The VLDB Journal (2019) 28:329–350
https://doi.org/10.1007/s00778-018-0531-8

REGULAR PAPER

Cascade-aware partitioning of large graph databases

Gunduz Vehbi Demirci[1] · **Hakan Ferhatosmanoglu**[2] · **Cevdet Aykanat**[1]

Received: 26 January 2018 / Revised: 23 October 2018 / Accepted: 29 November 2018 / Published online: 13 December 2018

Abstract
Graph partitioning is an essential task for scalable data management and analysis. The current partitioning methods utilize the structure of the graph, and the query log if available. Some queries performed on the database may trigger further operations. For example, the query workload of a social network application may contain re-sharing operations in the form of cascades. It is beneficial to include the potential cascades in the graph partitioning objectives. In this paper, we introduce the problem of cascade-aware graph partitioning that aims to minimize the overall cost of communication among parts/servers during cascade processes. We develop a randomized solution that estimates the underlying cascades, and use it as an input for partitioning of large-scale graphs. Experiments on 17 real social networks demonstrate the effectiveness of the proposed solution in terms of the partitioning objectives.

Keywords Graph partitioning · Propagation models · Information cascade · Social networks · Randomized algorithms · Scalability

1 Introduction

Distributed graph databases employ partitioning methods to provide data locality for queries and to keep the load balanced among servers [1–5]. Online social networks (OSNs) are common applications of graph databases where users are represented by vertices and their connections are represented by edges/hyperedges. Partitioning tools (e.g., Metis [6], Patoh [7]) and community detection algorithms (e.g., [8]) are used for assigning users to servers. The contents generated by a user are typically stored on the server that the user is assigned.

Graph partitioning methods are designed using the graph structure, and the query workload (i.e., logs of queries executed on the database), if available [9–14]. Some queries performed on the database may trigger further operations. For example, users in OSNs frequently share contents generated by others, which leads to a propagation/cascade of re-shares (cascades occur when users are influenced by others and then perform the same acts) [15–17]. The database needs to copy the re-shared contents to the servers that contain the users who will eventually need to access this content (i.e., at least a record id of the original content needs to be transferred).

Many users in a cascade process are not necessarily the neighbors of the originator. Hence, the graph structure, even with the influence probabilities, would not directly capture the underlying cascading behavior, if the link activities are considered independently. We first aim to estimate the cascade traffic on the edges. For this purpose, we present the concept of random propagation trees/forests that encodes the information of propagation traces through users. We then develop a cascade-aware partitioning that aims to optimize the load balance and reduce the amount of propagation traffic between servers. We discuss the relationship between the cascade-aware partitioning and other graph partitioning objectives.

To get insights into the cascade traffic, we analyzed a query workload from `Digg`, a news sharing-based social network [18]. The data include cascades with a depth of up to six

✉ Hakan Ferhatosmanoglu
Hakan.F@warwick.ac.uk

Gunduz Vehbi Demirci
gunduz.demirci@cs.bilkent.edu.tr

Cevdet Aykanat
aykanat@cs.bilkent.edu.tr

[1] Department of Computer Engineering, Bilkent University, Ankara, Turkey

[2] Department of Computer Science, University of Warwick, CV4 7AL Coventry, UK

links, i.e., the maximum path length from the initiator of the content to the users who eventually get the content. When we partitioned the graph by just minimizing the number of links straddling between 32 balanced partitions (using Metis [6]), the majority of the traffic remained between the servers, as opposed to staying local. This traffic goes over a relatively small fraction of the links. Only 0.01% of the links occur in 20% of the cascades, and these links carry 80% of the traffic observed in these cascades. It is important to identify the highly active edges and avoid placing them crossing the partitions.

We draw an equivalence between minimizing the expected number of cut edges in a random propagation tree/forest and minimizing communication during a random propagation process starting from any subset of users. A probability distribution is defined over the edges of a graph, which corresponds to the frequency of these edges being involved in a random propagation process. #P-Hardness of the computation of this distribution is discussed, and a sampling-based method, which enables estimation of this distribution within a desired level of accuracy and confidence interval, is proposed along with its theoretical analysis.

Experimentation has been performed both with theoretical cascade models and with real logs of user interactions. The experimental results show that the proposed solution performs significantly better than the alternatives in reducing the amount of communication between servers during a cascade process. While the propagation of content was studied in the literature from the perspective of data modeling, to the best of our knowledge, these models have not been integrated into database partitioning for efficiency and scalability.

The rest of the paper is organized as follows. Table 1 displays the notation used throughout the paper. Section 2 provides the background material and summarizes the related work. Section 3 presents a formal definition for the proposed problem. Section 4 describes the proposed solution for the problem, gives a theoretical analysis and explains how it achieves its objectives. Section 5 presents a discussion for some of the limitations and extensions of the cascade-aware graph partitioning algorithm. Section 6 presents the results of experiments on real-world datasets and demonstrates the effectiveness of the proposed solution. Section 7 concludes the paper.

2 Background

2.1 Graph partitioning

Let $G = (V, E)$ be an undirected graph such that each vertex $v_i \in V$ has weight w_i and each undirected edge $e_{ij} \in E$ connecting vertices v_i and v_j has cost c_{ij}. Generally, a K-way partition $\Pi = \{V_1, V_2 \ldots V_K\}$ of G is defined as follows:

Table 1 Notations used

Variable	Description
$\Pi = \{V_1, \ldots, V_k\}$	A K-way partition of a graph $G = (V, E)$
V_k	Part k of partition Π
$\chi(\Pi)$	Cut size under partition Π
$I_g(v)$	Random propagation tree
$\lambda_g^{\Pi}(v)$	Communication operations induced by propagation tree $I_g(v)$ under Π
$g \sim G$	Unweighted directed graph g drawn from the distribution induced by G
$\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)]$	Expected number of communication operations during a propagation process
w_{ij}	Propagation probability along edge e_{ij}
p_{ij}	Probability of edge e_{ij} being involved in a random propagation process
I	The set of random propagation trees generated by the estimation technique
$F_I(e_{ij})$	The number of trees in I that contains edge e_{ij}
N	The size of set I (i.e., $N = \lvert I \rvert$)

Each part $V_k \in \Pi$ is a non-empty subset of V, all parts are mutually exclusive (i.e., $V_k \cap V_m = \emptyset$ for $k \neq m$), and the union of all parts is V (i.e., $\bigcup_{V_k \in \Pi} V_k = V$).

Given a partition Π, weight W_k of a part V_k is defined as the sum of the weights of vertices belonging to that part (i.e., $W_k = \sum_{v_i \in V_k} w_i$). The partition Π is said to be balanced if all parts $V_k \in \Pi$ satisfy the following balancing constraint:

$$W_k \leq W_{\text{avg}}(1 + \epsilon), \quad \text{for } 1 \leq k \leq K \tag{1}$$

Here, W_{avg} is the average part weight (i.e., $W_{\text{avg}} = \sum_{v_i \in V} w_i / K$) and ϵ is the maximum imbalance ratio of a partition.

An edge is called cut if its endpoints belong to different parts and uncut otherwise. The cut and uncut edges are also referred to as external and internal edges, respectively. The cut size $\chi(\Pi)$ of a partition Π is defined as

$$\chi(\Pi) = \sum_{e_{ij} \in \mathcal{E}_{\text{cut}}^{\Pi}} c_{ij} \tag{2}$$

where $\mathcal{E}_{\text{cut}}^{\Pi}$ denotes the set of cut edges.

In the multi-constraint extension of the graph partitioning problem, each vertex v_i is associated with multiple weights w_i^c for $c = 1, \ldots, C$. For a given partition Π, W_k^c denotes the weight of part V_k on constraint c (i.e., $W_k^c = \sum_{v_i \in V_k} w_i^c$). Then, Π is said to be balanced if each part V_k satisfies $W_k^c \leq W_{\text{avg}}^c(1 + \epsilon)$, where W_{avg}^c denotes the average part weight on constraint c.

The graph partitioning problem, which is an NP-Hard problem [19], seeks to compute a partition Π^* of G that

minimizes the cut size $\chi(\cdot)$ in Eq. (2) while satisfying the balancing constraint in Eq. (1) defined on part weights.

2.2 Related work

2.2.1 Graph partitioning and replication

Graph partitioning has been studied to improve scalability and query processing performances of the distributed data management systems. It has been widely used in the context of social networks. Pujol et al. [10] propose a social network partitioning solution that reduces the number of edges crossing different parts and provides a balanced distribution of vertices. They aim to reduce the amount of communication operations between servers. It is later extended in [9] by considering replication of some users across different parts. SPAR [11] is developed as a social network partitioning and replication middleware.

Yuan et al. [13] propose a partitioning scheme to process time-dependent social network queries more efficiently. The proposed scheme considers not only the spatial network of social relations but also the time dimension in such a way that users that have communicated in a time window are tried to be grouped together. Additionally, the social graph is partitioned by considering two-hop neighborhoods of users instead of just considering directly connected users. Turk et al. [14] propose a hypergraph model built from logs of temporal user interactions. The proposed hypergraph model correctly encapsulates multi-user queries and is partitioned under load balance and replication constraints. Partitions obtained by this approach effectively reduces the number of communications operations needed during executions of multicast and gather type of queries.

Sedge [3] is a distributed graph management environment based on Pregel [20] and designed to minimize communication among servers during graph query processing. Sedge adopts a two-level partition management system: In the first level, complementary graph partitions are computed via the graph partitioning tool Metis [6]. In the second level, on-demand partitioning and replication strategies are employed. To determine cross-partition hotspots in the second level, the ratio of number of cut edges to uncut edges of each part is computed. This ratio approximates the probability of observing a cross-partition query and later is compared against the ratio of the number of cross-partition queries to internal queries in a workload. This estimation technique differs from our approach, since we estimate an edge being included in a cascade process, whereas this approach estimates the probability of observing a cross-partition query in a part and does not consider propagation processes.

Leopard is a graph partitioning and replication algorithm to manage large-scale dynamic graphs [1]. This algorithm incrementally maintains the quality of an initial partition via dynamically replicating and reassigning vertices. Nicoara et al. [21] propose Hermes, a lightweight graph repartitioner algorithm for dynamic social network graphs. In this approach, the initial partitioning is obtained via Metis and as the graph structure changes in time, an incremental algorithm is executed to maintain the quality of the partitions.

For efficient processing of distributed transactions, Curino et al. [4] propose SCHISM, which is a workload-aware graph model that makes use of past query patterns. In this model, data items are represented by vertices and if two items are accessed by the same transaction, an edge is put between the respective pair of vertices. In order to reduce the number of distributed transactions, the proposed model is split into balanced partitions using a replication strategy in such a way that the number of cut edges is minimized.

Hash-based graph partitioning and selective replication schemes are also employed for managing large-scale dynamic graphs [2]. Instead of utilizing graph partitioning techniques, a replication strategy is used to perform cross-partition graph queries locally on servers. This method makes use of past query workloads in order to decide which vertices should be replicated among servers.

2.2.2 Multi-query optimization

Le et al. [22] propose a multi-query optimization algorithm which partitions a set of graph queries into groups where queries in the same group have similar query patterns. Their partitioning algorithm is based on k-means clustering algorithm. Queries assigned to each cluster are rewritten to their cost-efficient versions. Our work diverges from this approach, since we make use of propagation traces to estimate a probability distribution over edges in a graph and partition this graph, whereas this approach clusters queries based on their similarities.

2.2.3 Influence propagation

Propagation of influence [15] is commonly modeled using a probabilistic model [23,24] learnt over user interactions [25,26]. Influence maximization problem is first studied by Domingos and Richardson [27]. Kempe et al. [28] proved that the influence maximization problem is NP-Hard under two influence propagation models such as Independent Cascade (IC) and Linear Threshold (LT) models. The Influence spread function defined in [28] has an important property called submodularity, which enables a greedy algorithm to achieve $(1 - 1/e)$ approximation guarantee for the influence maximization problem. However, computing this influence spread function is proven to be #P-Hard [17], which makes the greedy approximation algorithm proposed in [28] infeasible for larger social networks. Therefore, more efficient heuristic algorithms are targeted in the literature [17,29–32].

More recently, algorithms that run nearly in optimal linear time and provide $(1 - 1/e)$ approximation guarantee for the influence maximization problem are proposed in [33–35].

