information

Article

A New Algorithm Framework for the Influence Maximization Problem Using Graph Clustering

Agostinho Agra [1,*,†] and Jose Maria Samuco [2,†]

[1] Department of Mathematics, University of Aveiro, 3810-193 Aveiro, Portugal
[2] Center for Research & Development in Mathematics and Applications, University of Aveiro, 3810-193 Aveiro, Portugal; jsamuco@ua.pt
* Correspondence: aagra@ua.pt; Tel.: +351-234370670
† These authors contributed equally to this work.

Abstract: Given a social network modelled by a graph, the goal of the influence maximization problem is to find k vertices that maximize the number of active vertices through a process of diffusion. For this diffusion, the linear threshold model is considered. A new algorithm, called ClusterGreedy, is proposed to solve the influence maximization problem. The ClusterGreedy algorithm creates a partition of the original set of nodes into small subsets (the clusters), applies the SimpleGreedy algorithm to the subgraphs induced by each subset of nodes, and obtains the seed set from a combination of the seed set of each cluster by solving an integer linear program. This algorithm is further improved by exploring the submodularity property of the diffusion function. Experimental results show that the ClusterGreedy algorithm provides, on average, higher influence spread and lower running times than the SimpleGreedy algorithm on Watts–Strogatz random graphs.

Keywords: cluster; influence maximization; greedy algorithm; linking set; linear threshold model

check for updates

Citation: Agra, A.; Samuco, J.M. A New Algorithm Framework for Influence Maximization Problem Using Graph Clustering. *Information* **2024**, *15*, 112. https://doi.org/10.3390/info15020112

Academic Editor: Christos Gogos

Received: 5 January 2024
Revised: 6 February 2024
Accepted: 7 February 2024
Published: 14 February 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The influence maximization (IM) problem has attracted many researchers in the last few decades, especially due to its applications in various situations, such as viral marketing [1–3], terrorist attack prevention [4], and network control [5]. Viral marketing has proved to be more interesting for researchers. Chen et al. [6] justify the strong motivation for studying information and influence diffusion models with viral marketing. That is, if a company needs to introduce a new product in the market through word-of-mouth effects in a social network, it must select influencers in the network and convince them to adopt the new product. The goal is that the selected influencers will generate a very big cascade effect in the network, driving other people to use the new product. Thus, diffusion is the transmission of information among persons, such as the transmission of viruses.

The literature on IM is very extensive. Recent studies range from using reinforcement learning to solve IM problems [7] to investigating the equilibrium between information influence and the cocoon effect [8]. Several surveys on IM have been published; see [9–14].

The IM problem is a discrete optimization problem that was enunciated by Kempe et al. [15]. Given a social network modelled by a directed graph $G = (V, E)$ with influence weights on edges, in the influence maximization problem, we are interested in finding the k (pre-defined parameter) nodes of the network which maximize the spread of the influence once activated under a propagation model. The influence diffusion process occurs in discrete time steps. Here, for the propagation, we consider the linear threshold model, which is classified as a progressive diffusion model, since activated nodes cannot be deactivated [11].

The IM problem is NP-hard under the linear threshold model (and under other diffusion models such as the Independent Cascade model); see [15] for details. Therefore, heuristics play an important role in the solution techniques. As stated in [11], "existing

works focus on approximate solutions, and a keystone of these algorithmic IM studies is the greedy framework". Among these heuristics, probably the most well-known is the SimpleGreedy algorithm proposed by Kempe et al. [15]. This algorithm employs a greedy approach by selecting the node with the highest marginal gain over the network and including it in the seed set at each iteration.

In this paper, we propose a new algorithm framework based on greedy approaches for the IM problem. This framework consists of creating a partition of the graph nodes into clusters, solving the IM problem within each cluster to obtain a set of candidate nodes from each cluster, and choosing the seed set from those candidate nodes. To solve the IM problem in each cluster, we use a greedy approach. Following the taxonomy proposed in [11], the proposed algorithms are classified into simulation-based methods, since Monte-Carlo simulations are employed for evaluating the influence spread of the selected seed set.

First, we introduce the ClusterGreedy algorithm for the IM problem under the LT diffusion model. The ClusterGreedy creates a partition of the given network and implements the SimpleGreedy in the smaller networks resulting from that partition. Then, it solves a subproblem to determine the seed set of the original network from the seed set obtained for each of the smaller networks. With this approach, we solve, using a greedy approach, more problems and, consequently, more simulations (one for each iteration of the SimpleGreedy). However, each problem is solved in a smaller network, which is important in order to reduce the running times of the simulations, since this step is computationally very expensive and depends on the size of the network. By exploring the submodularity property of the diffusion function, we improve the ClusterGreedy by reducing the number of iterations that need to be performed. The resulting algorithm, called Improved ClusterGreedy, solves a few more iterations than the SimpleGreedy; however, all simulations run in smaller networks resulting from the clustering operation. The computational experiments show that this improved algorithm allows for a reduction in the runtime to 25% when we compare it with the algorithm proposed by Kempe et al. [15], under Watts–Strogatz random graphs and 54% in the three datasets selected. One important aspect in our approach is that by splitting the network into smaller networks and analysing each network separately, we lose the interactions between nodes in different clusters. This may be observed as a disadvantage. However, as reported in the computational section, we observe that there may be an advantage in that operation, since analysing each network separately to find the candidate solutions for the seed set in each cluster and choosing the seed set later by combining the different sets of candidates helps in avoiding the myopic behaviour of the greedy approach.

This paper is structured as follows: In Section 2, we review the literature most related to our work. The IM problem, the diffusion model, and the algorithm proposed by Kempe et al. [15] are described in Section 3. In Section 4, the ClusterGreedy and the Improved ClusterGreedy algorithms are introduced, and the main components, namely the Markov cluster algorithm and the subproblem to determine the final seed set from the seed of each smaller network, are explained. Experimental results are presented in Section 5. Finally, Section 6 summarizes the main conclusions of this paper.

2. Related Literature

The literature for IM is quite vast. In this section, we review the literature that is most related to our work. In particular, we focus on works related to greedy approaches.

Kempe et al. [15] introduced the SimpleGreedy algorithm to solve the problem of maximizing the influence. This algorithm uses the concept of marginal gain, which is the influence spread increase obtained by adding a node to the seed set. The SimpleGreedy algorithm employs a greedy approach by selecting the node with the highest marginal gain over the network and including it in the seed set at each iteration. This process is repeated k times [16]. Given the spread function σ, the marginal gain of a given node u can be computed by $\sigma(S \cup \{u\}) - \sigma(S)$, which Kempe et al. [15] estimated with Monte Carlo simulations. This algorithm is known to have a poor running time performance.

That inefficiency has motivated several alternative approaches and motivated our study to reduce the computation time.

The CELF (Cost-Effective Lazy Forward) algorithm was introduced by Leskovec et al. [17]. In the first round, the algorithm calculates an estimate of the influence spread of each node using Monte Carlo simulations. However, since the marginal gain in each iteration is not superior to its marginal gain of previous iterations, they reduced the number of Monte Carlo simulations [18]. CELF was presented as being 700 times faster than the SimpleGreedy. Goyal et al. [19] introduced CELF++ and described it as being 35–55% faster than CELF.

The NewGreedy algorithm was introduced by Chen et al. [20]. This algorithm improved the one proposed by Kempe et al. [15]. NewGreedy reuses the results of the Monte Carlo simulations from previous iterations to calculate the influence spread for each node in the same iteration [18].

Cheng et al. [21] introduced the StaticGreedy algorithm. They proposed a solution to the warranty submodularity and monotonicity in an effective way. Instead of generating a significant quantity of Monte Carlo simulations in each iteration, a not-particularly large number of Monte Carlo snapshots are generated at the start and then reused in all subsequent iterations. This approach guarantees that the estimated influence spreads of any seed set remain consistent over successive iterations. It also ensures that the features of submodularity and monotonicity are upheld.

The SIMPATH algorithm was introduced by Goyal et al. [22]. The algorithm uses simpath-spread and is based on the assumption that the influence propagated from a node can be calculated as the sum of the weights of all arcs from that node. When the precedent idea is combined with the CELF algorithm, the resulting algorithm can be applied to solve the IM problem.

