex of $\mathcal{S}'$ are assigned using function $mergeVP$; and *(iv)* the weights of the edges of $\mathcal{S}'$ are assigned by function $weight$. In the following, given $v' \in V'$, we denote by $\pi^{-1}(v')$ the set $\{v \in V \mid \pi(v) = v'\}$.

Algorithm 4 is a general algorithm for coarsening a social network. It starts by initializing the mapping function π as the identity function. In each iteration, a vertex v in the current SN is randomly selected, and the set of vertices to be merged with it is computed by the function $getMergingSet$. This function computes the set U' of all v's neighbors that are similar to v according to the similarity function SIM received as input, and returns a subset M of U' whose size is a percentage ρ

Algorithm 4 CoarsenSN

Input: A social network $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a GAP Π, a contraction factor $\theta \in (0, 1]$,
a similarity function SIM, a neighbors threshold $\rho \in (0, 1]$,
a vertex properties merging function $mergeVP$, and an edge $weight$ function.
Output: A coarsening $\mathcal{S}' = (V', E', \mathsf{VL}', \omega')$ and $\pi : V \to V'$ of $\mathcal{S}$.

$\mathcal{S}' = (V', E', \mathsf{VL}', \omega') = (V, E, \mathsf{VL}, \omega)$;
$\pi(v) \leftarrow v$ for all $v \in V$;
$R \leftarrow V'$, $cont \leftarrow true$;
while $((|V'| > \theta \cdot |V|) \wedge cont)$ **do**
  Randomly select a vertex $v \in R$;
  $M \leftarrow getMergingSet(v, \mathcal{S}', \Pi, SIM, \rho)$;
  $R \leftarrow R \setminus (M \cup \{v\})$;
  if $(M \neq \emptyset)$ **then**
    $\mathsf{VL}'(v) \leftarrow mergeVP(\{v\} \cup M, \mathsf{VL}')$;
    $\pi(v') \leftarrow v$ for all $v' \in \pi^{-1}(v)$;
    $\pi(v') \leftarrow v$ for all $v' \in M$;
    $V' \leftarrow V' \setminus M$;
    $\langle E', \omega' \rangle \leftarrow UpdateEdges(E', \omega', v, M, weight, \pi)$;
  if $(R = \emptyset)$ **then**
    $cont \leftarrow false$;
return $\langle \mathcal{S}', \pi \rangle$;

getMergingSet$(v, \mathcal{S}', \Pi, SIM, \rho)$
  $U =$ all neighbors of vertex v in $\mathcal{S}'$;
  $U' = U \cap SIM(v)$;
  Randomly select a set $M \subseteq U'$ of size $|M| = \rho \cdot |U'|$;
  return M;

updateEdges$(E', \omega, v, M, weight, \pi)$
  $E'' = \{\langle \pi(v_1), \pi(v_2), ep \rangle \mid \langle v_1, v_2, ep \rangle \in E' \wedge \pi(v_1) \neq \pi(v_2)\}$;
  for each $e' = \langle \pi(v_1), \pi(v_2), ep \rangle \in E''$ **do**
    $\omega'(e') \leftarrow weight(possEdges(\pi^{-1}(v_1), \pi^{-1}(v_2), ep))$;
  return $\langle E'', \omega' \rangle$;

of $|U'|$. If the set of vertices M is not empty, vertex v is merged with M, otherwise a new vertex v is randomly selected. If v is merged with M, the new vertex properties of v are computed using the $mergeVP$ function, the mapping π is updated, and the set of edges is updated by the function $UpdateEdges$. This function first computes the new set of edges so that if there was an edge between a vertex in $M \cup \{v\}$ and another vertex u, now there is an edge between v and u, while the new edge weight is computed by using the function $weight$ received in input. The algorithm's iterations stop when either the number $|V'|$ of vertices in the current SN is less than or equal to $\theta \cdot |V'|$, where θ is the contraction factor establishing the desired size of the coarsened network, or it is not possible to merge any other vertex. Algorithm 4 can be used with our similarity function SIM_{Π}.

6.2. Adapting the DC Definition to the Coarsened Network

We now adapt diffusion centrality to the case of coarsened networks. This intermediate step will later be used in the computation of diffusion centrality for vertices of the original network. When a set of vertices M in the original network $\mathcal{S}$ is merged into a single vertex v' in the coarsened one, v' represents the network $\mathcal{S}_{v'}$ consisting of all vertices in M and all edges among them from the original network. Thus, even if two vertices v' and v'' have the same diffusion centrality on the coarsened network, the actual diffusion of the property of interest in the original network among the vertices belonging to $\mathcal{S}_{v'}$ and $\mathcal{S}_{v''}$ may be different and depends on the properties of the two subnetworks $\mathcal{S}_{v'}$ and $\mathcal{S}_{v''}$ (number of vertices, edges, etc.). To take this into account, we assign a weight to each vertex v' in the coarsened network representing the importance of v' w.r.t. to the original network.

Definition 12 (Diffusion centrality on a coarsened network). *Let $\mathcal{S}$ be a social network, $\mathcal{S}_C = (V_C, E_C, \mathsf{VL}_C, \omega_C)$ be a coarsening of $\mathcal{S}$ with vertex mapping π, and mvw be a function assigning a weight to each vertex in V_C. Then, the diffusion centrality of a vertex v in the coarsened SN is defined as*

$$\begin{aligned} \mathsf{dc}'_{\Pi,p,\mathcal{S}_C}(v) \quad = \quad & \Sigma_{v' \in V_C \setminus \{v\}} \big(mvw(v') \cdot \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}_C \oplus p(v)})(p(v')) \big) - \\ & \Sigma_{v'' \in V_C \setminus \{v\}} \big(mvw(v'') \cdot \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}_C \ominus p(v)})(p(v'')) \big) \end{aligned}$$

Function mvw, which we call *merged vertex weight* function, can be defined in several ways. Below we provide three alternative definitions. Consider an SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, and let $\mathcal{S}_C$ and π be a coarsening of $\mathcal{S}$. Given a vertex v of $\mathcal{S}_C$,

- $mvw_0(v) = 1$. In this case, the original definition of DC is used.
- $mvw_1(v) = |V_v| \times \frac{|E_v|}{|V_v|^2 - |V_v|} = \frac{|E_v|}{|V_v| - 1}$,
  where $V_v = \pi^{-1}(v)$ and $E_v = \{\langle v_1, v_2, ep \rangle \mid v_1, v_2 \in V_v \wedge \langle v_1, v_2, ep \rangle \in$

$E\}$. Thus, the weight of v is given by the number of vertices in the original SN that were merged into v multiplied by the density of $\mathcal{S}_v$. The idea is that if $\mathcal{S}_v$ has high density and many vertices, the weight of v should be high.

- $mvw_2(v) = \ln(mvw_1(v)) + 1$. Here, we consider the fact that the diffusion of the property p can rapidly decrease according to the distance of vertices in V_v from the diffusion source vertex v.

6.3. CBAF: Approximately Solving the kDCP Problem

In this section, we show how to approximately compute the top-k diffusion centrality vertices in a social network. First, we introduce some definitions.

Definition 13 (Induced social network). *Given an SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, the SN* induced *from $\mathcal{S}$ by a set of vertices $V_I \subseteq V$ is $\mathcal{S}_I = (V_I, E_I, \mathsf{VL}_I, \omega_I)$, where $E_I = \{\langle v_1, v_2, ep\rangle \mid \langle v_1, v_2, ep\rangle \in E \wedge v_1, v_2 \in V_I\}$, $\mathsf{VL}_I(v) = \mathsf{VL}(v)$ for all $v \in V_I$, and $\omega_I(e) = \omega(e)$ for all $e \in E_I$.*

Given an SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a set of vertices $T \subseteq V$, and a positive integer d, we denote by $nbrs(T, d, \mathcal{S})$ the set of the neighbors of the vertices in T at a distance no greater than d. If $d = 0$, then $nbrs(T, d, \mathcal{S}) = \emptyset$.

