The Filter-Placement Problem and its Application to Content De-Duplication

Azer Bestavros best@bu.edu
Dóra Erdős edori@bu.edu
Vatche Ishakian visahak@bu.edu
Andrei Lapets lapets@bu.edu
Evimaria Terzi evimaria@bu.edu

Computer Science Department, Boston University
Boston, MA 02215, USA

ABSTRACT

In many information networks, data items – such as updates in social networks, news flowing through interconnected RSS feeds and blogs, measurements in sensor networks, route updates in ad-hoc networks, *etc.* – propagate in an uncoordinated manner: nodes often relay information they receive to neighbors, independent of whether or not these neighbors received such information from other sources. This uncoordinated data dissemination may result in significant, yet unnecessary communication and processing overheads, ultimately reducing the utility of information networks. To alleviate the negative impacts of this *information multiplicity* phenomenon, we propose that a subset of nodes (selected at key positions in the network) carry out additional information de-duplication functionality – namely, the removal (or significant reduction) of the duplicative data items relayed through them. We refer to such nodes as *filters.* We formally define the Filter Placement problem as a combinatorial optimization problem, and study its computational complexity for different types of graphs. We also present polynomial-time approximation algorithms for the problem. Our experimental results, which we obtained through extensive simulations on synthetic and real-world information flow networks, suggest that in many settings a relatively small number of filters is fairly effective in removing a large fraction of duplicative information.

1. INTRODUCTION

Information networks arise in many applications, including social networks, RSS-feed and blog networks, sensor networks, ad-hoc networks, among many others. In information networks, content propagates from content creators, *i.e.*, sources, to content consumers through directed links connecting the various nodes in the network. The utility of an information network has been long associated with its ability to facilitate *effective information propagation.* A network is considered highly functional, if all its nodes are up-to-date and aware of newly-generated content of interest.

This research was supported in part by NSF awards #0720604, #0735974, #0820138, #0952145, #1012798, and #1017529

Motivation: A common attribute of many information networks is that content propagation is not coordinated: nodes relay information they receive to neighbors, independent of whether or not these neighbors have received such information from other sources. This lack of coordination may be a result of node autonomy (as in a social network), limited capabilities and lack of local resources (as in a sensor network), or absence of topological/routing information (as in an ad-hoc network). As a result of uncoordinated information propagation, nodes in an information network may end up receiving the same content repeatedly, through different paths in the network. We call this phenomenon *information multiplicity.* The redundancy caused by information multiplicity results in significant, yet unnecessary communication and processing overheads, ultimately reducing the utility of information networks.

As an illustration of information multiplicity, consider the toy news network shown in Figure 1, in which branded syndicated content propagates over directed edges. In this example, node S is the originator of new information – the syndicated content – whereas nodes x and y are distributors (*e.g.*, newspapers) that may add branding or advertisement to the syndicated content received from S. All the other nodes in our illustrative example (*e.g.*, portals) do not generate new content; rather they utilize their connections to relay content to other nodes along directed links.

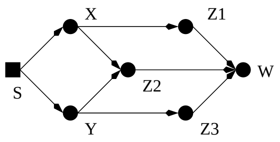


Figure 1: Illustration of information multiplicity.

Now assume that a single news item i reaches x and y after its generation by S. Nodes x and y forward i to their neighbors, and as a result, z_1, z_2 and z_3 receive i as well. In fact, z_2 receives (unnecessarily) two copies of i; one from x and one from y. Even worse, if z_1, z_2 and z_3 forward whatever they get to w, then w receives $(1+2+1)$ copies of i. Clearly, to inform w, one copy of i is enough.

To alleviate the negative implications from *information multiplicity*, we propose that a subset of nodes be strategically selected and equipped with additional "de-duplication" functionality, namely the removal (or significant reduction) of duplicative data items relayed through them. We refer to such nodes as *filters.* Filters de-duplicate information items they receive (*e.g.*, updates in social networks, news flowing through feeds and blogs networks, measurements in sensor networks, route updates in ad-hoc networks, *etc.*) and they never propagate to their neighbors the same item twice.[1] We refer to the problem of identifying the set of filter nodes in a given network as the Filter Placement problem.

Notice that the placement of filters does not impact the utility of the network in any other way; filters simply de-duplicate content. In the example above, placing two filters at z_2 and w completely alleviates redundancy.

Applications: Information multiplicity arises in many networks, including social networks, RSS-feed and blog networks, sensor networks and ad-hoc networks. In social networks, users typically share content (*e.g.*, news articles, links, videos) by forwarding such content to friends (using Facebook walls, group mailing lists, *etc.*) without checking whether such content was already forwarded. In news networks, nodes may play multiple roles. They may be sources of original content, portals that access such sources to publish branded variants of the original content, aggregators that harvest branded content from various publishers, and end-user clients (software) that subscribe to portals, publishers, aggregators, *etc.* In such an automated setting, there are no provisions for de-duplication of the content flowing trough different paths in the network. Here we note that de-duplication may entail significant overheads to identify similar but not identical content (*e.g.*, content with different branding or presentation). In sensor networks, nodes are small devices that relay their own measurements along with the measurements they collect from other nearby devices. In disseminating sensory data, these devices do not check for duplication, primarily because performing such checks requires significant storage and processing resources, which are beyond what is typically available in a sensor device. A salient feature of all of these applications is the sheer number of ultimate consumers of content (social network users, consumers of news and RSS feeds, *etc.*). This makes the deployment of filters at all end-points impractical, especially when this filtering functionality involves significant processing, and/or when the end-points are computationally impoverished. Rather, we contend that these information network applications (among others), stand to benefit significantly from the placement of "filter" functionality at key network nodes, not only to reduce the number of duplicate items received by end-points, but also to reduce the overall traffic flowing through the network.

Paper Contributions: To the best of our knowledge we are the first to address the issue of network information multiplicity, and the first to propose a solution based on the deployment of de-duplication functionality at strategic locations in these information networks. We formally define the Filter Placement problem as a combinatorial optimization problem, and study its computational complexity for different types of graphs. We present polynomial-time constant-factor approximation algorithms for solving the problem, which is NP-hard for arbitrary graph topologies. Our experimental results, which we obtained through extensive simulations on synthetic and real-world information flow networks, suggest that in many settings a relatively small number of filters is fairly effective in removing a large fraction of duplicative information.

[1]Without loss of generality, de-duplication is "perfect".

Paper Outline: The rest of the paper is organized as follows: In Section 2 we survey related work, in Section 3 we introduce the necessary notation, formally define the Filter Placement problem and establish its complexity. In Section 4 we propose various Filter Placement algorithms for trees (4.1), DAGs (4.2) and arbitrary graphs (4.3). In Section 5, we present our experimental results and we conclude in Section 6 with a summary of current and on-going work.

2. RELATED WORK

While our work is the first to motivate, formalize, and study the Filter Placement problem, the literature is full of other works in various application domains that address two related problems: the use of coordinated content dissemination, and/or the selection of nodes to carry out special functionalities. In this section, we discuss examples of such approaches in the context of social networks, sensor networks, and content networks.

Social Networks: The problem of identifying k *key* nodes or agents has been addressed in the context of many different social-network studies and applications.

For example, a number of studies focused on the identification of k influential nodes such that information seeded (*e.g.*, advertisements) at these nodes would be maximally spread out throughout the network [5, 9, 11, 25]. All existing variants of this influence-maximization problem are concerned with improving the extent of information spread in the network, even if such maximization results in significant information redundancy. Our work is complementary in that it does not aim to change or improve information spread. Rather, our work aims to identify the placement of k filter nodes so as to minimize duplicate communication and processing of information *without* changing the original extent of information spread.