Cluster algorithms for graphs are also known as community detection algorithms for networks. Community-based algorithms detect communities on social networks and solve the IM problem through the structure of the given network data; see [23–26]. Rahimkani et al. [23] consider an approach that first identifies the communities on social networks and then creates a new network based on those communities. In the new network, each node represents a community. Then, the most central nodes of the new network are selected based on the measures of betweenness and centrality. Each node in the new network includes nodes in the corresponding community. Therefore, a number of nodes, proportional to the size of the community, was selected based on the degree centrality to form the candidate set. The k most influential nodes are then selected from the set based on the betweenness centrality measure. Community-based algorithms should not be confused with our clustering approach. Although such algorithms use clusters (corresponding to communities), their approach is quite different from ours. We split the graph into clusters and consider them separately. In the Community-based algorithms, the clusters are simplified by choosing a subset of nodes that represents the cluster.

Ghayour-Baghbani et al. [27] propose an efficient method using linear programming (LP) for the IM problem in the linear threshold propagation model. The method works in three steps: first, the sparse matrix multiplication is used to estimate the influence graph from the social network graph; second, the influence graph is used to model the IM problem in the LT model as a linear program. Finally, the candidate is created. The linear program's output is seeded using a randomized rounding approach.

Some methods solve the IM problem by LP in the LT model, but either their runtime is higher than the greedy algorithm or they tend to produce worse solutions (the spread of the produced seed set is less than the greedy algorithm [28–31]).

3. Preliminary

In this section, we formulate the IM problem and provide an overview of the linear threshold model and the SimpleGreedy algorithm. Table 1 provides the notation used in this paper.

Table 1. Notation used.

Notation	Description
$G = (V, E)$	Directed graph G with vertex set V and edge set E
n	Number of vertices in the graph G
m	Number of clusters
S	Seed set
S^t	Set of activated nodes at step t of SimpleGeedy algorithm
S_j	Set of activated nodes in cluster j
S_j^t	Set of activated nodes in cluster j at step t of SimpleGeedy algorithm
$\|A\|$	Cardinality of set A
k	Number of seeds to be selected ($k =\mid S \mid$)
r	Number of Monte Carlo simulations
$\sigma(S)$	The total number of nodes activated by the set of nodes S
$N^{\text{in}}(v)$	Set of vertices u of directed graph $G = (V, E)$ such that the arc $(u, v) \in E$

Given a graph $G = (V, E)$, a stochastic diffusion model on G, and a budget k, the IM problem is the stochastic optimization problem of finding a set $S \subseteq V$ with $|S| \leq k$, such that the influence spread of $S, \sigma(S)$, under the given stochastic diffusion model is maximized. That is, determine $S^* \subseteq V$, such that

$$S^* = \underset{S \subseteq V, |S|=k}{\operatorname{argmax}} \sigma(S). \tag{1}$$

Regarding the problem formulation above, it should be noted that finding a group of k nodes with the greatest combined influence is not the same as finding the top k nodes with the greatest individual influences. It makes sense to identify two top influencers who both have a significant impact on the same group of individuals as top influencers, but only one of them should be chosen as a seed to maximize the influence.

One of the two primary diffusion models, the independent cascade (IC) model, can be used to explain simple contagions where activations might result from a single source, like the spread of viruses or information. However, in several circumstances, people need to be exposed to various independent sources for their behavior to change. The linear threshold (LT) model was introduced by Kemp et al. [15] as a stochastic diffusion model to represent this kind of behavior.

In this work, the LT diffusion model is considered. Each arc $(u, v) \in E$ in the LT model has an effect weight $w_{uv} \in [0, 1]$, which indicates how likely u influences v. The weights are normalized such that $\sum_{u \in N^{in}(v)} w_{uv} \leq 1$, for all $v \in V$ [6]. Formally, we have the following definition [6]:

Definition 1 (Linear Threshold Model)**.** *Given a graph $G = (V, E)$, with influence weights w_a, $a \in E$ assigned to the arcs, and a seed set S^0, the LT model creates the active sets S^t, for all $t \geq 1$ repeatedly, using the following randomized operation procedure: first, a threshold value θ_v is created for each $v \in V$ using a uniform distribution ranging from 0 to 1. For each time step $t \geq 1$, the set S^t is initialized to the same value as S^{t-1}. For each node v in the set $V \setminus S^{t-1}$ that is not active, if the combined weight of the arcs from its active neighbors is equal to or greater than θ_v, denoted as*

$$\sum_{u \in S^{t-1} \cap N^{in}(v)} w_{(u,v)} \geq \theta_v,$$

then node v is included in S^t (i.e., node v becomes active at time t).

An important property in IM is submodularity,

Definition 2 (Submodularity [6])**.** *Let V be a finite set, and denote by 2^V the power set of V. A function $f : 2^V \longrightarrow \mathbb{R}$ is called submodular if, for each $S \subseteq T \subseteq V$ and any $v \in V \setminus T$, we have*

$$f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T) \tag{2}$$

In the context of IM, this property says that the marginal gain obtained by adding a node to the seed set decreases with the increase in the seed set.

Kempe et al. [15] proved the following theorem:

Theorem 1. *The influence spread function $\sigma(\cdot)$ in the linear threshold model is monotone and submodular.*

Based on the monotony, non-negativity and submodularity properties of the spread function σ, Kempe et al. [15] devised an algorithm to address the influence maximization problem—the SimpleGreedy algorithm (Algorithm 1). The algorithm iteratively selects the node with the highest marginal gain and includes it in the seed set. The algorithm stops when the seed set has k elements. Under the LT model (and IC model), computing exact marginal gains, i.e., the exact expected spread, is #P−hard [6].

Since the SimpleGreedy algorithm needs to compute the exact marginal gains, or the exact expected spread (Line 3 of Algorithm 1), Kempe et al. proposed to estimate them by Monte Carlo simulations, with r runs. With this change, we obtain Algorithm 2.

Algorithm 1 Simple Greedy (k, σ)

Input: k: cardinality of the returning set; σ: monotone and submodular set function
Output: seed set S

1: initialize $S \leftarrow \emptyset$
2: **for** $i = 1$ to k **do**
3: $\quad u \leftarrow \underset{w \in V \setminus S}{\text{argmax}}(\sigma(S \cup \{w\}) - \sigma(S))$
4: $\quad S \leftarrow S \cup \{u\}$
5: **end for**
6: **return** S

Algorithm 2 SimpleGreedy (G, k)

Input: $G = (V, E)$: social graph; arc weights $w_e, e \in E$; k: cardinality of the returning set
Output: seed set S

1: **function** MC-ESTIMATE(S, G)
2: $\quad$ counter $\leftarrow 0$
3: $\quad$ **for** $j = 1$ to r **do**
4: $\quad\quad$ Perform a simulation of the diffusion process on the graph G using the seed set S.
5: $\quad\quad$ $n_a \leftarrow$ The number of active nodes once the diffusion concludes.
6: $\quad\quad$ counter $\leftarrow$ counter $+ n_a$
7: $\quad$ **end for**
8: **return** counter $/r$
9: **end function**
10: initialize $S \leftarrow \emptyset$
11: **for** $i = 1$ to k **do**
12: $\quad u \leftarrow \underset{w \in V \setminus S}{\text{argmax}}$ MC-ESTIMATE$(S \cup \{w\}, G)$
13: $\quad S \leftarrow S \cup \{u\}$
14: **end for**
15: **return** S

Notice that both Algorithms 1 and 2 generate a sequence of sets $S^0, S^1, \ldots, S^k$ and the corresponding estimative of spread. This observation will be used later. Algorithm 2 has time complexity $\mathcal{O}(krne)$, where n is the number of vertices and e is the number of edges.