We are now ready to present our CBAF algorithm shown in Algorithm 5. CBAF takes as input a social network $\mathcal{S}$, a GAP Π, a property p, an integer k, and a set of options $opts$ as described in Table 2. It returns an (approximate) set of top-k diffusion centrality vertices over $\mathcal{S}$. The first step of the algorithm filters out the original SN $\mathcal{S}$ by removing all unnecessary vertices, obtaining the network $\mathcal{S}'$ (line 1). Then $\mathcal{S}'$ is coarsened into a smaller social network $\mathcal{S}_C$ (line 2), and the exact set T_C of top-k vertices over $\mathcal{S}_C$ is computed (line 3) by running HyperDC to find the diffusion centrality of all vertices in $\mathcal{S}'$ and choosing the top-k. At this point the set T_C is extended with its neighbors in $\mathcal{S}_C$ at a distance no greater than d_C, in order to limit the bias of the vertices in T_C in the following computation (line 4). Next, the above set of vertices is mapped back to the vertices of the original SN $\mathcal{S}$ and is extended with its neighbors on $\mathcal{S}$ at a distance no greater than d_I obtaining the set of vertices V_I (line 5)—with a slight abuse of notation, we use $\pi^{-1}(T_C)$ to denote $\cup_{v \in T_C} \pi^{-1}(v)$. After that, the social network $\mathcal{S}_I$ induced from the vertices in V_I on $\mathcal{S}$ is computed (line 6), and the algorithm returns the approximate set of top-k vertices over $\mathcal{S}$ as the exact set of top-k diffusion centrality vertices computed over $\mathcal{S}_I$ (line 7). Note that CBAF first evaluates diffusion centrality of all vertices in the coarsened network $\mathcal{S}_C$ (which is typically small) and then diffusion centrality of all vertices in a subgraph of the original network corresponding to the neighborhood of vertices associated with the solution found

Algorithm 5 CBAF (Approximate Top-k)

Input: An SN $\mathcal{S} = (V, E, \mathsf{VL}, \omega)$, a GAP Π, a property p, an integer k, a set of options $opts$ as described in Table 2.
Output: An approximate set of top-k vertices.
1: $\mathcal{S}' = (V', E', \mathsf{VL}', \omega') \leftarrow networkFiltering(\mathcal{S}, \Pi, p)$
2: $\langle \mathcal{S}_C = (V_C, E_C, \mathsf{VL}_C, \omega_C), \pi \rangle \leftarrow CoarsenSN(\mathcal{S}', \Pi, opts)$
3: $T_C \leftarrow computeTopK(\mathcal{S}_C, k, mvw)$ % Use HyperDC
4: $T_C = T_C \cup nbrs(T_C, d_C, \mathcal{S}_C)$
5: $V_I = \pi^{-1}(T_C) \cup nbrs(\pi^{-1}(T_C), d_I, \mathcal{S})$
6: $\mathcal{S}_I = (V_I, E_I, \mathsf{VL}_I, \omega_I) \leftarrow$ SN induced from $\mathcal{S}$ by the vertices in V_I
7: **return** $computeTopK(\mathcal{S}_I, k, mvw_0)$

$\theta \in (0, 1]$	contraction factor	mvw	merged vertex weight function
SIM	similarity function	d_C	neighbors distance for extending the set T_C
$\rho \in (0, 1]$	neighbors threshold		
$mergeVP$	vertex properties merging function	d_I	neighbors distance for extending the set $\pi^{-1}(T_C)$
$weight$	edge weight function		

Table 2: Input options $opts$ for Algoritm 5.

in line 3 of the CBAF algorithm. This is typically a small fraction of the vertices in the original social network $\mathcal{S}$.

7. Experimental Evaluation

This section contains a detailed report on our experiments.

1) Exact Computation with HyperDC. We compared the runtime and spread generated by diffusion centrality against classical centrality measures (Section 7.2), using networks with up to around 265K vertices and 440K edges.

2) Approximate Computation with CBAF. We tested scalability, runtime, and spread of CBAF with networks of up to 2M vertices and 20M edges (Section 7.3).

3) Comparing spread of memes by high DC vertices against low DC vertices. In order to test whether diffusion centrality captured real spread, we ran a test with MemeTracker data where we knew who had initiated a meme. We tested the hypothesis that vertices with high centrality would be more influential than those with low centrality according to diffusion centrality as well as classical centrality measures. Kolmogorov-Smirnov tests validate both findings: *(i)* high DC vertices diffuse memes better than low DC vertices, and *(ii)* diffusion centrality does a better job explaining real meme diffusion than classical centrality measures.

We implemented HyperLFP (Algorithm 1), HyperDC (Algorithm 2), and CBAF (Algorithm 5) in Java. To compute degree, eigenvector, PageRank, closeness,

and betweenness centrality we used the Java Universal Network/Graph Framework (JUNG)[3]. All experiments were run on an Intel Xeon @ 2.40 GHz, 24 GB RAM.

7.1. Experimental Setup

Social Networks. Our experiments used several real-world social networks summarized in Table 3. The networks are taken from the *Stanford Large Network Dataset Collection* [38]. We also considered an additional online game dataset called STEAM [6], which contains friendship relations (represented as directed edges) between players (vertices). Each player has several vertex properties and we selected country, group(s), games played, and total time played per game. We extracted 10 subnetworks from the whole STEAM dataset by choosing different games and selecting, for each game, all players who played and all edges between the players. The features of the extracted networks are reported in Table 4.

Network	Description	Type	# Vertices	# Edges	Avg. Degree	Density
BlogCatalog	friendship	undirected	10,312	333,983	64.78	6.28E-03
e-mail Enron	e-mail communications	undirected	36,692	183,831	10.02	2.73E-04
Douban	friendship	undirected	154,907	654,188	8.45	5.45E-05
wiki-Vote	Wikipedia vote relationships	directed	7,115	103,689	14.57	2.05E-03
soc-Epinions	Trust relationships	directed	75,879	508,837	6.71	8.84E-05
email-EuAll	e-mail communications	directed	265,214	420,045	1.58	5.97E-06

Table 3: Non-Game Social Networks used in the experiments.

Social Network	# Vertices	# Edges	Avg. Degree	Density
GAME8690	4,083	21,447	5.25	1.29E-03
GAME50510	28,872	115,254	3.99	1.38E-04
GAME6850	52,879	164,702	3.11	5.89E-05
GAME1500	66,571	303,240	4.56	6.84E-05
GAME24420	82,377	437,554	5.31	6.45E-05
GAME11450	89,942	327,258	3.64	4.05E-05
GAME17300	122,467	441,657	3.61	2.94E-05
GAME420	1,307,335	12,431,564	9.51	7.27E-06
GAME220	2,030,579	20,596,331	10.14	5.00E-06

Table 4: STEAM networks used in the experiments.

Diffusion Models. We ran experiments with a conditional probability model (hereafter referred to as the "Flickr model" [13]), the Jackson-Yariv tipping model [28], and the SIR model of disease spread [2]. We generalized these diffusion models by adding an additional condition $q(u) : \mu$ in rule bodies. When all vertices have the property q, then we have the original diffusion models. The q-condition determines when the diffusion process can happen. More specifically, for the Flickr

[3] jung.sourceforge.net

model, only vertices satisfying property q can spread the diffusive property, while for the Jackson-Yariv and SIR models, only vertices satisfying property q can get the diffusive property. In the Flickr model, q is used to represent the "willingness" of a person to share her/his information (and thus influence other people). In the Jackson-Yariv and SIR models, q is used to represent a precondition for a vertex to adopt a behavior or get a disease. Thus, property q is useful when expressing the fact that only some vertices with some characteristics (modeled by q) can infect or be infected by the diffusive property. Note that edges originating from nodes that do not have property q can still play a role in the diffusion depending on the diffusion model. In our experiments, we compared DC with other classical centrality measures by varying the percentage of vertices in the network with property q. The diffusion models are reported in Appendix A.

7.2. Diffusion Centrality vs. Classical Centrality Measures

As described earlier, the Flickr, Jackson-Yariv, and SIR models assume that some set of vertices in the network have property p and that only vertices satisfying some property q can spread (in the case of the Flickr model) or receive (in the case of Jackson-Yariv and SIR models) p. For STEAM data, we were able to use known properties of the vertices for q but we were not able to do this for the other datasets.

Before getting into the details of our experimental evaluation, we summarize the high level conclusion of these comparative experiments (experiments on scalability using the CBAF algorithm are in the next subsection): *(i)* the runtime of HyperDC is better than betweenness and closeness centrality and comparable with the others, and *(ii)* the spread achieved by diffusion centrality is almost always better than those achieved by classical centrality measures.

7.2.1. STEAM Data

We used all but the last two (very large) STEAM data sets (Table 4) for the comparative analysis as some classical centrality measures could not finish the computation even on the first seven STEAM data sets. Consistent with the data, we set the percentage δ_p of vertices that *initially* have property p to 0% and varied the percentage δ_q of vertices having property q by using real properties of vertices in the STEAM data. In the STEAM data, q was taken to be the playing time of users in the game and was assigned as follows: we sorted the players in descending order according to playing time and assigned q to the first δ_q users. We varied $\delta_q \in \{1\%, 2\%, 3\%, 4\%, 5\%, 10\%, 15\%, 20\%, 25\%, 30\%\}$. The last two STEAM subnetworks, which are substantially larger, were used for the evaluation of the CBAF algorithm (cf. Section 7.3).

Runtime. We compared the time to compute DC w.r.t. the three diffusion models against the time to compute classical centrality measures. Figure 3 shows how the

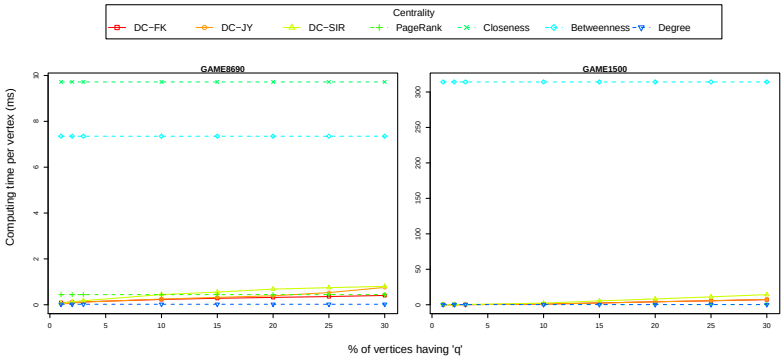

Figure 3: STEAM Data: Runtimes (ms) averaged per vertex when $\delta_q \in \{1\%, 2\%, 3\%, 4\%, 5\%, 10\%, 15\%, 20\%, 25\%, 30\%\}$.

runtimes vary w.r.t. δ_q for two representative STEAM games.

