# Bridging Community Detection and Influence Maximization: A Generalized Approach

Máté Vass[1], Miklós Krész[2,3,4], László Hajdu[2,3], András Bóta[6*]

[1]Department of Computer Algorithms and Artificial Intelligence, University of Szeged, Árpád tér 2, Szeged, HU-6720, Hungary.
[2]Innorenew CoE, Livade 6a, Izola, SI-6310, Slovenia.
[3]Faculty of Mathematics, Natural Sciences and Information Technologies, University of Primorska, Glagoljaška 8, Koper, SI-6000, Slovenia.
[4]Department of Applied Informatics, University of Szeged, Boldogasszony sgt. 6, Szeged, H-6725, Hungary.
[5]Department of Computer Science, Electrical and Space Engineering, Embedded Intelligent Systems Lab, Luleå University of Technology, Luleå, SE-97187, Sweden.

*Corresponding author(s). E-mail(s): andras.bota@ltu.se;
Contributing authors: vassmate@inf.u-szeged.hu;
miklos.kresz@innorenew.eu; laszlo.hajdu@innorenew.eu;

**Abstract**

In recent years, many papers have focused on community detection and influence spread in social networks. The influence maximization problem involves determining a set of nodes from which the maximum expected value of influenced nodes is generated through a given influence process. Our intuition was that in social networks, entities with important community roles are crucial for solving this problem. However, most of the existing approaches focus on specific models and their unique attributes. Our goal was to design an algorithm that is generalized in a way that it is independent of the specifics of these models. We have developed a general community detection method that can use any influence model as its input, and based on the found communities, it can narrow down the search space of the influence maximization problem. We only select nodes from the best candidates sorted by their community roles and try to maximize the expected final influence value. With our proof of concept approach, we show the efficiency of our algorithm on artificial and real-life benchmark graphs for the popular Independent Cascade, Linear Threshold and Only-Listen-Once influence models.

**Keywords:** network science, graph mining, influence modelling, community detection, influence maximization

## 1 Introduction

Networks are powerful modelling tools due to their ability to provide visual and mathematical representations of complex systems. They are often used in applications to represent interactions and relationships between people (Borgatti et al., 2009), financial and technological connections between companies (Hajdu and Krész, 2020; Bóta et al., 2015), relationships between words in our language (Bóta and Kovács, 2014; Kovács et al., 2021) and in many other ways (Newman and Girvan, 2004). Due to the ever-increasing number of internet users and the rise of social media, social network analysis has become an important topic in data science. In this context, networks can help identify closely knit communities, influential nodes or the spread of diseases and information.

In network science by communities we mean groups of nodes which are more densely connected to each other than to the rest of the network. Many real-life networks, including social networks, show community structure, meaning their edge distribution is both globally and locally inhomogeneous. A large variety of detection algorithms have been proposed to identify communities, and these can be categorized into different families (Fortunato, 2010). One fundamental difference between existing methods is whether or not they allow overlaps to exist between communities.

Modelling the spread of information (diffusion) and influence maximization are two closely related fields in science. The most commonly used network-based information spreading models come from 1. the field of epidemic spreading, e.g., the SEIR compartmental models (Kermack and McKendrick, 1927; Brauer and Castillo-Chavez, 2001), and their derivative, the Independent Cascade Model (Domingos and Richardson, 2001; Kempe et al., 2015), or 2. from the field of sociology, namely the Linear Threshold Model (Granovetter, 1978; Kempe et al., 2015) and its extensions. Various other network-based alternative methods have been proposed such as the Only-Listen-Once model (Kempe et al., 2015). Since the computation of transmission probabilities is NP-hard for most of the popular models (Kempe et al., 2015; Krész and Pluhár, 2018), approximation methods are often used to calculate these values (Li et al., 2018). It is also shown in (Wasserman and Faust, 1994; Kempe et al., 2015) that an adequate number of simulations is enough to quickly converge to accurate solutions.

The influence maximization problem aims to find the set of $K$ initially influenced (or active) nodes influencing (or activating) the largest fraction of nodes in the network, where $K$ is a parameter of the problem (Kempe et al., 2015). In the same paper where the problem was introduced, the authors also proposed a greedy heuristic algorithm which still serves as a benchmark today. The field attracted widespread interest and many algorithms have been proposed to solve the problem (Li et al., 2018).

Recent studies have shown that influence maximization and community detection can be used in combination to achieve better performance on both artificially generated and real-world graphs (Kovács et al., 2010; Hajdu et al., 2021; Rajeh and Cherifi, 2022). Kovács et al. (Kovács et al., 2010) showed that one possible way to combine these two methods is to use the results of a simulated influence process, usually by the Independent Cascade model, as the input for the community detection algorithm. The node modules (communities) can be identified by locating hills in the community landscape. One limitation of this approach is that while the Independent

Cascade model is the most popular influence model in network science, it does not incorporate threshold characteristics and its single-chance-of-influence behaviour is not realistic in many real-life situations. In a recent study, Rajeh and Cherifi (Rajeh and Cherifi, 2022) proposed a novel node ranking strategy exploiting the ubiquity of the community structure in real-world networks by selecting distant spreaders.

Hajdu et al. (Hajdu et al., 2021) defined so-called community values to rank influential nodes in networks. First, they applied overlapping community detection on the entire graph and then they counted how many communities a single node belongs to. The vertices could be ranked based on these values: the ones that were part of many communities had higher values assigned to them, thus having more influence on the network. They proposed that using these community values, the search space of the greedy influence maximization method can be efficiently narrowed. The idea was tested with numerous community detection algorithms on the Independent Cascade model.

In this paper, our main contribution is to propose a general community detection algorithm that is capable of using the output of any influence spreading algorithm to create communities. The algorithm repeatedly simulates influence processes on the input graph, and measures how much each individual edge contributes to the spreading process. A new influence graph is then constructed from these edge influence values, and a community detection method is applied to generate communities in an agglomerative fashion using a local expansion rule. We demonstrate the generality of this approach using a variety of influence spreading models: 1. Independent Cascade, 2. Linear Threshold and 3. Only-Listen-Once models.

As our secondary contribution, following the approach of Hajdu et al., we use the output of our newly proposed community detection algorithm to improve the performance of the greedy influence maximization algorithm of Kempe et al. We show that regardless of the influence model, choosing the most influential nodes based on their community value is efficient for the influence maximization problem. We test the performance of our approach on a variety of artificially generated and real-life benchmark networks. We have to highlight that it is not possible to directly compare our results to (Hajdu et al., 2021) since they used undirected networks in their study compared to our directed ones.

The remainder of the paper is structured as follows. In Section 2 we introduce the main concepts of our research and the three influence models. Section 3 is the main contribution of the article where our methodology is presented in detail. We showcase our approach to identify communities according to three different influence models and then use the results to enhance the performance of the previously described greedy influence maximization algorithm. Furthermore, in Section 4 we define our particular experimental setup and provide the datasets used for evaluation. Our results are demonstrated in Section 5 where we test our methodology on different artificially generated and real-life networks as well. Finally, we wrap up our study with Section 6, concluding our study with summing up our contributions, limitations and avenues for future research.

## 2 Background

Before describing our newly proposed community detection algorithm, in this section, we provide a short overview on the existing models and algorithms we use during our work. Namely, the three influence models (Independent Cascade, Linear Threshold and Only-Listen-Once models), and the greedy influence maximization algorithm.