The notion of influence and its propagation processes have also been used to detect communities in social networks. Zhou et al. [36] find community structure of a social network by grouping users that have high influence-based similarity scores. Similarly, Lu et al. [37] and Ghosh et al. [38] consider finding community partition of a social network that maximizes different influence-based metrics within communities. Barbieri et al. [39] propose a network-oblivious algorithm making use of influence propagation traces available in their datasets to detect community structures.

3 Problem definition

In this section, we present the graph partitioning problem within the context of content propagation in a social network where the link structure and the propagation probability values associated with these links are given. Let an edge-weighted directed graph $G = (V, E, w)$ represent a social network where each user is represented by a vertex $v_i \in V$, each directed edge $e_{ij} \in E$ represents the direction of content propagation from user v_i to v_j and each edge e_{ij} is associated with a content propagation probability $w_{ij} \in [0, 1]$. We assume that the w_{ij} probabilities associated with the edges are known beforehand as in the case of Influence Maximization domain [28,29,34]. Methods for learning the influence/content propagation probabilities between users in a social network are previously studied in the literature [25,26]. In this setting, a partition Π of G refers to a user-to-server assignment in such a way that a vertex v_i assigned to a part $V_k \in \Pi$ denotes that the user represented by v_i is stored in the server represented by part V_k.

We adopt a widely used propagation model, the IC model, with propagation processes starting from a single user for ease of exposition. As we discuss later, this can be extended to other popular models such as the LT model and propagation processes starting from multiple users as well. Under the IC model, a content propagation process proceeds in discrete time steps as follows: Let a subset $S \subseteq V$ consist of initially active users who share a specific content for the first time in a social network (we assume $|S| = 1$ for ease of exposition). For each discrete time step t, let set S_t consists of users that are activated in time step $t \geq 0$, which indicates that $S_0 = S$ (i.e., a user becomes activated meaning that this user has just received the content). Once activated in time step t, each user $v_i \in S_t$ is given a single chance of activating each of its inactive neighbor v_j with a probability w_{ij} (i.e., user v_i activates user v_j meaning that the content propagates from v_i to v_j). If an inactive neighbor v_j is activated in time step t (i.e., v_j has received the content), then it becomes active in the next time step $t + 1$ and added to the set S_{t+1}. The same process continues until there are no new activations (i.e., until $S_t = \emptyset$).

Kempe et al. [28] define an equivalent process for the IC model by generating an unweighted directed graph g from G by independently realizing each edge $e_{ij} \in E$ with probability w_{ij}. In the realized graph g, vertices reachable by a directed path from the vertices in S can be considered as active at the end of an execution of the IC model propagation process starting with the initially active users in S. As a result of the equivalent process of the IC model, the original graph G induces a distribution over unweighted directed graphs. Therefore, we use the notation $g \sim G$ to indicate that we draw an unweighted directed graph g from the distribution induced by G by using the equivalent process of IC model. That is, we generate a directed graph g via realizing each edge $e_{ij} \in G$ with probability w_{ij}.

3.1 Propagation trees

Given a vertex v, we define the propagation tree $I_g(v)$ to denote a directed tree rooted on vertex v in graph g. The tree $I_g(v)$ corresponds to an IC model propagation process, when v is used as the initially active vertex, in such a way that each edge $e_{ij} \in I_g(v)$ encodes the information that the content propagated to v_j from v_i during this process. Here, there can be more than one possible propagation trees for v on g, since g may not be a tree itself. One of the possible trees can be computed by performing a breadth-first search (BFS) on g starting from vertex v, since IC model does not prescribe an order for activating inactive neighbors of the newly activated vertices. Note that generating a graph g and performing a BFS on a vertex v are equivalent to performing a randomized BFS algorithm starting from the vertex v. The difference between the randomized BFS algorithm and usual BFS algorithm is that each edge $e_{ij} \in E$ is searched with probability w_{ij} in the randomized case. That is, during an iteration of BFS, if a vertex v_i is extracted from the queue, each of its outgoing edge(s) e_{ij} to an unvisited vertex v_j is examined and added to the queue with a probability w_{ij}.

Here, we also define a fundamental concept called random propagation tree which is used throughout the text. A random propagation tree is a propagation tree that is generated by two levels of randomness: First, a graph g is drawn from the distribution induced by G, then a vertex $v \in V$ is chosen randomly, and its propagation tree $I_g(v)$ on g is computed. It is important to note that a random propagation tree is equivalent to an IC model propagation process starting from a randomly chosen vertex. Here, the concept of random propagation trees has similarities to reverse-reachable sets previously proposed in [33,34]. Reverse-reachable sets are built on transpose G^T of directed graph G by performing a randomized BFS starting from a vertex v and including

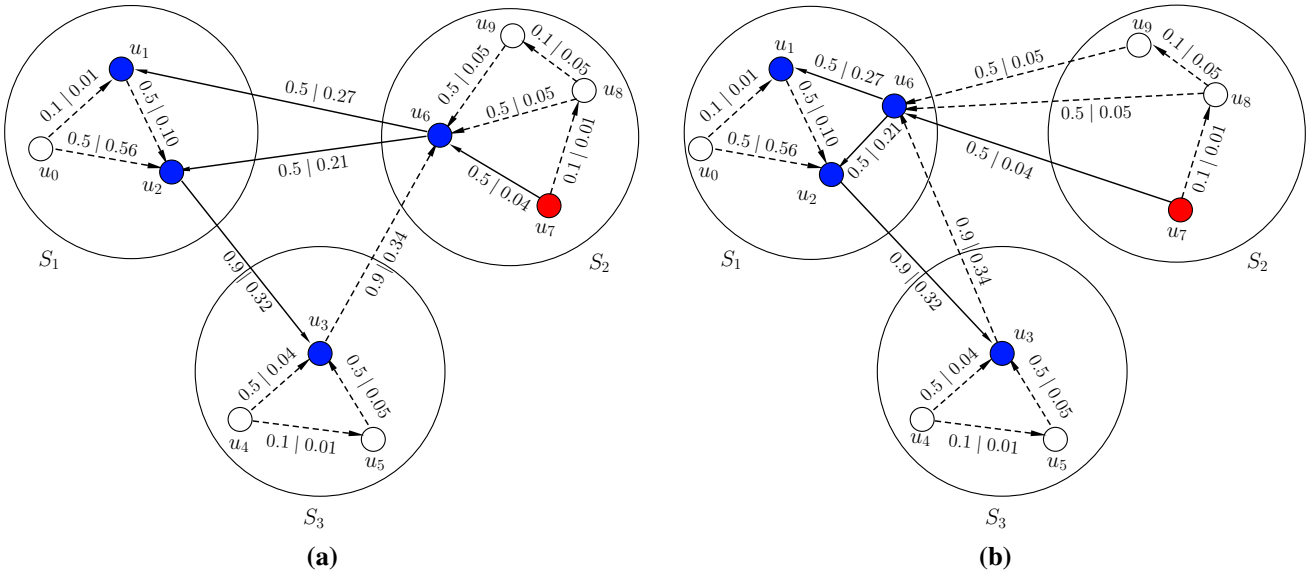

Fig. 1 An IC model propagation instance starting with initially active user u_7. Dotted lines denote edges that are not involved in the propagation process, and straight lines denote edges activated in the propagation process. **a**, **b** Display the same social network under two different partitions $\{S_1 = \{u_0, u_1, u_2\}, S_2 = \{u_6, u_7, u_8, u_9\}, S_3 = \{u_3, u_4, u_5\}\}$ and $\{S_1 = \{u_0, u_1, u_2, u_6\}, S_2 = \{u_7, u_8, u_9\}, S_3 = \{u_3, u_4, u_5\}\}$, respectively

all BFS edges. Hence, reverse-reachable sets are different from propagation trees in the ways that they do not constitute directed trees and they are built on the structure of G^T instead of G.

From a systems perspective, if a content propagation occurs between two users located on different servers, we assume this causes a communication operation. This is depicted in Fig. 1 which displays a graph with its edges denoting directions of content propagations between users. In this figure, two different partitionings of the same social network are given in Fig. 1a, b. In Fig. 1a, users are partitioned among three servers as $S_1 = \{u_0, u_1, u_2\}$, $S_2 = \{u_6, u_7, u_8, u_9\}$ and $S_3 = \{u_3, u_4, u_5\}$. In Fig. 1b, user u_6 is moved from S_2 to S_1 where S_3 remains the same. In the figure, a content shared by user u_7 propagates through four users u_6, u_1, u_2 and u_3 under the IC model. Here, the straight lines denote the edges along which propagation events occurred and these lines constitute the propagation tree formed by this propagation process (probability values associated with the edges will be discussed later in the next section). The dotted lines denote the edges that are not involved in this propagation process. Therefore, in accordance with our assumption, straight lines crossing different parts necessitate communication operations. For instance, in Fig. 1a, the propagation of the content from u_7 to u_6 does not incur any communication operation, whereas the propagation of the same content from u_6 to u_1 and u_2 incurs two communication operations. For the whole propagation process initiated by user u_7, the total number of communication operations are equal to 3 and 2 under the partitions in Fig. 1a, b , respectively.

Given a partition Π of G and a propagation tree $I_g(v)$ of vertex v on a directed graph $g \sim G$, we define the number of communication operations $\lambda_g^{\Pi}(v)$ induced by the propagation tree $I_g(v)$ under the partition Π as

$$\lambda_g^{\Pi}(v) = |\{e_{ij} \in I_g(v) \mid e_{ij} \in \mathcal{E}_{\text{cut}}^{\Pi}\}|. \tag{3}$$

That is, the number of communication operations performed is equal to the number of edges in $I_g(v)$ that are crossing different parts in Π. It can be observed that each different partition Π of G induces a different communication pattern between servers for the same propagation process.

3.2 Cascade-aware graph partitioning

In the cascade-aware graph partitioning problem, we seek to compute a partition Π^* of G that achieves the following two objectives:

(i) Under the IC model, the expected number of communication operations to be performed between servers during a propagation process starting from a randomly chosen user should be minimized.
(ii) The partition should distribute the users to servers as evenly as possible in order to ensure a balance of workload among them.

The first objective reflects the fact that many different content propagations, starting from different users or subsets of users, may simultaneously occur during any time interval in a social network and in order to minimize the total communication between servers, the expected number of communication operations in a random propagation process can be minimized. It is worth to mention that, due to the

equivalence between random propagation trees and randomized BFS algorithm, the first objective is also equivalent to minimizing the expected number of cross-partition edges traversed during a randomized BFS execution starting from a randomly chosen vertex.

To give a formal definition for the proposed problem, we redefine the first objective in terms of the equivalent process of the IC model. For a given partition Π of G, we write the expected number of communication operations to be performed during a propagation process starting from a randomly chosen user as $\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)]$. Here, subscripts v and $g \sim G$ of the expectation function denote the two levels of randomness in the process of generating a random propagation tree. As mentioned above, a random propagation tree $I_g(v)$ is equivalent to a propagation process that starts from a randomly chosen user in the network. Therefore, the expected value of $\lambda_g^{\Pi}(v)$, which denotes the expected number of cut edges included in a random propagation tree, is equivalent to the expected number of communication operations to be performed. Due to this correspondence, computing a partition Π^* that minimizes the expectation $\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)]$ achieves the first objective (i) of the proposed problem. Consequently, the proposed problem can be defined as a special type of graph partitioning in which the objective is to compute a K-way partition Π^* of G that minimizes the expectation $\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)]$ subject to the balancing constraint in Eq. (1). That is,

$$\Pi^* = \underset{\Pi}{\operatorname{argmin}}\, \mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)] \tag{4}$$

subject to $W_k \leq W_{\text{avg}}(1+\epsilon)$ for all $V_k \in \Pi$. Here, we designate weight $w_i = 1$ of each vertex $v_i \in V$ and define the weight W_k of a partition $V_k \in \Pi$ as the number of vertices assigned to that part (i.e., $W_k = |V_k|$). Therefore, this balancing constraint ensures that the objective (ii) is achieved by the partition Π^*.