Of the 7 STEAM games we tested in this experiment, closeness centrality computation finished only in the smallest case (GAME8690). HyperDC is much faster than betweenness and closeness centrality for all the STEAM networks and for all values of δ_q. Even when $\delta_q = 30\%$, the runtime of HyperDC is still low.

HyperDC is faster than PageRank over networks GAME8690 and GAME50510 (the two smallest STEAM networks) for the Flickr and Jackson-Yariv models, and also faster than degree over network GAME50510 when δ_q is low.[4] However, when the number of vertices increases (i.e., for the STEAM datasets other than GAME8690 and GAME50510), HyperDC becomes slower than PageRank and degree for all three diffusion models. For instance, in GAME1500, a crossover occurs when $\delta_q = 3\%$ for the SIR model but not for the Jackson-Yariv model. There are two main reasons for that: *(i)* the size of $I_{\Pi,p}$ is larger for the SIR model (recall that the smaller $I_{\Pi,p}$ is, the more effective the optimization of Proposition 3 will be), and *(ii)* the SIR model is neither p-monotonic nor p-dwindling (and thus optimizations are less effective, cf. Section 5.1).

Runtime of HyperDC Tends to be Linear. We note from Proposition 6 that in the worst case, the complexity of HyperDC is nonlinear. In order to assess actual runtime characteristics of HyperDC, we ran an experiment on the STEAM data when the number of vertices increases and as δ_q varies. We used the networks

[4]Though it may seem surprising that HyperDC sometimes beats degree centrality, the reason for this is that when δ_q is low (below 3%), HyperDC only needs to compute diffusion centrality for a small number of vertices. However, when δ_q is larger, this is not the case.

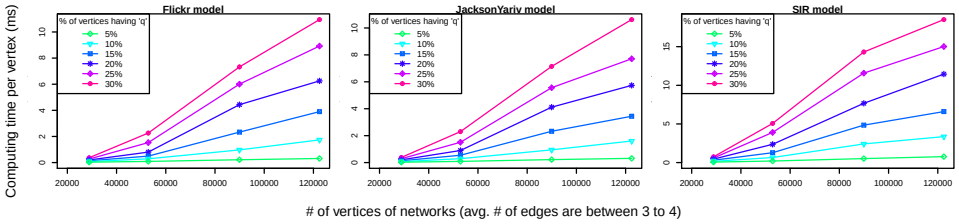

Figure 4: STEAM Data: Average Runtime (per vertex) of HyperDC as the number of vertices is increased and for different δ_q using Flickr, Jackson-Yariv, and SIR models.

from GAME 50510, 6850, 11450, and 17300 which all have a similar average degree (in the 3-4 range). Figure 4 shows the number of vertices in the network on the x-axis and the average runtime per vertex on the y-axis. Different curves show varying values of δ_q in the range 5-30%. We see that irrespective of the diffusion model used and the value of δ_q, HyperDC runs in linear time in practice.

Higher values of δ_q lead to higher running times, because when more vertices have property q, diffusion unfolds more and thus HyperDC takes more time to converge to the least fixed point. In several real scenarios, information spreading is limited to individuals who are within close proximity (e.g., see [13]).

Spread. We now compare the diffusion of property p when we choose the top-k central vertices according to DC vs. using classical centrality measures on the STEAM data. In each case, the top-k vertices are called *seeds*. We vary k from 10 to 100 in steps of 10. The *spread* w.r.t. a given set of seeds is the expected number of vertices with property p (after diffusion) assuming the seeds have property p minus the expected number of vertices with property p (after diffusion) in the original social network. This difference is normalized. By the expected number of vertices with property p for an SN $\mathcal{S}$ after diffusion of Π we mean $\sum_{v \in V} \mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}})(p(v))$, where V is the set of vertices of $\mathcal{S}$. Our spread experiments are presented in two ways. In the first, we show how spread varies by averaging over the number of seeds for fixed δ_q values. In the second, we do the opposite.

Spread Experiments Averaged over Varying k values for specific δ_q values. We varied δ_q over the set $\{1\%, 2\%, 3\%, 4\%, 5\%, 10\%, 15\%, 20\%, 25\%, 30\%\}$. For each selection of δ_q, we considered every value of k from the set $\{10, 20, \ldots, 100\}$. Then for a fixed $\delta_q, k, \mathit{Diffusion\text{-}Model}$ triple, we computed the ratio of the spread using DC to the best spread achieved by any of the classical centrality measures. Table 5 reports the average of these ratios. Thus, any ratio greater than 1 shows that DC achieves a higher spread than all of the classical centrality measures.

For the Flickr and Jackson-Yariv models, DC always achieved a better spread than all classical centrality measures. For the SIR model, on average, DC achieves

Table 5: Spread over STEAM networks for different δ_q values.

		STEAM Data: Average ratio of the spread generated by diffusion centrality to the best spread generated by any of the classical centrality measures for different δ_q.									
Model	Network	1%	2%	3%	4%	5%	10%	15%	20%	25%	30%
FKModel	GAME8690	19.4	2.4	2.0	1.8	1.9	1.9	1.6	1.5	1.4	1.3
	GAME50510	18.1	22.5	23.8	26.5	19.5	5.0	3.6	3.0	2.7	2.4
	GAME6850	436.6	654.2	11.6	10.4	10.7	8.1	6.4	5.6	4.1	3.2
	GAME1500	677.6	89.9	48.3	49.9	46.6	48.8	50.0	56.2	55.5	3.1
	GAME24420	125.6	6.5	6.1	6.3	5.7	4.9	3.1	2.3	1.9	1.7
	GAME11450	512.0	585.5	804.3	8.9	9.7	6.9	5.5	5.1	4.9	3.7
	GAME17300	288.7	363.5	6.6	6.7	6.8	5.4	5.1	4.4	3.7	2.8
	Average	296.9	246.4	129.0	15.8	14.4	11.6	10.8	11.2	10.6	2.6
JYModel	GAME8690	2.8	1.7	1.4	1.2	1.3	1.2	1.1	1.1	1.1	1.2
	GAME50510	11.0	5.0	5.1	4.3	4.0	2.7	1.9	1.5	1.1	1.0
	GAME6850	8.1	5.8	4.9	4.5	3.6	2.5	2.1	1.8	1.7	1.6
	GAME1500	8.8	6.7	4.6	3.7	3.6	2.3	2.0	1.9	1.7	0.9
	GAME24420	5.4	3.3	2.8	2.6	2.2	1.6	1.4	1.3	1.3	1.3
	GAME11450	10.9	7.7	6.2	5.1	4.2	2.9	2.2	1.9	1.7	1.6
	GAME17300	9.0	4.5	2.4	2.2	2.2	1.6	1.2	1.2	1.2	1.2
	Average	8.0	5.0	3.9	3.4	3.0	2.1	1.7	1.5	1.4	1.3
SIRModel	GAME8690	1.2	0.8	0.8	0.8	0.8	0.8	0.9	0.9	0.8	0.8
	GAME50510	10.1	2.6	3.0	2.2	2.2	1.4	1.1	1.0	0.9	0.9
	GAME6850	4.9	3.6	2.8	2.6	2.5	1.8	1.7	1.4	1.3	1.2
	GAME1500	5.7	3.3	2.2	2.4	2.1	1.7	1.2	1.1	1.0	0.8
	GAME24420	2.0	1.4	1.5	1.5	1.4	1.1	0.8	0.8	0.8	0.8
	GAME11450	6.7	4.4	2.7	1.7	1.8	1.4	1.1	1.1	1.0	0.9
	GAME17300	4.3	2.1	1.0	1.1	1.2	0.7	0.6	0.6	0.6	0.6
	Average	5.0	2.6	2.0	1.8	1.7	1.3	1.1	1.0	0.9	0.9

better spreads than classical centrality measures as long as $\delta_q \leq 15\%$ — at 20%, they are even, and at 25-30%, classical centrality measures achieve a better spread.

Not surprisingly, this spread ratio decreases as δ_q increases because when δ_q is large, more vertices get infected regardless of how the seeds were chosen and thus the difference between DC and other centrality measures decreases.

Interestingly, spread ratios for the Flickr model are very large compared to those for the Jackson-Yariv and SIR models. This is because the expected number of vertices which have property p after diffusion is much higher in the case of the Flickr model than in the other two cases.

Spread Experiments Averaged over Varying δ_q values for specific k values. Here we selected values of k as before and averaged over different possible values of δ_q drawn from the set $\{1\%, 2\%, 3\%, 4\%, 5\%, 10\%, 15\%, 20\%, 25\%, 30\%\}$. Table 6 summarizes the results. As in the previous case, a ratio exceeding 1 implies that DC outperforms all classical centrality measures. On average, for all values of k and for all three diffusion models, DC achieves a better spread than all the classical centrality measures, with only a few exceptions in the SIR model.