4. Cluster Greedy Algorithm

This paper presents a novel algorithm named ClusterGreedy (Algorithm 3) that combines the SimpleGreedy (Algorithm 2) with graph clustering. The main idea is to create a partition of the set of nodes using clustering techniques. Each subset of nodes obtained from the partition induces a smaller subgraph. Then, we apply the SimpleGreedy to each subgraph to generate a set of candidate nodes for the seed set. Then, a subproblem called the linking set problem is solved to obtain the best seed set from the combination of the sets of candidate nodes generated for each subgraph. We present an overview of the ClusterGreedy algorithm in Figure 1. The purpose of this approach is twofold. On one hand, we aim to reduce the running times since the evaluation of the marginal gains (which is the most expensive task from the computational point of view) is conducted on smaller subgraphs. On the other hand, as the SimpleGreedy approach is heuristic and known to be myopic, by leaving to a second step, the choice of the final seed set may help to obtain better solutions.

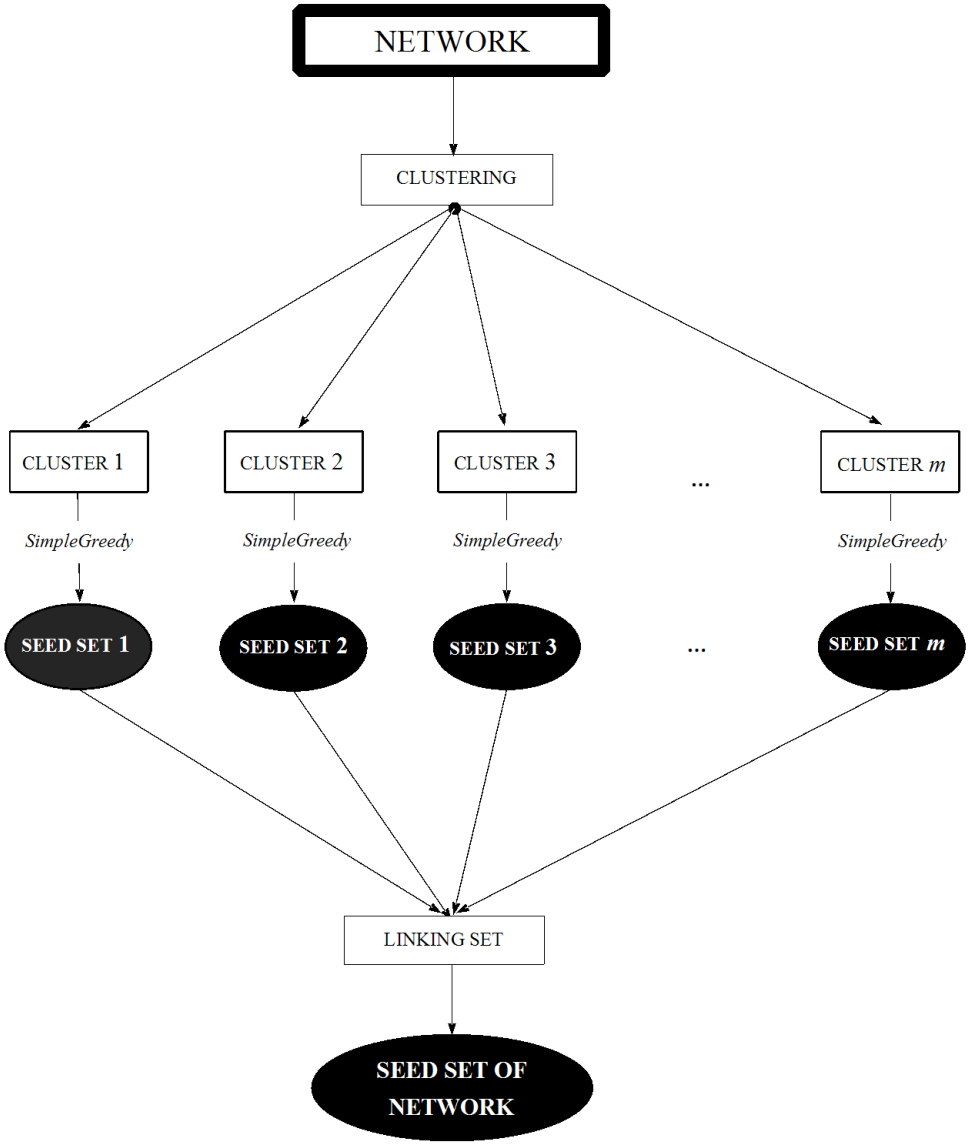

Figure 1. Scheme of Algorithm 3.

Next, the three main components of the proposed algorithm are presented. First, the ClusterGreedy algorithm is described, then the subproblem to combine the candidate sets of nodes is modelled, and, finally, the full algorithm is detailed.

Algorithm 3 ClusterGreedy (G, k).

Input: $G = (V, E)$: social graph; arc weights $w_e, e \in E$; k: cardinality of the returning set
Output: seed set S

generate the clusters of graph G using the MCL algorithm
set m to the number of clusters of G
for $j = 1$ to m **do**
    $G_j \leftarrow$ the subgraph induced by the set of nodes in cluster j
    $S_j =$SimpleGreedy(G_j, k) (Algorithm 2)
end for
compute $c_{ij} \geq 0$, the value of spread within cluster j, with seed set S_j^i
solve the linear integer program defined by Equations (3)–(6)
Let $\overline{x}$ be the optimal solution obtained
for $i = 1$ to k **do**
    for $j = 1$ to m **do**
        if $\overline{x}_{ij} = 1$ **then**
            $S \leftarrow S_j^i$
        end if
    end for
end for
return S

4.1. Markov Cluster Algorithm

Clustering aims to partition a given data set into clusters where the elements inside each cluster exhibit a similarity or connectivity according to a preset criterion [32]. However, there exist graphs that lack a naturally clustered structure. Nevertheless, a clustering algorithm produces a clustering for every input graph. If the graph's structure is completely uniform, with edges evenly distributed across the nodes, the clustering produced by any technique will be somewhat arbitrary. This constitutes no major issue for our approach since the main goal is to create several smaller graphs.

In our case, the clustering is performed using the Markov clustering (MCL) algorithm [33]. The MCL algorithm emulates stochastic walks on the underlying interaction network, employing two alternating operations: expansion and inflation [33]. Firstly, loops are incorporated into the provided graph. The highest weight among the weights of the edges connecting to the vertex determines the weight of each loop. Next, the graph is converted into a stochastic "Markov" matrix, denoted as A, which captures the probabilities of transitioning between each pair of vertices. The probability of a random walk of length λ between any two vertices is then determined using a method called expansion, which involves calculating A^λ. The resulting values are used for forming the clusters in such a way that frequent random walks on the graph occur between nodes belonging to the same cluster.

In contrast to other clustering algorithms, such as K-means clustering, the MCL algorithm does not require the number of groups to be predetermined. Nevertheless, the value of this number can be indirectly regulated by manipulating the inflation parameter. The level of inflation directly impacts the granularity or resolution of the clustering result. Lower inflation values (1.3, 1.4) result in fewer and larger clusters, whereas higher inflation values (5, 6) lead to more and smaller clusters [34,35].

4.2. Linking Set

In order to obtain the seed set for the original network from the candidate seed sets obtained for each cluster, it is necessary to solve a subproblem. Given m clusters, the aim is to find the best seed set of k nodes selected among the m candidate seed sets obtained for each cluster.

To model this subproblem as an integer linear model, let us consider non-negative weights c_{ij}, for all $i \in K = \{1, \ldots, k\}$ and all $j \in J = \{1, \ldots, m\}$, which represent the

expected spread in cluster j, considering the seed set of cardinality i, denoted by S_j^i. Also, consider the binary variables x_{ij} for all $i \in K$ and $j \in J$ indicating whether we select i nodes from the cluster j. The subproblem can be formulated as follows:

$$\max \sum_{j=1}^{m} \sum_{i=1}^{k} c_{ij} x_{ij}, \tag{3}$$

$$\sum_{j=1}^{m} \sum_{i=1}^{k} i x_{ij} = k, \tag{4}$$

$$\sum_{i=1}^{k} x_{ij} \leq 1, \quad j \in J, \tag{5}$$

$$x_{ij} \in \{0,1\}, \quad i \in K, \quad j \in J \tag{6}$$

The objective function aims to maximize the estimated spread. Equation (4) means that k nodes are chosen. On the left-hand side, the number of nodes selected from each cluster is added. The total sum must be k. Constraints (5) impose that at most one variable is positive. Notice that if all of the variables are null for a given j, then no node is selected from cluster j. Constraints (6) are the integrality conditions of the variables.