Another line of work focuses on the identification of k nodes that need to be monitored (and/or immunized) in order to detect contamination (prevent epidemics) [2, 12, 15, 23]. Here, the goal from selecting these key nodes is to inhibit as much as possible the spread of harmful information content (*e.g.*, viruses) by insuring that the selected nodes act as barriers that stop/terminate the flow of such content. In our model, filters do not terminate or inhibit the propagation of information. Rather, they remove redundancy in order to propagate a streamlined/sanitized ("cleaner") version of the information content. As a result, the underlying combinatorial problem we encounter when solving the Filter Placement problem is intrinsically different from the combinatorial problems addressed before.

Sensor Networks: At a high level, our work is related to mechanisms for information flow management in sensor networks. In sensor networks, nodes are impoverished devices – with limited battery life, storage capacity, and processing capabilities – necessitating the use of in-network data aggregation to preserve resources, and/or the use of coordination of node operations for effective routing and query processing. In-network aggregation is not aimed at de-duplication, but at the extraction of aggregate statistics from data (*e.g.*, sum, average, *etc*). Therefore, studies along these lines focus on the design of data-aggregation strategies [4, 8, 10, 16] as well as associated routing of queries and query results [13] in order to allow for reduced communication costs. Here we note that there is an implicit tradeoff between resource consumption and the quality of information flow. An aggregate is effectively a caricature (an approximation) of the information; aggressive aggregation implies a reduction in the quality (and hence utility) of information flow.

Coordinated communication of sensory data is exemplified by recent work that went beyond standard aggregation and focused on minimizing the communication cost required for the evaluation of multi-predicate queries scheduled across network nodes [3]. In that work, a dynamic-programming algorithm is used to determine an optimal execution order of queries. Examples of other studies that proposed coordination strategies include techniques that ensure efficient storage, caching, and exchange of spatio-temporal data [19, 20] or aggregates thereof [18].

In addition to their focus on balancing information fidelity and resource consumption, there is an implicit assumption in all of these studies that all nodes in the network are under the same administrative domain, and hence could be expected to coordinate their actions (*e.g.*, as it relates to routing or wake-sleep cycles). In our work, we consider settings in which coordination of node operation is not justified (since nodes are autonomous). Instead, we focus on reducing overheads by adding functionality to a subset of nodes, without concern to resource or energy constraints.

Content Networks: A common challenge in large-scale access and distribution networks is the optimal placement of servers to optimize some objective (*e.g.*, minimize average distance or delay between end-points and the closest content server) – namely, the classical facility location and k-median problems [17], for which a large number of centralized [17, 6] and distributed [21, 7] solutions have been developed. Conceptually, the Filter Placement problem could be seen as a facility location problem, wherein filters constitute the facilities to be acquired. However, in terms of its objective, the Filter Placement problem is fundamentally different since there is no notion of local measures of "distance" or "cost" between installed facilities and end-points. Rather, in our setting, the subject of the optimization is a global measure of the impact of *all* facilities (and not just the closest) on the utility that end-points derive from the network.

3. THE FILTER PLACEMENT PROBLEM

Propagation model: In this paper, we consider networks consisting of an interconnected set of entities (*e.g.*, users, software agents) who relay information items (*e.g.*, links, ideas, articles, news) to one another. We represent such a network as a directed graph $G(V, E)$, which we call the *communication graph* (c-graph). Participants in the network correspond to the nodeset V. A directed edge $(u, v) \in E$ in the graph represents a link, along which node v can propagate items to u. Some nodes of G generate new items by virtue of access to some information origin; we call these nodes *sources*. Sources generate distinct items – *i.e.*, any two items generated by the same source are distinct. Once an item is generated, it is propagated through G as follows: every node that receives an item blindly propagates copies of it to its outgoing neighbors. Since the propagation is blind, a node might receive many copies of a single item, leading to a *information multiplicity.*

Our information propagation model is deterministic in the sense that every item reaching a node is relayed to all neighbors of that node. In reality, links are associated with probabilities that capture the tendency of a node to propagate messages to its neighbors. Although our results (both theoretical and experimental) continue to hold under a probabilistic information propagation mode, for ease of presentation and without loss of generality we adopt a deterministic propagation model in this paper. Moreover, even though the propagation model defined by G could be used to communicate multiple items, in this paper we focus on a single item i. The technical results are identical for the multiple-item version of the problem.

Filters: Consider the production of a new item i by source s. In order for node v to receive this item, i has to travel along a directed path from s to v. Since there are several paths from s to v, node v will receive multiple copies of i. Moreover if v has children, then v propagates every copy of i it receives to each one of its children. To reduce the amount of redundancy (underlying information multiplicity), our goal is to add a filtering functionality to some of the nodes in G. We use *filters* to refer to nodes augmented with such filtering functionality.

A filter is a function that takes as input a multiset of items I and outputs a set of items I', such that the cardinality of I' is less than the cardinality of I. The choice of the filtering function depends strongly on the application it is used for. For ease of exposition, we will fix the filter function to be the function that eliminates *all* duplicates:[2] for every item the filter node receives, it will perform a check to determine if it has relayed that exact same item before. If not, it propagates the item to all of its neighbors. A filter node never propagates an already propagated item.

Objective Function: Let $v \in V$ be an arbitrary node in the graph. Define $\Phi(\emptyset, v)$ as the number of (not necessarily distinct) items that node v receives, when no filters are placed. Let $A \subseteq V$ be a subset of V. Then $\Phi(A, v)$ denotes the number of items node v receives, when filters are placed in the nodes in A. For a subset $X \subseteq V$, let $\Phi(A, X) = \sum_{x \in X} \Phi(A, x)$.

For a given item of information, the total number of messages that nodes in V receive in the absence of filters is $\Phi(\emptyset, V)$. For filter set A, the total number of messages that nodes in V receive is $\Phi(A, V)$. Thus, our goal is to find the set A of k nodes where filters should be placed, such that the difference between $\Phi(\emptyset, V)$ and $\Phi(A, V)$ is maximized – *i.e.*, the set A results in maximal de-duplication.

Problem 1 (Filter Placement–FP). *Given directed c-graph $G(V, E)$ and an integer k, find a subset of nodes $A \subseteq V$ of size $|A| \leq k$, which maximizes the function*

$$F(A) = \Phi(\emptyset, V) - \Phi(A, V)$$

The objective function F is always positive and monotone, since placing an additional filter can only reduce the number of messages. This also implies that F is bounded: $F(\emptyset) = 0 \leq F() \leq F(V)$. In addition, function F is submodular, since for every $X \subset Y \subset V$ and $v \notin Y$, it holds that $F(X \cup \{v\}) - F(x) \geq F(Y \cup \{v\}) - F(Y)$.

In the definition of Problem 1 there is a bound k on the size of the filter set A. In the next proposition we show that when the number of filters is not bounded, finding the minimal size filter placement that maximizes FP is trivial. Throughout the paper we use $n = |V|$ to denote the number of nodes in G.

[2]Generalizations that allow for a percentage of duplicates to make it through a filter are straightforward.

PROPOSITION 1. *Let $G(V,E)$ be a directed c-graph. Finding the minimal sized set of filters $A \subseteq V$, such that $F(A) = F(V)$ takes $O(|E|)$ time.*

PROOF. Let A be the nodes $v \in V$ that are not sinks and $d_{\text{in}}(v) > 1$, *i.e.*, $A = \{v \in V | d_{\text{in}}(v) > 1 \text{ and } d_{\text{out}}(v) > 0\}$. Turning these nodes into filters is enough to ensure that only one copy of an item propagates on every edge. □

Despite this result, FP for a filterset of fixed k-size on an arbitrary graph is NP-complete. Observe that the number of items propagating in the graph is infinite along any directed cycle. Our proof of Theorem 1 reduces SETCOVER to the relaxed version of the FP problem, where we want to find a filter set of size A, such that $\Phi(A,V)$ is finite.

