

Graph-Aware Evolutionary Algorithms for Influence Maximization

Kateryna Konotopska
University of Trento
Trento, Italy
konotopska.k@gmail.com

Giovanni Iacca
University of Trento
Trento, Italy
giovanni.iacca@unitn.it

ABSTRACT

Social networks represent nowadays in many contexts the main source of information transmission and the way opinions and actions are influenced. For instance, generic advertisements are way less powerful than suggestions from our contacts. However, this process hugely depends on the influence of people who disseminate these suggestions. Therefore modern marketing often involves paying some targeted users, or *influencers*, for advertising products or ideas. Finding the set of nodes in a social network that lead to the highest information spread –the so-called Influence Maximization (IM) problem– is therefore a pressing question and as such it has recently attracted a great research interest. In particular, several approaches based on Evolutionary Algorithms (EAs) have been proposed, although they are known to scale poorly with the graph size. In this paper, we tackle this limitation in two ways. Firstly, we use approximate fitness functions to speed up the EA. Secondly, we include into the EA various graph-aware mechanisms, such as smart initialization, custom mutations and node filtering, to facilitate the EA convergence. Our experiments show that the proposed modifications allow to obtain a relevant runtime gain and also improve, in some cases, the spread results.

CCS CONCEPTS

• **Human-centered computing** → **Social networks**; *Social network analysis*; • **Theory of computation** → **Evolutionary algorithms**; • **Mathematics of computing** → Graph algorithms.

KEYWORDS

Social Network, Influence Maximization, Evolutionary Algorithm, Graph Theory, Combinatorial Optimization.

1 INTRODUCTION

Online social networks have become a relevant part in the daily life of the majority of people. Social networks are used every day to maintain contact with other people, read news, search for a job, watch movies, or do shopping. All these things are usually done based on what other people say about products, jobs, events, and so on. More formally, when the activity of a user u implies an action of the user v , there is an edge $u \rightarrow v$ in the graph representing the social network. This edge might indicate, for example, a voter supporting a candidate for an election, or the fact that if a customer u buys a product, the same product will be bought by the user v . In general, the notation $u \rightarrow v$ indicates that information spreads from u to v with a certain probability. In case of success, v is said to be *activated* by u . As we will see later, different probabilistic models, called *diffusion models*, are now well-established tools to study how influence propagates over social networks.

When one wants to find which set of nodes has more chances of producing the maximum possible number of activations in a social network, we are dealing with a problem known as the *Influence Maximization* (IM) problem. This is a crucial problem for instance to win political campaigns, perform targeted marketing to trigger word-of-mouth, etc. Usually, one has a limited budget on the number of nodes that can be initially activated (the so-called *seed* nodes). This is a particularly challenging combinatorial problem that, being proved to be NP-hard [20], has attracted a great research interest in the past few years. As we will briefly summarize in Section 3, various algorithms have been recently proposed for solving the IM problem, either heuristics with provable guarantees, or metaheuristics such as Evolutionary Algorithms (EAs). However, while several successful techniques exist, these usually suffer from the so-called *curse of dimensionality*, i.e., they either require too much time to converge, or produce limited-quality results.

Here, we aim at improving the performance of EAs to solve the IM problem. To do that, we first investigate two different approximations of the influence spread simulation. Then, we introduce several graph-aware mechanisms to facilitate the EA convergence. These contributions represent the main novelty of this work: to the best of our knowledge, a thorough investigation of these directions has not been done in the current literature. Overall, our experiments show that the proposed modifications produce a relevant runtime gain and also improve, in some cases, the spread results.

The rest of the paper is organized as follows. In the next Section, we provide the background concepts. Section 3 gives an overview of the state-of-the-art in the field of IM. Sections 4 describes our methods, while Section 5 presents our experimental setup and the numerical results. Finally, Section 6 summarizes our main findings and discusses the possible future work directions.

2 BACKGROUND

As said earlier, the IM problem consists in finding the set of nodes that will lead to the maximal number of node activations according to a given influence spread model. More specifically, the problem can be formulated as follows: given a network, represented with a graph G , a diffusion model m , and a budget k , find a set of k initially active nodes, such that the influence spread is maximized. Here, we use the Independent Cascade (IC) and Weighted Cascade (WC) models [20]. In IC, at each timestep t a node can be either active or inactive. A node can transit from inactive to active, but it cannot become inactive again. At the beginning, only the nodes in the seed set S are in the active state. Then, at each timestep each active node u has a chance of activating its neighboring nodes v (if the edge $u \rightarrow v$ exists) with a probability p , which is the same for all the nodes in the graph and is a given parameter. The simulation stops

once a convergence condition is met, see Algorithm 1 in Section 4.1. Influence spread is thus given by the number of active nodes at the end of the simulation. In WC, influence spread is calculated in the same way as in IC, the only difference being the probability of node activation: given a node v , the probability of being activated by one of its neighboring nodes u is $1/(\text{indegree}(v))$.