This problem is a particular case of the multiple-choice knapsack problem, and it is known as the linking set problem (LSP) [36], which can be solved by the following dynamic program (which is an adaptation of the one given in [36] to our case).

Let $\alpha(j,i)$ denote the optimal value of the subproblem when considering the first j clusters and i seed nodes. For $j = 1$, we obtain:

$$\alpha(1,i) = \begin{cases} c_{i1}, & i \in K, \\ +\infty, & \text{otherwise} \end{cases}$$

For $j = 2, \ldots, m$ the recursive relations are given by

$$\alpha(j,i) = \begin{cases} \max_{w \in \{0,\ldots,i-1\}} \{\alpha(j-1, i-w) + c_{ij}\}, & i \in K, \\ +\infty, & \text{otherwise} \end{cases}$$

where $c_{i0} = 0$.

The optimal value of the LSP is given by $\alpha(m,k)$. This dynamic program can be solved in $\mathcal{O}(mk^2)$ operations.

It should be noticed that the optimal value of the LSP should be an under-estimative of the value of $\sigma(S)$, since the spread of each set S_j^i is estimated within each subgraph, not in the entire graph G, and the interactions between activated nodes in different subgraphs are not taken into account.

An important property of the IM models is submodularity; see Section 3. This property also holds when $\sigma(\cdot)$ is applied to each cluster. Consequently, that property implies that for each cluster j the gains for including a new node in the seed set are decreasing, that is,

$$c_{i,j} - c_{i-1,j} \geq c_{i+1,j} - c_{i,j} \tag{7}$$

where we assume again $c_{0,j} = 0$ for all $j \in J$.

When this property holds, the linking set problem can be solved more efficiently using the following greedy algorithm (Algorithm 4) where, in each iteration, the marginal gain is obtained from selecting a node from each cluster, and selecting the node with the largest gain. This algorithm has time complexity $\mathcal{O}(mk)$.

The proof of this theorem is left in Appendix B. If the values c_{ij} are obtained from the true function $\sigma(\cdot)$ or from a greedy algorithm, then Property (7) holds and Theorem 2 can be applied.

Theorem 2. *If condition* (7) *holds then Algorithm* 4 *solves the LSP.*

Algorithm 4 Greedy algorithm for the linking set problem.

Set $value := 0$; $t := 0$; and, for $j \in J$, $i_j := 0$,
while $t \leq k$ **do**
    compute $j^* = argmax_{j \in J} \left\{ c_{i_j+1,j} - c_{i_j,j} \right\}$;
    $value = value + c_{i_{j^*}+1,j^*} - c_{i_{j^*},j^*}$;
    $i_{j^*} := i_{j^*} + 1$;
    $t = t + 1$.
end while
for $j \in J$ **do**
    set $x_{ij} = 1$, if $i = i_j$ and $x_{ij} = 0$, otherwise.
end for

4.3. The ClusterGreedy Algorithm

The full ClusterGreedy is described in Algorithm 3.

First, the algorithm applies the MCL algorithm to define the clusters of graph G. Then, it applies the SimpleGreedy to each subgraph of G induced by the clusters obtained by the graph clustering of the main network. SimpleGreedy obtains a candidate seed set S_j for each cluster j with cardinality k. Remember that SimpleGreedy (see Algorithm 2) forms set S_j in a constructive manner, including a node at each iteration. Hence, it generates a sequence of sets S_j^i, for $i = 1, \ldots, k$ containing the seed set with i nodes. In addition, it estimates the spread of each set, which is given by c_{ij}. These values are the input parameters to the LSP presented in Section 4.2. Then, the seed set S is obtained from the optimal solution to the LSP, from the combination of all sets generated for each cluster. This set is the union of sets S_j^i, such that $x_{ij} = 1$.

The time complexity of the ClusterGreedy is the maximum of the time complexity of the dynamic program for the linking set ($\mathcal{O}(km)$) and the time complexity of the SimpleGreedy ($\mathcal{O}(kr \sum_{j=1}^{m} n_j e_j)$, where n_j and e_j are the numbers of nodes and edges, respectively, of the subgraph induced by cluster j. From this complexity, we observe that the size of the largest cluster will be an important factor in determining the running time of the algorithm.

By observing that when the values of c_{ij} are obtained from the greedy algorithm and Property (7) holds, we can improve the ClusterGreedy. Instead of solving, for each cluster, the SimpleGreedy for a seed set of k nodes, we merge the SimpleGreedy with the Greedy algorithm for the LSP. The full algorithm, denoted by Improved ClusterGreedy, is described in Algorithm 5.

Algorithm 5 starts by computing the gain from selecting a node from each cluster. Then, in each iteration, we choose the node from the cluster that provides a larger marginal gain and apply a new iteration of the greedy algorithm to that cluster.

This algorithm allows us to reduce the number of iterations performed in the calls to the SimpleGreedy. In particular, instead of performing mk calls to the function MC-ESTIMATE(S, G) it performs only $k + m$ calls. Since the function MC-ESTIMATE (S, G) is the most time-consuming operation with $\mathcal{O}(re)$ operations (where e is the number of edges of graph G), this reduction allows for substantial time savings.

Algorithm 5 Improved ClusterGreedy (G, k).

Input: $G = (V, E)$: social graph; arc weights $w_e, e \in E$; k: cardinality of the returning set
Output: seed set S

generate the clusters of graph G using the MCL algorithm
set m to the number of clusters of G
for $j = 1$ to m **do**
    $G_j \leftarrow$ the subgraph induced by the set of nodes in cluster j
    $S_j =$SimpleGreedy$(G_j, 1)$ (Algorithm 2)
    compute c_{ij}, the value of spread within cluster j, with seed set S_j^i, for $i = 1$
end for
Set $value := 0$; $t := 0$; and, for $j \in J$, $i_j := 0$,
while $t \leq k$ **do**
    compute $j^* = argmax_{j \in J}\left\{c_{i_j+1,j} - c_{i_j,j}\right\}$;
    $value = value + c_{i_{j^*}+1,j^*} - c_{i_{j^*},j^*}$;
    $i_{j^*} := i_{j^*} + 1$;
    $t = t + 1$.
    $S_j =$SimpleGreedy(G_j, i_{j^*}) (Algorithm 2)
    Set $C_{i_{j^*},j}$ to the optimal value of SimpleGreedy(G_j, i_{j^*})
end while
$S \leftarrow \emptyset$
for $j \in J$ **do**
    $S \leftarrow S_j^i$
end for
return S

5. Computational Experiments

In this section, the computational experiments conducted to evaluate the proposed approach are reported. These experiments compare the performance of the ClusterGreedy algorithm (Algorithm 3) with the SimpleGreedy (Algorithm 2) and to evaluate the gains of using the Improved ClusterGreedy algorithm when compared to the ClusterGreedy.

These experiments were conducted in a Desktop DELL Intel Core i7-11700 2.50GHz 16 GB 500 GB Win11Pro. All the implementations were conducted in the Julia language.

We consider two sets of instances. A set of instances based on Watts–Strogatz random graphs was generated using the package Graphs.jl (version v1.8.0) [37] and a set of real datasets.

The LSP was solved using the packages JuMP.jl (version v1.11.1) and HiGHS.jl (version v1.5.2). The clusters were generated by MCL [34] (available in the Clustering Julia package/version v0.15.2). Based on some preliminary tests to control the number of clusters, we set the inflation parameter to $I = 5.5$.

We report the results with Watts–Strogatz random graphs on Section 5.1 and the results on a real data set on Section 5.2.

5.1. Computational Results with Watts–Strogatz Random Graphs

For n equal to 3000, 4000, 5000, and 6000, we obtained, on average, 51, 62, 58, and 60 clusters (Figure 2), respectively. The runtime of MCL was 28.65 s, on average, which is very small when compared to the total running time of the algorithms. Figure 3 shows the average runtime of cluster generation for n equal to 3000, 4000, 5000, and 6000.