THEOREM 1. *The* FP *problem on an arbitrary c-graph $G(V,E)$ is NP-complete.*

PROOF. First of all observe that for the communications graph G the set $A \subseteq V$ maximizes $F(.)$ whenever $\Phi(A,V)$ is minimized. Hence we will prove the hardness of Problem 1 by showing that finding a placement of k filters A such that $\Phi(A,V)$ is minimized is NP-complete. We prove this by showing that finding the smallest integer k, for which the number of received items in graph $G(V,E)$ is finite, is equivalent to the SETCOVER problem. An instance of SETCOVER consists of the universe $U = \{u_1, u_2, \ldots, u_m\}$ and a set $S = \{S_1, S_2, \ldots, S_n\}$, where $\forall i, S_i \subseteq U$ is a subset of U and k is an integer. The goal is to find a subset $S' \subseteq S$ such that $|S'| \leq k$ and $\{u_j \in U : u_j \in \cup_{S_i \in S'} S_i\} = U$. Define the instance of FPas follows. First, define graph G by creating a node v_i for every set $S_i \in S$. Fix an arbitrary cyclic order σ of the nodes v_i. A cyclic order is the same as a linear order with the additional constraint that $\sigma(n+1) = \sigma(1)$. For every instance $u_j \in U$ add an edge $v_{j_1} \to v_{j_2}$ whenever $u \in S_{j_1}$, $u \in S_{j_2}$ and $\sigma(v_{j_1}) < \sigma(v_{j_2})$. Observe that this adds a directed cycle to the graph for every $u \in U$. Also add a source v_s to the graph and add an edge from the source to all other nodes in the graph. Imagine now that the source creates one single item and propagates that to its children. Observe that now infinite number of items will propagate on every directed cycle. Let k be the integer, specified in the instance of SETCOVER and let l be an arbitrary finite integer. Now for the decision version of the FPproblem if the answer tot he question "*Is there a filter assignment A with $|A| \leq k$, such that $\Phi(A,v) \leq l$?*" is "*YES*", then this also implies a solution with k sets for the SETCOVER problem. Since the decision version of SETCOVER is NP-complete this reduction shows that FPis also NP-complete. □

4. FILTER PLACEMENT ALGORITHMS

In this section we present algorithms for the FP problem on different types of c-graphs, namely trees, DAGs and arbitrary directed graphs.

4.1 Filter Placement in a Tree

While FP is hard on arbitrary graphs, and as we will show also on DAGs, it can be solved in polynomial time with dynamic programing on c-trees. We call a graph $G(V,E)$ a *communication tree* (c-tree), if in addition to being a c-graph, G is a tree when removing the source node. The recursion of the dynamic programming algorithm is done on the children of every node. Transforming the input c-tree G into a binary tree makes it computationally more feasible. This transformation can be done in the following way: for every node $v \in V$, if $d_{\text{out}}(v) \leq 2$ then do nothing. Otherwise, fix an arbitrary ordering $v_1, v_2, \ldots v_r$ of the children of v, where $d_{\text{out}}(v) = r$. Let v_1 be the left child of v. Create a new node u_1 and let that be the right child of v. Let the remaining children of v be the children of u_1. Repeat these steps until u_{r-1} has only two children: v_{r-1} and v_r. The edges adjacent to the source will continue to be connected to the nodes in V. The resulting binary tree is G'. Observe that the number of nodes in G' is at most twice as much as the number of nodes in G. Also notice that if the maximum out-degree in tree G is Δ then the height of G' is at most a factor of $\log(\Delta)$ larger than G.

We apply dynamic programming on G': Let $\text{OPT}(v,i,A)$ be the function that finds an optimal filter set A of size $|A| \leq i \leq k$ in the subtree rooted in v. The recursion is as follows: for every $i = 0 \ldots k$ we can compute $\text{OPT}(v,i,A)$ by

$$\begin{aligned}
&\text{OPT}(v,i,A) = \max\{ \\
&\qquad \max_{j=0\ldots i}\{\text{OPT}(v_l,j,A) + \text{OPT}(v_r,i-j,A)\}, \\
&\max_{j=0\ldots i-1}\{\text{OPT}(v_l,j,A\cup\{v\}) + \text{OPT}(v_r,i-1-j,A\cup\{v\})\}\}
\end{aligned}$$

where v_l and v_r denote the left and right child of v. The first term of this recursion corresponds to the case where we do not place a filter in v, hence a total number of i filters can be placed in the subtrees. The second term corresponds to the case when we do place a filter in v and only $i-1$ can be placed in the subtrees. The optimal solution for the whole tree is then $\text{OPT}(r,k,A)$ where $r \in V$ is the root of the tree. The above recursion does not guarantee that we do not choose any dump nodes of the binary tree. For this reason, we omit the second term of the recursion when v is a dump node. Building the binary tree takes $O(n\Delta)$ time. We have to evaluate the recursion $O(k)$ times for every node. One evaluation takes $O(k)$ computations. There are $O(n\log(\Delta))$ nodes in G', which makes the total running time $O(n\Delta + k^2 n\log(\Delta))$.

4.2 Filter Placement in DAGs

Consider c-graphs $G(V,E)$, which are directed and acyclic (DAGs). Although DAGs seem to have simpler structures than arbitrary graphs, the FP problem is NP-complete even on DAGs.

THEOREM 2. *The* FP *problem is NP-complete when the c-graph $G(V,E)$ is a DAG.*

PROOF. We reduce the NP-complete VERTEXCOVER problem to the FP problem on DAGs. We say that for an undirected graph $G(V,E)$, a set $A \subseteq V$ is a *vertex cover* of G, if every edge in E is incident to at least one node in A. For an instance of the VERTEXCOVER problem, let $G(V,E)$ be an undirected graph and k an integer. The decision version of the problem asks for a set $A \subseteq V$ of size k that is a vertex cover.

Define the DAG $G'(V',E')$ of the corresponding FP problem as follows. Let $V' = V \cup \{s,t\}$ contain the nodes in G, an additional source node s and an additional sink t. Let E' contain all edges in E. In addition to that, add an edge from the source to every node, and from every node to the sink. Fix an arbitrary order σ of the nodes in V', such that s is the first and t is the last in this ordering. Then direct every edge $(u,v) \in E'$ from u to v if $\sigma(u) < \sigma(v)$, otherwise from v to u. This will naturally result in a DAG. Let m be an arbitrary integer such that $m > \Omega(|V'|^{10})$. We will replace every directed edge in E' (including the edges incident to s and t) with the following *multiplier* tool (Figure 2). For every edge (u,v) we add m new nodes: $w_1, w_2, \ldots, w_m$, and $2m$ new directed edges: (u, w_i) and (w_i, v). Observe, that by this exchange, the size of the graph only changes by a polynomial factor of the original size. Now we will proof that there exists a vertex cover A of size at most k for this instance of the VertexCover problem if and only if there exists an FP A' of size k where $\Phi(A', V') < O(m^3)$. In addition we claim that $A' \subseteq V$ and thus $A = A'$ is the desired solution for the VertexCover.

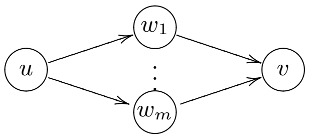


Figure 2: "Multiplier edge" construction for G'. When x items leave u, $x \cdot m$ items arrive at v.