Interestingly, the spread ratio for GAME8690 is consistently the lowest according to each of three diffusion models and in each setting of both Tables 5 and 6. This game has just 4083 vertices (so it is very small and dense). Our choice of q is based on the amount of playing time of a player and there is strong correlation between that and the number of friends. As a consequence, having multiple seeds does not greatly increase diffusion of property p because of overlaps between the

Table 6: STEAM Data: Average ratio of the spread generated by diffusion centrality to the best spread generated by any of the classical centrality measures for different k values.

		Average ratio of the DC spread to the best spread of other centrality measures for different k values.									
Model	Network	10	20	30	40	50	60	70	80	90	100
FlickrModel	GAME8690	16.5	2.1	2.2	2.1	2.1	2.0	2.0	2.1	2.1	2.1
	GAME50510	48.1	6.8	9.0	10.5	7.7	8.6	9.2	9.8	10.3	7.2
	GAME6850	41.4	67.3	86.1	101.2	114.9	127.9	138.5	149.6	157.8	166.2
	GAME1500	364.0	94.3	58.4	67.8	74.2	81.0	88.3	93.5	99.9	104.7
	GAME24420	48.2	72.2	4.4	5.2	5.5	5.7	5.5	5.7	6.0	5.7
	GAME11450	70.9	116.4	155.1	187.7	216.8	241.5	265.2	287.8	310.0	95.3
	GAME17300	63.3	101.0	134.8	164.0	187.8	8.0	8.0	8.5	8.9	9.4
	Average	93.2	65.7	64.3	76.9	87.0	67.8	73.8	79.6	85.0	55.8
JYModel	GAME8690	1.9	1.5	1.3	1.2	1.3	1.4	1.4	1.4	1.4	1.4
	GAME50510	3.3	3.6	3.9	3.7	4.1	3.7	3.9	4.0	3.7	3.8
	GAME6850	2.6	3.3	3.7	3.7	3.8	3.5	3.8	4.0	4.2	4.0
	GAME1500	3.1	3.3	3.9	3.9	4.0	3.7	3.7	3.6	3.5	3.5
	GAME24420	2.2	2.2	2.2	2.3	2.4	2.4	2.5	2.4	2.4	2.3
	GAME11450	3.0	4.3	4.7	4.5	4.7	4.5	4.6	4.8	4.8	4.6
	GAME17300	2.1	1.9	2.5	2.8	2.8	2.8	3.0	3.0	2.9	3.0
	Average	2.6	2.9	3.2	3.2	3.3	3.1	3.3	3.3	3.3	3.2
SIRModel	GAME8690	1.2	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.9	0.9
	GAME50510	2.1	2.0	2.4	2.4	2.8	2.5	2.7	3.0	2.7	2.7
	GAME6850	2.2	2.4	2.5	2.4	2.4	2.2	2.3	2.5	2.5	2.4
	GAME1500	2.5	2.4	2.6	2.2	2.2	2.1	2.1	1.9	1.8	1.8
	GAME24420	1.4	1.4	1.4	1.3	1.3	1.1	1.1	1.1	1.0	1.0
	GAME11450	2.1	2.8	2.4	2.3	2.2	2.2	2.1	2.2	2.2	2.2
	GAME17300	1.3	1.2	1.3	1.4	1.3	1.2	1.3	1.3	1.3	1.4
	Average	1.8	1.9	1.9	1.8	1.9	1.7	1.8	1.8	1.8	1.8

vertices that may be influenced by the seeds.

7.2.2. Non-Game Social Network Data

In this section, we perform experiments similar to those reported above on the networks in Table 3. For these networks, the data did not have associated vertex properties. We looked at two cases.

Case 1. We randomly selected $\delta_p = 0.1\%$ of the vertices to have a synthetic property p (in 5 runs). In each run, we varied $\delta_q \in \{1\%, 2\%, 3\%, 4\%, 5\%, 10\%, 15\%, 20\%, 25\%, 30\%\}$ and selected $\delta_q\%$ of the vertices to have a synthetic property q.

Case 2. We randomly selected $\delta_q = 3\%$ of the vertices and varied $\delta_p \in \{0.1\%, 0.2\%, 0.3\%, 0.4\%, 0.5\%\}$, associating synthetic properties p and q with vertices as above.

Recall that, as mentioned before, vertices that do not have the q-property can still play a role in spreading p depending on the diffusion model.[5]

Runtime. *Case 1.* Figure 5 shows how runtime varies w.r.t. δ_q for Case 1 above. The networks are sorted from left to right according to the number of vertices, with

[5]For instance, consider an SN with vertices v_1, v_2, v_3, v_4, and edges (v_1, v_2), (v_2, v_3), (v_3, v_4), where only v_4 has property q. Consider a diffusion model saying that if X has property p, X is connected with Y, Y is connected with Z, and Z is connected to W, then if W has property q he gets property p. If, for instance, v_1 has property p, then v_4 gets p too. Notice that each of the four vertices and their connections play a role in that. So, if for instance we delete v_2 and the edge (v_2, v_3) (e.g., just because v_2 does not have property q), we lose the real behavior discussed above.

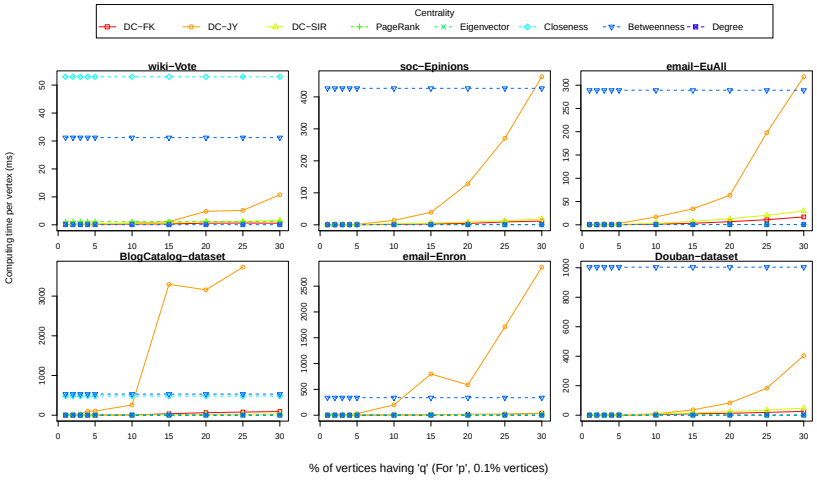

Figure 5: Non-Game SNs: Runtime per vertex when varying δ_q from 1% to 30% and $\delta_p = 0.1\%$.

directed networks in the first row and undirected networks in the second row. As in the case of our experiments with the STEAM data, closeness and betweenness centrality are time consuming compared to the other algorithms (including HyperDC).

HyperDC with the Flickr model is faster than PageRank in the majority of cases considered, and sometimes faster than eigenvector centrality. HyperDC with the SIR model is faster than PageRank for directed networks when $\delta_q \leq 4\%$. As in the case of the STEAM data described earlier, this is because the network filtering step can eliminate many vertices. For the Flickr and SIR models, the runtimes increase slightly as δ_q increases because of the higher diffusion that takes place.

However, as in the case of the STEAM data, HyperDC with the Jackson-Yariv model often takes the most time. By increasing the percentage of vertices having property q, computing diffusion centrality for the Jackson-Yariv model becomes slower than for the other diffusion models, because the diffusion process is determined by the sum of neighbors' diffusion probability and now we have that 0.1% of the vertices initially have the property p. Thus, every time that a vertex's probability is updated, all of its neighbors who have q are updated in the next step.

Case 2. Figure 6 shows how runtime varies as δ_p varies. The runtime for closeness and betweenness centrality are very high (worse by 1 to 3 orders of magnitude) and hence we do not report runtimes for them.

HyperDC's runtime for the Flickr and SIR models do not vary much with δ_p. HyperDC with the Flickr model is faster than PageRank in almost all networks and settings. Compared to degree centrality, HyperDC with Flickr exhibits competitive runtimes, being faster in about half the settings considered and comparable or

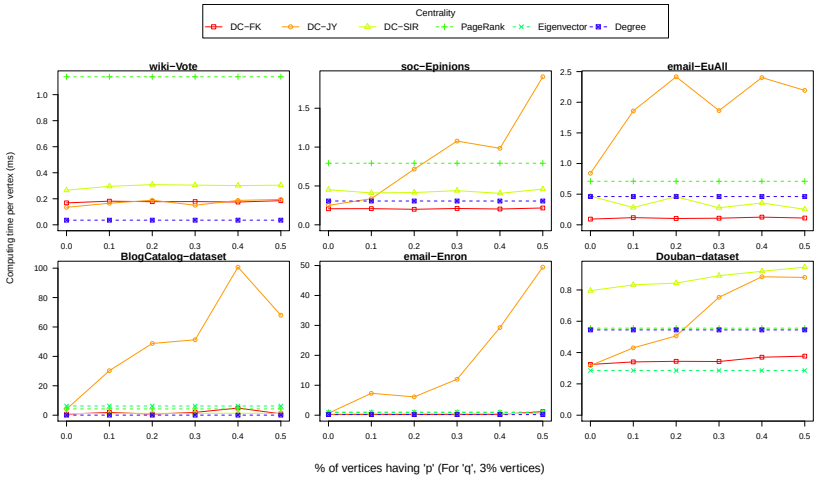

Figure 6: Non-Game SNs: Runtime per vertex when varying δ_p from 0.1% to 0.5% with $\delta_q = 3\%$.

slightly worse in the others. HyperDC with the SIR model is faster than PageRank in all data sets except the Douban data set.