The experimental results show that the ClusterGreedy is faster than SimpleGreedy. For $r = 50$ (10 result) and $r = 100$ (10 result) simulations, both algorithms were tested for n equal to 3000, 4000, 5000, and 6000. ClusterGreedy used, on average, 35%, 36%, 27%, and 22% of the SimpleGreedy runtime, respectively, to obtain the seed set S (Figures 4 and 5). It corresponds, globally, to 25%. That is, on average, ClusterGreedy required only 25% of the runtime of SimpleGreedy to achieve the solution to the problem. More importantly, there is a clear trend showing that the gains in the running time increase

as n increases. The Wilcoxon signed-rank test used to compare the running times of both algorithms provides a p value close to zero for all sets of instances, showing that the gains are statistically meaningful.

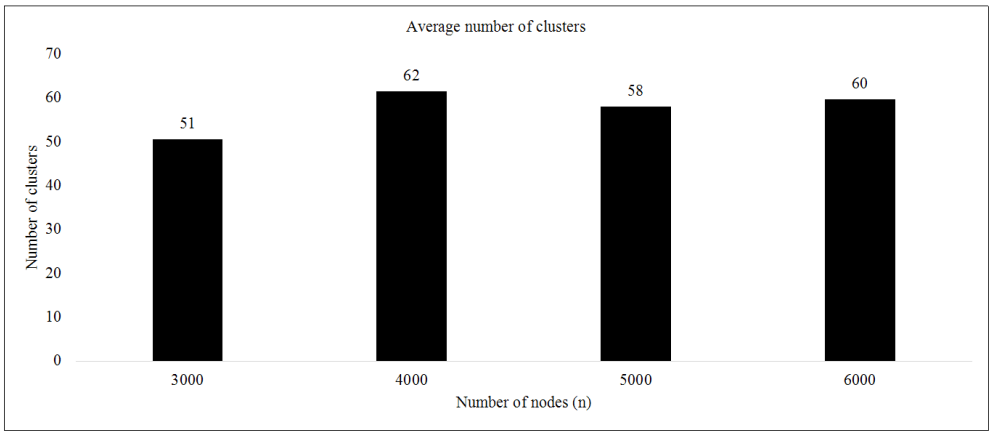

Figure 2. Average number of clusters generated by the MCL algorithm.

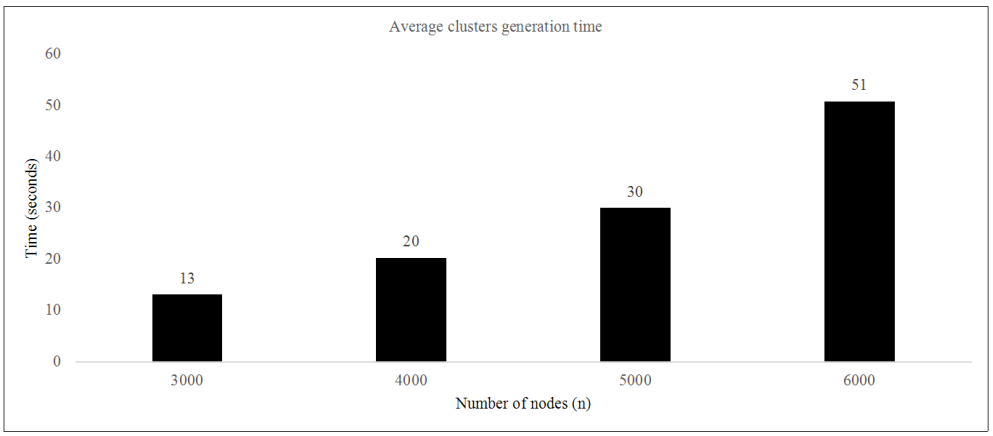

Figure 3. Average runtime of clusters' generation.

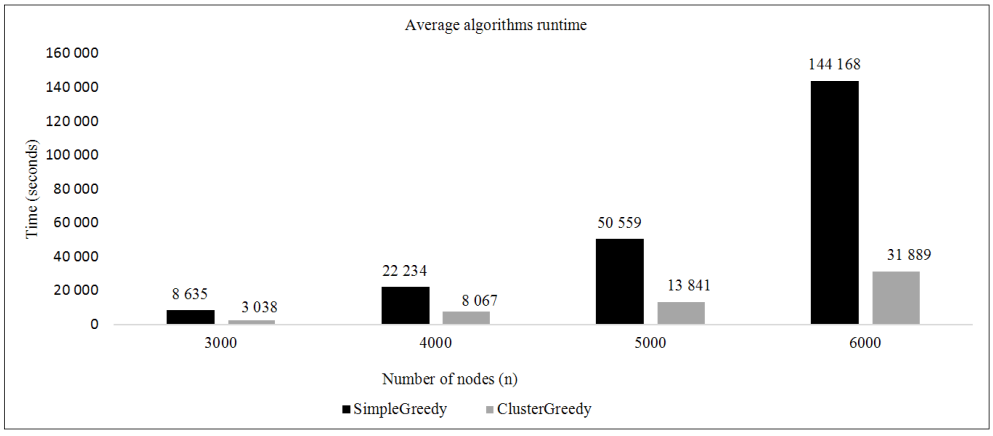

Figure 4. Comparison of the average execution time of SimpleGreedy and ClusterGreedy algorithms.

To evaluate the spread of the seed set S in the network, we simulate the propagation of the seed set S in the network with 1000 runs and calculate the average value. The results show that the seed sets obtained with ClusterGreedy provide a gain of about 3%, on average, for the estimated spread when compared with the seed sets obtained with SimpleGreedy (Figure 6). The Wilcoxon signed-rank test provides a p value lower than 5%

for all sets of instances except for $n = 3000$, which is 0.09, meaning that for these larger instances, the improvement in the objective function value is statistically meaningful.

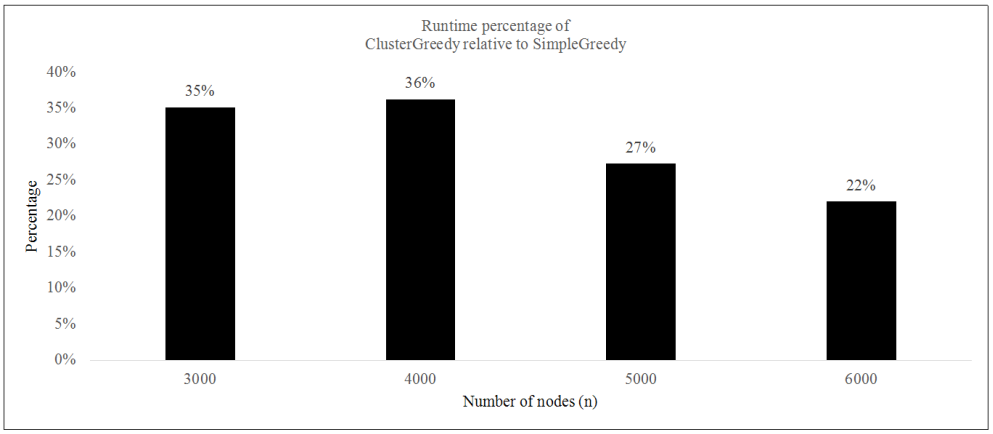

Figure 5. Percentage of average runtime of ClusterGreedy compared with SimpleGreedy.