Let us assume A' is the solution of size k for the FP problem and $\Phi(A', V') < \Omega(m^3)$. We will show that $A' \subseteq V$ and that A is a vertex cover of G. In special we will show that, if there is an edge (u,v) in E that is not incident to any node in A, then $\Phi(A', V') > O(m^3)$. As seen in Proposition 1, it is more advantageous to put the filter in the parent of a node with indegree 1, than in the node itself. For this reason, we can assume that filters are only placed in the nodes w_i of the multiplier tool, if all nodes with indegree larger than 1 are already filters. In this case, all nodes in $V \subseteq V'$ would be filters, and then A is a trivial vertex cover. Thus we can assume $A' \subseteq V$. Now we will show that A is a vertex cover. Let us consider the subgraph G_{uv} depicted in Figure 3, corresponding to the nodes $u, v \in V$ and the adjacent edges. σ_i depicts the number of incoming items on that edge. Let $\Sigma_u = \sigma_1^u + \sigma_2^u + \ldots + \sigma_u^u$ and $\Sigma_v = \sigma_1^v + \sigma_2^v + \ldots + \sigma_v{}^v$ be the total number of items u and v receive from other nodes.

Let us assume that for every edge $(u,v) \in E$ at least one $\{u,v\}$ is in A'. Then we can compute an upper bound on the number of items propagating in this subgraph, with respect to A'. There are three possible cases:

case $u \in A'$, $v \in A'$**:** In this case the total number of items propagating on the edges of G_{uv} is $\Sigma_u + m + m + 2m + \Sigma_v + m + m + m + m$, which is $O(m^2)$.

case $u \in A'$, $v \notin A'$**:** Then the total number of items propagating on the edges of G_{uv} is $(\Sigma_u + m) + 2m + (\Sigma_v + m) + ((m + (\Sigma_v + m)) \cdot m)$, which is $O(m^2)$;

case $u \notin A'$, $v \in A'$**:** Here the total number of items propagating on the edges of G_{uv} is $(\Sigma_u + m) + 2 \cdot ((\Sigma_u + m) \cdot m) + (\Sigma_v + m) + m$, which is $O(m^2)$;

The total number of subgraphs G_{uv} in G' is $|E|$. Thus if A is a vertex cover in G, then for A' the number of items is bounded $\Phi(A', V') = O(n^2 \times m^2) < m^3$. On the other hand let us assume that $A = A'$ is not a vertex cover in G. This means that there is at least one edge $(u,v) \in E$ for which $u \notin A'$ and $v \notin A'$. Then:

case $u \notin A'$, $v \notin A'$**:** The total number of items propagating on the edges of G_{uv} is $(\Sigma_u + m) + 2 \cdot ((\Sigma_u + m) \cdot m) + (\Sigma_v + m) + ((((\Sigma_u + m) \cdot m) + (\Sigma_v + m)) \cdot m)$, which is $O(m^4)$ for a worst-case Σ_u,Σ_v and $O(m^3)$ in the best case;

In this case $\Phi(A', V') = \Omega(m^3)$ in contradiction with our assumption. In this proof we showed that there exists a vertex cover of size k for $G(V,E)$ if and only if there exists an FP A' of size k for $G'(V', E')$, with a bounded number of total items $\Phi(A', V') = O(m^2)$. Thus the FP problem is NP-complete.

□

In the remainder of this section, we propose polynomial-time algorithms, that result in solutions for the FP problem, that we prove to be effective in our experiments.

First, we start with a naive approach, `Greedy_1`[3] (See Algorithm 1). Consider node $v \in V$; v will receive items on its incoming edges and will forward a copy of every item on every one of its outgoing edges. We can now compute a lower bound on the number of copies of an item that v is propagating: $m(v) = d_{\text{in}}(v) \times d_{\text{out}}(v)$. `Greedy_1` computes $m(v)$ for every $v \in V$ and chooses the k nodes with the highest $m()$ values. Computing $m()$ depends on the way the graph is stored. In general it takes $O(|E|)$ time, since we need to compute the in and outdegree of every node. Finding the k largest values of $m()$ requires $O(kn)$ computations, which makes the total running time of `Greedy_1` $O(kn + |E|)$.

Algorithm 1 `Greedy_1`

```
   Input: DAG $G(V,E)$ and integer $k$.
   Output: set of filters $A \subseteq V$ and $|A| \leq k$.
1: while V.hasMoreElements() do
2:     v = V.nextElement()
3:     $M \leftarrow (v, d_{\text{in}}(v) \times d_{\text{out}}(v))$
4: $A \leftarrow \{v : k$ highest $M(v)$ value$\}$
5: return A
```

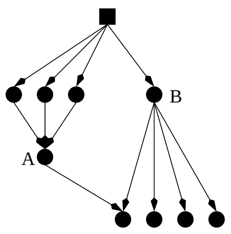


Figure 4: for $k = 1$ `Greedy_1` places a filter in B while the optimal solution would be to place a filter in A.

Although `Greedy_1` is simple and efficient, for many graphs it does not yield an optimal solution as exemplified in Figure 4. Without any filters, the total number of received

[3]The name will be meaningful later, when we propose heuristics that can be viewed as extensions of `Greedy_1`.

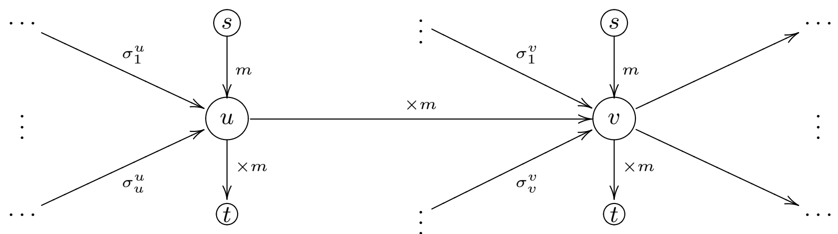


Figure 3: Isolated subgraph $G_{uv} = \{v, v, s, t\}$ of G'.

items in the graph is 14. `Greedy_1` would place a filter in node B; this is because $m(B) = 1 \times 4$ is the largest $m()$ value in this graph. However the optimal solution would be to place a filter in A, for which $m(A) = 3 \times 1$. Placing a filter in B leaves the number of received items unchanged. Making node A a filter instead, would yield a total number of 12 received items.

In light of this illustrative example, we propose `Greedy_All`, which is an improved extension of `Greedy_1`. First, let us look at the propagation of an item i that is created by the source. In order for i to be received at node v it has to propagate along a path from s to v. In fact, v will receive as many copies of i as the number of paths from s to v. We will define this as the Prefix of v: $\text{Prefix}(v) = \#\text{path}(s, v)$. Likewise, we denote by Suffix the number of paths that go from v to any other node: $\text{Suffix}(v) = \sum_{u \in V} \#\text{path}(v, u)$. The product of these two values expresses the effect that node v has on the number of copies of i that propagate in the graph. We call this value the *impact* of v.

Definition 1. *Let $G(V, E)$ be a DAG. We call $I(v) = \textit{Prefix}(v) \times \textit{Suffix}(v)$ the* Impact *of node $v \in V$.*

Observe that placing a filter in node v reduces the number of items that v induces to $1 \times \text{Suffix}(v)$. To put this in terms of the objective function: $F(v) = \Phi(V, \emptyset) - \Phi(V, \{v\}) = (\text{Prefix}(v) - 1) \times \text{Suffix}(v)$. This is the concept underlying our `Greedy_All` algorithm. The algorithm first chooses the node with the highest impact. Placing a filter at that node might change the impact of other nodes. Hence an update of the impact of every node is required. Then `Greedy_All` chooses again the node with the highest impact. The algorithm repeats this for k steps. Observe that `Greedy_1` only looks at the immediate neighbors of every node, whereas `Greedy_All` is more exhaustive and considers all the nodes in the graph.