4 Solution

The proposed approach is to first estimate a probability distribution for modeling the propagation and use it as an input to map the problem into a graph partitioning problem. Given an edge-weighted directed graph $G = (V, E, w)$ representing an underlying social network, the first stage of the proposed solution consists of estimating a probability distribution defined over all edges of G. For that purpose, we define a probability value p_{ij} for each edge $e_{ij} \in E$ apart from its content propagation probability w_{ij}. The value p_{ij} of an edge e_{ij} is defined to be the probability that the edge e_{ij} is involved in a propagation process that starts from a randomly selected user. Equivalently, when a random propagation tree $I_g(v)$ is generated by the process described in Sect. 3, the probability that the edge e_{ij} is included in the propagation tree $I_g(v)$ is equal to p_{ij}. It is important to note that the value w_{ij} of an edge e_{ij} corresponds to the probability that the edge e_{ij} is included in a graph $g \sim G$, whereas the value p_{ij} is defined to be the probability that e_{ij} is included in a random propagation tree $I_g(v)$ rooted on a randomly selected vertex v in graph g. For now, we delay the discussion on the computation of p_{ij} values for ease of exposition and assume that we are provided with the p_{ij} values. Later in this section, we provide an efficient method that estimates these values.

The function $\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)]$ corresponds to the expected number of cut edges in a random propagation tree $I_g(v)$ under a partition Π. In other words, if we draw a graph g from the distribution induced by G and randomly choose a vertex v and compute its propagation tree $I_g(v)$, then the expected number of cut edges included in $I_g(v)$ is equal to $\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)]$. On the other hand, the value p_{ij} of an edge e_{ij} is defined to be the probability that the edge e_{ij} is included in a random propagation tree $I_g(v)$. Therefore, given a partition Π of G, the function $\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)]$ can be written in terms of p_{ij} values of all cut edges in $\mathcal{E}_{\text{cut}}^{\Pi}$ as follows:

$$\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)] = \sum_{e_{ij}\in\mathcal{E}_{\text{cut}}^{\Pi}} p_{ij} \tag{5}$$

In Eq. (5), the expected number of cut edges in a random propagation tree is computed by summing the p_{ij} value of each edge $e_{ij} \in \mathcal{E}_{\text{cut}}^{\Pi}$, where the value p_{ij} of an edge e_{ij} is defined to be the probability that the edge e_{ij} is included in a random propagation tree. Hence, the main objective becomes to compute a partition Π^* that minimizes the total p_{ij} values of edges crossing different parts in Π^* and satisfies the balancing constraint defined over the part weights. That is,

$$\Pi^* = \underset{\Pi}{\operatorname{argmin}} \sum_{e_{ij}\in\mathcal{E}_{\text{cut}}^{\Pi}} p_{ij} \tag{6}$$

subject to the balancing constraint defined in the original problem. As mentioned earlier, each vertex v_i is associated with a weight $w_i = 1$ and part weight W_i of a part V_i is defined to be the number of vertices assigned to V_i (i.e., $W_i = |V_i|$).

As a result of Eq. (6), the problem can be formulated as a graph partitioning problem for which successful tools exist [6,40]. However, the graph partitioning problem is usually defined for undirected graphs, whereas G is a directed graph and p_{ij} values are associated with the directed edges of G. To that end, we build an undirected graph $G' = (V, E')$ by symmetrizing directed graph G through computing the cost of each edge $e'_{ij} \in E'$ as $c_{ij} = p_{ij} + p_{ji}$.

Let Π be a partition of G'. Since G' and G consist of the same vertex set V, Π induces a set $\mathcal{E}_{\text{cut}}^{\Pi}$ of cut edges for the original graph G. Due to the cost definitions of edges in E', the cut size $\chi(\Pi)$ of G' under partition Π is equal to the sum of the p_{ij} values of cut edges in $\mathcal{E}_{\text{cut}}^{\Pi}$ which is shown to be equal to the value of the main objective function in Eq. (4). That is,

$$\chi(\Pi) = \sum_{e_{ij} \in \mathcal{E}_{\text{cut}}^{\Pi}} p_{ij} = \mathbb{E}_{v,g \sim G}[\lambda_g^{\Pi}(v)] \tag{7}$$

Hence, a partition Π^* that minimizes the cut size $\chi(\cdot)$ of G' also minimizes the expectation $E_{v,g \sim G}[\lambda_g^{\Pi}(v)]$ in the original social network partitioning problem. In other words, if the partition Π^* for G' is an optimal solution for the partitioning of G', it is also an optimal solution for Eq. (4) in the original problem. Additionally, the equivalence drawn between the graph partitioning problem and the cascade-aware graph partitioning problem also proves that the proposed problem is NP-Hard even the p_{ij} values were given beforehand.

In Fig. 1, the main objective of cascade-aware graph partitioning is depicted as follows: Each edge in the figure is associated with a content propagation probability along with its computed p_{ij} value (i.e., each edge e_{ij} is associated with "$w_{ij} \mid p_{ij}$"). The partitioning in Fig. 1a provides a better cut size in terms of both number of cut edges and the total propagation probabilities of edges crossing different parts. However, we assert that the partitioning in Fig. 1b provides a better partition for objective function 4, at the expense of providing worse cut size in terms of other cut size metrics (i.e., the sum of p_{ij} values of cut edges is less in the second partition).

4.1 Computation of the p_{ij} values

We now return to the discussion on the computation of the p_{ij} values defined over all edges of G and start with the following theorem indicating the hardness of this computation:

Theorem 1 *Computation of the p_{ij} value for an edge e_{ij} of G is a #P-Hard problem.*

Proof Let function $\sigma(v_k, v_i)$ denote the probability of there being a directed path from vertex v_k to vertex v_i on a directed graph g drawn from the distribution induced by G. Assume that the only path goes from v_k to v_j is through v_i on each possible g. That is v_j is only connected to v_i in G. (This simplifying assumption does not affect the conclusion we draw for the theorem.) Hence, the probability of v_i included in a propagation tree $I_g(v_k)$ is $\sigma(v_k, v_i)$. Let p_{ij}^k denote the probability of e_{ij} is included in $I_g(v_k)$. We can compute p_{ij}^k as

$$p_{ij}^k = w_{ij} \cdot \sigma(v_k, v_i) \tag{8}$$

since inclusion of e_{ij} in g and formation of a directed path from v_k to v_i on g are two independent events; therefore, their respective probability values w_{ij} and $\sigma(v_k, v_i)$ can be multiplied. As mentioned earlier, the value p_{ij} of an edge e_{ij} is defined to be the probability of edge e_{ij} included in a random propagation tree. Therefore, we can compute the value p_{ij} of an edge e_{ij} as follows:

$$p_{ij} = \frac{1}{|V|} \sum_{v_k \in V} p_{ij}^k \tag{9}$$

Here, to compute the p_{ij} value of edge e_{ij}, we sum the conditional probability $\frac{1}{|V|} \cdot p_{ij}^k$ for all $v_k \in V$. Due to the definition of random propagation trees, selections of v_k in a graph $g \sim G$ are all mutually exclusive events with equal probability $\frac{1}{|V|}$. Therefore, we can sum the terms $\frac{1}{|V|} \cdot p_{ij}^k$ for each $v_k \in V$ to compute the total probability p_{ij}.

In order to prove the theorem, we present an equivalence between the computation of function $\sigma(\cdot, \cdot)$ and the s,t-connectedness problem [41], since the p_{ij} value of an edge e_{ij} depends on the computation of $\sigma(v_k, v_i)$ for all $v_k \in V$. The input to the s,t-connectedness problem is a directed graph $G = (V, E)$, where each edge $e_{ij} \in E$ may fail randomly and independently from each other. The problem asks to compute the total probability of there being an operational path from a specified source vertex s to a target vertex t on the input graph. However, computing this probability value is proven to be a #P-Hard problem [41]. On the other hand, the function $\sigma(v_k, v_i)$ denotes the probability of there being a directed path from v_k to v_i in a $g \sim G$, where each edge in g is realized with probability w_{ij} randomly and independently from other edges. It is obvious to see that the computation of function $\sigma(v_k, v_i)$ is equivalent to the computation of the s,t-connectedness probability. (We refer the reader to [17] for a more formal description for the reduction of $\sigma(v_k, v_i)$ to s,t-connectedness problem). This equivalence implies that the computation of function $\sigma(v_k, v_i)$ is #P-Hard even for a single vertex v_k and therefore implies that the computation of p_{ij} value for any edge e_{ij} is also #P-Hard. □

Theorem 1 states that it is unlikely to devise a polynomial time algorithm which exactly computes p_{ij} values for all edges in G. Therefore, we employ an efficient method that can estimate these p_{ij} values for all edges in G at once. These estimations can be made within a desired level of accuracy and confidence interval, but there is a trade-off between the runtime and the estimation accuracy of the proposed approach. On the other hand, the quality of the results produced by the overall solution is expected to increase with increasing accuracy of the p_{ij} values.

The proposed estimation technique employs a sampling approach that starts with generating a certain number of random propagation trees. Recall that a random propagation tree

is generated by first drawing a directed graph $g \sim G$ and then computing a propagation tree $I_g(v)$ on g for a randomly selected vertex $v \in V$. Let I be the set of all random propagation trees generated for estimation and let N be the size of this set (i.e., $N = |I|$). After forming the set I, the value p_{ij} of an edge e_{ij} can be estimated by the frequency of that edge's appearance in random propagation trees in I as follows: Let function $F_I(e_{ij})$ denote the number of random propagation trees in I that contains edge e_{ij}. That is,

$$F_I(e_{ij}) = |\{I_g(v) \in I \mid e_{ij} \in I_g(v)\}| \tag{10}$$

Due to the definition of p_{ij}, the appearance of edge e_{ij} in a random propagation tree $I_g(v) \in I$ can be considered as a Bernoulli trial with success probability p_{ij}. Hence, the function $F_I(e_{ij})$ can be considered as the number of total successes in N Bernoulli trials with success probability p_{ij}, which implies that $F_I(e_{ij})$ is Binomially distributed with parameters N and p_{ij} (i.e., $F_I(e_{ij}) \sim \text{Binomial}(p_{ij}, N)$). Therefore, the expected value of the function $F_I(e_{ij})$ is equal to $p_{ij}N$, which also implies that

$$p_{ij} = \mathbb{E}[F_I(e_{ij})/N] \tag{11}$$

As a result of Eq. (11), if an adequate number of random propagation trees are generated to form the set I, the value $F_I(e_{ij})/N$ can be an estimation for the value of p_{ij}. Therefore, the estimation method consists of generating N random propagation trees that together form the set I, and computing the function $F_I(e_{ij})$ according to Eq. (10) for each edge $e_{ij} \in E$. After computing the function $F_I(e_{ij})$ for each edge e_{ij}, we use $F_I(e_{ij})/N$ as an estimation for the p_{ij} value.

4.2 Implementation of the estimation method

We seek an efficient implementation for the proposed estimation method. The main computation of the method consists of generating N random propagation trees. A random propagation tree can be efficiently generated by performing a randomized BFS, starting from a randomly chosen vertex, in G. It is important to note that the randomized BFS algorithm starting from a vertex v is equivalent to drawing a graph $g \sim G$ and performing a BFS starting from the vertex v on g. That is, the randomized BFS algorithm is equivalent to the method introduced in Sect. 3 to generate a propagation tree $I_g(v)$ rooted on v. Therefore, forming the set I can be accomplished by performing N randomized BFS algorithms on G starting from randomly chosen vertices. Moreover, the computation of the function $F_I(\cdot)$ for all edges in E can be performed while forming the set I with a slight modification to the randomized BFS algorithm. For this purpose, a counter for each $e_{ij} \in E$ can be kept in such a way that its value is incremented each time the corresponding edge is traversed during a randomized BFS execution. This counter denotes the number of times an edge is traversed during the performance of all randomized BFS algorithms. Therefore, after N randomized BFS executions, the function $F_I(e_{ij})$ for an edge e_{ij} is equal to the value of the counter maintained for that edge.

4.3 Algorithm

The overall cascade-aware graph partitioning algorithm is described in Algorithm 1. In the first line, the set I is formed by performing N randomized BFS algorithms, where the function $F_I(e_{ij})$ is computed for each edge $e_{ij} \in E$ during these randomized BFS executions. In lines 2 and 3, an undirected graph $G' = (V, E')$ is built via composing a new set E' of undirected edges, where each undirected edge $e'_{ij} \in E'$ is associated with a cost of c_{ij} using the estimations computed in the first step. In line 4, each vertex $v_i \in V$ is associated with a weight $w_i = 1$ in order to ensure that the weight of a part is equal to the number of vertices assigned to that part. Lastly, a K-way partition Π of the undirected graph G' is obtained using an existing graph partitioning algorithm and returned as a solution for the original problem. Here, the graph partitioning algorithm is executed with the same imbalance ratio as with the original problem.