As in the case of the STEAM data, HyperDC with the Jackson-Yariv is the worst performer w.r.t. runtime (excluding betweenness and closeness centrality which we eliminated earlier due to their high running times) because the computation of diffusion centrality for the Jackson-Yariv model becomes slower when many vertices have the diffusion property p.

Spread. In both Cases 1 and 2, on average, experimental results showed that the ratio of spread generated by HyperDC to the spread generated by the best classical measure exceeds 1. As in the case of the STEAM data, the best ratios are for the Flickr model.

7.3. CBAF Algorithm: Performance Experiments

In this section, we describe experiments we performed to compare CBAF with HyperDC in terms of both runtime and spread. Simply put, these experiments show that CBAF almost always achieves the same spread as HyperDC with a runtime that is always lower (with the correct choice of settings) than HyperDC—moreover, sometimes it takes less than half the runtime of HyperDC.

Input Options for CBAF. There are a number of input options for CBAF that influence its performance (cf. Table 2). In order to find the best possible input options, we ran extensive experiments in which we considered the 972 different candidate option sets determined by the candidate values reported in Table 7. Each option set was tested 5 times (because of the random component of the algorithm) over the

		Best input options		
Option	Possible values	OP_1	OP_2	OP_3
d_C	0,1,2	1	1	1
d_I	0,1,2	0	0	0
$weight$	min, max, $average$	max	max	$average$
$mergeVP$	$union$, $intersect$, $majority$	$majority$	$intersect$	$majority$
mvw	mvw_0, mvw_1, mvw_2	mvw_1	mvw_1	mvw_0
ρ	One vertex, 10%, 50%, 100%	100%	100%	100%

Table 7: Candidate and best values of the input options for CBAF

GAME17300 network. Candidate option sets were compared on the basis of their running time and the quality of their results, measured in terms of spread, recall, Kendall and Spearman's rank correlation coefficients. The three option sets that achieved a good balance between runtime and spread are reported in Table 7—these were the options we used in our experiments with CBAF. Specifically, $d_C = 1$ and $d_I = 0$ showed better running times than the other values with little difference in terms of quality; $weight = min$ was disregarded as its computing time was slightly better than max and $average$, while the quality was much worse; running times for $mergeVP = union$ and $mvw = mvw_2$ were worse than the other values while the qualities were similar; $\rho = 100\%$ showed slightly higher running times, but much better quality than the other values. In all experiments, we used the vertex similarity function SIM_{Π} defined earlier. The contraction factor θ was chosen from the set $\{0.2, 0.3, 0.4\}$.

CBAF Evaluation Measures. We used two measures to evaluate CBAF. The time ratio (denoted $ratio_{time}$) is simply the ratio of time taken by CBAF vs. HyperDC. The spread ratio (denoted $ratio_{spread}$) is the ratio of spread according to CBAF (assuming the top-k vertices have property p) vs. that according to HyperDC.

We compared HyperDC and CBAF over the two largest STEAM networks (GAME420 with 1.3M vertices and 12.43M edges, and GAME220 with 2.03M vertices and 20.59M edges) and the 2 largest non-game networks (Email-Eu and Douban), together with the Flickr, Jackson-Yariv, and SIR models, using the three best option sets of Table 7. In all the experiments we set $\delta_p = 0$ and $\delta_q = 5\%$. For the Email-Eu and Douban networks, we set $k = 100$ and $\theta \in \{0.4, 0.3, 0.2\}$. For the two STEAM networks, we ran the experiments with $\theta = 0.4$, and $k \in \{130, 650, 1300\}$ for GAME420 and $k \in \{200, 1000, 2000\}$ for GAME220.

Runtime. Option OP_3 is the one that consistently yielded the fastest runtimes for CBAF and its results are reported in Table 8 (we present the results only for the network GAME220, which is the largest one, because the results for GAME420 are quite similar to the ones for GAME220). The time ratios are always less than one in all networks and for all diffusion models with the exception of the Jackson-

Table 8: Results over the Douban, Email-eu, and GAME220 networks (Option set OP_3).

		Douban network		Email-eu network		GAME220 network		
Model	θ	$ratio_{time}$	$ratio_{spread}$	$ratio_{time}$	$ratio_{spread}$	k	$ratio_{time}$	$ratio_{spread}$
Flickr	0.4	0.63	0.96	0.83	0.61	200	0.52	0.99
	0.3	0.62	0.97	0.82	0.67	1000	0.45	0.99
	0.2	0.64	0.97	0.83	0.62	2000	0.60	0.94
JY	0.4	0.93	0.93	0.52	0.82	200	0.95	0.97
	0.3	0.91	0.94	0.55	0.82	1000	0.95	1.00
	0.2	0.91	0.94	0.66	0.83	2000	1.04	1.00
SIR	0.4	0.46	0.99	0.72	0.90	200	0.66	0.97
	0.3	0.48	0.99	0.76	0.89	1000	0.72	0.98
	0.2	0.47	0.99	0.76	0.91	2000	0.64	1.00

Yariv model in the huge network GAME220 with $k = 2000$. In particular, on the Douban network with the SIR model the time ratio is less than 50%, while delivering an almost perfect spread ratio of 0.99. On the Email-Eu network using the Jackson-Yariv model, it runs in just over 50% of the time taken by HyperDC. On the huge GAME220 network using the Flickr model it runs in under 60% of the time taken by HyperDC. CBAF does not work well for the Jackson-Yariv model in the two huge networks because of the high average degree of these networks.

Spread. In all cases, all three options yield approximately the same spread. In the huge networks and the Douban network, the spread ratio is always close to one. On the Email-Eu network, the spreads range from 0.8-0.9 for the Jackson-Yariv and SIR models. However, for the Flickr model, the spread is lower, mostly in the 0.6-0.7 range. The reason for this is that the Email-Eu network has a very low average degree which leads to merged vertices (in the coarsened networks) representing only small sets of vertices of the original network so that the induced network is small and not representative enough to compute approximate top-k vertices well.

The performance of CBAF w.r.t. runtime and spread depends on the input options. In general, we have observed that the more aggressive is the coarsening step, the faster is the algorithm, but the quality (spread) gets worse. In the step computing the induced SN, if we add more neighbors the quality gets better, but running times become higher.

7.4. Testing the quality of DC in MemeTracker data

We also tested the quality of diffusion centrality in the real context of memes diffusion through the Web. We used the MemeTracker data[6] consisting of a set of

[6] http://www.memetracker.org/data.html

172M news articles and blog posts from 1M online sources collected from September 1 2008 till August 31 2009. For each article/post the dataset contains timestamp, phrases contained in the document and hyper-links. In addition, phrases have been clustered together and this information is available in the data, too. We considered all phrases in the same cluster as the same meme. We built a network from the raw phrase data where vertices are online websites and edges are hyperlinks. More specifically, we selected as vertices the top 10,000 sites w.r.t. number of hyperlinks, and inserted a direct edge (u, v) between two sites u and v if there is a webpage on site u having a hyperlink to a webpage on site v.

Our aim was to show that diffusion centrality correlates well with the spread generated by vertices: vertices with high diffusion centrality spread more than vertices with low diffusion centrality. To show that, we restricted our attention to source vertices, i.e. online websites that firstly showed a meme m, and for each of them we computed the actual spread (number of websites infected by m). We assumed that a site u infected a site v with meme m if there is an edge from u to v, m appeared on u at time $t1$ and on v at time $t2$, and $t1 < t2$. When many webpages belonging to the same website u are infected by the same meme m, we consider the smaller webpage timestamp as timestamp for the infection of u. Moreover, our analysis focused only on the top 5,000 most spread memes.

We assumed that the memes diffusion through the websites is described by a (cascade) diffusion model for which we estimated the parameters by using the actual spread of memes (the identification of the actual model for memes diffusion is out of the scope of this paper).

By using the MemeTracker data and the above diffusion model, we performed a set of experiments as follows. We do not use a k-fold cross validation since we have temporal data, but we considered a time window (t_1, t_2) to determine training/testing data that we moved in steps of one day, whose size (in days) assumed values in the set $\{30, 60, 90\}$. For each time window, we considered the data whose timestamp is in the first 80% of the days as a training set to estimate the parameter of the diffusion model, and the last part as a test set. For each meme m in the test set we computed the tuple (m, v, c, s), where v is the source of m, c is its centrality value (we computed diffusion centrality, PageRank, degree, betweenness and closeness centrality), and s is the actual spread of m.