Algorithm 2 `Greedy_All`

```
   Input: DAG $G(V, E)$ and integer $k$.
   Output: set of filters $A \subseteq V$ and $|A| \leq k$.
1: find topological order $\sigma$ of nodes
2: for $i = 1 \ldots k$ do
3:     for $j = 1 \ldots n$ do
4:         compute $I(v_j)$
5:     $A \leftarrow \text{argmax}_{v \in V} I(v)$
6: return A
```

Because of the properties of $F()$, we can use the well-known result by Nemhauser *et al.* [22], which states that for any maximization problem, where the objective function is a positive, monotone and submodular set-function, the greedy approach yields a $(1 - \frac{1}{e})$-approximation.

Theorem 3. *The `Greedy_All` algorithm for problem 1 is an $(1 - \frac{1}{e})$-approximation.*

In order to compute the impact, we need to compute the Prefix and Suffix of every node. This can be done recursively in the topological order σ of the nodes, in $O(n|E|)$ time with some clever bookkeeping: for every node we keep track of the number of items it receives, and we keep a list containing the number of paths leading from every ancestor.[4] Let P_v denote the set of parents of node v, and let $v[a]$ denote the entry in the list of v for ancestor a. Then

$$\text{Prefix}(v) = \sum_{u \in P_v} \text{Prefix}(u) \tag{1}$$

$$\forall a \in V : v[a] = \sum_{u \in P_v} u[a] \tag{2}$$

When evaluating formulas (1), (2) we need to do an addition for every edge, and an addition for every node in the list. The Suffix of node v can be computed as the sum of all list entries corresponding to v. This can be maintained and updated while computing the recursion. To get an accurate count, every node v's list contains itself and is set to one, $v[v] = 1$. Overall, this yields an $O(n|E|)$ running time for the computation of the impact. Observe now that placing a filter in node f affects the paths leading through f. We can capture this effect, by setting all values in f's list to zero, before using this list in the recursion. Observe that the change in f's list changes the Suffix of all nodes preceding f, and the Prefix of all nodes succeeding it. For this reason, we need to make a pass over the whole graph when updating the impact. `Greedy_All` updates the impact k times. Therefore the overall time complexity is $O(kn|E|)$. This can be $O(kn^3)$ in worst case, but for most real-life graphs $|E| < O(n\log n)$.

An alternative to `Greedy_All` is the `Local` algorithm. `Local` takes as input a set of k filter *candidates* and makes a pass over the nodes in V in the fixed topological order σ. Let $v \in V \setminus A$ be the next node in σ that is not handled yet. v is exchanged with a member of A in the candidate set, if this change results in an increase in the objective function. v is exchanged with that node in A that yields the largest

[4]In a topological order of nodes, every edge is directed from a smaller to a larger ranked node in the ordering. This order can be found in $O(|E|)$ time.

improvement. For every node $v \in V \setminus A$ we need to compute $F()$ k times, once for every $a \in A$. As we have seen in the previous sections, $F()$ can be computed by the recursive formula (1) using $\text{Prefix}(a) = 1$ for every filter in $a \in A$. Observe that since we are examining the nodes in V in the topological order σ, in iteration i we only need to update the Prefix() value for nodes $u \in V$ when $\sigma(v_i) < \sigma(u)$. This implies a running time of $O(|E_i|)$ for the update of Prefix(), where E_i is the set of edges below v_i with respect to σ. Since A has size k, this recomputation needs to be done k times for every $v \in V$. Hence, the total running time is $O(k \sum_{v \in V} |E_v|)$.

As we can see in Section 5, `Local` (see Algorithm 3) on its own performs quite similar to `Greedy_All`. We could also use it as a post-processing step, by using the resulting filter set of `Greedy_All` as the input candidate set for `Local`. Since `Local` monotonically increases $F()$, this must yield an improved result. To compare `Local` and `Greedy_All`, look at the example in Figure 5. Node A has the largest overall impact. For this reason, `Greedy_All` will choose A as its first filter. Then, for $k = 2$ `Greedy_All` places filters in nodes A and C. Alternatively, `Local` cuts A out of the final filter set, and chooses B and C.

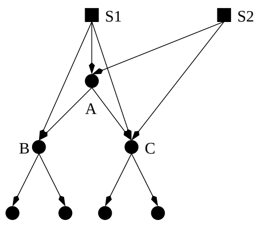


Figure 5: For k=2, `Greedy_All` will choose nodes A and C, while `Local` puts the filters in B and C. The optimal filter set on this graph is $\{B, C\}$

Algorithm 3 `Local`

```
    Input: DAG $G(V, E)$, topological order $\sigma$ and candidate filter set $A$.
    Output: set of filters $A' \subseteq V$.
 1: for $i = n - k \dots n$ do
 2:     $a = 0$
 3:     $F = F(A)$
 4:     for $j = 1 \dots k$ do
 5:         if $F(A \setminus \{a_j\} \cup \{\sigma(i)\}) > F$ then
 6:             $a = a_j$
 7:             $F = F(A \setminus \{a_j\} \cup \{\sigma(i)\})$
 8:     if $F \neq F(A)$ then
 9:         $A = A \setminus \{a\} \cup \{\sigma(i)\}$
10: return A
```

4.3 Filter Placement in Arbitrary Graphs

We have seen that solving FP on arbitrary graphs is NP-hard. In this section we propose a heuristic to choose an acyclic subgraph from any graph. This allows us to apply the algorithms designed for DAGs on this subgraph (See Figure 6). Let c-graph $G'(V, E')$ be an arbitrary directed graph. Fix an arbitrary ordering σ of the nodes in V. We call an edge $(v, u) \in E'$ a *forward* edge if $\sigma(v) < \sigma(u)$; otherwise it is a *backward* edge. A well-known 2-approximation (Algorithm 4) for choosing an acyclic subgraph is the following greedy algorithm: fix an arbitrary order σ of the nodes. Let F be the set of forward edges with respect to σ, and let B be the set of backward edges. If $|F| > |B|$ then choose the DAG $G(V, F)$, else choose the edges in B: $G(V, B)$. The drawback of this algorithm is that it does not guarantee the resulting DAG to be connected. For this reason we develop our own algorithm, `Acyclic` to choose a maximal acyclic subgraph.

Algorithm 4 Greedy algorithm for choosing an acyclic subgraph.

```
    Input: directed graph $G'(V, E')$
    Output: acyclic directed graph $G(V, E)$
 1: fix an arbitrary ordering $\sigma$ of the nodes in $v$
 2: $F = \{(u, v) \in E' | \sigma(u) < \sigma(v)\}$
 3: $B = \{(u, v) \in E' | \sigma(u) > \sigma(v)\}$
 4: if $|F| > |B|$ then
 5:     return $G(V, F)$
 6: else
 7:     return $G(V, B)$
```

`Acyclic` (Algorithm 5) leverages the idea of forward and backward edges, but it also ensures a connected subgraph. It finds the strongly connected components $G_1, G_2, \dots G_r$ of G' by running two subsequent DFS searches: one in the direction of the edges and one in the opposite direction of the edges. Finding the DFS tree takes only time proportional to the number of edges, which is $O(|E'_i|)$. For every strong component $G_i \subseteq G'$, fix the same order σ_i of the nodes in which they were found by the first DFS search. Next, we add the larger set of forward or backward edges to the DAG. For this we need to go once more through all the edges in G_i, which again takes $O(|E'_i|)$ time. Since by definition of strong connectedness, there is no directed cycle in G', that contains nodes from multiple strong components, we also add every edge that connects nodes in different strong components of G'. The total running time of the `Acyclic` algorithm is $O(|E'|)$. The resulting graph $G(V, E)$ is maximal; no other edge can be added without creating a directed cycle. There are examples, where starting the DFS search from different nodes in the graph, creates DAGs with different numbers of edges. In the graph in Figure 6 starting the DFS from node 1 results in a DAG containing all edges except edge $(n, 1)$. However, starting the DFS from node $n/2$ would result in a DAG that does not contain edges going from any node in $\{1, 2, \dots n/2\}$ to a node in $\{\frac{n}{2}, \frac{n}{2} + 1, \dots n\}$. Note that in our experiments we do not start from arbitrary nodes. We choose nodes that are either known to be the initiator of items or that are based on their timestamps: Nodes that have acquire the item first are very likely to be the initiators of that item.