3 RELATED WORKS

The existing approaches for solving the IM problem can be divided into two main categories: approximation algorithms with provable guarantees, and metaheuristics. The former provide good results, but their long runtime make them impractical on large-scale networks. The latter compromise runtime with approximate solutions, although they suffer from curse of dimensionality. We provide below an overview of the related works from both categories.

3.1 Approximation algorithms

The IM problem was formulated for the first time as a combinatorial optimization problem in [20], and proven to be NP-hard. The authors also proposed a greedy algorithm that yields an $(1 - 1/e - \epsilon)$ approximation. The submodular property of the objective function was later used in the CELF algorithm [23], producing a 700x speedup w.r.t. a greedy approach. This property was further explored in CELF++ [13], with an additional 35-55% gain in runtime.

More recent algorithms, namely TIM and TIM+ [27], and its improved version IMM [26], use a different approach to calculate the nodes with the maximum spread, based on Reverse Reachable (RR) sets. Each RR set is obtained by taking the set of nodes reached by the spread from one node (one for each RR set) and by putting them in an hypergraph as an hyperedge. The solution is then calculated iteratively by selecting the node with the highest degree in the hypergraph and removing then that node with all its incident edges. The difference between the three algorithms consists in the ways of calculating the sufficient number of RR sets and the method used to sample nodes for their generation. All three algorithms outperform both CELF and CELF++, while IMM outperforms both TIM and TIM+. A further improvement of IMM is BCT [16], specifically designed for cost-aware targeted viral marketing, where each node has a benefit and a cost associated to it. BCT uses this information in order to first sample nodes with higher benefit for creation of the RR sets, resulting in an algorithm significantly faster than IMM.

3.2 Metaheuristics

The first application of EAs to the IM problem was proposed in [3], in which a basic EA without any domain knowledge outperformed the greedy algorithm [20] in terms of runtime while obtaining comparable influence spread results. Since that, a plethora of improvements were done in this direction. In [29], for instance, a GPU-parallelized EA outperformed the greedy algorithm given the same runtime, taking up to 34 times less time to archive the same result. The impact of the EA parameters on the IM problem was studied instead in [30]. Another recent study [10] investigated the effect of two selection schemes used for solving the IM problem with EAs, and found that the (μ, λ) scheme performs better than $(\mu + \lambda)$ when the size of the seed set increases.

Other works have investigated the use of *smart initialization* of the initial EA population. In [31], different initialization strategies were compared, with PageRank-based initialization [25] giving the worst results. A smart initialization approach was also adopted in [11], that unlike standard high-degree initialization can guarantee a higher diversity of the solutions. In the same work, the authors used 2-hop spread [22] as approximated fitness function, and a similarity-based local search. Their results were quite promising, with a 10x speedup w.r.t. CELF and comparable influence results. In another work, the same authors proposed a discrete Particle Swarm Optimization algorithm (DPSO) [12], which made use of 2-hop spread approximation and a smart initialization based on degree discount. DPSO outperformed CELF++ in terms of runtime, while the influence results were similar. Smart initialization techniques were also studied in [21], where each node has a probability to be inserted into the initial population proportional to its degree, and [9], producing better results than the basic EA.

Custom genetic operators were used in [24], where an informed mutation operator was introduced, based on a neural network predicting which nodes to change in a solution using centrality metrics as inputs. In [32], the EA premature convergence problem was handled using multi-population competition with specific crossover and mutation operations among populations. In other studies the IM problem was formulated instead as a multi-objective problem. In [15] a multi-objective differential evolution was used to maximize influence while minimizing the information delivery cost. Similarly, a multi-objective EA was used in [4] to maximize influence while minimizing the size of the seed set.

4 METHODS

We now explain the main modifications we applied to the basic EA proposed in [3]. This algorithm uses a direct encoding in which each individual genotype is simply a vector of IDs of nodes in the social network graph, i.e., a seed set. Initialization is done by randomly selecting a seed set of k nodes from the graph. At each generation, parents are selected by performing tournament selection, which then reproduce by means of one-point crossover operator. Differently from [3], the crossover operator adopted here has two additional constraints: 1) to produce solutions without node repetitions, and 2) to produce solutions different from those given in input, in order to improve diversity. To satisfy the former condition, we swap only nodes which are not in common between the two parents, while maintaining the other nodes in common. Regarding the latter property, we simply force a mutation of at least one random node in each child produced by crossover. This is then followed by a random mutation that simply resets each node, with a given probability, to one of the other possible node IDs (excluding those already present in the seed set). The new population is then obtained with generational replacement with elitism. The evolution is continued until a maximum number of generations is reached, or stopped earlier if there is not any improvement for α consecutive generations, where α was set to 10% of the maximum number of generations. In our experiments, we set *population_size*=100, *generational_budget*=100 generations, *crossover_rate*=1.0, *mutation_rate*=0.1, *tournament_size*=5, and *num_elites*=1. This parametrization follows the one used in [3],

to obtain a fair comparison with the basic EA. A specific parameter tuning, as done in [30], is instead out of the scope of this work.