4.4 Determining the size of set *I*

As mentioned earlier, the accurate estimation of the p_{ij} values is a crucial step to compute "good" solutions for the proposed problem, since the graph partitioning algorithm used in the second step makes use of these p_{ij} values to compute the costs of edges in G'. The total cost of cut edges

Algorithm 1 Cascade-Aware Graph Partitioning

Input: $G = (V, E, w)$, K, ϵ
Output: Π

1: Form a set I of N random propagation trees by performing randomized BFS algorithms on G and compute $F_I(e_{ij})$ for each $e_{ij} \in E$ according to Eq. (10)
2: Build an undirected graph $G' = (V, E')$ where edge set E' is composed as follows:

$$E' = \{e'_{ij} \mid e_{ij} \in E \vee e_{ji} \in E\} \tag{12}$$

3: Associate a cost c_{ij} with each $e'_{ij} \in E'$ as follows:

$$c_{ij} = \begin{cases} F_I(e_{ij})/N + F_I(e_{ji})/N, & \text{if} \quad e_{ij} \in E \wedge e_{ji} \in E \\ F_I(e_{ij})/N, & \text{if} \quad e_{ij} \in E \wedge e_{ji} \notin E \\ F_I(e_{ji})/N, & \text{if} \quad e_{ij} \notin E \wedge e_{ji} \in E \end{cases} \tag{13}$$

4: Associate each vertex $v_i \in V$ with weight $w_i = 1$.
5: Compute a K-Way partition Π of G' using an existing graph partitioning algorithm (using the same imbalance ration ϵ).
6: **return** Π

in G' represents the value of the objective function in Eq. (4). Therefore, the p_{ij} values need to be accurately estimated so that the graph partitioning algorithm correctly optimizes the objective function.

Estimation accuracies of the p_{ij} values depend on the number of random propagation trees forming the set I. As the size of the set I increases, more accurate estimations can be obtained. However, we want to compute the minimum value of N to get a specific accuracy within a specific confidence interval. More formally, let $\hat{p}_{ij}$ be the estimation computed for the p_{ij} value of an edge $e_{ij} \in E$ (i.e., $\hat{p}_{ij} = F_I(e_{ij})/N$), and we want to compute the minimum value of N to achieve the following inequality:

$$Pr[|\hat{p}_{ij} - p_{ij}| \leq \theta \,, \forall e_{ij} \in E] \geq 1 - \delta. \tag{14}$$

That is, with a probability of at least $1 - \delta$, we want the estimation $\hat{p}_{ij}$ to be within θ of p_{ij} for each edge $e_{ij} \in E$. For that purpose, we make use of well-known Chernoff [42] and Union bounds from probability theory. Chernoff bound can be used to find an upper bound for the probability that a sum of many independent random variables deviates a certain amount from their expected mean. In this regard, due to the function $F_I(\cdot)$ being Binomial, Chernoff bound can guarantee the following inequality:

$$Pr\left[|F_I(e_{ij}) - p_{ij}N| \geq \xi \; p_{ij}N\right] \leq 2\exp\left(-\frac{\xi^2}{2+\xi} \cdot p_{ij}N\right) \tag{15}$$

for each edge $e_{ij} \in E$. Here, ξ denotes the distance from the expected mean in the context of Chernoff bound.

In Eq. (15), dividing both sides of the inequality $|F_I(e_{ij}) - p_{ij}N| \geq \xi \; p_{ij}N$ in the function $Pr[\cdot]$ by N and taking $\xi = \theta/p_{ij}$ yields

$$\begin{aligned} Pr[|\hat{p}_{ij} - p_{ij}| \geq \theta] &\leq 2\exp\left(-\frac{\theta^2}{2p_{ij}+\theta} \cdot N\right) \\ &\leq 2\exp\left(-\frac{\theta^2}{2+\theta} \cdot N\right) \end{aligned} \tag{16}$$

which denotes the upper bound for the probability that the accuracy θ is not achieved for a single edge e_{ij} (the last inequality in Eq. (16) follows, since $p_{ij} \leq 1$). Moreover, RHS of Eq. (16) is independent from the value of p_{ij} and its value is the same for all edges in E, which enables us to apply the same bound for all of them. However, our objective is to find the minimum value of N to achieve accuracy θ for all edges simultaneously with a probability at least $1 - \delta$. For that purpose, we need to find an upper bound for the probability that there exists at least one edge in E for which the accuracy θ is not achieved. We can compute this upper bound using Union bound as follows:

$$Pr[|\hat{p}_{ij} - p_{ij}| \geq \theta \,, \exists e_{ij} \in E] \leq 2|E|\exp\left(-\frac{\theta^2}{2+\theta} \cdot N\right) \tag{17}$$

Here, we simply multiply RHS of Eq. (16) by $|E|$, since for each edge in E, the accuracy θ is not achieved with a probability at most $2\exp(-\frac{\theta^2}{2+\theta} \cdot N)$. In order to achieve Eq. (14), RHS of Eq. (17) needs to be at most δ. That is,

$$2|E|\exp\left(-\frac{\theta^2}{2+\theta} \cdot N\right) \leq \delta \tag{18}$$

Solving this equation for N yields

$$N \geq \frac{2+\theta}{\theta^2} \cdot \ln\frac{2|E|}{\delta} \tag{19}$$

which indicates the minimum value of N to achieve θ accuracy for all edges in E with a probability at least $1 - \delta$.

The accuracy θ determines how much error is made by the graph partitioning algorithm while it performs the optimization. As shown in Eq. (7), for a partition Π of G' obtained by the graph partitioning algorithm, the cut size $\chi(\Pi)$ is equal to the value of main objective function (4). However, the cost values associated with the edges of G' are estimations of their exact values, and therefore, the partition cost $\chi(\Pi)$ might be different from the exact value of the objective function. In this regard, the difference between the objective function and the partition cost can be expressed as follows:

$$\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(v)] - \chi(\Pi) \leq \theta \cdot |\mathcal{E}_{\text{cut}}^{\Pi}| \tag{20}$$

Here, the error is computed by multiplying the accuracy θ by the number of cut edges of G under the partition Π, since for each edge in $\mathcal{E}_{\text{cut}}^{\Pi}$, at most θ error can be made with a probability at least $1-\delta$. Therefore, even if it were possible to solve the graph partitioning problem optimally, the solution returned by Algorithm 1 would be within $\theta \cdot |\mathcal{E}_{\text{cut}}^{\Pi}|$ of the optimal solution for the original problem with a probability at least $1 - \delta$. Consequently, as the value of θ decreases, the partition obtained by Algorithm 1 will incur less error for the main objective function, which will enable the graph partitioning algorithm to perform a better optimization for the original problem.

4.5 Complexity analysis

The proposed algorithm consists of two main computational phases. In the first phase, for an accuracy θ with confidence δ, the set I is generated via performing at least $N = \frac{2+\theta}{\theta^2} \cdot \ln\frac{2|E|}{\delta}$ randomized BFS algorithms and each of

these BFS executions takes $\Theta(V + E)$ time. The second phase of the algorithm performs the partitioning of the undirected graph G' which is constructed from the directed graph G by using $F_I(e_{ij})$ values computed in the first phase. The construction of the graph G' can be performed in $\Theta(V + E)$ time. The partitioning complexity of the graph G', however, depends on the partitioning tool used. In our implementation, we preferred Metis which has a complexity of $\Theta(V+E+K \log K)$, where K is the number of partitions. Therefore, if θ and δ are assumed to be constants, the overall complexity of Algorithm 1 to obtain a K-way partition can be formulated as follows:

$$\Theta\left(\frac{2+\theta}{\theta^2}\ln\frac{2|E|}{\delta}(V+E)\right)+\Theta(V+E+K\log K)$$
$$=\Theta((V+E)\log E+K\log K). \quad (21)$$

Equation (21) denotes serial execution complexity of Algorithm 1. The proposed algorithm's scalability can be improved even further via parallel processing, since the estimation technique is embarrassingly parallel. Given P parallel processors, N propagation trees in I can be computed without necessitating any communication or synchronization (i.e., each processor can generate N/P trees by separate BFS executions). The only synchronization point is needed in the reduction of $F_I(e_{ij})$ values computed by these processors. This reduction operation, however, can be efficiently performed in $\log P$ synchronization phases. Additionally, there exist parallel graph partitioning tools (e.g., ParMetis [43]) which can improve the scalability of the graph partitioning phase.

4.6 Extension to the LT model

Even though we have illustrated the problem and solution for the IC model, both our problem definition and proposed solution can be easily extended to other models such as the LT (linear threshold) model. It is worth to mention that the proposed solution does not depend on the IC model or the probability distribution defined over edges (i.e., w_{ij} probabilities). As long as the random propagation trees can be generated, the proposed solution does not require any modification for the use of any different cascade model or the probability distribution defined over edges.

We skip the description for the LT model and just provide the equivalent process of LT model proposed in [28]. In the equivalent process of the LT model, an unweighted directed graph g is generated from G by realizing only one incoming edge of each vertex in V. That is, for each vertex $v_i \in V$, each incoming edge e_{ji} of vertex v_i has probability w_{ji} of being selected and only the selected edge is realized in g. Given a directed graph g generated by this equivalent process, a propagation tree $I_g(v)$ rooted on vertex v again can be computed by performing a BFS starting from v on g. Different from the equivalent process of IC model, there can be only one propagation tree for each vertex, since all vertices have only one incoming edge to these vertices. However, a propagation tree $I_g(v)$ under LT model still encodes the same information as in IC model; that is, each edge $e_{ij} \in I_g(v)$ encodes the information that a content propagates from v_i to v_j.

In the problem definition part, we make use of the notion of propagation trees in such a way that edges in a propagation tree that are crossing different parts are assumed to necessitate communication operations between servers. This assumption also holds for the LT model, since propagation trees generated by the equivalent processes of IC and LT models encode the same information. Therefore, minimizing the expected number of communication operations during an LT propagation process starting from a randomly chosen user still corresponds to minimizing the expected number of cut edges in a random propagation tree. In this regard, we do not need any modification for the objective function (4) and we still want to compute a partition Π^* that minimizes the expected number of cut edges in a random propagation tree. (The only difference is in the process of computing a random propagation tree under LT model.)

In the solution part, we generate a certain number of random propagation trees in order to estimate a probability distribution defined over all edges in E. The estimated probability distribution associates each edge with a probability value denoting how likely an edge is included in a random propagation tree under the IC model. The associated probability values are also later used as costs in the graph partitioning phase. However, both the estimation method and the overall solution do not depend on anything specific to the IC model and only require a method for generating random propagation trees which is mentioned above. Moreover, concentration bounds attained for the estimation of the probability distribution still holds under the LT model and the number of random propagation trees forming the set I in Algorithm 1 should satisfy Eq. (19).

4.7 Processes starting from multiple users

The method proposed for the propagation processes starting from a single user can be generalized for propagation processes that start from multiple users as follows: Instead of the definition of random propagation trees, we define random propagation forest $I_g(S)$ for a randomly selected subset of users $S \subseteq V$. The only difference between the two definitions is that a random propagation forest consists of multiple propagation trees that are rooted on the vertices in S. However, these propagation trees must be edge-disjoint and if a vertex is reachable from two different vertices in S, this

vertex can be arbitrarily included in one of the propagation trees rooted on these vertices. As noted earlier, the IC model does not prescribe an order for activating inactive neighbors; therefore, a random propagation forest over the set S can be computed by first drawing a graph $g \sim G$ and then performing a multi-source BFS on g starting from the vertices in S. The order of execution of multi-source BFS determines the form of propagation trees in the propagation forest $I_g(S)$.

In a partition Π of propagation forest $I_g(S)$, each cut edge incurs one communication operation. So, the total number of communication operations induced by Π is defined to be the number of cut edges which we denote as $\lambda_g^{\Pi}(S)$. These new definitions do not require any major modification for the optimization problem introduced in Eq. (4), and we just replace the expectation function with $\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(S)]$. That is, our objective becomes computing a partition that minimizes the expected number of cut edges in a random propagation forest.

To generalize the proposed solution, we redefine p_{ij} value of an edge e_{ij} as the probability of edge e_{ij} included in a random propagation forest instead of a random propagation tree. With this new definition of p_{ij} values, Eqs. (5) and (6) can still be satisfied; hence, a partition Π^* that minimizes the sum of p_{ij} values of edges crossing different parts also minimizes the expectation $\mathbb{E}_{v,g\sim G}[\lambda_g^{\Pi}(S)]$.