5. EXPERIMENTAL EVALUATION

In this section, we present a comparative experimental evaluation of the various Filter Placement algorithms, using synthetic data sets as well as real data sets capturing the propagation of information in real-life scenarios.

Performance Metric: To measure and compare the effectiveness of various algorithms, we define the `Filter Ratio` (FR) as the ratio between the objective function using filters

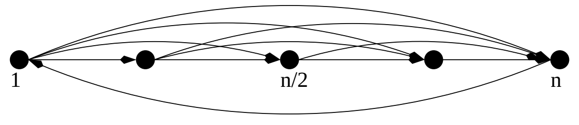


Figure 6: `Acyclic` finds a DAG with all but one edges if started from node 1. It will find a DAG with only half the edges if started from node $n/2$

```
Algorithm 5 Acyclic
    Input: directed graph $G'(V, E')$
    Output: acyclic directed graph $G(V, E)$
 1: find the strongly connected components $G_1, G_2 \dots G_r \subseteq G'$ with help of DFS
 2: for every component $G_i$ fix the ordering $\sigma_i$ of the nodes defined by the first DFS
 3: $E_i = \text{argmax}\{|F_i|, |B_i|\}$
 4: $E = \cup_{i=1}^{r} E_i \cup E' \setminus \{\cup_{i=1}^{r} G_i\}$
 5: return $G(V, E)$
```

```
Algorithm 6 FP on an arbitrary graph
    Input: directed graph $G'(V, E')$, integer $k$.
    Output: acyclic directed graph $G(V, E)$, filter set $A$
 1: $G(V, E) = \texttt{Acyclic}(G'(V, E'))$
 2: $A$ = solve FP on $G$
 3: return $G(V, E)$ and $A$
```

deployed in a subset of nodes (A) and the maximum value of the objective function –*i.e.*, the level of de-duplication delivered by installing filters at the nodes in A. A FR of 1 underscores complete de-duplication.

$$FR(A) = \frac{F(A)}{F(V)}$$

Baseline Algorithms: In our experiments a number of deterministic and random algorithms serve as baselines, against which we compare our more sophisticated methods. As we find, these baseline algorithms by themselves often reveal interesting insights.

`Greedy_Max` (`G_Max`) starts by computing the impact of nodes in a similar way to `Greedy_All`(`G_All`), but once the impact of all nodes is calculated, `Greedy_Max` selects the k nodes with the highest impact as filters.

`Greedy_L` (`G_L`) computes a simplified impact for every node: it computes $I'(v) = \text{Prefix}(v) \times d_{\text{out}}(v)$, which is the number of items v propagates to its immediate children. It selects as filters the top k nodes with respect to I'.

`Random_k` (`Rand_K`) chooses k filters from V uniformly at random.

`Random_Independent` (`Rand_I`) is very similar to `Random_k`. Every node decides by itself with probability $\frac{k}{n}$ whether or not to become a filter.

`Random_Weighted` (`Rand_W`) weights $w(v) = \sum_{u \in C_v} \frac{1}{d_{\text{in}}(u)}$ are assigned to every node, where $C_v = \{u \in V | (v, u) \in E\}$ is the set of children of v. Then every node decides with probability $w(v) \times \frac{k}{n}$ to become a filter. The intuition behind this is that the influence of node v, on the number of items that its child u receives is inversely proportional to the indegree of u.

Note that while we cannot guarantee that the number of filters for randomized algorithms is k, they are designed so that the expected number of filters is k. We run the randomized algorithms 25 times and then average the results.

Results Using Synthetic Data Sets: To test some basic properties of our algorithms we generate synthetic graphs. First we assign nodes to 10 levels randomly, so that the expected number of nodes per level is 100. Next, we generate directed edges from every node v in level i to every node u in level $j > i$ with probability $p(v, u) = \frac{x}{y^{j-i}}$. For $\frac{x}{y} = \frac{1}{4}$ we generate a graph with 1026 nodes and 32427 edges. For $\frac{x}{y} = \frac{3}{4}$ we generate 1069 nodes and 101226 edges. The CDFs of the indegree are shown in Figure 7. The CDFs of the outdegree are quite similar, and thus omitted due to space limitations. Observe that nodes on the same level have similar properties; the expected number and length of paths going through them is the same.

Figures 8 and 9 show a gradual increase in FR as a function of the number of filters. This implies that there are several nodes, that cover distinct portions of the paths in the graphs, with roughly the same size.

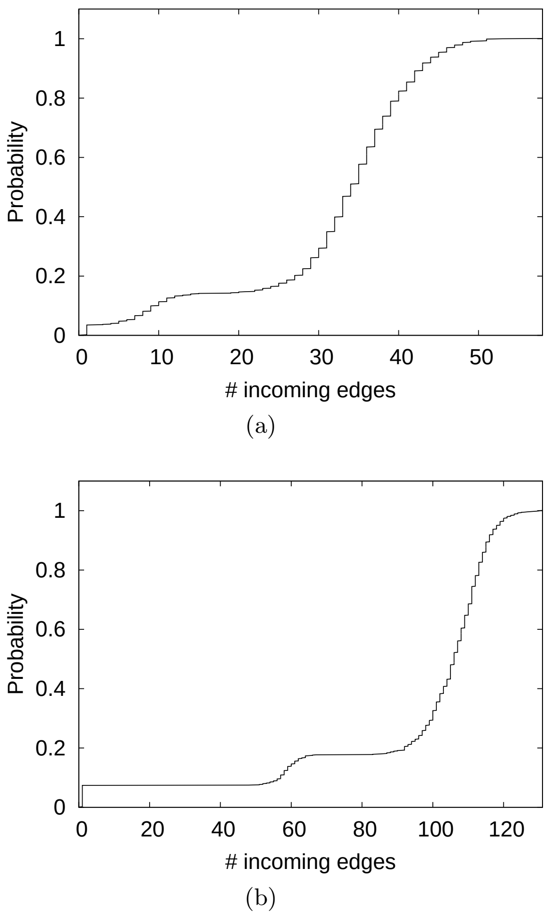


Figure 7: CDF of indegrees for synthetic graphs generated with (a) $x = \frac{1}{4}$ and (b)$x = \frac{3}{4}$. The CDF of the outdegrees for these graphs are quite similar.