### 2.1 Influence models

Graph-based influence models (also known as diffusion models) can be used to model a large variety of real-life phenomena. In this paper, we focus on the spreading opinion and influence on graphs.

Let $G(V, E)$ denote graph $G$, with $V(G)$ as the set of nodes and $E(G)$ as the set of edges connecting them. The influence models used in this paper require an influence transmission value $0 < w_{u,v} \leq 1$ to be assigned to each edge of the graph. Furthermore, all models assign states to the nodes of the network: they can be *active* or *inactive*. These models also require a set of initially influenced nodes $A_0$ to be selected before the spreading process begins. How other nodes become active varies depending on the influence model. One similarity among the chosen models is that they are *progressive* i.e. active nodes keep their state until the end of the process, meaning they can't become inactive again. The spreading process takes place in discrete time steps or iterations, and each iteration can be characterized with the number of active nodes. Progressive models are clearly finite, the influence spreading process stops after a finite number of iterations. Our developed methodology considers the so-called *Generalized Threshold Model* as a framework: the decision of a node $v$ decision to become active can be based on an arbitrary monotone function of the set of neighbours of $v$ that are already active.(Kempe et al., 2015) The three models chosen for demonstrating the effectiveness of our methodology fit into this framework with an appropriate formulation. (Kempe et al., 2015)

#### 2.1.1 Independent Cascade model

In the Independent Cascade (IC) model (Kempe et al., 2015) nodes try to activate their inactive neighbors according the the influence transmission values between them starting from the set of initially active nodes $A_0$. However, in any iteration $i$, only the newly activated nodes, activated in iteration $i - 1$, are able to activate their inactive neighbors. This effectively creates another state for the nodes: activated, newly activated and inactive. These newly activated nodes have one chance to activate each of their inactive neighbors according to the $w_{u,v}$ influence probability assigned to the edge connecting them. If the attempt is successful, the target node will become active in iteration $i$. If a node has multiple newly activated neighbors, the activation attempts are made in an arbitrary order independently of each other.

### 2.1.2 Linear Threshold model

The Linear Threshold model, initially proposed by Granovetter in (Granovetter, 1978) was later reformulated for graphs in (Kempe et al., 2015). The model has additional requirements for the influence transmission values (weights): for each node $v$, $\sum_{w \in N_v^+} w_{u,v} \leq 1$: the weights on the in-edges of a node may sum up to 1 at maximum. This model also assigns thresholds to each node at the beginning of the process either randomly or with a user specified strategy. These thresholds are denoted as $\theta_v$ for all $v \in V(G)$. The model starts from the set of initially active nodes $A_0$ at iteration 0. At iteration $i$ all nodes remain active that were active in the previous iteration, and any node $v$ becomes active if the total weight of its active neighbors exceed $\theta_v$: $\sum_{w \in N_v^+} w_{u,v} \geq \theta_v$.

### 2.1.3 Only-Listen-Once model

The Only-Listen-Once model was also introduced in (Kempe et al., 2015) and - like in the Linear Threshold model - each node $n$ has a weight $0 < p_n \leq 1$ assigned to them. In this model, the first active neighboring node of $n$ tries to activate it with probability $p_n$ and any further attempt will inevitably fail. This means that $n$ only ”listens” to the first neighbor that tries to activate it. The whole activation process unfolds the same way as in the Independent Cascade or Linear Threshold model in finite, discrete time steps.

## 2.2 Influence maximization

The influence maximization problem proposed by Kempe et al. in (Kempe et al., 2015) was inspired by the work of Domingos and Richardson on virus marketing (Domingos and Richardson, 2001). The goal of their study was to identify a few key individuals that have the greatest influence on the network of buyers. The proposed methodology can easily be applied in other settings, such as the spreading of influence or opinion on networks.

For a set of initially active nodes $A_0$, let the expected number of active nodes at the end of a spreading process be $\sigma(A_0)$, and let the cardinality of $A_0$ be $K$. The objective of the influence maximization problem is to maximize the expected number of activated nodes $\sigma(A_0)$ when choosing the set of $K$ initially active nodes $A_0$. In the same paper where they introduced the problem, Kempe et. al. (Kempe et al., 2015) proved that greedy maximization guarantees an $1 - 1/e$ approximation for a wide subclass of the General Threshold Model (Submodular Threshold Model), which is the best possible theoretical bound (assuming that $P \neq NP$). Since then, a great variety of algorithms have been designed to solve the influence maximization problem (Li et al., 2018), but the greedy algorithm remains a fundamental benchmark.

### 2.2.1 Greedy algorithm of Kempe et al.

The greedy influence maximization heuristic of Kempe et al. (Kempe et al., 2015) starts with an empty initial set $A_0$ and iteratively adds nodes to it until $|A_0| = K$, in each step maximizing $\sigma(A_0)$ with greedy decisions. Algorithm 1 shows the pseudocode of the greedy method.

**Algorithm 1** Greedy method

**Input:** $G(V, E)$ benchmark graph, $K$: desired size of $A_0$
**Output:** $A_0$ initially influenced set

```
1: $A_0 \leftarrow \emptyset$
2: While $|A_0| \leq K$
3:     $A_0 = A_0 \cup \; arg\ max_{v \in V(G) \setminus A_0)} \sigma(A_0 \cup \{v\})$
```

The algorithm starts with $A_0 = \emptyset$. In each iteration $i$, a node $v_i$ is selected from $V$, so that $\sigma(A_{i-1} \cup v_i)$ is maximal. This process stops when the size of $A_i$ reaches $K$. The proposed method can be computationally expensive for two main reasons. Computing $\sigma$ itself can be time consuming especially for large graphs. Furthermore, in each iteration, the greedy algorithm has to evaluate $\sigma$ for $|V(G)| \setminus A_i$, which again, for large graphs, can be time consuming. We refer to two heuristics to solve these issues in the next section.

### 2.2.2 Influence approximation with simulation

Computing the transmission probabilities is an NP-hard problem for the influence models used in our study (Kempe et al., 2015). For example, for any initially influenced node set, computing the influence functions is #P-complete for the Independent Cascade model (Chen et al., 2010). This is the reason why many heuristics and approximation methods emerged in the past decades to tackle this problem (Li et al., 2018). Others have also shown that it is possible to quickly converge to accurate solutions with an adequate number of simulations (Wasserman and Faust, 1994; Kempe et al., 2015). While simulating influence spread, we also reach back to these heuristics: First, at the generation of influence graphs (Algorithm 2), then during the calculation of the final influence values for a given initially influenced node set (Algorithms 1 and 4).

## 3 Methodology

In this section, we are going to describe the the core concepts of our algorithm that aims to provide a general approach for the influence maximization problem. First, we begin by introducing our influence simulation method that creates networks which we call influence graphs. These networks will serve as the input for our novel community detection algorithm. This method is general in a way that it works on any directed and weighted network, and the influence models only play a role in generating the input graphs themselves. Finally, we introduce community values and showcase our methodology that narrows down the search space of the previously mentioned greedy influence maximization algorithm.