The new definition of p_{ij} values necessitates some modifications to the estimation method proposed earlier. Recall that, for the previous definition of p_{ij} values, we generate a set I of random propagation trees and compute the function $F_I(\cdot)$ for each edge e_{ij}. For the new definition of p_{ij} values, the estimations can be obtained with a similar approach; however, the set I must now consist of random propagation forests and $F_I(\cdot)$ must denote frequencies of edges to appear in these random propagation forests. Therefore, the only modification required for Algorithm 1 is to replace the step that the set I is generated by performing N randomized BFS algorithms. The new set I of random propagation forests can be obtained with a similar approach such that instead of performing randomized single-source BFS algorithms, we perform randomized multi-source BFS algorithms. These two BFS algorithms are essentially the same except that multi-source BFS starts execution with its queue containing a randomly selected subset of vertices instead of a single vertex. The new definitions together with the modifications performed on the overall solution do not affect the concentration bounds obtained in Eq. (19).

5 Extensions and limitations

Here, we show how the proposed cascade-aware graph partitioning algorithm (CAP) can be incorporated into other graph partitioning objectives.

5.1 Non-cascading queries

Queries such as "reading-friend's-posts" and "read-all-posts-from-friends" can be observed more frequently than cascading (i.e., re-share) operations in a typical OSN application. The number of communication operations for such non-cascading queries may require minimizing the number of cut edges if query workload is highly changing or not available, or minimizing the total traffic crossing different parts if it can be estimated. The cascade-aware graph partitioning aims at reducing the cut edges that have high probability of being involved in a random propagation process under a specific cascade model. Assigning unit weights to all edges (i.e., $c_{ij} = 1$ for each edge e_{ij}) makes the objective same as minimizing the number of cut edges. A combination of objectives can be achieved by assigning each edge cost $c_{ij} = 1 + \alpha(p_{ij} + p_{ji})$, where α determines the relative weight of traffic/cascade-awareness.

5.2 Intra-propagation balancing among servers

This paper considers the number of nodes/users as the only balancing criteria for the proposed cascade-aware partitioning. On the other hand, the proposed formulation can be enhanced to handle balance on multiple workload metrics via a multi-constraint graph partitioning. For example, a balanced distribution of the number of content propagation operations within servers can be attained via the following two-constraint formulation. We assign the following two weights to each vertex v_i:

$$w_i^1 = 1 \text{ and } w_i^2 = \sum_{e_{ki} \in E} p_{ki}. \tag{22}$$

Here, the summation in the second weight represents the sum of p probabilities of the incoming edges of vertex v_i. Under this vertex weight definition, the two-constraint partitioning maintains balance on both the number of users assigned to servers and the number of intra-propagation operations to be performed within servers. The latter balancing holds because of the fact that the expected number of propagations within a part V_k is

$$\sum_{e_{ij} \in \mathcal{E}(V_k)} p_{ij} \tag{23}$$

where $\mathcal{E}(V_k)$ denotes the set of edges pointing toward the vertices in V_k.

5.3 Repartitioning

As graph databases are usually dynamic, i.e., new vertices and edges are added or removed, etc., repartitioning is nec-

essary [1–3,21]. Repartitioning methods aims to maintain the quality of an initial partition via reassigning vertices to parts as the graph structure changes. However, the costs of new edges should be computed for repartitioning. That is, if a new direct edge is established in G, then its p value needs to be computed before repartitioning. The p_{ij} value of a new edge e_{ij} can be computed using p_{ki} value of each incoming edge e_{ki} of vertex v_i as follows:

$$p_{ij} = w_{ij} \times \left[1 - \prod_{e_{ki} \in E} (1 - p_{ki}) \right] \tag{24}$$

That is, the content propagation probability w_{ij} is multiplied by the probability of there being at least one edge e_{ki} incoming to vertex v_i is activated during a random propagation process. It is important to note that establishing the new edge e_{ij} also affects p_{jk} value of each outgoing edge e_{jk} of vertex v_j. If these values also need to be updated during repartitioning, Eq.(24) can be applied for each edge e_{jk}, in succession for updating the value of p_{ij}. In short, while moving vertices between parts during repartitioning, the p_{ij} value of any edge e_{ij} can be updated via applying Eq. (24) in a correct order. By updating p_{ij} values on demand, the existing repartitioning approaches can be adapted for the cascade-aware graph partitioning problem.

5.4 Replication

Replication strategies need some modifications in order to be used for the cascade-aware graph partitioning. It should be noted that, even though the cut size of graph G' can be reduced via replication of some vertices among multiple parts, this approach also incurs additional communication operations. This is because, when a replicated vertex becomes active during a content propagation process, the content needs to be transferred to each server that the vertex is replicated.

6 Experimental evaluation

In this section, we experimentally evaluate the performance of the proposed solution on social network datasets. We develop an alternative solution, which produces competitive results, as a baseline algorithm in our experiments. The baseline algorithm directly makes use of propagation probabilities between users in the partitioning phase (i.e., w_{ij} values). Additionally, we also test various algorithms previously studied in the literature [10,13] and compared them with the proposed solution.

6.1 Experimental setup

6.1.1 Datasets

Table 2 displays the properties of the real-world social networks used in our experiments. Many of these datasets are used in the context of influence maximization research [34]. The first 13 datasets (`Facebook–LiveJournal`) are collected from Stanford Large Network Dataset Collection[1] [45], and they contain friendship, communication or citation relationships between users in various real-world social network applications. `Twitter (large)` is collected from [46], `uk-2002` and `webbase-2001` are collected from Laboratory for Web Algorithmics[2] [47], and `sinaweibo` is collected from Network Repository[3] [48]. Additionally, we also make use of a synthetic graph, named as `random-social-network`, which we generate by using graph500 [49] power law random graph generator. The graph500 tool is initialized with two parameters, namely as *edge-factor* and *scale*, in order to produce graphs with 2^{scale} vertices and *edge-factor*$\times 2^{\text{scale}}$ directed edges. We set both *scale* and *edge-factor* to 16 to produce `random-social-network` dataset.

All datasets are provided in the form of a graph, where users are represented by vertices and relationships by directed or undirected edges. To infer the direction of content propagation between users, we interpret these social networks as follows: For directed graphs, we assume that a propagation may occur only in the direction of a directed edge, whereas for undirected graphs, we assume that a propagation may occur in both directions along an undirected edge. Therefore, we did not apply any modifications to the directed graphs, whereas we modified the undirected graphs by replacing each undirected edge with two opposite directed edges.

In the datasets in Table 2, the information about the content propagation probabilities between users is not available. Therefore, for each dataset, we draw values uniformly at random from the interval [0, 1] and associate these values with the edges connecting its pairs of users as the propagation probabilities. We repeat this process five times for each dataset and obtain five different versions of the same social network having different propagation probabilities associated with its edges. Using these versions of the social network, we performed the same set of experiments on each different version and reported the averages of the results obtained for that specific dataset.

Given an underlying social network with its associated propagation probabilities, our aim is to find a user partition that minimizes the expected number of communication oper-

[1] https://snap.stanford.edu/data.

[2] http://law.di.unimi.it.

[3] http://networkrepository.com.

Table 2 Dataset properties

	Number of		In degree		Out degree		Descripton
	Vertices	Edges	max	avg	max	avg	
Facebook	4039	176, 468	1045	44	1045	44	Social network from Facebook
wiki-Vote	7115	103, 689	893	15	457	15	Wikipedia who-votes-on-whom network
HepPh	34, 546	421, 534	411	12	846	12	Arxiv High Energy Physics paper citation network
email-Enron	36, 692	367, 662	1383	10	1383	10	Email communication network
Epinions	75, 879	508, 837	1801	7	3035	7	Who-trusts-whom network of Epinions.com
Twitter (small)	81, 306	1, 768, 135	1205	22	3383	22	Social network from Twitter
Slashdot	82, 168	870, 161	2510	11	2552	11	Slashdot social network from February 2009
email-EuAll	265, 214	418, 956	929	2	7631	2	Email communication network
dblp	317, 080	2, 099, 732	343	7	343	7	DBLP collaboration network
youtube	1, 134, 890	5, 975, 248	28, 754	5	28, 754	5	Youtube online social network
Pokec	1, 632, 803	30, 622, 564	8763	19	13, 733	19	Pokec online social network
wiki-Talk	2, 394, 385	5, 021, 410	100, 022	2	3311	2	Wikipedia talk (communication) network
LiveJournal	4, 847, 571	68, 475, 391	20, 292	14	13, 905	14	LiveJournal online social network
Twitter (large)	11, 316, 811	85, 331, 843	564, 512	7	214, 381	7	Social network from Twitter
uk-2002	18, 484, 123	292, 243, 668	194, 942	16	2449	16	Web graph crawled (2002) under `.uk` domain
sinaweibo	58, 655, 849	522, 642, 104	278, 490	9	278, 490	9	Sina Weibo online social network
webbase-2001	118, 142, 155	1, 019, 903, 190	816, 127	8	3841	8	Web graph crawled (2001) by WebBase [44]
random-social-network	65, 536	910, 479	9613	19	3233	19	Generated by graph500 power law graph generator

ations during a random propagation process under a specific cascade model. There have been effective approaches in the literature to learn the propagation probabilities between users in a social network [25,26]. Inferring these probability values using logs of user interactions is out of the scope of this paper. However, we also work on a real-world dataset, from which real propagation traces can be deduced, to test the proposed solution.

6.1.2 Baseline partitioning (BLP) algorithm

One can partition the input graph in such a way that the edges with high propagation probabilities are removed from the cut as much as possible. To achieve this, the sum of propagation probabilities of the cut edges can be considered as an objective function to be minimized in the graph partitioning problem. The baseline algorithm also builds an undirected graph from a given social network and makes use of a graph partitioning tool. Instead of computing a new probability distribution over all edges (i.e., the p_{ij} values), the baseline algorithm directly makes use of propagation probabilities associated with edges (i.e., the w_{ij} values). That is, the cost c_{ij} of an undirected edge e'_{ij} of G' is determined using w_{ij} and w_{ji} values instead of p_{ij} and p_{ji} values of edges e_{ij} and e_{ji}, respectively. By this way, the graph partitioner minimizes the sum of propagation probabilities associated with the edges crossing different parts. The difference between the baseline algorithm and the proposed solution is the cost values associated with the edges of the undirected graph provided to the graph partitioner.

6.1.3 Other tested algorithms

In our experiments, we also test three previously studied social network partitioning algorithms for comparison purposes. The first of these algorithms (CUT) is given in [10] and aims to minimize the number of links crossing different parts (i.e., basically minimizes the number of cut edges). The second algorithm (MO+) [10] makes use of a community detection algorithm and performs partitioning based on the community structures inherent to social networks.

As the third algorithm, we consider the social network partitioning algorithm provided in [13]. The social graph is partitioned in such a way that two-hop neighborhood of a user is kept in one partition, instead of the one-hop network. For that purpose, an activity prediction graph (APG) is built and its edges are associated with weights that are computed using the number of messages exchanged between users in a time period. Since the w_{ij} values can not be directly considered as the number of messages exchanged between users, we make use of $F_I(e_{ij})$ values computed by CAP algorithm. That is, we designate the number of messages exchanged in a time period between users as $F_I(e_{ij})$. Additionally, to compute edge weights, the algorithm uses two parameters which are the total number of past periods considered and a scaling constant (these parameters are referred to as K and C in [13]). We set these parameters to *one*, since we can not partition $F_I(e_{ij})$ values into time periods. Using these values, we construct the same APG graph and partition this graph. We refer to this algorithm as 2Hop in our experiments.

6.1.4 Content propagations

To evaluate the qualities of the partitions obtained by the tested algorithms, we performed a large number of experiments based on both real and IC-based traces of propagation on real-world social networks. We generated the IC-based propagation data as follows: First, we generate a randomly selected subset of users and then execute an IC model propagation process starting from the users in this set. The size of the set is randomly determined and chosen uniformly at random from the interval $[1, 50]$. During this propagation process, we count the total number of propagation events that occurred between the users located on different parts. As mentioned earlier, such propagation events cause communication operations between servers according to our problem definition. For each of the datasets, we perform 10^5 such experiments and compute the average of the total number of communication operations performed under a given partition. This average corresponds to an estimation for the expected number of communication operations during a random propagation process.

6.1.5 Partitioning framework

The graphs generated by algorithms, except MO+, are partitioned using state-of-the-art multi-level graph partitioning tool Metis [6] using the following set of parameters: We specify partitioning method as multi-level k-way partitioning, type of objective as edge-cut minimization and the maximum allowed imbalance ratio as 0.10. All the other parameters are set to their default values. We implemented MO+ algorithm by using a community detection algorithm[4] provided in [50] with its default parameters.