Results Using the `Quote` Data Set: The `Quote` data set by Leskovec *et al.* [14] contains the link network of mainstream media sites ranging from large news distributors to

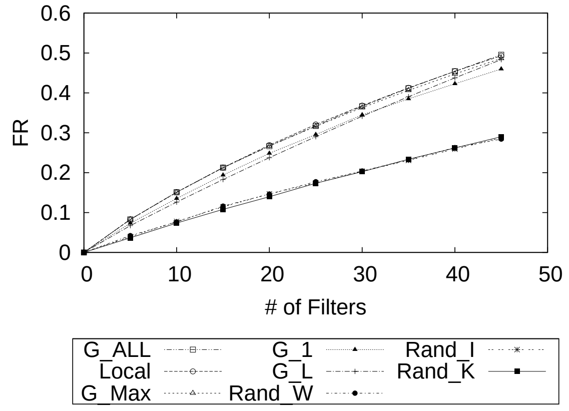


Figure 8: FR **for synthetic graph generated with** $x = \frac{1}{4}$**. X-axis corresponds to the number of filters, Y-axis corresponds to the** FR **for different algorithms**

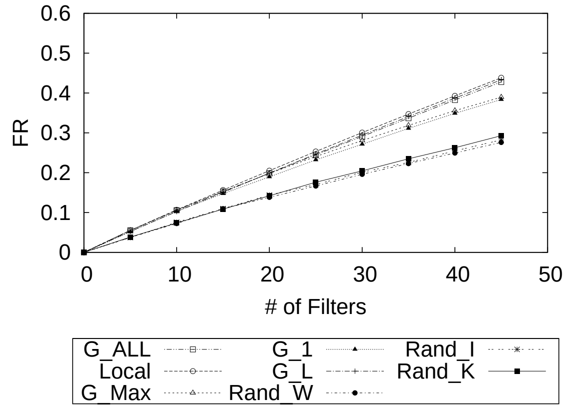
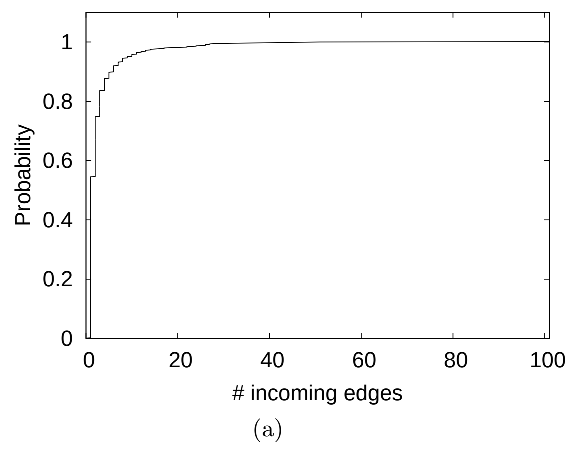


Figure 9: FR **for synthetic graph generated with** $x = \frac{3}{4}$**. X-axis corresponds to the number of filters, Y-axis corresponds to the** FR **for different algorithms**



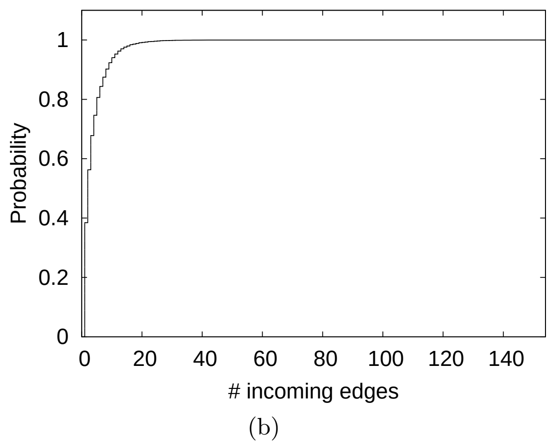


Figure 10: CDF of node indegree of real dataset based c-graphs. (a) `G_Phrase`**, (b)** `G_Citation`

personal blogs, along with timestamped data, indicating the adoption of topics or different phrases by the sites. We utilize monthly traces from August 2008 to April 2009 to generate a c-graph `G_Phrase`. Since the `Quote` graph, which has over $400K$ edges, is very large, in our experiments we select a subgraph. The subgraph we chose contains the nodes and adjacent edges, corresponding to sites that use the phrase "lipstick on a pig". Sites may freely link to each other, which might result in cycles. Thus we run `Acyclic` to find a maximal acyclic subgraph in this graph. Since the phrase was used by a candidate during the presidential campaign in 2008, there is no clear initiator of the phrase in the blogosphere. For this reason, we initiate a run of `Acyclic` from every node in the graph, and then choose the largest resulting DAG. This DAG has a single source: the node `Acyclic` was started from. It contains 932 nodes and 2,703 edges.

Figure 10(a) shows the CDF of the nodes' in-degree in `G_Phrase`. We found that almost 70% of the nodes are sinks and almost 50% of the nodes have in-degree one. There are a number of nodes, which have both high in- and out-degrees. These are potentially good candidates to become filters. The steep curve of FR for `G_Phrase` in Figure 11 confirms our intuition: indeed there are a couple of central nodes that achieve perfect de-duplication. The fact, that the different versions of the greedy algorithm and `Local` give slightly different results shows that these central nodes are also interconnected among themselves. We tracked the connection between the node chosen first by `Greedy_All` (node A), and the node chosen first by `Greedy_L` (node B). These nodes were connected by a path of length 2. The impact of A is larger then the impact of B, since A also influences all of B's children. However B is chosen over A by `Greedy_L`, since the prefix of B is much larger than that of A. The central nodes also explain why `Random_Weighted` performs so well: nodes with large weights (and thus high probability of becoming filters) are those nodes with a large number of children. The randomized algorithms `Random_k` and `Random_Independent` perform significantly worse than all others because of the high fraction of sink nodes in the graphs.

Results Using APS Research Data Set: The APS research dataset [1] contains of the citation network of over 450,000 articles from Physical Review Letters, Physical Review, and Reviews of Modern Physics, dating back to 1893. The dataset consists of pairs of APS articles – one citing the other. This can be viewed as a graph with a directed

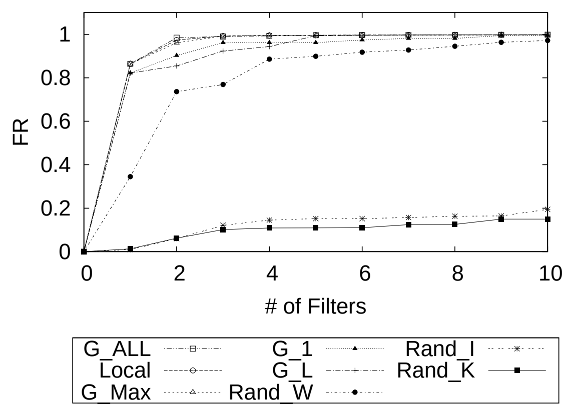


Figure 11: FR **for** `G_Phrase` **on the** `Quote` **dataset. X-axis corresponds to the number of filters, Y-axis corresponds to the** FR **for different algorithms**

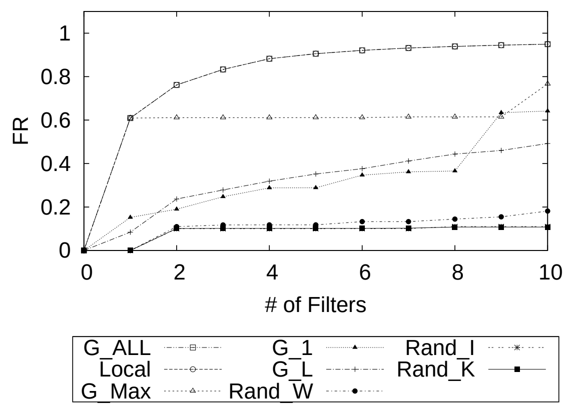


Figure 12: FR **for** `G_Citation` **in the** `APS` **dataset. X-axis corresponds to the number of filters, Y-axis corresponds to the** FR **for different algorithms**

edge from node A to B if B cites A. We select article [24], published in Physical Review as the source node, and take the subgraph of nodes that can be reached from this node through directed paths. In this case only the node corresponding to paper [24] is connected to the source. The resulting subgraph is intended to portray the propagation of an original concept or finding in this paper through the physics community: a filter in this setting can be seen as an opportune point in the knowledge transfer process to purge potentially redundant citations of the primary source (*e.g.*, derivative work).[5] The resulting citation graph `G_Citation` is acyclic and contains 9,982 nodes and 36,070 edges. Figure 10(b) shows the CDF of the nodes' in-degree. Observe that almost 99% of the nodes have degree less than ten.