### 3.1 Influence simulation

As the first step of our method, an influence model has to be chosen. This can be any model that fulfils the following criteria: 1. it describes the spread of influence in discrete steps on a certain network, 2. the number of steps is finite and 3. an equivalent instance of the so-called triggering model (a well-known subclass of Submodular Threshold Models) exists (Kempe et al., 2015) for the chosen model. As a result of the simulation we obtain the *influence graph* in which the weight of an edge directed from node $v$ to $w$ is the approximate probability of $w$ being activated by the influence process starting from $v$. We note that the methodology is not restricted to the triggering model; its choice is technical concerning the unified implementation framework. To showcase the abilities of our algorithm, we decided to consider three popular models, namely the Independent Cascade (Domingos and Richardson, 2001; Kempe et al., 2015), Linear Threshold (Granovetter, 1978; Kempe et al., 2015) and Only-Listen-Once models (Kempe et al., 2015) described in detail in Section 2.1.

After implementing the above mentioned models, we run influence simulations on the input benchmark graphs. Algorithm 2 shows the pseudocode of our approach.

**Algorithm 2** Influence simulation

**Input:** $G = (V, E)$ benchmark graph, $M$: influence model $I$: number of iterations, $d_{max}$: maximum influence distance
**Output:** $G'' = (V, E'')$ influence graph

```
1:  $D \leftarrow dictionary()$
2:  $i \leftarrow 0$
3:  while $i < I$ do
4:      $i \leftarrow i + 1$
5:      $E' \leftarrow triggering(G, M)$
6:      $G' \leftarrow (V, E')$
7:      for each $v_1, v_2 \in V, v_1 \neq v_2$ do
8:          if $distance(G', v_1, v_2) \leq d_{max}$ then
9:              if $(v_1, v_2) \notin D_{keys}$ then
10:                 $add(D, (v_1, v_2), 1)$
11:             else
12:                 $increment(D, (v_1, v_2))$
13:             end if
14:         end if
15:     end for
16: end while
17: for each $value \in D_{values}$ do
18:     $value \leftarrow value \div I$
19: end for
20: $E'' \leftarrow pairs(D)$
21: $G'' \leftarrow (V, E'')$
```

Our influence simulation algorithm takes a benchmark graph $G$ and an influence model $M$ as its two main inputs to generate the output influence graph $G''$. First we initiate an empty dictionary $D$ with the *dictionary*() function before we begin the $I$ number of iterations. In each iteration we construct a graph instance by simulating a single influence process. In these instances we keep or delete each edge $e \in E$ based on rules that involve randomness, which we will discuss later. We achieve this using the triggering model equivalent $triggering(G, M)$ of the input model $M$ described in detail in (Kempe et al., 2015). This is where we use the vertex and node weights of the benchmark graphs.

The second part of the iteration is finding each directed path in the graph instance that is at most $d_{max}$ long. We store each path in $D$ with its endpoint node pairs as keys and we count how many instances contain them. As we are going through the iterations, if there is a directed $(v_1, v_2)$ path in the instance and $|(v_1, v_2)| \leq d_{max}$ is fulfilled, we modify the dictionary. If $(v_1, v_2) \in D_{keys}$ already then we increase the value associated with the key by 1 in $increment(D, (v_1, v_2))$, otherwise we add this new key $(v_1, v_2)$ to $D$ with value 1 in $add(D, (v_1, v_2), 1)$. $D$ can be interpreted as a structure where the keys are directed edges and the values are the edge weights. In this representation there is a directed edge between two nodes $v_1, v_2 \in V$ if the influence would spread from $v_1$ to $v_2$ supposing $v_1$ was influenced.

After every iteration is done we can create the output influence graph using $D$. First, we divide each value $d \in D_{values}$ by $I$. This way $\forall d \in D_{values} : 0 < d \leq 1$ is guaranteed. As the final step, we take the *keys* from the dictionary as directed edges and the *value* associated with the key as their weight to get $E''$ in $pairs(D)$. With vertices $V$ from the input benchmark graph and $E''$ we have generated the output influence graph $G'' = (V, E'')$.

## 3.2 Community detection

After generating the influence graph for the given influence model, the resulting networks will serve as the input for our novel community detection algorithm. From this point onwards the methodology is generalized in a way that it will not use influence models anymore. All the influence characteristics are stored purely in the influence graphs. To experiment with a new model only the influence graph has to be generated from the mentioned influence model. The pseudocode of our community detection method is shown in Algorithm 3.

To generate communities, our method takes the influence graph $G''$ and three other parameters, namely $\kappa_{max}$, $cp$ and $ta$ as its input. $G''$ is the influence graph generated with Algorithm 2, and $\kappa_{max}$ is the desired maximum size of the communities we are looking for. The remaining parameters, $cp$ and $ta$ are responsible for building up the communities bottom-up, we will describe them in detail later. We store the found communities as sets of nodes, and collect them in the output community set $C$. In this representation, $C$ is a set of sets, where each $c \in C$ refers to a certain community. While our algorithm is running, we will call these non-final sets of nodes *community initiatives*. As an initialization step, we fill-up $C$ with each $v \in V$ vertex from the graph as a set of only one node. It means that in the beginning, we consider each node as a separate community.

To expand these community initiatives further, we execute $\kappa_{max}$ iterations. In each iteration, we try to extend each community initiative found in the previous iteration with one node that doesn't belong to it. The newly extended community initiatives are stored in $C_{new}$ which is reset each iteration. Each $c \in C, |c| = \kappa$ community initiative and $v \in V, v \notin C$ vertex is paired to examine whether they can form a new community

**Algorithm 3** Community detection

**Input:** $G'' = (V, E'')$ influence graph, $\kappa_{max}$: maximum community size, *cp*: connected percent, *ta*: times average

**Output:** $C$ set of communities

```
 1: $C \leftarrow \emptyset$
 2: for each $v \in V$ do
 3:     $C \leftarrow C \cup \{v\}$
 4: end for
 5: $\kappa \leftarrow 0$
 6: while $\kappa < \kappa_{max}$ do
 7:     $\kappa \leftarrow \kappa + 1$
 8:     $C_{new} \leftarrow \emptyset$
 9:     for each $c \in C, |c| = \kappa$ and $v \in V, v \notin C$ do
10:         if $isConnected(c, v, cp)$ and $isWeight(c, v, ta)$ then
11:             $C_{new} \leftarrow C_{new} \cup \{c \cup v\}$
12:         end if
13:     end for
14:     for each $c \in C$ and $c_{new} \in C_{new}$ do
15:         if $c \subset c_{new}$ then
16:             $C \leftarrow C \setminus c$
17:         end if
18:     end for
19:     for each $c_{new} \in C_{new}$ do
20:         $C \leftarrow C \cup c_{new}$
21:     end for
22: end while
```

initiative together. Notice that $v$ comes from the set of all nodes in the graph, which means that our algorithm is capable of finding overlapping communities.

The above mentioned extension step depends on two criteria given in the form of parameters $cp \in (0, 1]$ and $ta \in \mathbb{R}$. The $isConnected(c, v, cp)$ function checks how many directed edges are there from $v$ to any of the nodes in $c$. If this number is greater or equal than $|c| \times cp$, then the first criteria is met. The $isWeight(c, v, ta)$ function calculates the average edge weight for $G''$ in $avg(G'')$ and for the subgraph formed by the nodes $c \cup v$ in $avg(c \cup v)$. If $avg(c \cup v) > avg(G'') \times ta$, then the second criteria is also met. If both criteria are fulfilled, we have found a new community initiative in the iteration as the set of nodes $c \cup v$.