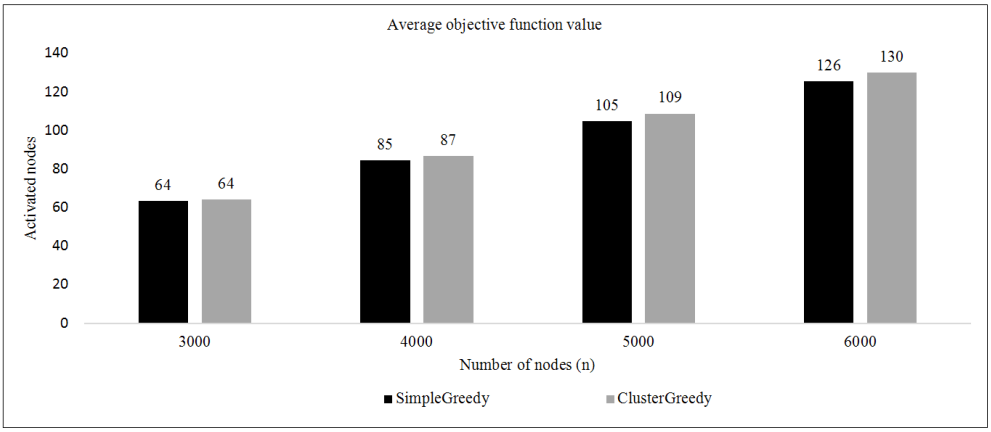

Figure 6. Comparison of the average objective function value given by SimpleGreedy and ClusterGreedy algorithms.

In the experimental results (in bold), reported in Appendix A, numbers 4, 11, 13 (Table A1), 4, 15, 16 (Table A2), 1, 6, 15 (Table A3), 2, 4, 11, 13, 16, and 17 (Table A4), the runtime of the ClusterGreedy algorithm is not more than 15% of the runtime of the SimpleGreedy algorithm. These were the best results obtained. In opposition to these, results with numbers 3, 18 (Table A1), 20 (Table A2), and 5 (Table A4) were recorded, in which the execution time of the ClusterGreedy algorithm was longer than SimpleGreedy's.

In order to compare the Improved ClusterGreedy with the ClusterGreedy, we test the running times for $n = 3000, 4000, 5000, 6000$ and $r = 50$. The running times of the Improved ClusterGreedy were 4.1%, 2.8%, 2.5%, 1.6%, repectively, and, on average, 2.8%. In relation to the SimpleGreedy, the running times of the Improved ClusterGreedy were, on average, 0.5%.

5.2. Computational Results with Real Datasets

- Email-eu-core [38]: Email records from a major European research institution were utilized to create the network. If individual u has transmitted at least one email to individual v, then there is an edge (u, v) in the network. The dataset excludes any inbound or outgoing messages from or to the rest of the world; the emails exclusively document communication among members of the institution (the core). "Ground-truth" community memberships of the nodes are also included in the dataset. At the research institute, each individual is affiliated with precisely one of the 42 departments.

- Adolescent health [39]: The construction of this network was informed by a survey done between 1994 and 1995. Each student was directed to create a list of their top five female acquaintances and their top five male acquaintances. A node represents a student, and an edge (u, v) linking two students signifies that student u has chosen student v as a friend.
- Gnutella peer-to-peer (8 August 2002) [38]: An assemblage of the Gnutella peer-to-peer file-sharing network captures the month span of August 2002. The Gnutella network topology is composed of nodes, which symbolize hosts and edges, which symbolize connections among the hosts.

The statistics of the real–world graph datasets that we employ are outlined in Table 2.

The experimental results demonstrate that the ClusterGreedy algorithm exhibits a higher speed compared to the SimpleGreedy algorithm. Both methods underwent testing using the three chosen real data sets. The ClusterGreedy algorithm utilized 4% of the SimpleGreedy runtime in the Email-eu-core network, 70% in Adolescent health, and 60% in Gnutella peer-to-peer to obtain the seed set S (see to Table 3). Globally, it represents 54%. On average, ClusterGreedy achieved the solution to the problem in just 54% of the duration required by SimpleGreedy, based on the selected real data sets. The highest performance was attained in the dataset with the most significant average clustering coefficient (Email-eu-core network).

Table 2. Dataset statistics.

Network Statistics	Email-Eu-Core Network	Adolescent Health Network	Gnutella Peer-to-Peer Network
Nodes	1005	2539	8114
Edges	25,571	12,969	26,013
Average clustering coefficient	0.3994	0.1419	0.0095
Type	Directed	Directed	Directed

Table 3. Computational results of evaluation of ClusterGreedy in real data sets.

Network	n	m	k	r	Runtime (s)			Objective Function Value		
					MCL	SimpleGreedy	ClusterGreedy	Linking Set	SimpleGreedy	ClusterGreedy
Email-eu-core	1005	181	10	100	0.30	60,630.30	2327.60	131.268	530.814	411.78
Adolescent health	2539	406	25	50	25.87	2511.18	1768.04	82.181	176.257	157.273
Gnutella peer-to-peer	8114	13	81	50	59.87	482,067.82	288,061.54	244.29	763.558	771.724
Average					29	181,736	97,386	153	490	447

6. Conclusions

We present a novel algorithmic strategy to solve the problem of maximizing the influence that is based on the idea of creating a partition of the set of nodes of the original network into smaller subsets, solving the IM problem in the subgraph induced for each subset of nodes, and, finally, combining the solution obtained for the several subgraphs in order to generate a final solution (seed set). This general idea was applied using greedy algorithms, and the results showed that:

- The proposed algorithm (called ClusterGreedy) needed only, on average, 25% of the classical greedy (SimpleGreedy) runtime to achieve the solution of the influence maximization problem;

- under the general condition that the marginal gains decrease with the increase in the size of the seed set (which is satisfied by the greedy algorithm and by the diffusion function under the linear threshold model), then combining the solutions from the different clusters can be conducted in a much more efficient way. Moreover, the two-phase approach can be improved and a new algorithm that searches for the highest marginal gain between the different clusters can be used. This improved algorithm runs, on average, in 4% of the running time of the ClusterGreedy algorithm.

Overall, the proposed approach is more efficient than the classical one of solving the IM problem. It should be important to notice that this approach can be applied using other algorithms for the IM problem, not necessarily the greedy one. This future research direction includes the use of both proxy-based and sketch-based approaches, see [11]. However, for those cases, the necessary condition for the improved algorithm may not hold. Another direction of research is to explore the relation of the IM problem with the p-median problem, where a given set of p nodes must also be selected. In Ref. [40], a decomposition approach for the p-median problem on disconnected graphs is introduced. This decomposition allows for solving large-scale problems. In our approach, we obtain disconnected graphs after performing the clustering. Hence, a similar decomposition approach can be employed for the IM problem.

Author Contributions: Conceptualization, J.M.S. and A.A.; methodology, J.M.S. and A.A.; software, J.M.S.; validation, J.M.S. and A.A.; formal analysis, J.M.S. and A.A.; investigation, J.M.S. and A.A.; data curation, J.M.S.; writing—original draft preparation, J.M.S.; writing—review and editing, J.M.S. and A.A.; visualization, J.M.S.; supervision, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by The Center for Research and Development in Mathematics and Applications (CIDMA) through the Portuguese Foundation for Science and Technology (FCT—Fundação para a Ciência e a Tecnologia), references https://doi.org/10.54499/UIDB/04106/2020 and https://doi.org/10.54499/UIDP/04106/2020 (accessed on 6 February 2024).

Data Availability Statement: The Julia code used to obtain reported results can be shared by sending an email to jsamuco@gmail.com.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CELF	Cost-effective lazy forward
IM	Influence maximization
LT	Linear threshold
LSP	Linking set problem
LP	Linear programming
MCL	Markov clustering

Appendix A

Here, we present the tables with the full detailed results.

Tables A1–A4 provide detailed information regarding the running times and objective function values for $n = 3000$, $n = 4000$, $n = 5000$, $n = 6000$, respectively, for the instances based on Watts–Strogatz random graphs. Column m provides the number of clusters, column k provides the number of elements in the seed set, and column r provides the number of Monte Carlo simulations. We can observe that the objective function value obtained from the LSP is an underestimation of the estimated spread, as expected.