In order to observe the variation of the relative performance of the algorithms, each graph instance is partitioned K-way for $K = 32, 64, 128, 256, 512$ and 1024, respectively. In order to observe the performance gain achieved by intelligent partitioning algorithms, all graph instances are also partitioned random-wise, which is referred to as random partitioning (RP) algorithm.

[4] http://www.mapequation.org/code.html.

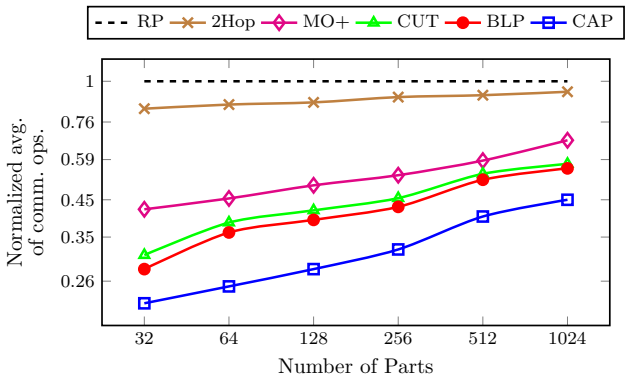

Fig. 2 The geometric means of the communication operation counts incurred by the partitions obtained by BLP, CUT [10], CAP, MO+ [10] and 2Hop [13] normalized with respect to those by RP

6.2 Experimental results

Figure 2 compares the performance of the proposed CAP algorithm against the existing algorithms 2Hop, MO+, CUT as well as BLP. In the figure, we display the geometric means of the ratios of the communication operation counts incurred by the partitions obtained by CAP, BLP, CUT, MO+ and 2Hop to those by RP, for each different K value. We run CAP algorithm with accuracy $\theta = 0.01$ and $\delta = 0.05$. As seen in the figure, BLP performs much better than both 2Hop and MO+, whereas it performs slightly better (6%–9% on average) than CUT. These experimental results justify the use of BLP as a baseline algorithm to test the validity of the proposed CAP algorithm. As also seen in the figure, the proposed CAP algorithm performs significantly better than all algorithm.

Table 3 compares the performance of the proposed CAP algorithm against BLP and RP on each graph for each K value, in terms of average number of communication operations during IC model propagation simulations. Here, partitioning of a graph for each different K value constitutes a partitioning instance. For each K value, the last column entitled "%imp" displays the percent improvement of CAP over BLP for each dataset in terms of the number of communication operations. For each K value, the last row entitled "norm avgs wrto RP" displays the geometric means of the ratios of the communication operation counts incurred by the partitions obtained by BLP and CAP to those by RP. The table also contains a "cut" column which displays the ratio of the number of cut edges to the total number of edges for each partitioning instance.

As seen in Table 3, BLP performs significantly better than RP in all partitioning instances. This is because BLP successfully reduces the sum of propagation probabilities of cut edges and reduces the chances for propagation events to occur between different parts. On average, partitions obtained by BLP incurs 4.76x, 3.84x, 3.57x, 3.22x, 2.77x and 2.63x less communications than RP for $K = 32, 64, 128, 256, 512$ and 1024 servers, respectively. The decrease in the performance gap between BLP and RP with increasing K can be attributed to the performance degradation of the graph partitioning tool for high K values. In particular, whenever the average number of vertices assigned to parts (i.e., $|V|/K$) decreases below some certain threshold (e.g., for $K = 1024$ and $K = 512$ on `Facebook` and `wiki-Vote` datasets), improvements of Metis significantly degrades as can be seen from Table 3. However, for the case of web graphs (e.g., `uk-2002` and `webbase-2001`), Metis provides significantly better partitions, providing cut ratios below 0.1 (i.e., structures of graphs also have effect on quality of partitions produced by Metis). As a result, the number of inter-partition communication operations is significantly less in cases of these graphs as compared to other cases.

As seen in Table 3, CAP performs significantly better than BLP in all of the partitioning instances. If the cut ratio values are closely inspected, partitions obtained by CAP leave more edges in the cut (i.e., higher cut ratios); but these partitions incur less communication operations. On average, CAP achieves 25.16%, 31.82%, 32.04%, 29.97%, 27.36% less communication operations than BLP for $K = 32, 64, 128, 256, 512$ and 1024 servers, respectively. In particular, the best improvement is obtained on `email-EuAll` social network for $K = 64$, where the partitions obtained by CAP achieve 88% less communication operations than those by BLP. Also in this partitioning instance, CAP achieves a cut ratio of 0.35 which is significantly less than 0.75 of BLP. However, as the value of K increases, the improvement of CAP over BLP decreases for some social networks, especially for `wiki-Talk` and `wiki-Vote`, where 19.11% and 19.70% improvements of CAP over BLP for $K = 32$, respectively, decrease to 1.11% and 1.27% for $K = 1024$. This can be attributed to Eq. (20), since as the value of K increases, the number of cut edges is also expected to increase. As shown in Eq. (20), the number of cut edges directly affects the error made by CAP algorithm: the upper bound of the error made by CAP algorithm is shown to be proportional to the number of cut edges. Indeed, the performance improvement of CAP over BLP is observed to be the lowest on the partitioning instances for which CAP incurs the highest cut ratio. For instance, on datasets `Facebook`, `wiki-Talk` and `wiki-Vote` for $K = 1024$, partitions generated by CAP have cut ratios of 0.97, 0.97 and 0.92, respectively.

The performance decrease of CAP can be alleviated by making more accurate estimations for the p_{ij} values and decreasing the value of θ. However, the cut ratio depends on the graph partitioning tool performance, dataset characteristics and imbalance ratios used during partitioning. In order to get better cut ratios, the imbalance constraint can be relaxed and increased to higher values (e.g., we used imbalance ratio of 0.1 in our experiments).

Table 3 Average number of communication operations during IC model propagations under partitions obtained by RP (Random partitioning), BLP (Baseline partitioning) and CAP (proposed cascade-aware graph partitioning) algorithms

	$K = 32$						$K = 64$					
	RP	BLP		CAP			RP	BLP		CAP		
	comm	comm	cut	comm	cut	%imp	comm	comm	cut	comm	cut	%imp
Facebook	3787	1818	0.32	1647	0.38	9.40	3854	2344	0.52	2187	0.57	6.66
wiki-Vote	2127	1681	0.74	1360	0.82	19.11	2160	1824	0.82	1497	0.87	17.91
HepPh	12,048	4092	0.29	3155	0.44	22.91	12,580	5095	0.36	3858	0.52	24.28
email-Enron	25,153	6539	0.39	5083	0.45	22.27	25,514	8073	0.45	6193	0.52	23.29
Epinions	29,801	12,781	0.59	8290	0.66	35.14	30264	13,232	0.64	8699	0.70	34.26
Twitter (small)	68,391	11,901	0.18	8827	0.23	25.83	69,470	14,348	0.22	10,379	0.27	27.66
email-EuAll	27,543	3709	0.21	2592	0.31	30.11	28,050	24,421	0.75	2934	0.35	87.99
Slashdot	57,426	29,724	0.71	21,497	0.77	27.68	58,313	29,416	0.74	22,433	0.79	23.74
dblp	240,487	36,939	0.17	33,853	0.20	8.36	244,170	40,818	0.19	37,270	0.22	8.69
youtube	648,399	147,215	0.33	122,535	0.40	16.76	659,107	163,117	0.38	134,613	0.46	17.47
Pokec	429,046	129,345	0.26	117,943	0.31	8.82	435,335	154,124	0.33	141,410	0.39	8.25
wiki-Talk	729,773	115,825	0.44	93,005	0.47	19.70	740,897	125,724	0.47	98,511	0.50	21.64
LiveJournal	1,152,319	279,846	0.24	234,804	0.30	16.10	1,168,167	317,001	0.29	267,697	0.36	15.55
Twitter (large)	4,183,322	1,950,300	0.57	1,194,675	0.75	38.74	4,250,880	1,942,093	0.68	1,268,234	0.79	34.70
uk-2002	6,933,807	107,593	0.01	60,539	0.03	43.73	7,046,263	114,284	0.01	64,713	0.03	43.38
sinaweibo	40,403,232	12,192,759	0.46	9,541,550	0.56	21.74	40,404,365	12,356,095	0.50	10,169,184	0.61	17.70
webbase-2001	29,301,906	614,458	0.02	329,646	0.03	46.35	29,774,985	651,124	0.02	358,722	0.04	44.91
norm avgs wrto RP	**1.00**	**0.21**		**0.16**		**25.16**	**1.00**	**0.26**		**0.17**		**31.82**
	$K = 128$						$K = 256$					
Facebook	3884	2774	0.70	2629	0.73	5.24	3893	3107	0.82	2980	0.85	4.09
wiki-Vote	2174	1931	0.88	1837	0.96	4.89	2184	2125	0.96	2101	0.97	1.13
HepPh	12,409	5954	0.44	4389	0.60	26.29	12,850	7034	0.53	5048	0.68	28.23
email-Enron	25,714	9596	0.53	7636	0.59	20.42	25,820	11,505	0.59	10,208	0.65	11.27
Epinions	30,506	13,285	0.68	9094	0.72	31.55	30,653	13,635	0.71	9574	0.74	29.78
Twitter (small)	69,993	17,111	0.27	12,336	0.34	27.90	70,169	20,687	0.34	14,709	0.41	28.89

Table 3 continued

	$K = 128$						$K = 256$					
email-EuAll	28,266	24,241	0.79	3231	0.41	86.67	28,348	24,172	0.81	4207	0.47	82.60
Slashdot	58,716	29,501	0.76	23,149	0.81	21.53	58,946	30,265	0.79	23,667	0.81	21.80
dblp	245,923	43,376	0.20	39,053	0.24	9.97	247,006	45,240	0.21	40,534	0.25	10.40
youtube	664,146	177,097	0.43	146,781	0.51	17.12	666,821	195,400	0.49	158,057	0.54	19.11
Pokec	438,930	182,158	0.40	168,153	0.49	7.69	440,504	210,461	0.48	201,080	0.61	4.46
wiki-Talk	747,415	130,856	0.49	111,842	0.51	14.53	747,916	142,779	0.50	138,546	0.52	2.97
LiveJournal	1,177,061	347,405	0.33	295,910	0.40	14.82	1,182,573	368,749	0.36	318,370	0.44	13.66
Twitter (large)	4,284,834	3,449,396	0.86	1,308,157	0.81	62.08	4,301,563	3,610,392	0.93	1,343,795	0.83	62.78
uk-2002	7,101,990	117,968	0.01	68,482	0.03	41.95	7,129,786	126,795	0.01	71,213	0.03	43.84
sinaweibo	40,726,570	12,836,470	0.53	10,679,002	0.63	16.81	40,886,216	13,232,762	0.57	11,027,580	0.65	16.66
webbase-2001	30,011,343	682,865	0.02	383,526	0.04	43.84	30,129,511	717,763	0.02	405,536	0.04	43.50
norm avgs wrto RP	**1.00**	**0.28**		**0.19**		**32.04**	**1.00**	**0.31**		**0.21**		**29.97**
	$K = 512$						$K = 1024$					
Facebook	3912	3625	0.95	3555	0.95	1.91	3914	3721	0.97	3687	0.97	0.91
wiki-Vote	2186	2135	0.97	2131	0.97	0.21	2190	2138	0.96	2115	0.97	1.11
HepPh	12,822	7956	0.61	5555	0.76	30.18	12,863	8910	0.70	6075	0.82	31.82
email-Enron	25,835	13,560	0.66	12,733	0.71	6.10	25,897	15,679	0.72	14,621	0.75	6.75
Epinions	30,631	14,396	0.74	10,320	0.75	28.32	30,674	15,302	0.77	11,424	0.77	25.34
Twitter (small)	70,309	24,979	0.42	18,020	0.50	27.86	70,430	33,555	0.55	23,861	0.64	28.89
email-EuAll	28,388	24,255	0.83	6109	0.58	74.81	28,390	25,421	0.86	10,569	0.70	58.42
Slashdot	59,053	30,532	0.81	24,340	0.82	20.28	59,161	30,944	0.82	24,938	0.83	19.41
dblp	247,367	47,106	0.22	41,875	0.26	11.11	247,658	49,605	0.24	43,372	0.27	12.57
youtube	668,146	200,723	0.52	166,605	0.56	17.00	668,800	212,618	0.55	180,943	0.58	14.90
Pokec	441,892	232,194	0.55	214,309	0.67	7.70	442,150	249,415	0.60	223,356	0.72	10.45
wiki-Talk	752,402	635,183	0.90	622,024	0.92	2.07	752,048	672,420	0.93	663,852	0.92	1.27
LiveJournal	1,185,101	389,357	0.39	340,986	0.48	12.42	1,185,655	412,393	0.42	362,391	0.52	12.12
Twitter (large)	4,310,133	3,498,242	0.94	1,472,766	0.85	57.90	4,314,342	3,917,869	0.97	3,397,889	0.94	13.27
uk-2002	7,144,002	138,545	0.02	75,149	0.04	45.76	7,150,971	156,001	0.02	82,082	0.05	47.38
sinaweibo	40,966,473	13,630,831	0.60	11,516,372	0.67	15.51	40,983,365	14,209,656	0.63	11,904,051	0.69	16.23
webbase-2001	30,188,276	735,969	0.02	427,812	0.04	41.87	30,218,102	760,751	0.02	441,437	0.04	41.97
norm avgs wrto RP	**1.00**	**0.36**		**0.26**		**27.36**	**1.00**	**0.38**		**0.30**		**22.14**