After the extension step, we have found each new community initiative that fulfills our two criteria. Before adding these to the output community set $C$, we delete any $c \in C$ from $C$ that is a subset of any $c_{new} \in C_{new}$. Without this step, $C$ would contain numerous community initiatives that are subsets of bigger ones. Finally, we can add each newly found $c_{new} \in C_{new}$ community initiative to $C$. After $\kappa_{max}$ iterations are done, the output is produced in the form of the final community set $C$.

## 3.3 Community values

Existing literature (Chen et al., 2014) has established that nodes, which belong to multiple communities, may play a significant role in the influence maximization problem. Thus we decided to assign a number called community value to each vertex in the graph. This value represents the number of communities a certain node belongs to. In other words, we defined a function $f(v) : v \to Z$ that assigns this integer community value to each vertex. We took the community detection results for each benchmark graph – influence model pair $(G, M)$ and then we created the $L_{GM}$ ordered lists of nodes based on their $f(v)$ community values in decreasing order. These lists will be used to generate the input of the narrowed search space greedy approach described in detail in 3.4.

## 3.4 Greedy with narrowed search space

We took inspiration from Hajdu et al. (Hajdu et al., 2021) in narrowing down the search space of the greedy influence maximization algorithm since we wanted to see how this approach affects performance and runtime compared to the original greedy method in (Kempe et al., 2015). The intuition behind this heuristic is that nodes that are part of numerous communities should be prioritized when choosing the most influential initial subset. The method is described in detail in Algorithm 4.

**Algorithm 4** Greedy with narrowed search space

```
Input: $G(V, E)$ benchmark graph, $K$: desired size of $A_0$, $V^*(G)$: narrowed node set
Output: $A_0$ initially influenced set
1: $A_0 \leftarrow \emptyset$
2: While $|A_0| \leq K$
3:     $A_0 = A_0 \cup \ arg\ max_{v \in V^*(G) \setminus A_0} \sigma(A_0 \cup \{v\})$
```

Algorithm 4 with the narrowed search space is also approximated with the same technique described in Section 2.2.2 and only differs in two minor details from the original greedy approach in Algorithm 1. The first difference is that the narrowed approach takes a narrowed node set $V^*(G) \subset V(G)$ as its input. This set contains the top $X\%$ nodes from the $L_{GM}$ ordered lists. Then, in each iteration the algorithm adds a single node to $A_0$ so that $\sigma(A_{i-1} \cup v_i)$ is maximal. The second difference is that this vertex can only be chosen from the smaller set $V^*(G)$, not from all of the nodes $V(G)$ in the graph. With less choices in each iteration, the runtime of the whole greedy process can be significantly reduced.

## 3.5 Methodology summary

To summarize, our aim is to create a novel process using influence models and community detection to locate the most influential nodes in directed and weighted networks. The developed method can take any discrete and finite influence model as its input that has a triggering model equivalent. From that point onward, the algorithm is general in that it always uses the same techniques and criteria to detect the above-mentioned communities. To demonstrate the potential of our concept, we choose the well-known influence maximization problem and narrow down its search space with the help of our community detection results. Regarding evaluation, we seek to observe how this new approach affects performance and runtime compared to the original greedy solution.

## 4 Experimental setup

For testing and evaluating our previously described method, we use artificial and real-life benchmark networks. In this section, we give a brief summary of the datasets considered in our research.

### 4.1 Artificial graphs

Before evaluating our approach on real-life networks, we conduct preliminary testing on smaller artificially created graphs. For this purpose, we take the graph generator algorithm proposed by Lancichinetti and Fortunato in (Lancichinetti and Fortunato, 2009). We use the same networks from our previous paper (Hajdu et al., 2021), with the difference that in (Hajdu et al., 2021) the instances were transformed to undirected graphs for methodological reasons. The parameters of the generation are the following:

– $N$: 1000 (number of nodes)
– $d$: 7 (average degree)
– $d_{max}$: 9 (maximum degree)
– $t_1$: $-2$ (exponent for the degree sequence)
– $t_2$: $-1.5$ (exponent for the community size distribution)
– $c_{min}$: 10 (minimum community size)
– $c_{max}$: 50 (maximum community size)
– $o_n$: $0.1, 0.2, \ldots, 0.6$ (fraction of overlapping nodes)
– $o_m$: $2, 3, 4$ (number of memberships of the overlapping nodes)
– $\mu$: $0.1, 0.2, \ldots, 0.6$ (mixing parameter)

These parameters ensure that the resulting networks exhibit a wide range of community structures. Specifically, six distinct values are chosen for the parameter $o_n$ controlling the fraction of overlapping nodes, and three different values for the parameter $o_m$ determining the number of communities a node can belong to. Moreover, we provide six possible values for the mixing parameter $\mu$, which affects the degree of connections within and between communities. Given the possible combinations of these parameters, we end up with $6 * 3 * 6 = 108$ test graphs in total.

By default, this method only outputs graphs without node and edge weights, both of which are essential to test our methodology. To overcome this issue, we generate these weights manually using uniform distribution with the following parameters:

– node weights: $min$: 0.05, $max$: 0.1
– edge weights: $min$: 0, $max$: 0.2

### 4.2 Real-life networks

After evaluating on smaller artificial graphs, we experiment with significantly larger real-life ones. We select three benchmark networks from the Stanford Large Network Dataset Collection (Leskovec and Krevl, 2014), namely the following:

– *cit-HepPh*, an Arxiv High Energy Physics paper citation network (34546 nodes, 421578 edges)

– *soc-Epinions1*, a Who-trusts-whom network of Epinions.com (75879 nodes, 508837 edges)

– *email-EuAll*, an Email network from a EU research institution (265214 nodes, 420045 edges)

We select these real-life networks to showcase a wide variety of node and edge counts along with some potential applications of our algorithm. All of the above-mentioned graphs are directed and unweighted by definition in (Leskovec and Krevl, 2014). To generate weights for the nodes and edges, we use the same procedure as for the artificial graphs described in 4.1.

## 5 Results

As a proof-of-concept approach, we use our novel community detection method to rank the most influential nodes in networks and narrow the search space of the greedy influence maximization algorithm. Here we show how shrinking the possible node set affects overall performance and runtime compared to the greedy method selecting nodes from the whole the graph.

### 5.1 Results on artificial graphs

We begin our evaluation on the networks introduced in Section 4.1. First, we run the influence simulation algorithm described in 3.1 to create the influence graphs that will be used as the input for our community detection method. We proceed with this for all three selected influence models. Regarding the parameters, we select the number of iterations to be $I = 10000$ and the maximum influence distance is set to $d_{max} = 2$. After creating the influence graphs, we draw the following inferences: there is 1. a considerable increase in the densities (number of edges) and average clustering coefficients and 2. a noticable decrease in the diameters and average edge weights compared to the input graphs. These observations can easily be explained with the characteristics of Algorithm 2: in the influence graphs, every directed influence path from each simulation is taken into consideration, thereby creating additional edges compared to the original graph.