`G_Citation` unveils important insights about effective filter placement. As evident in Figure 12, `Greedy_All` and `Local` perform much better then `Greedy_1`, `Greedy_L` or even `Greedy_Max`. The long range over which `Greedy_Max` is constant prompted us to examine the structure of the graph for an explanation. As sketched in Figure 13, we found that there is a set of nine nodes, all of which were on the same path and had an in-degree of one. They were all high-impact nodes since they receive a large number of items from above and all have high out-degrees. However, placing a filter in the first node highly diminishes the impact of the remaining nodes. This also explains the bad performance of `Greedy_1` and `Greedy_L`. The best node to act as a filter is the first of the nine nodes. However, given its relatively low out-degree, this node is not be chosen by these algorithms.

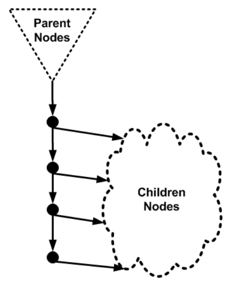


Figure 13: Sketch of `APS` **graph**

Summary of Comparative Performance: Comparing the results for the synthetic data sets (Figures 8 and 9) to the results for the real datat sets (Figures 11 and 12) reveals a significant difference in the marginal utility from added filters (the steepness of the performance curve). For the synthetic data sets, there is a slow gradual improvement in performance for all algorithms as the number of filters increases, suggesting a fairly constant marginal utility of additional filters throughout. In contrast, for the `Quote` data set, almost all duplication can be filtered out with at most ten filters, implying no marginal utility from added filters beyond that point. For the `APS` data set, there is more of a spread in the performance of the algorithms, but the best algorithms have very steep performance curves as well.

This disparity between the performance using synthetic and real data sets can be explained by the structure of the graphs underlying these data sets. Synthetic graphs are densely connected, and as a result paths cannot be covered with a small number of filters. On the other hand, the real data sets have a small set of "key" nodes that cover all paths in the graph. This indicates that while our methods can be used effectively on all types of graphs, placing filters is more effective when operating over sparse graphs (which are more prevalent in many information networks).

6. CONCLUSION

Networks in which nodes indiscriminately disseminate information to their neighbors are susceptible to information multiplicity – *i.e.*, a prevalence of duplicate content communicated over a multitude of paths. In this paper, we proposed a particular strategy to mitigate the negative implications from information multiplicity. Our approach relies on the installation of filters (de-duplication agents) at key positions in the information network, and accordingly defined the Filter Placement problem. In addition to characterizing the computational complexity of Filter Place-

[5]In a live corpora of interconnected documents and citations, filters can be seen as the key documents in which to consolidate references to a specific source.

MENT (polynomial for trees, but NP-hard for DAGs and arbitrary graphs), we devised a bounded-factor approximation as well as other heuristic algorithms, which we evaluated experimentally using syntehtic and real data sets. Our results suggest that in practice, our FILTER PLACEMENT algorithms scale well on fairly large graphs, and that typically a small number of filters is sufficient for purposes of content de-duplication in real-world (typically sparse) information networks.

Our current and future work is focusing on extensions to the information propagation model adopted in this paper to take into consideration multirate information sources. In addition, we are investigating different formulations of the FILTER PLACEMENT problem in which the filter functionality goes beyond the deterministic and probabilistic de-duplication considered in this paper.

7. REFERENCES

[1] Aps data sets for research. https://publish.aps.org/datasets.

[2] J. Aspnes, K. Chang, and A. Yampolskiy. Inoculation strategies for victims of viruses and the sum-of-squares partition problem. In *YALEU/DCS/TR-1295*, 2004.

[3] G. Chatzimilioudis, H. Hakkoymaz, N. Mamoulis, and D. Gunopulos. Operator placement for snapshot multi-predicate queries in wireless sensor networks. In *Proceedings of the 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, MDM '09, pages 21–30. IEEE Computer Society, 2009.

[4] K. Dasgupta, K. Kalpakis, and P. Namjoshi. An efficient clustering-based heuristic for data gathering and aggregation in sensor networks. In *in Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC*, pages 1948–1953, 2003.

[5] P. Domingos and M. Richardson. Mining the network value of customers. In *Seventh International Conference on Knowledge Discovery and Data Mining*, 2001.

[6] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(6):992–1009, 1978.

[7] C. Frank and K. Römer. Distributed facility location algorithms for flexible configuration of wireless sensor networks. In *Proceedings of the 3rd IEEE international conference on Distributed computing in sensor systems*, pages 124–141. Springer-Verlag, 2007.

[8] A. Goel and D. Estrin. Simultaneous optimization for concave costs: Single sink aggregation or single source buy-at-bulk. In *In Proc. of the 14 th Symposium on Discrete Algorithms (SODA*, pages 499–505, 2003.

[9] J. Goldberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, 2001.

[10] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming, 2001.

[11] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence throw a social network. In *ACM SIGKDD*, 2003.

[12] A. Krause and C. Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, 2007.

[13] B. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. *Distributed Computing Systems Workshops, International Conference on*, 0:575, 2002.

[14] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 497–506. ACM, 2009.

[15] J. Leskovec, A. Krause, C.Guestrin, C.Faloutsos, J.VanBriesen, and N.Glance. Cost-effective outbreak detection in networks. In *ACM SIGKDD*, 2007.

[16] H. Luo, Y. Liu, and S. K. Das. Distributed algorithm for en route aggregation decision in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 8, January 2009.

[17] P. Mirchandani, R. Francis, et al. *Discrete location theory*, volume 504. Wiley New York, 1990.

[18] H. Morcos, G. Atia, A. Bestavros, and A. Matta. An Information Theoretic Framework for Field Monitoring Using Autonomously Mobile Sensors. In *Proceedings of DCOSS'08: The 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems*, Santorini, Greece, June 2008.

[19] H. Morcos, A. Bestavros, and A. Matta. Amorphous Placement and Informed Diffusion for Timely Field Monitoring by Autonomous, Resource-Constrained, Mobile Sensors. In *Proceedings of SECON'08: The IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 469–477, San Francisco, CA, June 2008.

[20] H. Morcos, A. Bestavros, and I. Matta. Preferential Field Coverage Through Detour-Based Mobility Coordination. In *Proceedings of Med-Hoc-Net'10: The IFIP/IEEE Mediterranean Ad Hoc Networking Worshop*, Jun-Les-Pins, France, June 2010.

[21] T. Moscibroda and R. Wattenhofer. Facility location: distributed approximation. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 108–117. ACM, 2005.

[22] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[23] R. Pastor-Satorras and A. Vespignani. Immunization of complex networks. *Physical Reviews E*, 65, 2002.

[24] O. Rader, W. Gudat, C. Carbone, E. Vescovo, S. Blügel, R. Kläsges, W. Eberhardt, M. Wuttig, J. Redinger, and F. J. Himpsel. Electronic structure of two-dimensional magnetic alloys:c(2x2) mn on cu(100) and ni(100). *Phys. Rev. B*, 55(8):5404–5415, Feb 1997.

[25] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Eighth International Conference on Knowledge Discovery and Data Mining*, 2002.