"%imp": improvement of CAP over BLP, "cut": ratio of number of cut edges to total number of edges, K: number of parts/servers

Table 4 Results for IC model propagations on `random-social-network`. "cut" column denotes the fraction of edges remain in the cut after partitioning, "comm" column denotes the average number of communication operations and "%imp" column denotes the percent improvement of CAP over BLP

K	RP		BLP		CAP		%imp
	cut	comm	cut	comm	cut	comm	
32	0.97	24,690	0.85	20,981	0.91	11,890	43.33
64	0.98	25,119	0.92	18,638	0.93	12,317	33.91
128	0.99	25,328	0.93	18,484	0.94	12,635	31.64
256	0.99	25,378	0.94	18,546	0.94	12,961	30.11
512	0.99	25,400	0.95	19,911	0.95	13,515	32.12
1024	0.99	25,511	0.95	20,279	0.95	14,229	29.83

To observe how the improvement of CAP algorithm changes with respect to the cut ratio, we perform the same set of experiments also on `random-social-network`. As seen in Table 4, partitions obtained by CAP algorithm cause 43% less communication operations for $K = 32$ even though the fraction of edges are 6% more than that of BLP. As noted in previous experimental results, the improvement of CAP over BLP decreases as the value of K and the cut ratio increases: the percent improvement of CAP over BLP decreases from 43% to 30% as the fraction of cut edges increases from 0.91 to 0.95 on $K = 32$ and $K = 1024$, respectively.

6.2.1 Sensitivity analysis

We performed experiments to see how the accuracy parameter θ affects the performance of the CAP algorithm. For different values of θ and K, we compare the performance of CAP against RP on `random-social-network`. In Fig. 3, we designate the size of set I as $|I| = 10, 10^2, 10^3, 10^4$ and 10^5, respectively. Experiments are performed under K-way partitions for $K = 32, 64, 128$ and 256. We plot the percent increase in the performance of CAP over RP on y-axis. The accuracy values are computed for confidence $\delta = 0.05$ and displayed on the right side of the figure. As seen in the figure, with increasing size of set I, the value of θ decreases exponentially and the improvement of CAP increases logarithmically. Additionally, as also observed earlier, the relative performance of CAP decreases with increasing K. The best performance improvement is obtained for $K = 32$ where CAP performs 2x better than RP. These results can be attributed to the results of Eq. 20, since for higher values of K, both the cut ratio and the error made by the CAP algorithm increase. However, as the accuracy increases, the error made by CAP decreases and the overall optimization quality improves.

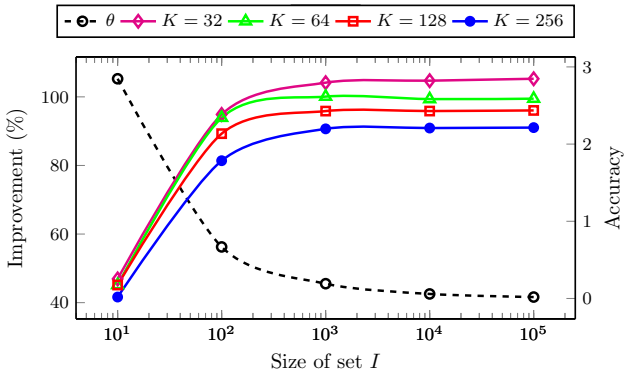

Fig. 3 Variation in the improvement of CAP over RP with different sizes of set I. Dashed curve denotes the accuracy θ, whereas solid lines denote variations in the improvements for `random-social-network` on $K = 32, 64, 128$ and 256 parts/servers

6.2.2 Relationship with minimizing cut edges

We also performed experiments to observe how much the cascade-based estimation of traffic is related to the performance measure of minimizing the number of cut edges. Previously, we asserted that different objectives can be encoded in the same cut size definition through assigning different weights to costs associated with edges by each objective (i.e., $c_{ij} = 1 + \alpha(p_{ij} + p_{ji})$). The parameter α controls how much the cascade-based estimation of the traffic is considered. Figure 4 displays the average number of communication operations and ratio of cut edges, by varying parameter α, obtained by CAP for `HepPh` dataset on $K = 32$ parts/servers (i.e., the α value is multiplied by $F_I(e_{ij})$ value of each edge). As seen in the figure, with increasing α, the average number of communication operations decreases, whereas the ratio of the number of cut edges increases. On the other hand, the increase in the cut size slows down after $\alpha = 10^{-2}$ and cut ratio becomes at most 0.44, since the cut size also has effect on the average number of communication operations. Note that for the smallest value of α, CAP becomes almost equivalent to CUT, providing equally well partitions in terms of number of cut edges (black-dashed curve denotes the cut value obtained by CUT algorithm). If the query workload is dominated by non-cascading queries and there is comparably small number of cascades, then α value can be set to smaller values, or vice versa.

6.2.3 Running times

We performed our experiments on a server having 256 GB main memory and *two Intel(R) Xeon(R) CPU E7-8860 v4 @ 2.20GHz* processors, each of which consists of 18 cores and is able to run 36 threads concurrently. One of the largest social networks we used in our experiments, `sinaweibo`, consists of approximately 58 M vertices and 522 M edges. For this

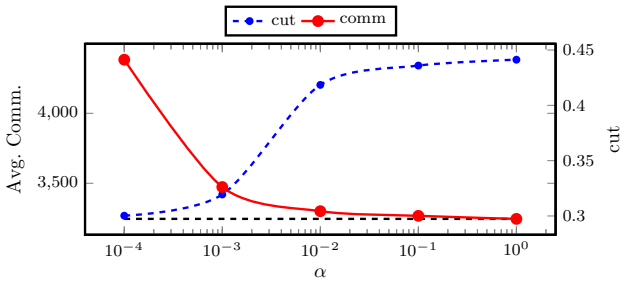

Fig. 4 Variation of average number of communication operations and cut values obtained by CAP for different α values. Experiments are performed for `HepPh` dataset on $K = 32$ parts/servers. Solid curve denotes communication values, and dashed curve denotes cut values. (Black-dashed curve denotes the cut value obtained by CUT algorithm)

social network, the estimation of p_{ij} values took approximately 5.9 h and required 59 GB main memory. For the estimation phase, we employed shared memory parallelism and generated the set I via using 72 threads. The partitioning phase of undirected graph G' into $K = 1024$ parts via Metis took approximately 1, 25 hours and required 71 GB main memory. It is worth to mention that partitioning time of `sinaweibo` via Pulp [51], which is a label propagation-based partitioning tool, took approximately 1.1 h in case of BLP algorithm. Here, both Metis and pulp were run in serial mode for a fair comparison. Similarly, the biggest graph we used in our experiments, `webbase-2001`, has 118 M vertices and 1B edges. For this dataset, the estimation phase took approximately 2.2 h and required 130 GB main memory. However, even though `webbase-2001` has more vertices and edges than `sinaweibo`, partitioning this graph into $K = 1024$ parts took approximately 0.5 h and required 51 GB main memory. Note that the estimation phase of CAP takes only 4.7x and 4.4x more time than the partitioning phase on the two large datasets `sinaweibo` and `webbase-2001`, respectively. These relative runtime comparisons suggest that cascade-aware graph partitioning, considered as an offline process that will be used in relatively long time intervals, runs in reasonable time limits for large-scale social networks.

6.3 Experiments on digg social network with real propagation traces

In this section, we use actual propagation traces collected from `Digg`[5] [18] social news portal. `Digg` is an OSN where users can share and vote for news stories, and designate others as friends to inform about their activities. Friendships can be designated as one-way relationships in such a way that if a user v_i declares another user v_j as a friend, then v_i is informed of activities of v_j but not vice versa. Users follow activities of their friends in their news feeds where the stories their friends shared or voted for are displayed. With these properties of `Digg` social network, a story can propagate through users once it is shared or voted for. Propagation of news stories can be considered as propagation of contents in our problem definition.

The `Digg` dataset contains a directed graph $G = (V, E)$ representing the underlying social network which consists of 71,367 users and 1,731,658 friendship links. As friendships are formed as one-way relationships, they are represented by directed edges. Each directed edge $e_{ij} \in E$ means that user v_j is following the activities of user v_i; therefore, the content propagation can occur in the direction of v_i to v_j. Additionally, the dataset contains log $\mathcal{L}$ of past activities of users over a set $\mathcal{N}$ of news stories. Each entry $(v_i, n_k, t_i) \in \mathcal{L}$ means that user $v_i \in V$ has voted for news story $n_k \in \mathcal{N}$ at time t_i. The dataset contains 3,018,197 votes made on 3,553 news stories (i.e., $|\mathcal{L}| = 3{,}018{,}197$ and $|\mathcal{N}| = 3{,}553$).

In order to deduce the content propagation traces from log $\mathcal{L}$, we follow the approach proposed in [39]. In this approach, if user v_i votes for the news story n_k, then it is assumed that v_i is probably influenced by one of its friends that have voted for the same story before. However, in order for v_i to be influenced by its friends, the difference between their voting times should be within a time window t_Δ. Let P_i^k denote the set of users that potentially influence user v_i in voting for news story n_k, we define P_i^k as

$$P_i^k = \{v_j \in V \mid (v_j, v_i) \in E \wedge t_i - t_j \leq t_\Delta\}. \tag{25}$$

In our experiments, we set the time window t_Δ as *one* month following the approach in [39]. The set P_i^k induces a subgraph $g_k = (V, E_k)$ of G, where potential influencers of each user are denoted by the directed edges in E_k as follows:

$$E_k = \{(v_j, v_i) \in E \mid v_j \in P_i^k\}. \tag{26}$$

The subgraph g_k is reminiscent of a directed graph $g \sim G$, where each directed edge e_{ij} is associated with a propagation probability w_{ij} and g is generated by the equivalent process of the IC model as described in Sect. 3. Note that g is also a subgraph where each user may have multiple potential influencers and one of them can be arbitrarily selected to generate a propagation tree/forest. Therefore, we generate a propagation forest for the news story n_k on g_k as follows: Let $I_k(S)$ denote a propagation forest on g_k, where propagation trees are rooted on vertices in S and the set S is composed of vertices that are having no incoming edges (i.e., users in the set S do not have any potential influencers).

The propagation forest $I_k(S)$ can be computed by performing a multi-source BFS starting from vertices in S on g_k as if a random propagation tree is built from a $g \sim G$. It is important to note that multiple propagation forests can be

[5] www.isi.edu/~lerman/downloads/digg2009.html.