Regarding community detection, there are few quantitative metrics that can be analysed without actually using the community values in Algorithm 4. Our primary observation is that different values set for the two hyperparameters *cp* and *ta* substantively affect runtime. Consequently, a timeout threshold is implemented to disregard any executions exceeding this time limit. For the Independent Cascade and Linear Threshold models, we omit around 10% of the results due to long runtime. However, this is a drastically larger 60% when the Only-Listen Once model is used for community detection. Our explanation for this phenomenon is that the average edge weights in the influence graphs for this model are relatively small compared to the weights for the other two models.

After locating overlapping communities in our networks, we can calculate the community values for each node and sort them in descending order. Then, we use this ordering of vertices to narrow the search space of the greedy influence maximization algorithm. In our evaluation, we select $K = 50$ nodes from 1000 (5%) in each graph and run influence simulations with these nodes being initially active. In each simulation, we count how many nodes become active and after every iteration is done, we calculate the average final influence value for the graph. The goal is to achieve as high final influence values as possible.

We compare three different settings in our analysis:

1. *Full greedy.* As a benchmark, we execute the greedy approximation algorithm by Kempe et al. for each input network. This heuristic selects vertices from the entire graph. In each greedy selection, we generate 100 instances, and the final influence value is calculated with 10000 instances.

2. *Narrow greedy.* To demonstrate the effectiveness of our general community detection algorithm, we narrow the search space of the greedy approximation algorithm to the top 20% nodes (regarding the artificially generated graphs, it means 200 vertices in each graph) based on their community values. Then, the greedy heuristic can only use this subset to select $K = 50$ vertices from the graph. In each greedy selection, we generate 100 instances, and the final influence value is calculated with 10000 instances.

3. *Community values.* We also run influence simulations on the top 5% of the nodes (it means 50 vertices for the artificially generated graphs) to see whether the greedy approximation algorithm performs better or community values by themselves can identify the most influential nodes in our networks.

Before actually comparing the performance of the three above mentioned settings, we have to fine-tune the two hyperparameters *cp* and *ta* of our community detection algorithm. We set 5-5 possible parameters for both variables, resulting in 25 community results in total for each graph. We execute the *Narrow greedy* on each of them and take the hyperparameter combination that belongs to the highest average final influence value. Figure 1 shows which combinations turn out to be the best for different models to maximize influence in this setting. For different models, different hyperparameter combinations stand out in terms of effectiveness for narrowing the search space of the greedy algorithm. For comparison, we create the same heatmaps for the *Community values* setting. The resulting plots are shown in Figure 2.

The same conclusions can be derived from both heatmaps regarding further hyperparameter tuning: our results indicate that other parameter intervals should also be considered in future research. For example, trying higher *ta* values for the Linear Threshold or higher *cp* values for the Only-Listen-Once models could balance these charts towards the center. However, our findings - which are presented later in this

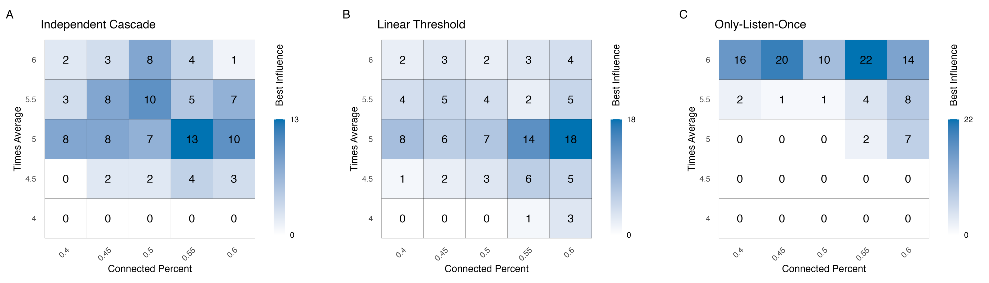

**Fig. 1** Heatmaps for each influence model showing the best community detection hyperparameter combinations to use in the *Narrow greedy* setting for the artificial graphs

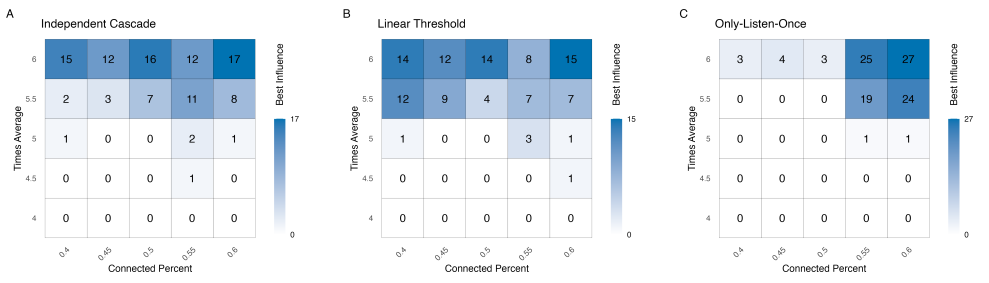

**Fig. 2** Heatmaps for each influence model showing the best community detection hyperparameter combinations to use in the *Community values* setting for the artificial graphs

chapter - suggest that the chosen hyperparameters were capable of producing adequate results.

To compare performance on different models and types of graphs, we create the following 9 plots below in Figure 3. In each row, a certain graph parameter is fixed at a pre-defined value, and the other two parameters are shown on the $x$ and $y$ axis. These are the parameters that are used for artificially generating the graphs described in Section 4.1. The columns represent our three chosen influence models. The three graph generation parameters combined with the three influence models result in $3 * 3 = 9$ plots. The white rectangles represent the lowest, and the red ones the highest final influence values in each plot.

As we can see, there are no distinct patterns that can be explicitly identified. It means that our *Narrow greedy* solution doesn't really depend on the parameters of the input benchmark graphs.

Figure 4 shows a performance comparison of different settings and models. We take the highest final influence values for each graph in each setting, and calculate an average value. Our main observation is that our *Narrow greedy* setting achieves considerably better results than the *Community values* setting, and is just a little worse than the *Full greedy* setting. These results confirm our hypothesis that vertices