Figure 7 (left) shows the distribution of centrality values for DC, PageRank, closeness, betweenness, and degree centrality as well as the distribution of the actual spread over all the memes in the dataset and by considering a sliding time window of size 30 days. The plot in Figure 7 (right) shows the value of the distance (measured by using the Kolmogorov-Smirnov statistic) between the actual spread distribution and the distribution of PageRank, closeness, diffusion, betweenness, and degree centrality. The figure shows that the distribution of DC values is the

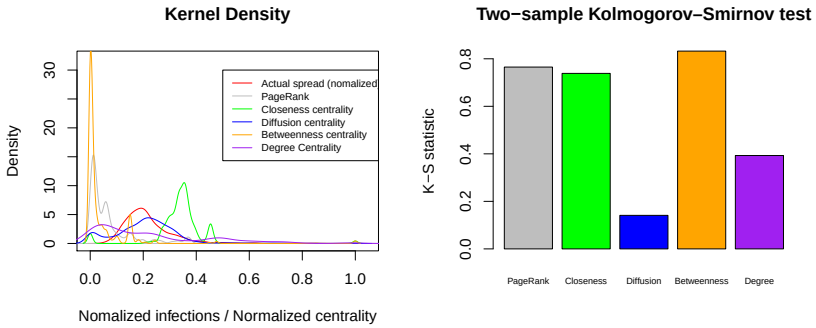

Figure 7: (Left) Actual spread and centrality measures' distributions. (Right) Kolmogorov-Smirnov statistic between actual spread distribution and centrality measures' distributions from the left plot.

closest to the one of the actual spread and its distance is the smallest. This result intuitively suggests that an initiator with a high value of diffusion centrality is more likely to reach a high value of spread than one with a low value of diffusion centrality, while this does not hold for the other centrality measures.

7.5. Summary of Experiments

In this section, we conducted three experiments. The first compares HyperDC with classical centrality measures. The experiment shows that: The runtime of HyperDC is better than betweenness and closeness centralities and comparable with other centrality measures. The spread achieved by diffusion centrality is almost always better than those achieved by classical centrality measures.

The second experiment compares HyperDC with CBAF. We show that CBAF almost always achieves the same spread as HyperDC with a runtime that is always lower (with the correct choice of settings) than HyperDC.

Recall that in our experimental setup, we considered an initial distribution of a property q, which models a characteristic that a vertex should have to infect or be infected by the diffusive property. As also discussed before, running times increase as the percentage δ_q of vertices having property q increases. This is because diffusion unfolds more with more vertices having property q, which means that the time to compute DC increases. Nevertheless, HyperDC showed good performances with values of δ_q up to 30%. Lower percentages of vertices having property q essentially mean that propagation unfolds less, and thus the fixpoint of HyperDC is reached sooner. However, it is worth mentioning that this is the case in several real scenarios. For instance, [13] analyzed propagation in the Flickr social network and found out that even the more popular photos have substantially limited popularity outside the immediate network neighborhood of the uploader.

The runtime of DC is expected to be higher with more complex diffusion models. A structural analysis of the corresponding GAPs can provide insights on what to expect from the diffusion model in terms of running time to evaluate it. In this regard, the subclasses of GAPs and the optimizations introduced in Section 5 are useful tools. For instance, if a GAP is p-monotonic and/or p-dwindling, we can lower running times, as we can apply further optimizations and because the HyperLFP converges more quickly. Another useful parameter to analyze is the size of $I_{\Pi,p}$, which roughly speaking consists of the predicate symbols "interfering" with p; we can expect better performances when $I_{\Pi,p}$ is small. These analyses are useful in practice to "tune" the diffusion model and find a good balance between accurateness of the model and time to evaluate it. For instance, one might want to simplify the considered diffusion model to make it p-monotonic, sacrificing its accurateness in describing the diffusion process, but obtaining a more efficient evaluation.

To strengthen our claims about the advantage of DC in predicting the spread initiated by given vertices, we used the MemeTracker data where diffusion occurred naturally without our intervention. The parameters of the diffusion model were estimated from the data (making sure to separate learning and testing data). The results of this experiment showed that there is a correlation between the DC values and the actual diffusion of memes: high diffusion centrality vertices diffuse memes better than low diffusion centrality vertices. It also showed that diffusion centrality is a better predictor of real meme diffusion than classical centrality measures.

8. Conclusion

Centrality and importance in social networks are closely interlinked concepts. Central vertices are assumed to be important and vice versa. However, in real-world online social networks, people who are considered important or authoritative on some topics may be considered very unimportant on others. For instance, an influential sports commentator is not likely to be an influential movie critic. Moreover, the importance of an individual should be measured by an influence related factor. A person who can influence 1000 people about movies should be considered more important (with regard to movies) than one who can influence just 500. Diffusion models are a mechanism to capture the spread of concepts or phenomena through a network. Many diffusion models have been developed to successfully predict the extent of spread of memes and concepts through networks.

In this paper, we propose the novel concept of *Diffusion Centrality* and show how it can be computed with respect to a wide array of diffusion models. [51] shows that the framework of generalized annotated programs (GAPs) can express most known diffusion models.

. In this paper, we increase the breadth of knowledge about GAPs by introducing novel specific classes of GAPs and present the HyperDC algorithm that exactly computes the top-k diffusion centrality vertices w.r.t. any GAP-expressible diffusion models. In addition, we present a novel (but approximate) Coarsening Back and Forth (CBAF) algorithm that allows us to take a huge social network, and reduce it to a manageable size to efficiently solve (in an approximate way) the problem of finding the top-k vertices.

We conduct a very detailed experimental study on several real-world social networks. A first set of experiments compares the runtime and spread generated via the HyperDC algorithm with classical centrality measures. Our results show that HyperDC is efficient and produces better spread than current centrality measures. A second set of experiments looks at the scalability of CBAF, showing that it almost always has a lower runtime than HyperDC, while achieving high spreads. In particular, CBAF was tested on networks with over 2M vertices and over 20M edges and achieved acceptable runtime. Using MemeTracker data, we show that diffusion centrality captures the importance of people who are truly responsible for the spread of a meme more effectively than past centrality measures.

This work opens up a dramatic new set of possible diffusion models that can be automatically learned from data that take into account the rich semantics that can be associated with vertices and edges in modern social networks like Twitter, Facebook, and LinkedIn. For instance, in the case of predicting election outcomes using Twitter data [30], we might learn rules that identify the likelihood that person P will vote for a candidate C by taking their gender, demographic factors, tweets, and the tendencies of their neighbors to vote for C. Alternatively, we might look at the diffusion of banking crises throughout the world's nations by considering network flows of exposures—the nodes in such a network would be countries and the edges would be labeled with the exposure a country has to another [44]. High exposure of country A to country B would suggest that a systemic banking crisis in country B could lead to default, triggering a systemic banking crisis in country A. Clearly, there are many other factors to be considered. For instance, political factors (represented as semantic properties of vertices) might capture the likelihood of country A taking intelligent steps to forestall a crisis. Mutual trust (a property of the edge) might determine whether A and B can work together to address the problem. Such a diffusion model will use the rich semantic opportunities offered by research in knowledge representation via GAPs, as well as other paradigms, to learn more fine-grained diffusion models than the relatively coarse grained diffusion models that were developed in the past. We believe this would form a rich line of inquiry for the future.

Acknowledgement. Some of the authors were partly supported by ARO grants W911NF11103, W911NF1410358, and W911NF09102, by Maafat, and by Israel

Science Foundation (grant #1488/14).

References

[1] Adali, S., Lu, X., Magdon-Ismail, M., 2014. Local, community and global centrality methods for analyzing networks. Social Network Analysis and Mining 4 (1), 210.

[2] Anderson, R. M., May, R. M., 1979. Population biology of infectious diseases: Part I. Nature 280 (5721), 361–367.

[3] Aral, S., Muchnik, L., Sundararajan, A., 2009. Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. Proc. National Academy of Sciences 106 (51), 21544–21549.

[4] Barbieri, N., Bonchi, F., Manco, G., 2013. Topic-aware social influence propagation models. Knowledge and Information Systems 37 (3), 555–584.

[5] Beauchamp, M. A., 1965. An improved index of centrality. Behavioral Science 10 (2), 161–163.

[6] Becker, R., Chernihov, Y., Shavitt, Y., Zilberman, N., 2012. An analysis of the steam community network evolution. In: Proc. Convention of Electrical & Electronics Engineers in Israel. pp. 1–5.

[7] Bonacich, P., 1972. Factoring and weighting approaches to status scores and clique identification. Journal of Mathematical Sociology 2 (1), 113–120.

[8] Brandes, U., 2008. A graph-theoretic perspective on centrality. Social Networks 30 (2), 136–145.

[9] Brin, S., Page, L., 1998. The anatomy of a large-scale hypertextual web search engine. Computer Networks 30 (1-7), 107–117.

[10] Broecheler, M., Shakarian, P., Subrahmanian, V. S., 2010. A scalable framework for modeling competitive diffusion in social networks. In: Proc. International Conference on Social Computing. pp. 295–302.

[11] Çigdem Aslay, Barbieri, N., Bonchi, F., Baeza-Yates, R. A., 2014. Online topic-aware influence maximization queries. In: Proc. International Conference on Extending Database Technology. pp. 295–306.

[12] Centola, D., 2010. The spread of behavior in an online social network experiment. Science 329 (5996), 1194–1197.

[13] Cha, M., Mislove, A., Gummadi, P. K., 2009. A measurement-driven analysis of information propagation in the flickr social network. In: Proc. International World Wide Web Conference. pp. 721–730.

[14] Chen, N., 2009. On the approximability of influence in social networks. SIAM Journal on Discrete Mathematics 23 (3), 1400–1415.