Table A1. Computational results for $n = 3000$.

m	k	r	Runtime (s)			Objective Function Value		
			MCL	SimpleGreedy	ClusterGreedy	Linking Set	SimpleGreedy	ClusterGreedy
46	30	50	9.63	5686.06	1272.13	52	63	66
74	30	50	8.49	6102.56	1205.10	52	63	64
35	30	50	10.47	**1842.13**	**2991.36**	55	63	66
54	30	50	7.18	**6448.28**	**646.86**	54	64	55
36	30	50	9.54	2507.60	1075.74	57	63	67
36	30	50	19.01	6646.61	2276.28	55	64	66
57	30	50	11.23	3806.67	1530.79	51	62	55
35	30	50	18.50	5989.16	2078.36	54	64	67
45	30	50	11.30	4338.80	2415.55	51	63	65
35	30	50	11.01	4388.51	1671.92	53	63	53
72	30	100	10.74	**17,638.40**	**2549,70**	54	64	63
36	30	100	10.32	13,198.84	7041.78	52	64	66
70	30	100	9.66	**23,175.31**	**2879.32**	56	64	68
35	30	100	42.80	18,236.07	5763.19	56	64	67
35	30	100	10.60	22,917.41	9680.69	55	65	64
71	30	100	17.54	6037.13	3449.89	55	64	65
72	30	100	11.30	5725.89	2720.86	54	63	66
35	30	100	10.68	**5859.43**	**6868.13**	55	64	66
63	30	100	12.22	6199.05	1556.98	56	64	68
71	30	100	11.39	5965.02	1091.29	54	64	68
		Average	13	8635	3038	54	64	64

Table A2. Computational results for $n = 4000$.

m	k	r	Runtime (s)			Objective Function Value		
			MCL	SimpleGreedy	ClusterGreedy	Linking set	SimpleGreedy	ClusterGreedy
37	40	50	20.61	19,183.01	6084.58	70	84	84
94	40	50	17.05	15,541.19	3535.49	71	83	88
47	40	50	24.58	17,288.80	3378.13	72	85	89
78	40	50	25.57	**19,902.08**	**2953.08**	70	84	87
70	40	50	20.28	19,153.86	4529.98	71	83	87
48	40	50	24.60	15,556.00	2729.04	73	84	88
95	40	50	20.66	9470.89	1879.85	72	84	89
96	40	50	23.28	10,521.98	1758.60	73	84	88
37	40	50	24.25	14,953.14	4982.72	72	84	88
83	40	50	26.16	8261.44	2475.15	73	84	88
47	40	100	15.73	37,350.58	16,860.93	75	85	89
64	40	100	17.51	13,187.33	6510.77	71	84	88
48	40	100	18.71	13,766.84	2511.51	72	85	88

Table A2. *Cont.*

m	k	r	Runtime (s)			Objective Function Value		
			MCL	SimpleGreedy	ClusterGreedy	Linking set	SimpleGreedy	ClusterGreedy
49	40	100	19.37	12,624.12	2842.77	71	84	88
98	40	100	18.29	**29,727.57**	**2001.41**	69	85	89
95	40	100	16.80	**31,476.40**	**3398.91**	72	85	85
38	40	100	15.49	30,891.72	10,749.67	70	85	89
42	40	100	16.95	31,523.74	18,893.21	70	86	88
48	40	100	21.75	56,815.60	18,476.38	72	86	88
18	40	100	19.64	**37,483.61**	**44,788.42**	64	86	68
		Average	20	22,234	8067	71	85	87

Table A3. Computational results for $n = 5000$.

m	k	r	Runtime (s)			Objective Function Value		
			MCL	SimpleGreedy	ClusterGreedy	Linking Set	SimpleGreedy	ClusterGreedy
72	50	50	35.50	**39,198.99**	**4090.11**	87	104	109
27	50	50	33.66	37,746.83	12,551.05	83	104	110
35	50	50	38.13	43,241.71	21,132.67	88	103	97
45	50	50	30.11	41,008.61	15,922.91	90	105	112
59	50	50	28.34	23,410.06	8635.88	94	103	107
88	50	50	26.32	**18,710.15**	**2134.49**	92	104	106
76	50	50	28.28	17,171.25	2656.90	87	105	109
50	50	50	39.54	24,120.51	15,279.60	86	103	106
60	50	50	20.85	12,712.06	2265.87	91	103	111
60	50	50	23.88	10,538.98	2520.88	88	104	110
61	50	100	27.45	48,493.67	14,665.65	93	107	113
61	50	100	27.15	37,923.94	16,476.38	92	106	104
46	50	100	29.05	49,783.73	15,885.76	84	107	109
59	50	100	28.31	78,956.37	19,807.02	89	105	110
76	50	100	27.38	**159,351.68**	**16,017.61**	91	106	109
58	50	100	29.39	79,943.63	22,085.49	90	108	110
60	50	100	29.24	37,985.19	14,601.94	92	106	112
59	50	100	31.62	82,649.48	25,357.64	91	105	112
51	50	100	33.72	89,812.26	16,172.17	86	107	111
59	50	100	35.05	78,412.95	28,552.08	92	105	112
		Average	30	50,559	13,841	89	105	109

Table A4. Computational results for $n = 6000$.

m	k	r	Runtime (s)			Objective Function Value		
			MCL	SimpleGreedy	ClusterGreedy	Linking Set	SimpleGreedy	ClusterGreedy
71	60	50	64.63	47,839.63	13,682.93	107	124	132
142	60	50	34.95	**47,010.98**	**2842.19**	106	124	128
47	60	50	36.24	83,701.97	33,636.33	101	126	133
72	60	50	66.51	**172,009.16**	**22,980.20**	108	126	129
17	60	50	66.52	**74,947.11**	**88,144.33**	98	126	92
20	60	50	34.84	29,437.34	16,364.70	101	125	129
71	60	50	34.83	57,536.98	10,625.88	109	124	132
39	60	50	76.57	48,285.85	33,088.12	103	123	130
63	60	50	77.48	48,952.40	19,807.15	105	125	134
61	60	50	35.40	52,906.38	15,269.56	102	125	132
71	60	100	44.85	**267,189.27**	**14,896.03**	108	129	134
58	60	100	37.08	243,731.77	50,383.76	110	126	135
72	60	100	38.53	**159,052.73**	**12,433.80**	108	127	135
42	60	100	38.51	180,923.76	82,078.75	99	126	131
9	60	100	54.29	296,129.56	59,096.51	102	125	132
70	60	100	34.30	**262,348.87**	**12,476.31**	112	128	136
54	60	100	59.83	**313,372.35**	**36,160.77**	105	126	133
72	60	100	72.20	101,866.35	37,024.97	111	126	134
73	60	100	72.34	106,270.40	25,122.75	112	126	128
71	60	100	38.18	289,850.34	51,670.68	109	126	134
		Average	51	144,168	31,889	106	126	130

Appendix B

Here, we prove Theorem 2.

Proof. First, consider the following reformulation of the LSP using variables $y_{ij}, j \in J, i \in K$, which indicate whether at least i nodes from cluster j are selected. The model is written using the marginal gains, as follows:

$$max \quad \sum_{j=1}^{m} \sum_{i=1}^{k} (c_{ij} - c_{i-1,j}) y_{ij}$$

$$\sum_{j=1}^{m} \sum_{i=1}^{k} y_{ij} = k, \tag{A1}$$

$$y_{ij} \leq y_{i-1,j}, \qquad j \in J, i \in K, \tag{A2}$$

$$y_{ij} \in \{0,1\}, \qquad i \in K, j \in J \tag{A3}$$

Constraints (A1) ensure that k nodes are selected, and constraints (A2) ensure that if at least i nodes are selected from cluster j, then at least $i-1$ nodes are selected from cluster j.