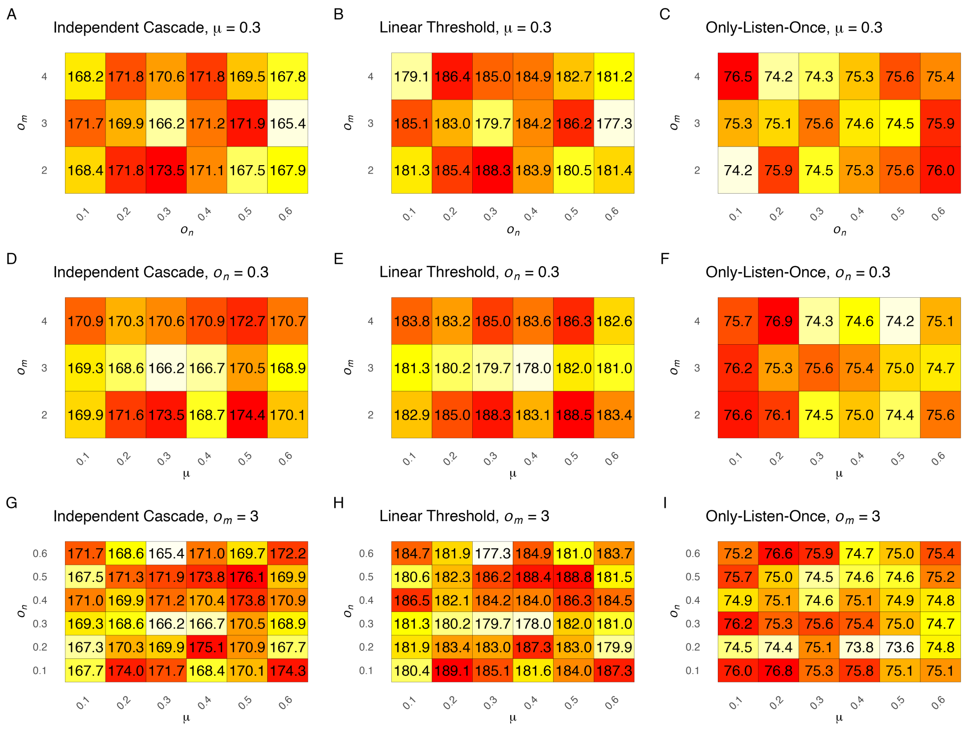

**Fig. 3** Heatmaps for different models and types of artificial graphs showing the highest achieved final influence values in the *Narrow greedy* setting

with significant community roles are of utmost importance in solving the influence maximization problem.

One notable advanatage of the *Narrow greedy* setting using our novel community detection algorithm is its runtime. Figure 5 demonstrates the average seconds required for each step in the pipeline, and their sum compared to the average runtime of the *Full greedy* setting. For two of the three influence models, we managed to save substantial amounts of execution time in exchange for a marginal performance loss.

Another interesting aspect of our testing is the degree of overlap between the best initially active node selections for the *Full greedy* and *Narrow greedy* settings. In Figure 6, the bars labeled as *Artificial (avg)* show the average amount of overlaps for the artificially generated benchmark graphs. For the Independent Cascade and Linear Threshold models, almost 40% of the selected nodes are the same, which is a notable result considering the size of the input graphs. For the Only-Listen-Once model, only 17% of the selected nodes are the same on average. These results are still worth mentioning, but not as outstanding as for the other models.

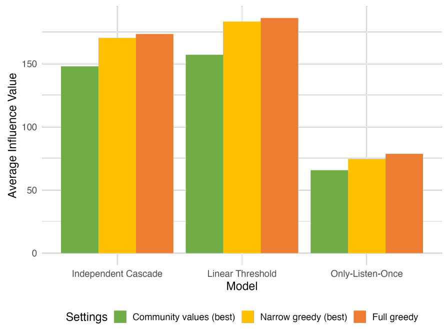

**Fig. 4** Comparison of final influence values across models for the artificial graphs

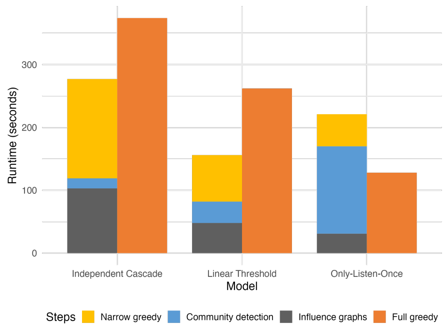

**Fig. 5** Average runtime of different steps across models for the artificial graphs

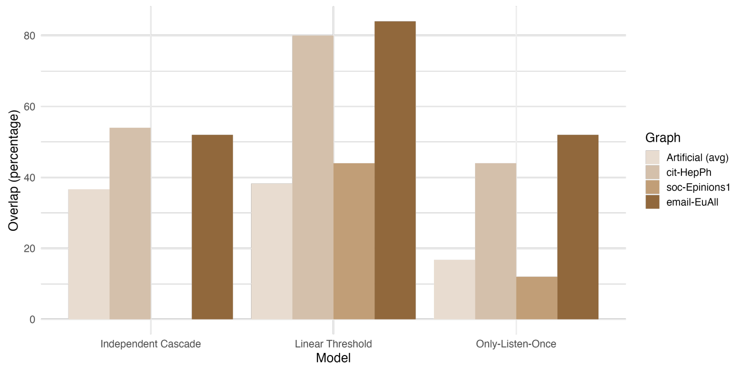

**Fig. 6** Overlap of the best initially active node selections between the *Full greedy* and *Narrow greedy* settings for the artificial and real-life networks

## 5.2 Results on real-life networks

We follow the same evaluation process for our real-life benchmark networks as we used in Section 5.1 above. First, we create infection graphs, then we run community detection on them. Finally, we compare our three pre-defined settings to each other regarding performance and runtime: *Full greedy*, *Narrow greedy* and *Community values*. The only difference for these bigger networks is the number of instances to simulate influence spread. In each greedy selection, we generate only 10 instances, and the final influence value is calculated with 100 instances. Selecting such small values for these parameters is due to runtime considerations.

To create the influence graphs, we use exactly the same parameters as we set for the smaller artificial networks. The number of iterations is $I = 10000$ and the maximum influence distance is set to $d_{max} = 2$ for each benchmark graph. Similar characteristics can be discovered for the real-life networks if we compare our results to the artificially generated ones: the influence graphs are notably denser and their average edge weights are considerably smaller compared to the input networks. One difference, however, is that none of the chosen real-life input graphs are strongly connected, making all of the influence graphs falling into multiple components as well.

For community detection, hyperparameter fine-tuning in this scenario is even more crucial thanks to the magnitude of nodes and edges in our chosen graphs. Instead of implementing timeouts with a threshold value, a more drastic heuristic is considered. When expanding the community initiatives, only a predefined number of neighbors are tested randomly. This way, some communities might not be found instantly, but we hope that the most important structures will still be enough for our solution. For the smallest *cit-HepPh* network, this $n_{max}$ parameter is set to 50, and for the other two bigger ones, namely *soc-Epinions1* and *email-EuAll*, $n_{max} = 25$ is used. Still, few quantitative metrics can be analysed here without using our communities for narrowing the search space in Algorithm 4. In our analysis regarding performance and runtime, the settings are the same as defined in Section 5.1 earlier. We want to highlight that in the *Narrow greedy* setting, we still use 20% of nodes to narrow the search space, which means different values for each real-life network (based on their number of nodes).

After the overlapping communities are found, we can calculate the community values for each node in our networks and sort them in descending order. This ordering of vertices is then used to narrow the search space of the greedy influence maximization algorithm. We try to select $K = 50$ nodes in each graph and run influence simulations with these nodes being initially active. At the end of each simulation, the number of active nodes is counted. After every iteration is done, we calculate an average final influence value for the graph. The goal of the initial vertex selection is to achieve as high final influence values as possible at the end of the whole process.

Fine-tuning hyperparameters *cp* and *ta* of our community detection algorithm is also an important challenge. Instead of 5-5 possible values for both variables, we experiment with 3-3 possibilities, resulting in 9 community results in total for each graph. Then, we execute the *Narrow greedy* on each of the community results and take the hyperparameter combination that belongs to the highest average final influence value. We proceed similalrly with the *Community values* setting. Tables 1 and 2 show which combinations turn out to be the best for different models to maximize influence in these settings. It turns out that very distinct choices of hyperparameters produce the best final infection results in each setting, there are huge differences even between the settings for the same graphs.