[15] Chen, W., Wang, C., Wang, Y., 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: Proc. International Conference on Knowledge Discovery and Data Mining. pp. 1029–1038.

[16] Chen, Y., Peng, W., Lee, S., 2012. Efficient algorithms for influence maximization in social networks. Knowledge and Information Systems 33 (3), 577–601.

[17] Chiang, C.-Y., Huang, L.-H., Li, B.-J., Wu, J., Yeh, H.-G., 2013. Some results on the target set selection problem. Journal of Combinatorial Optimization 25 (4), 702–715.

[18] Dolev, S., Elovici, Y., Puzis, R., 2010. Routing betweenness centrality. Journal of the ACM 57 (4).

[19] Fierens, D., den Broeck, G. V., Renkens, J., Shterionov, D. S., Gutmann, B., Thon, I., Janssens, G., Raedt, L. D., 2015. Inference and learning in probabilistic logic programs using weighted boolean formulas. Theory and Practice of Logic Programming 15 (3), 358–401.

[20] Freeman, L. C., 1977. A set of measures of centrality based on betweenness. Sociometry 40 (1), 35–41.

[21] Freeman, L. C., 1979. Centrality in social networks conceptual clarification. Social Networks 1 (3), 215–239.

[22] Fung, W. S., Hariharan, R., Harvey, N. J. A., Panigrahi, D., 2011. A general framework for graph sparsification. In: Proc. ACM Symposium on Theory of Computing. pp. 71–80.

[23] Goyal, A., Lu, W., Lakshmanan, L. V. S., 2011. SIMPATH: an efficient algorithm for influence maximization under the linear threshold model. In: Proc. International Conference on Data Mining. pp. 211–220.

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

[25] Graves, A., Adali, S., Hendler, J., 2008. A method to rank nodes in an RDF graph. In: Proc. International Semantic Web Conference.

[26] Hethcote, H. W., 1976. Qualitative analyses of communicable disease models. Mathematical Biosciences 28 (3-4), 335–356.

[27] Irfan, M. T., Ortiz, L. E., 2014. On influence, stable behavior, and the most influential individuals in networks: A game-theoretic approach. Artificial Intelligence 215, 79–119.

[28] Jackson, M., Yariv, L., 2005. Diffusion on social networks. Economie Publique 16 (1), 69–82.

[29] Jiang, Q., Song, G., Cong, G., Wang, Y., Si, W., Xie, K., 2011. Simulated annealing based influence maximization in social networks. In: Proc. AAAI Conference on Artificial Intelligence.

[30] Kagan, V., Stevens, A., Subrahmanian, V. S., 2015. Using Twitter sentiment to forecast the 2013 Pakistani election and the 2014 Indian election. IEEE Intelligent Systems 30 (1), 2–5.

[31] Kang, C., Molinaro, C., Kraus, S., Shavitt, Y., Subrahmanian, V. S., 2012. Diffusion centrality in social networks. In: Proc. International Conference on Advances in Social Networks Analysis and Mining. pp. 558–564.

[32] Karypis, G., 2011. METIS and parmetis. In: Encyclopedia of Parallel Computing. pp. 1117–1124.

[33] Kempe, D., Kleinberg, J. M., Tardos, É., 2003. Maximizing the spread of influence through a social network. In: Proc. International Conference on Knowledge Discovery and Data Mining. pp. 137–146.

[34] Kempe, D., Kleinberg, J. M., Tardos, É., 2005. Influential nodes in a diffusion model for social networks. In: Proc. International Colloquium on Automata, Languages and Programming. pp. 1127–1138.

[35] Kifer, M., Subrahmanian, V. S., 1992. Theory of generalized annotated logic programming and its applications. Journal of Logic Programming 12 (3&4), 335–367.

[36] Kimura, M., Saito, K., Motoda, H., 2009. Efficient estimation of influence functions for SIS model on social networks. In: Proc. International Joint Conference on Artificial Intelligence. pp. 2046–2051.

[37] Kimura, M., Saito, K., Nakano, R., 2007. Extracting influential nodes for information diffusion on a social network. In: Proc. AAAI Conference on Artificial Intelligence. pp. 1371–1376.

[38] Leskovec, J., 2015. The Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data/index.html`.

[39] Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J. M., Glance, N. S., 2007. Cost-effective outbreak detection in networks. In: Proc. International Conference on Knowledge Discovery and Data Mining. pp. 420–429.

[40] Lu, J. J., Nerode, A., Subrahmanian, V. S., 1996. Hybrid knowledge bases. IEEE Transactions on Knowledge and Data Engineering 8 (5), 773–785.

[41] Mathioudakis, M., Bonchi, F., Castillo, C., Gionis, A., Ukkonen, A., 2011. Sparsification of influence networks. In: Proc. International Conference on Knowledge Discovery and Data Mining. pp. 529–537.

[42] McPherson, M., Smith-Lovin, L., Cook, J. M., 2001. Birds of a feather: Homophily in social networks. Annual Review of Sociology 27, 415–444.

[43] Memory, A., Kimmig, A., Bach, S. H., Raschid, L., Getoor, L., 2012. Graph summarization in annotated data using probabilistic soft logic. In: Proc. International Workshop on Uncertainty Reasoning for the Semantic Web. pp. 75–86.

[44] Minoiu, C., Kang, C., Subrahmanian, V. S., Berea, A., 2015. Does financial connectedness predict crises? Quantitative Finance 15 (4), 607–624.

[45] Naseri, M. B., Elliott, G., 2011. Role of demographics, social connectedness and prior internet experience in adoption of online shopping: Applications for direct marketing. Journal of Targeting, Measurement and Analysis for Marketing 19 (2), 69–84.

[46] Nieminen, J., 1974. On the centrality in a graph. Scandinavian Journal of Psychology 15 (1), 332–336.

[47] Pearl, J., Russell, S., 1998. Bayesian Networks. Computer Science Department, University of California.

[48] Purohit, M., Prakash, B. A., Kang, C., Zhang, Y., Subrahmanian, V. S., 2014. Fast influence-based coarsening for large networks. In: Proc. International Conference on Knowledge Discovery and Data Mining.

[49] Sabidussi, G., 1966. The centrality index of a graph. Psychometrika 31, 581–603.

[50] Schelling, T., 1978. Micromotives and Macrobehavior. W.W. Norton and Co.

[51] Shakarian, P., Broecheler, M., Subrahmanian, V. S., Molinaro, C., 2013. Using generalized annotated programs to solve social network diffusion optimization problems. ACM Transactions on Computational Logic 14 (2), 10.

[52] Szczepanski, P. L., Michalak, T. P., Wooldridge, M., 2014. A centrality measure for networks with community structure based on a generalization of the Owen value. In: Proc. European Conference on Artificial Intelligence. pp. 867–872.

[53] Tang, J., Sun, J., Wang, C., Yang, Z., 2009. Social influence analysis in large-scale networks. In: Proc. International Conference on Knowledge Discovery and Data Mining. pp. 807–816.

[54] Toivonen, H., Zhou, F., Hartikainen, A., Hinkka, A., 2011. Compression of weighted graphs. In: Proc. International Conference on Knowledge Discovery and Data Mining. pp. 965–973.

[55] Toivonen, H., Zhou, F., Hartikainen, A., Hinkka, A., 2012. Network compression by node and edge mergers. In: Bisociative Knowledge Discovery - An Introduction to Concept, Algorithms, Tools, and Applications. pp. 199–217.

[56] Wang, C., Chen, W., Wang, Y., 2012. Scalable influence maximization for independent cascade model in large-scale social networks. Data Mining and Knowledge Discovery 25 (3), 545–576.

[57] Watts, D., Peretti, J., 2007. Viral marketing for the real world. Harvard Business Review.

Appendix A. Diffusion Models

Below we provide details on the diffusion models used in the experimental evaluation. The Flickr model consists of the following rule (see [51]):

$$p(v) : \mu_{v',v} \times \mu_p \times \mu_q \times dp_F \leftarrow e(v', v) : \mu_{v',v} \wedge p(v') : \mu_p \wedge q(v') : \mu_q$$

saying that if vertex v' has properties q and p then it can diffuse property p to its neighbor v. The value dp_F is a constant representing the probability that the vertex v will receive property p. This model falls into the category of cascade models.