Algorithm 2 Generating Propagation Trees/Forests from logs of past propagation traces

Input: $G = (V, E), \mathcal{L}, t_\Delta$
Output: I
1: Partition log $\mathcal{L}$ based on news stories and obtain $\mathcal{L}_k$ for each story $n_k \in \mathcal{N}$
2: Initialize an empty set I of propagation forests
3: **for each** $\mathcal{L}_k$ **do**
4: Sort entries $(v_i, n_k, t_i) \in \mathcal{L}_k$ according to their timestamps t_i in increasing order
5: Initialize a directed graph $g_k = (V, E_k)$, where $E_k = \emptyset$
6: **for each** entry $(v_i, n_k, t_i) \in \mathcal{L}_k$ **do**
7: Mark v_i as activated
8: **for each** $(v_j, v_i) \in E$ **do**
9: **if** v_j is activated **and** $t_j - t_i \leq t_\Delta$ **then**
10: $E_k = E_k \cup \{(v_j, v_i)\}$
11: Initialize set $S = \{v_i \mid \text{in-degree of } v_i = 0 \text{ in } g_k\}$
12: Perform multi-source BFS on g_k starting from the vertices in S and generate a propagation forest I_k
13: $I = I \cup \{I_k\}$
14: **return** I

generated depending on the execution of multi-source BFS on g_k. Edges in the propagation forest $I_k(S)$ still encode the information as to propagation traces through users. We generate propagation trees/forests for all news stories in Log $\mathcal{L}$ and use them instead of performing the IC model propagation simulations. Algorithm 2 presents computations we performed on log $\mathcal{L}$ to deduce the content propagation traces.

After generating the propagation trees/forests for all news stories available in log $\mathcal{L}$, we sample 90% of these trees/forests to use in Algorithm 1. That is, instead of randomly generating random propagation forests, we use real propagations in log $\mathcal{L}$ to compute the function $F_I(\cdot)$ and estimate p_{ij} values of edges in G. We use the remaining 10% of the propagation forests to test the qualities of the partitions returned by Algorithm 1. If an edge in a propagation forest crosses different parts, we count that edge as one communication operation.

We compare the qualities of the partitions produced by CAP algorithm against those of a slightly modified version of the BLP algorithm presented previously. In the modified version of BLP, we associate unit cost with each edge of the undirected graph that is produced from the input social graph. This modification causes BLP to regard only the friendship structure of `Digg` social network and produce partitions that minimize the number of friendship links crossing different parts. In this way, BLP and CUT algorithms become equivalent.

In Table 5, we present the results of the experiments on `Digg` social network. In addition to CAP and the modified version of BLP, we also include the results for partitions generated by RP. For each of these partitions, we compute the average number of communication operations induced on the propagation trees that are generated and sampled from 10 percent of log $\mathcal{L}$.

Table 5 Experimental results on Digg social network. For each tested algorithm, average number of communication operations induced during propagation of news stories are displayed. "%imp" column denotes the percent improvement of CAP over BLP

K	RP	MO+	2Hop	BLP	CAP	%imp
32	192	189	151	101	40	60.80
64	195	193	160	86	44	48.11
128	196	196	167	119	62	47.97
256	197	197	172	128	77	39.65
512	198	198	174	133	102	23.06
1024	198	198	177	152	131	13.53

As seen in Table 5, BLP performs much better than 2Hop, CUT, MO+ and RP algorithms. For $K = 32$, the partition generated by BLP incurs approximately 2x less communication operations than RP. The performance improvements of BLP is less for higher values of K. For example, BLP performs 2 times better than RP for $K = 1024$. CAP algorithm, on the other hand, consistently performs better than BLP for all values of K. In particular, for $K = 32$, CAP algorithm incurs 60% less communication operations. However, as the value of K increases from 32 to 1024, the overall improvement of CAP over BLP decreases to 13%. This is because the accuracy obtained by 90% of the propagation trees/forest sampled from log $\mathcal{L}$ remains constant as we increase the value of K and therefore the error made by CAP algorithm increases as we have shown in Eq. 20. Additionally, the performance of the graph partitioning tool is expected to decrease for higher values of K where the average number of vertices per part reduces below 100 for $K = 1024$.

Results displayed in Table 5 illustrates the effectiveness of the CAP algorithm in a case where actual propagation traces are used instead of the IC model simulations.

7 Conclusion

We studied the problem of cascade-aware graph partitioning, where we seek to compute a user-to-server assignment that minimizes the communication between servers/parts considering content propagation processes.

We employed a sampling-based method to estimate a probability distribution by which each edge of a graph is associated with a probability of being involved in a random propagation process. We use these estimates as part of the input of graph partitioning. The proposed solution works under various cascade models and requires that parameters of these models are given beforehand. Theoretic results that show how our solution achieves the stated objectives are also

derived. To the best of our knowledge, this is the first work that incorporates the models of graph cascades into a systems performance goal.

We performed experiments under the widely used IC model and evaluated the effectiveness of the proposed solution in terms of partitioning objectives. We implemented the solution over real logs of propagation traces among users, in addition to using their social network structure. Experiments demonstrate the effectiveness of the proposed solution in both the presence and absence of actual propagation traces.

References

1. Huang, J., Abadi, D.J.: Leopard: Lightweight edge-oriented partitioning and replication for dynamic graphs. Proc. VLDB Endow. **9**(7), 540–551 (2016)
2. Mondal, J., Deshpande, A.: Managing large dynamic graphs efficiently. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 145–156. ACM, (2012)
3. Yang, S., Yan, X., Zong, B., Khan, A.: Towards effective partition management for large graphs. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 517–528. ACM, (2012)
4. Curino, C., Jones, E., Zhang, Y., Madden, S.: Schism: a workload-driven approach to database replication and partitioning. Proc. VLDB Endow. **3**(1–2), 48–57 (2010)
5. Yaşar, A., Gedik, B., Ferhatosmanoğlu, H.: Distributed block formation and layout for disk-based management of large-scale graphs. Distrib. Parallel Databases **35**(1), 23–53 (2017)
6. Karypis, G., Kumar, V.: Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. (1995)
7. Çatalyürek, Ü., Aykanat, C.: Patoh (partitioning tool for hypergraphs). In: Encyclopedia of Parallel Computing, pp. 1479–1487. Springer, (2011)
8. Newman, M.E.J.: Modularity and community structure in networks. Proc. Natl. Acad. Sci. **103**(23), 8577–8582 (2006)
9. Pujol, J.M.., Siganos, G., Erramilli, V., Rodriguez, P.: Scaling online social networks without pains. In: Proceedings of NETDB (2009)
10. Pujol, J.M., Erramilli, V., Rodriguez, P.: Divide and conquer: Partitioning online social networks. arXiv preprint arXiv:0905.4918 (2009)
11. Pujol, J.M., Erramilli, V., Siganos, G., Yang, X., Laoutaris, N., Chhabra, P., Rodriguez, P.: The little engine (s) that could: scaling online social networks. ACM SIGCOMM Comput. Commun. Rev. **41**(4), 375–386 (2011)
12. Carrasco, B., Lu, Y., da Trindade, J.M.F.: Partitioning social networks for time-dependent queries. In: Proceedings of the 4th Workshop on Social Network Systems. ACM, (2011)
13. Yuan, M., Stein, D., Carrasco, B., Trindade, J.M.F., Lu, Yi: Partitioning social networks for fast retrieval of time-dependent queries. In: Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on, pp. 205–212. IEEE (2012)
14. Turk, A., Selvitopi, R.O., Ferhatosmanoglu, H., Aykanat, C.: Temporal workload-aware replicated partitioning for social networks. IEEE Trans. Knowl. Data Eng. **26**(11), 2832–2845 (2014)
15. Gruhl, D., Guha, R., Liben-Nowell, D., Tomkins, A.: Information diffusion through blogspace. In: Proceedings of the 13th International Conference on World Wide Web, pp. 491–501. ACM, (2004)
16. Bakshy, E., Rosenn, I., Marlow, C., Adamic, L.: The role of social networks in information diffusion. In: Proceedings of the 21st International Conference on World Wide Web, pp. 519–528. ACM, (2012)
17. Chen, W., Wang, C., Wang, Y.: Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: Proceedings of the 16th ACM SIGKDD International Conference On Knowledge Discovery and Data Mining, pp. 1029–1038. ACM, (2010)
18. Digg social news portal. http://digg.com/, (2017)
19. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified np-complete graph problems. Theor. Comput. Sci. **1**(3), 237–267 (1976)
20. Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 135–146. ACM, (2010)
21. Nicoara, D., Kamali, S., Daudjee, K., Chen, L.: Hermes: Dynamic partitioning for distributed social network graph databases. In: EDBT, pp. 25–36, (2015)
22. Le, W., Kementsietsidis, A., Duan, S., Li, F.: Scalable multi-query optimization for sparql. In: IEEE 28th International Conference on Data Engineering (ICDE), pp. 666–677. IEEE, (2012)
23. Goldenberg, J., Libai, B., Muller, E.: Talk of the network: a complex systems look at the underlying process of word-of-mouth. Mark. Lett. **12**(3), 211–223 (2001)
24. Granovetter, M.: Threshold models of collective behavior. American journal of sociology, pp. 1420–1443, (1978)
25. Goyal, A., Bonchi, F., Lakshmanan, L.V.S.: Learning influence probabilities in social networks. In: Proceedings of the third ACM International Conference on Web Search and Data Mining, pp. 241–250. ACM, (2010)
26. Saito, K., Nakano, R., Kimura, M.: Prediction of information diffusion probabilities for independent cascade model. In: International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, pp. 67–75. Springer, (2008)
27. Domingos, P., Richardson, M: Mining the network value of customers. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 57–66. ACM, (2001)
28. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 137–146. ACM, (2003)
29. Chen, W., Wang, Y., Yang, S.: Efficient influence maximization in social networks. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 199–208. ACM, (2009)
30. Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., Van Briesen, J., Glance, N.: Cost-effective outbreak detection in networks. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 420–429. ACM, (2007)
31. Wang, Y., Cong, G., Song, G., Xie, K.: Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1039–1048. ACM, (2010)

32. Li, H., Bhowmick, S.S., Sun, A., Cui, J.: Conformity-aware influence maximization in online social networks. VLDB J. **24**(1), 117–141 (2015)
33. Borgs, C., Brautbar, M., Chayes, J., Lucier, B.: Maximizing social influence in nearly optimal time. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 946–957. Society for Industrial and Applied Mathematics, (2014)
34. Tang, Y., Xiao, X., Shi, Y.: Influence maximization: Near-optimal time complexity meets practical efficiency. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 75–86. ACM, (2014)
35. Cohen, E., Delling, D., Pajor, T., Werneck, R.F.: Sketch-based influence maximization and computation: Scaling up with guarantees. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp. 629–638. ACM, (2014)
36. Zhou, Y., Liu, L.: Social influence based clustering of heterogeneous information networks. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 338–346. ACM, (2013)
37. Zaixin, L., Zhu, Y., Li, W., Weili, W., Cheng, X.: Influence-based community partition for social networks. Comput. Soc. Netw. **1**(1), 1 (2014)
38. Ghosh, R., Lerman, K.: Community detection using a measure of global influence. In: Advances in Social Network Mining and Analysis, pp. 20–35. Springer, (2010)
39. Barbieri, N., Bonchi, F., Manco, G.: Cascade-based community detection. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining, pp. 33–42. ACM, (2013)
40. Chevalier, C., Pellegrini, F.: Pt-scotch: a tool for efficient parallel graph ordering. Parallel Comput. **34**(6), 318–331 (2008)
41. Colbourn, C.J., Colbourn, C.J.: The Combinatorics of Network Reliability, vol. 200. Oxford University Press, New York (1987)
42. Motwani, R., Raghavan, P.: Randomized Algorithms. Chapman & Hall/CRC, Boca Raton (2010)
43. Karypis, G., Schloegel, K., Kumar, V.: Parmetis. Parallel graph partitioning and sparse matrix ordering library. Version, 2, (2003)
44. Boldi, P., Vigna, S.: The webgraph framework i: compression techniques. In: Proceedings of the 13th International Conference on World Wide Web, pp. 595–602. ACM, (2004)
45. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, (June 2014)
46. Zafarani, R., Liu, H.: Social computing data repository at ASU. http://socialcomputing.asu.edu, (2009)
47. Boldi, P., Vigna, S.: The WebGraph framework I: Compression techniques. In: Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004), pp. 595–601, Manhattan, USA, (2004). ACM Press
48. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, (2015)
49. Bader, D., Madduri, K., Gilbert, J., Shah, V., Kepner, J., Meuse, T., Krishnamurthy, A.: Designing scalable synthetic compact applications for benchmarking high productivity computing systems. CT Watch **2**, 1–10 (2006)
50. Rosvall, M., Axelsson, D., Bergstrom, C.T.: The map equation. Eur. Phys. J. Spec. Top. **178**(1), 13–23 (2009)
51. Slota, G.M., Madduri, K., Rajamanickam, S.: Pulp: Scalable multi-objective multi-constraint partitioning for small-world networks. In: Big Data (Big Data), 2014 IEEE International Conference on, pp. 481–490. IEEE, (2014)