Figures 7, 9 and 11 compare the final influence values of our defined settings, each graph separately. For these bigger networks, the *Community values* setting performs noticeably worse compared to the other two settings. This was not the case for the artificially generated small graphs, the results are closer to each other in Section 5.1. If we look at the *Narrow greedy* setting, the results emphasize our intuition again, that is significant community roles should be taken into consideration when selecting initially active nodes in the influence maximization problem. Our finding to highlight is that the *Narrow greedy* produces way better results in most cases compared to the *Community values* setting, while also being close to the *Full greedy* results. In some cases, it delivers even better results than the *Full greedy* in our evaluation. The explanation for this phenomenon is very simple: while the original greedy method

**Table 1** Best parameter combinations for each real-life network and influence model for the *Narrow greedy* setting

| Graphs | Independent Cascade | Linear Threshold | Only-Listen-Once |
|---|---|---|---|
| cit-HepPh | $(cp = 0.75,\ ta = 8)$ | $(cp = 0.75,\ ta = 17.5)$ | $(cp = 0.75,\ ta = 7)$ |
| soc-Epinions1 | $(cp = 0.75,\ ta = 36)$ | $(cp = 0.75,\ ta = 40.5)$ | $(cp = 0.80,\ ta = 38)$ |
| email-EuAll | $(cp = 0.7,\ ta = 15)$ | $(cp = 0.7,\ ta = 45.5)$ | $(cp = 0.75,\ ta = 24)$ |

**Table 2** Best parameter combinations for each real-life network and influence model for the *Community values* setting

| Graphs | Independent Cascade | Linear Threshold | Only-Listen-Once |
|---|---|---|---|
| cit-HepPh | $(cp = 0.75,\ ta = 8)$ | $(cp = 0.8,\ ta = 18)$ | $(cp = 0.8,\ ta = 8)$ |
| soc-Epinions1 | $(cp = 0.7,\ ta = 35)$ | $(cp = 0.8,\ ta = 40.5)$ | $(cp = 0.75,\ ta = 39)$ |
| email-EuAll | $(cp = 0.75,\ ta = 15.5)$ | $(cp = 0.8,\ ta = 45)$ | $(cp = 0.75,\ ta = 24.5)$ |

guarantees at least 63% approximation of the optimal solution, there is nothing that prevents other algorithms to produce better results. In these situations, it turns out that the original greedy method is not able to find the optimal solution, but our algorithm approximates it better.

Fortunately, our runtime statistics are also promising. Figures 8, 10 and 12 showcase that extensive amounts of runtimes can be saved with the *Narrow greedy* setting compared to the *Full greedy* setting. In our testing, we observe a positive correlation between the size and density of the networks and the corresponding time savings. For some models and benchmark graphs, around 90% of the runtime can be preserved with marginal, or even no performance loss at all.

In this section, we also have to refer back to Figure 6 where we showcase the degree of overlap between the best initially active node selections for the *Full greedy* and *Narrow greedy* settings. For two of our three selected graphs, namely *cit-HepPh* and *email-EuAll*, quite high overlap percentages can be observed, ranging from $45\% - 85\%$. These values are even higher than the ones for the smaller artificial networks. It means that in some cases, our community detection and search space narrowing solution combined selects almost the same nodes as the original greedy influence maximization method, but with considerably less runtime. We can also see that for the *soc-Epinions1* network and the Independent Cascade model, none of the selected nodes were the same for the two settings (0% overlap). Entirely different node sets resulted in almost the same final influence values, which might be the consequence of the structure of this certain network.

## 6 Conclusions

The results presented in this study demonstrate the efficiency and versatility of our proposed generalized community detection algorithm and its application for the influence maximization problem. By giving the outputs of the influence simulations to our community detection process, we assign community values to each node in the selected benchmark networks. Then, we take the *Narrow greedy* setting developed by Hajdu et

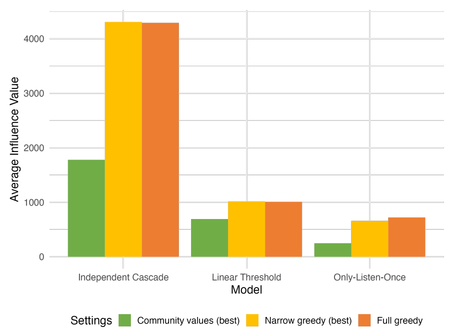

**Fig. 7** Comparison of final influence values across models for the cit-HepPh real-life graph

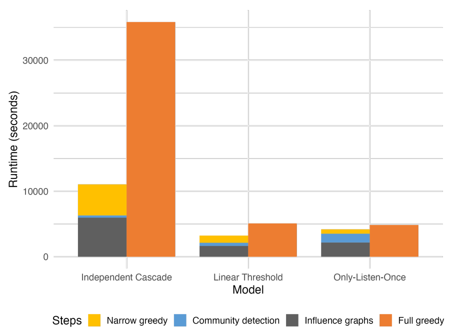

**Fig. 8** Average runtime of different steps across models for the cit-HepPh real-life graph

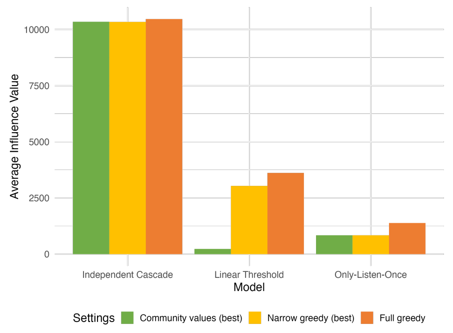

**Fig. 9** Comparison of final influence values across models for the soc-Epinions1 real-life graph

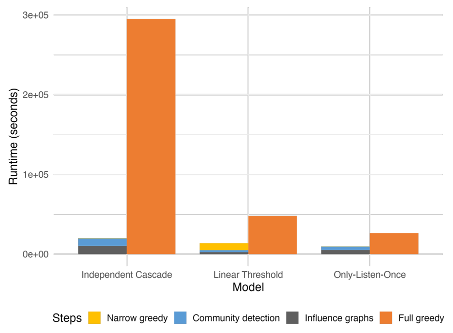

**Fig. 10** Average runtime of different steps across models for the soc-Epinions1 real-life graph

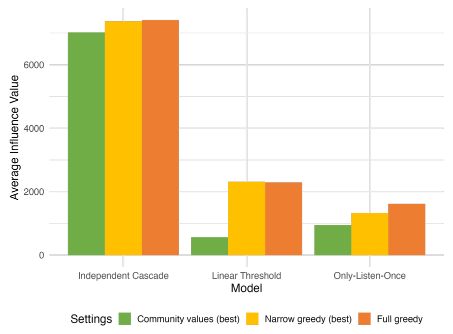

**Fig. 11** Comparison of final influence values across models for the email-EuAll real-life graph

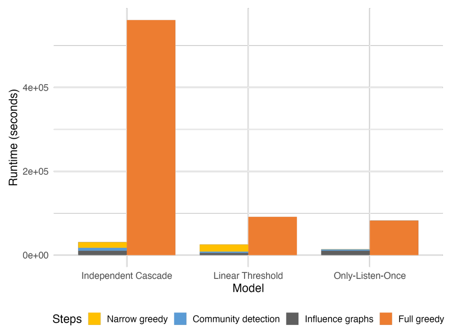

**Fig. 12** Average runtime of different steps across models for the email-EuAll real-life graph

al. that narrows the search space of the original greedy influence maximization algorithm. This approach significantly reduces computational costs, and has also proven effective across various network types, influence models, and benchmark datasets.