The Jackson-Yariv model is a diffusion model stating that a vertex will receive (adopt) a property p according to the cumulative effect of its neighbors and the ratio of the benefit to the cost of the vertex for adopting p. Suppose that v_i is an agent having a default behavior that can be changed in the new behavior p, and that v_i has specific cost c_i and benefit b_i for adopting the behavior p. Then, the Jackson-Yariv model can be expressed by the rule (see [51]):

$$p(v_i) : \tfrac{b_i}{c_i} \times r(\textstyle\sum_j E_j) \times \frac{\sum_j w_j}{\sum_j E_j} \times w_q \times dp_{_{JY}} \leftarrow \\ \bigwedge_{v_j|(v_j,v_i)\in E} \big(e(v_j, v_i) : E_j \wedge p(v_j) : w_j\big) \wedge q(v_i) : w_q$$

where *(1)* $r(\sum_j E_j)$ is a function describing how the number of neighbors of v_i affects the benefits to v_i for adopting behavior p, *(2)* $\frac{\sum_j w_j}{\sum_j E_j}$is the fraction of the neighbors of v_i having property p, and *(3)* $dp_{_{JY}}$ is a constant representing the probability that the vertex v will adopt the property p. In our experiments we set $r(\sum_j E_j)$ to be a logarithmic function normalized by the logarithm of the maximum in-degree d^{in}_{max} of the network, and having values within the interval $[0.1, 2]$, i.e., $r(\sum_j E_j) = 1.9 \times \frac{\ln(\sum_j E_j)}{\ln(d^{in}_{max})} + 0.1$. When the annotation μ of an atom $p(v)$, with $v \in V$, becomes greater than 1, then we set $\mu = 1$. Moreover, observe that the vertex v_i can adopt property p only if it also has property q. For the STEAM data, we set $\frac{b_i}{c_i} = 1$ for all vertices, while for the Non-Game networks we randomly assigned $\frac{b_i}{c_i}$ to the vertices according to a normal distribution with $0.5 \leq \frac{b_i}{c_i} \leq 1.5$.

The SIR model is a classic disease model which labels each vertex with *susceptible* if it has not had the disease but can receive it from one of its neighbors, *infectious* if it has caught the disease and t units of time have not expired, and *recovered* when the vertex can no longer catch or transmit the disease. The diffusion rules are following (see [51]):

$$p(v) :(1 - R) \times \mu^e_{v',v} \times \mu^p_{v'} \times (1 - R') \times \mu^q_v \times dp_{_{SIR}} \leftarrow \\ r_t(v) : R \wedge e(v', v) : \mu^e_{v',v} \wedge p(v') : \mu^p_{v'} \wedge r_t(v') : R' \wedge q(v) : \mu^q_v$$

$$r_i(v) :\mu^r_v \leftarrow r_{i-1}(v) : \mu^r_v,\ i \in [2, t]$$
$$r_1(v) :\mu^p_v \leftarrow p(v) : \mu^p_v$$

Here, only the vertices having property q can be susceptible and the diffusion property p represents that a vertex is infected. Properties r_i ($i \neq t$) express that a vertex is in the infectious state at time $t-1$ and r_t means that a vertex is recovered. In our experiments, we set $t = 2$. The constant $dp_{_{SIR}}$ is the probability that the vertex v will be infected. The SIR model falls into the category of cascade models.

We conclude this section by showing how the well-known linear threshold model can be expressed with GAPs. Recall that in the linear threshold model,

a vertex v is influenced by each neighbor w according to a weight $b_{w,v}$ such that $\sum_{w\ neighbor\ of\ v} b_{w,v} \leq 1$. Furthermore, each vertex v has a threshold $\theta_v \in [0,1]$, which is the weighted fraction of v's neighbors that must become active in order for v to become active, that is, v becomes active if $\sum_{w\ active\ neighbor\ of\ v} b_{w,v} \geq \theta_v$. The GAP below captures the behavior of the linear threshold model. For each vertex v, the GAP has a rule of the following form:

$$p(v) : \left\lceil \left(\sum_{w\ neighbor\ of\ v} B_{w,v} \times X_w \right) - \theta_v \right\rceil \leftarrow \bigwedge_{w\ neighbor\ of\ v} \big(e(w,v) : B_{w,v} \wedge p(w) : X_w\big)$$

Notice that the SN is assumed to be modeled with facts of the form $e(w,v) : b_{v,w}$, meaning that w is a neighbor of v and $b_{w,v}$ is the weight according to which w influences v. If a vertex v is active, then $p(v) : 1$ holds, otherwise $p(v) : 0$ holds.

Appendix B. Proofs

Proposition 3. *Consider an SN $\mathcal{S}$, a property p, and a GAP Π. Let ψ be the interpretation $\mathsf{lfp}(\Pi_{\mathcal{S}} \cup \Pi_p^*)$. Then, $\mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}})(p(v)) = \mathsf{lfp}((\Pi_p \setminus \Pi_p^*) \cup \{A : \psi(A) \mid A \in \mathcal{A}\})(p(v))$ for every vertex v of $\mathcal{S}$.*

PROOF. All the rules in $\Pi \setminus \Pi_p$ have a predicate in the head atom that cannot reach predicate p, and then these rules do not affect the value of $\mathsf{lfp}(\Pi \cup \Pi_{\mathcal{S}})(p(v))$. As we are interested in computing the diffusion centrality for property p, we can ignore these rules in the computation. Moreover, rules in Π_p can be partitioned into two sets of rules, namely Π_p^* and $(\Pi_p \setminus \Pi_p^*)$, and by definition of Π_p^*, we have that rules in $(\Pi_p \setminus \Pi_p^*)$ "depend on" rules in Π_p^* because an atom having predicate symbol $q \in I_{\Pi,p}$ may appear in the head of a rule in Π_p^* and in the body of a rule in $(\Pi_p \setminus \Pi_p^*)$, but not vice versa. Thus, the rules in $(\Pi_p \setminus \Pi_p^*)$ do not contribute to the least fixed point of Π_p^* and then the value of ψ can be pre-computed. □.

Proposition 4. *The Flickr model is p-monotonic and p-dwindling. The Jackson-Yariv model is p-monotonic but not p-dwindling. The SIR model is neither p-monotonic nor p-dwindling.*

PROOF. The GAP describing the Flickr model is p-monotonic because the function $\mu_{v',v} \times \mu_p \times \mu_q \times dp_F$ in the head of its unique rule is clearly monotonic. The GAP is also p-dwindling because the value of the function $\mu_{v',v} \times \mu_p \times \mu_q \times dp_F$ is less than or equal to any value that μ_p can assume, as the annotations $\mu_{v',v}$, μ_q and the constant dp_F can assume only values between 0 and 1.

The GAP describing the Jackson-Yariv model is p-monotonic too. In fact, the function in the head of its unique rule is monotonic as the term $\frac{b_i}{c_i} \times r(\sum_j E_j) \times \frac{1}{\sum_j E_j} \times dp_{JY}$ is a constant. However, the GAP describing the Jackson-Yariv model is not p-dwindling because of the term $\sum_j \omega_j$ in the head atom function.

The GAP describing the SIR model is not p-monotonic because of the terms $(1 - R)$ and $(1 - R')$ in the head atom annotation function of the first rule (which is $(1 - R) \times \mu^e_{v',v} \times \mu^p_{v'} \times (1 - R') \times \mu^q_v \times dp_{SIR}$). Moreover, the GAP is not p-dwindling. To see this, it is sufficient to note that, because of the presence of the terms $(1 - R)$ and $(1 - R')$ in the head atom annotation function of the first rule, we cannot say that the value assumed by this function is always less than or equal to R or R' (remember that $I_{\Pi_{SIR},p} = \{p, r_1, r_2\}$). □

Proposition 5. *The worst-case time complexity of Algorithm HyperLFP is* $O(|N|+\frac{1}{\alpha}\cdot|H|\cdot(\log|H|+U_{max}\cdot(S_{max}+\log|H|)))$*, where* $U_{max} = \max_{v\in V,q\in I_{\Pi,p}}\{|\{h \mid h \in H \wedge q(v) \in S(h)\}|\}$*,* $S_{max} = \max_{h\in H}\{|S(h)|\}$*, and* α *is the minimum value obtained at line 9 for* $(w - M'[q][v])$*.*

PROOF. The cost of making two copies of the matrix M at Lines 1-2 is $O(|N|)$. The loop at Line 5 is executed $|H|$ times, as at each iteration we remove an hyper-edge from $Heap$ (Line 6). Within this loop, the predominant costs are the cost of deleting the maximum from $Heap$ (Line 6), which is $O(\log|H|)$, and the cost of the loop at Line 10. This loop is executed for each hyper-edge $h' \in U[q][v]$ whose number is U_{max} in the worst-case and at each iteration the cost of computing the function W at Line 11 is $O(S_{max})$ in the worst-case, while the cost of adding $\langle h', w'\rangle$ either to $Heap$ (Line 16) or $Heap'$ (Line 18) is $O(\log|H|)$. Thus, the cost of executing Lines 5-19 is $O(|N| + |H| \cdot (log|H| + U_{max} \cdot S_{max}))$. Finally, the loop at Line 3 is executed $\frac{1}{\alpha}$ times in the worst-case as 1 is the maximum growth the annotation of an atom can have and α is the minimum increment step. □

Proposition 6. *The worst-case time complexity of Algorithm HyperDC is* $O(|V| \cdot (|N|+\frac{1}{\alpha}\cdot|H|\cdot(\log|H|+U_{max}\cdot(S_{max}+\log|H|))))$*, where* $U_{max} = \max_{v\in V}\{|\{h \mid h \in H \wedge v \in S(h)\}|\}$*,* $S_{max} = \max_{h\in H}\{|S(h)|\}$*, and* α *is defined as in Proposition 5.*

PROOF. By leveraging Proposition 1, the HyperDC algorithm computes one least fix point for each vertex, so its worst-case time complexity is given by the number of vertices ($|V|$) times the worst-case time complexity of the HyperLFP algorithm (which is $O(|N|+\frac{1}{\alpha}\cdot|H|\cdot(\log|H|+U_{max}\cdot(S_{max}+\log|H|)))$) called at line 17. □