Notice that y_{ij} variables can be related to variables x_{ij}, as follows:

$$x_{ij} = 1 \text{ iff } \ (y_{tj} = 1, \ t \leq i, \ \text{ and } \ y_{tj} = 0, \ t > i), \quad j \in J, i \in K \tag{A4}$$

Now, we can observe that the relaxation obtained from removing constraints (A2) is a uniform matroid which can be solved by the greedy algorithm. As the coefficients are decreasing for each j (since they satisfy Equation (7)), then the greedy algorithm follows exactly the steps of Algorithm 4. Now, observing that the solution obtained by this algorithm satisfies constraints (A2), we conclude that it is optimal for the LSP, which concludes the proof. □

References

1. 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, San Francisco, CA, USA, 26–29 August 2001; pp. 57–66.
2. Richardson, M.; Domingos, P. Mining knowledge-sharing sites for viral marketing. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 23–26 July 2002; pp. 61–70.
3. Rui, X.; Meng, F.; Wang, Z.; Yuan, G. A reversed node ranking approach for influence maximization in social networks. *Appl. Intell.* **2019**, *49*, 2684–2698. [CrossRef]
4. Huang, H.; Shen, H.; Meng, Z. Community-based influence maximization in attributed networks. *Appl. Intell.* **2020**, *50*, 354–364. [CrossRef]
5. Leskovec, J.; Adamic, L.A.; Huberman, B.A. The dynamics of viral marketing. *ACM Trans. Web TWEB* **2007**, *1*, 5. [CrossRef]
6. Chen, W.; Castillo, C.; Lakshmanan, L.V. *Information and Influence Propagation in Social Networks*; Morgan & Claypool Publishers: San Rafael, CA, USA, 2013.
7. Haleh, S.; Dizaji, S.; Patil, K.; Avrachenkov, K. Influence Maximization in Dynamic Networks Using Reinforcement Learning. *SN Comput. Sci.* **2024**, *5*, 169.
8. Ni, P.; Guidi, B.; Michienzi, A.; Zhu, J. Equilibrium of individual concern-critical influence maximization in virtual and real blending network. *Inf. Sci.* **2023**, *648*, 119646. [CrossRef]
9. Arora, A.; Galhotra, S.; Ranu, S. Debunking the myths of influence maximization: An in-depth benchmarking study. In Proceedings of the 2017 ACM International Conference on Management of Data, New York, NY, USA, 14–19 May 2002; pp. 651–666.
10. Guille, A.; Hacid, H.; Favre, C.; Zighed, D.A. Information diffusion in online social networks: A survey. In *SIGMOD Rec.* **2013**, *42*, 17–28. [CrossRef]
11. Li, Y.; Fan, J.; Wang, Y.; Tan, K.-L. Influence Maximization on Social Graphs: A Survey. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 1852–1872. [CrossRef]
12. Sun, J.; Tang, J. A survey of models and algorithms for social influence analysis. In *Social Network Data Analytics*; Springer: New York, NY, USA, 2011; pp. 177–214.
13. Tejaswi, V.; Bindu, P.V.; Thilagam, P.S. Diffusion models and approaches for influence maximization in social networks. In Proceedings of the 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, India, 21–24 September 2016; pp. 1345–1351.
14. Zheng, Y. *A Survey: Models, Techniques, and Applications of Influence Maximization Problem*; Southern University of Science and Technology: Shenzhen, China, 2018.
15. Kempe, D.; Kleinberg, J.; Tardos, E. Maximizing the spread of influence through a social network. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 24–27 August 2003; pp. 137–146.
16. Ko, Y.-Y.; Cho, K.-J.; Kim, S.-W. Efficient and effective influence maximization in social networks: A hybrid-approach. *Inf. Sci.* **2018**, *465*, 144–161. [CrossRef]
17. Leskovec, J.; Krause, A.; Guestrin, C.; Faloutsos, C.; VanBriesen, J.; Glance, N. Cost-effective outbreak detection in networks. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, CA, USA, 12–15 August 2007; pp. 420–429.
18. Taherinia, M.; Esmaeili, M.; Minaei Bidgoli, B. Optimizing CELF Agorithm for Influence Maximization Problem in Social Networks. *J. Data Min.* **2022**, *10*, 25–41.
19. Goyal, A.; Lu, W.; Lakshmanan, L.V.S. Celf++ optimizing the greedy algorithm for influence maximization in social networks. In Proceedings of the 20th International Conference Companion on World Wide Web, Hyderabad, India, 28 March–1 April 2011; pp. 47–48.
20. 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, Paris, France, 28 June–1 July 2009; pp. 199–208.
21. Cheng, S.; Shen, H.; Huang, J.; Zhang, G.; Cheng, X. Staticgreedy: Solving the scalability-accuracy dilemma in influence maximization. In Proceedings of the 22nd ACM International Conference on Information & Knowledge Management, San Francisco, CA, USA, 27 October–1 November 2013.
22. Goyal, A.; Lu, W.; Lakshmanan, L.V.S. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In Proceedings of the 2011 IEEE 11th International Conference on Data Mining, Vancouver, BC, Canada, 11–14 December 2011; IEEE: New York, NY, USA, 2011; pp. 211–220.

23. Rahimkhani, K.; Aleahmad, A.; Rahgozar, M.; Moeini, A. A fast algorithm for finding most influential people based on the linear threshold model. *Expert Syst. Appl.* **2015**, *42*, 1353–1361. [CrossRef]
24. Shang, J.; Zhou, S.; Li, X.; Liu, L.; Wu, H. COFIM: A community-based framework for influence maximization on large-scale networks. *Knowl. Based Syst.* **2017**, *117*, 88–100. [CrossRef]
25. Bozorgi, A.; Haghighi, H.; Zahedi, M.S.; Rezvani, M. Incim: A community-based algorithm for influence maximization problem under the linear threshold model. *Inf. Process. Manag.* **2016**, *52*, 1188–1199. [CrossRef]
26. Girvan, M.; Newman, M.E.J. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA* **2002**, *99*, 7821–7826. [CrossRef]
27. Ghayour-Baghbani, F.; Asadpour, M.; Faili, H. Mlpr: Efficient influence maximization in linear threshold propagation model using linear programming. *Soc. Netw. Anal. Min.* **2021**, *11*, 3. [CrossRef]
28. Baghbani, F.G.; Asadpour, M.; Faili, H. Integer linear programming for influence maximization. *Iran. J. Sci. Technol. Trans. Electr. Eng.* **2019**, *43*, 627–634. [CrossRef]
29. Keskin, M.E.; Güler, M.G. Influence maximization in social networks: An integer programming approach. *Turk. J. Electr. Eng. Comput.* **2018**, *26*, 3383–3396. [CrossRef]
30. Wu, H.-H.; Küçükyavuz, S. A Two-stage Stochastic Programming Approach for Influence Maximization in Social Networks. *Comput. Optim. Appl.* **2018**, *69*, 563–595. [CrossRef]
31. Ackerman, E.; Ben-Zwi, O.; Wolfovitz, G. Combinatorial model and bounds for target set selection. *Theor. Comput. Sci.* **2010**, *411*, 4017–4022. [CrossRef]
32. Schaeffer, S.E. Graph clustering. *Comput. Sci. Rev.* **2007**, *1*, 27–64. [CrossRef]
33. Vlasblom, J.; Wodak, S.J. Markov Clustering versus Affinity Propagation for the Partitioning of Protein Interaction Graphs. *BMC Bioinform.* **2009**, *10*, 99. [CrossRef]
34. Van Dongen, S. Graph Clustering Via a Discrete Uncoupling Process. *SIAM J. Matrix Anal. Appl.* **2008**, *30*, 121–141. [CrossRef]
35. Van Dongen, S. Graph Clustering by Flow Simulation. Ph.D. Thesis, Utrecht University, Utrecht, The Netherlands, 2000.
36. Agra, A.; Requejo, C. The linking set problem: A polynomial special case of the multiple-choice knapsack problem. *J. Math. Sci.* **2009**, *161*, 919–929. [CrossRef]
37. Fairbanks, J.; Besançon, M.; Simon, S.; Hoffiman, J.; Eubank, N.; Karpinski, S. Juliagraphs/Graphs.jl: An Optimized Graph Package for the Julia Programming Language. 2021. Available online: https://github.com/JuliaGraphs/Graphs.Jl (accessed on 6 February 2024).
38. Leskovec, J.; Krevl, A. SNAP Datasets: Stanford Large Network Dataset Collection. 2014. Available online: http://snap.stanford.edu/data (accessed on 6 February 2024).
39. Moody, J. Peer Influence Groups: Identifying Dense Clusters in Large Networks. *Soc. Netw.* **2001**, *23*, 261–283. [CrossRef]
40. Agra, A.; Cerdeira, J.O.; Requejo, C. A decomposition approach for the p-median problem on disconnected graphs. *Comput. Oper. Res.* **2017**, *86*, 79–85. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.