We demonstrated that our *Narrow greedy* setting achieves results comparable to the traditional *Full greedy* setting while significantly reducing computational overhead on both artificially generated and real-life networks. This efficiency gain is particularly critical for large-scale networks, as seen in the *cit-HepPh*, *soc-Epinions1* and *email-EuAll* datasets. In some cases, runtime was reduced by up to 90% with minimal performance trade-offs.

Our study highlights the critical role of community structure in influence maximization. By leveraging community values derived from overlapping community detection, we prove that nodes with significant community roles are highly influential in spreading processes. The *Narrow greedy* setting consistently produced influence values comparable to the original *Full greedy* algorithm, achieving this with substantially less runtime.

Compared to existing methods, our approach offers a unique advantage: it eliminates the dependency of specific influence models, allowing for a broader range of applications. While prior studies focused on predefined models or specific community detection techniques, our method provides a generalized framework capable of adapting to various network structures and dynamics. This adaptability was demonstrated in its consistent performance across with three influence models: the Independent Cascade, the Linear Threshold and Only-Listen-Once models.

Despite its strengths, our approach has certain limitations. First, the reliance on hyperparameter tuning for the variables *cp* and *ta* introduces an element of subjectivity and additional computational overhead during the fine-tuning process. Second, the Only-Listen-Once influence model presented runtime challenges due to its lower average edge weights, highlighting a potential area for optimization.

In terms of future work, expanding the range of compatible influence models can help capture even more diverse dynamics of influence propagation within networks. By refining our proposed methodology, it will be possible to develop more robust tools capable of handling the complexities of real-life scenarios, extending the scalability, precision, and practicality of the approaches introduced in our study.

## Acknowledgements

Máté Vass has been supported by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

Miklós Krész acknowledges the support of the BioLOG project: he is grateful for the support of the National Center of Science (NCN) through grant DEC-2020/39/I/HS4/03533, the Slovenian Research and Innovation Agency (ARIS) through grant N1-0223 and the Austrian Science Fund (FWF) through grant I 5443-N. He has been also supported by the research program CogniCom (0013103) at the University of Primorska.

## Declarations

### Author contributions

The authors contributed to this work as follows:

- **Conceptualization:** Máté Vass, Miklós Krész, László Hajdu
- **Data curation:** Máté Vass
- **Formal analysis:** Miklós Krész, András Bóta
- **Funding acquisition:** Miklós Krész, László Hajdu
- **Investigation:** Miklós Krész, András Bóta
- **Methodology:** Máté Vass, László Hajdu
- **Project administration:** Miklós Krész, András Bóta
- **Resources:** Miklós Krész, András Bóta
- **Software:** Máté Vass
- **Supervision:** Miklós Krész, László Hajdu, András Bóta
- **Validation:** Máté Vass, Miklós Krész, András Bóta
- **Visualization:** Máté Vass
- **Writing – original draft:** Máté Vass, András Bóta
- **Writing – review & editing:** Máté Vass, Miklós Krész, András Bóta

### Competing interests

The authors declare no competing interests.

## References

Brauer, F., Castillo-Chavez, C.: Mathematical Models in Population Biology and Epidemiology. Texts in Applied Mathematics. Springer, New York, NY (2001). https://doi.org/10.1007/978-1-4614-1686-9

Bóta, A., Csernenszky, A., Győrffy, L., Kovács, G., Krész, M., Pluhár, A.: Applications of the inverse infection problem on bank transaction networks. Central European Journal of Operations Research **23**, 345–356 (2015)

Bóta, A., Kovács, L.: The community structure of word association graphs. In: Proceedings of the 9th International Conference on Applied Informatics, vol. 1, pp. 113–120 (2014). https://doi.org/10.14794/ICAI.9.2014.1.113

Borgatti, S.P., Mehra, A., Brass, D.J., Labianca, G.: Network analysis in the social sciences. Science **323**(5916), 892–895 (2009) https://doi.org/10.1126/science.1165821

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. KDD '10, pp. 1029–1038. Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1835804.1835934

Chen, Y.-C., Zhu, W.-Y., Peng, W.-C., Lee, W.-C., Lee, S.-Y.: CIM: Community-based influence maximization in social networks. ACM Transactions on Intelligent Systems and Technology **5**(2) (2014) https://doi.org/10.1145/2532549

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. KDD '01, pp. 57–66. Association for Computing Machinery, New York, NY (2001). https://doi.org/10.1145/502512.502525

Fortunato, S.: Community detection in graphs. Physics Reports **486**(3), 75–174 (2010) https://doi.org/10.1016/j.physrep.2009.11.002

Granovetter, M.: Threshold models of collective behavior. American journal of sociology **83**(6), 1420–1443 (1978) https://doi.org/10.1086/226707

Hajdu, L., Krész, M.: Temporal network analytics for fraud detection in the banking sector. In: ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium, pp. 145–157. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55814-7_12

Hajdu, L., Krész, M., Bóta, A.: Evaluating the role of community detection in improving influence maximization heuristics. Social Network Analysis and Mining **11**(91) (2021) https://doi.org/10.1007/s13278-021-00804-5

Kovács, L., Bóta, A., Hajdu, L., Krész, M.: Networks in the mind–what communities reveal about the structure of the lexicon. Open linguistics **7**(1), 181–199 (2021)

Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. Theory of Computing **11**(4), 105–147 (2015) https://doi.org/10.4086/toc.2015.v011a004

Kermack, W.O., McKendrick, A.G.: A contribution to the mathematical theory of epidemics. Proceedings of the Royal Society of London. Series A, Containing papers of a mathematical and physical character **115**(772), 700–721 (1927) https://doi.org/10.1098/rspa.1927.0118

Krész, M., Pluhár, A.: Economic network analysis based on infection models. In: Encyclopedia of Social Network Analysis and Mining, pp. 707–715. Springer, New York, NY (2018). https://doi.org/10.1007/978-1-4939-7131-2_29

Kovács, I.A., Palotai, R., Szalay, M.S., Csermely, P.: Community landscapes: An integrative approach to determine overlapping network module hierarchy, identify key nodes and predict network dynamics. PLOS ONE **5**(9), 1–14 (2010) https://doi.org/10.1371/journal.pone.0012528

Lancichinetti, A., Fortunato, S.: Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Phys. Rev. E **80**(1) (2009) https://doi.org/10.1103/PhysRevE.80.016118

Li, Y., Fan, J., Wang, Y., Tan, K.-L.: Influence maximization on social graphs: A survey. IEEE Transactions on Knowledge and Data Engineering **30**(10), 1852–1872 (2018) https://doi.org/10.1109/TKDE.2018.2807843

Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection (2014). https://snap.stanford.edu/data/

Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**, 026113 (2004) https://doi.org/10.1103/PhysRevE.69.026113

Rajeh, S., Cherifi, H.: Ranking influential nodes in complex networks with community structure. PLOS ONE **17**(8), 1–26 (2022) https://doi.org/10.1371/journal.pone.0273610

Wasserman, S., Faust, K.: Social network analysis: Methods and applications. Structural Analysis in the Social Sciences, vol. 8. Cambridge University Press, Cambridge (1994). https://doi.org/10.1017/CBO9780511815478