In order to improve the basic EA, we collected the best ideas from the literature, discussed in the previous Section, and put them in practice by adapting them to the algorithm. As we will see in the next Section, some of them were more successful, while some others hardly produced any improvement. Nevertheless, we report all of our results in order to provide valuable knowledge on what are the most promising directions. In a nutshell, the objectives of our work are to: 1) reduce the runtime of the fitness function, the main bottleneck of the EA, by using fitness function approximations; 2) boost the EA convergence with smart initialization; 3) improve the search efficiency by using graph-aware mutations, also including a mechanism for dynamic selection of multiple mutations; 4) reduce the EA search space by filtering the most promising nodes before starting the search. Next, we discuss these four elements in detail.

4.1 Fitness function approximations

The influence spread computation under the IC and WC models was proven to be #P-complete in [13], and as such it is one of the main bottlenecks for the IM algorithms. This is usually calculated by using **Monte Carlo** (MC) sampling. According to this procedure, the influence spread is calculated by simulating the information spread in the graph, using the specified diffusion model, a given number of times (usually very large, > 10.000). The influence spread is then approximated by the mean spread across the simulations.

As seen earlier, an alternative to the expensive MC sampling is the use of RR sets [8]. A preliminary study of surrogate models was instead conducted in [5], but the results were contrasting. More promising solutions make use of spread function approximations, which are faster than MC sampling. This is the idea we have decided to adopt in this work. In particular, we use two different approximations. The first one is the **2-hop spread** approximation function introduced in [22], which as discussed earlier was used in [11] because of its efficiency. According to this approximation, the influence spread is given by:

$$\hat{\sigma}_S = \sum_{s \in S} \hat{\sigma}_{\{s\}} - \left(\sum_{s \in S} \sum_{c \in C_s \cap S} p(s, c) \left(\sigma_c^1 - p(c, s) \right) \right) - \chi,$$

where: $\chi = \sum_{s \in S} \sum_{c \in C_s \setminus S} \sum_{d \in C_c \cap S \setminus \{s\}} p(s, c) p(c, d)$, σ_u^1 is the one-hop influence spread of node u , defined as $\sigma_u^1 = 1 + \sum_{c \in C_u} p(u, c)$, and C_u denotes the set of neighbors of node u .

The second approximation is a variation of MC sampling, called **MC max-hop**. In this case, influence is propagated up to a maximum number of hops max_hop . If e.g. $max_hop = 2$, the influence is propagated up to the neighbors of the neighbors of the seed set S . The pseudo-code of the algorithm is shown in Algorithm 1, where setting max_hop to Inf produces the original MC sampling.

4.2 Smart initialization

Introducing custom-built solutions into the initial population is a common practice in Evolutionary Computation [6, 7, 18]. As we have seen earlier, several works [9, 11, 12, 21, 31] have reported a positive effect of smart initialization on the IM problem. Here we

Algorithm 1 Monte Carlo max-hop simulation.

input: seed set S , seed set size k , graph G , social model $model$, number of simulations $no_simulations$, maximum number of hops max_hop

output: spread mean value avg , spread standard deviation std

```

1: procedure MONTECARLOMAXHOPSIMULATION
2:    $sample \leftarrow initialize\_array(k)$ 
3:    $i \leftarrow 1$ 
4:   for  $i \leq no\_simulations$  do
5:      $A \leftarrow S$ 
6:      $B \leftarrow S$ 
7:      $converged \leftarrow False$ 
8:      $j \leftarrow max\_hop$ 
9:     while not converged and  $j \leq max\_hop$  do
10:       $nextB \leftarrow ()$ 
11:      for  $n$  in  $B$  do
12:         $neighbors \leftarrow get\_neighbors(n, G)$ 
13:         $not\_activated \leftarrow set\_difference(neighbors, A)$ 
14:        for  $m$  in  $not\_activated$  do
15:           $spread\_prop \leftarrow get\_influence(n, m, model)$ 
16:           $p \leftarrow random\_real(0, 1)$ 
17:          if  $p > spread\_prop$  then
18:             $add\_element\_to\_set(m, nextB)$ 
19:           $B \leftarrow nextB$ 
20:          if  $is\_empty(nextB)$  then
21:             $converged \leftarrow True$ 
22:           $A \leftarrow union(A, B)$ 
23:           $j \leftarrow j + 1$ 
24:           $sample[i] \leftarrow length(A)$ 
25:           $i \leftarrow i + 1$ 
26:    $avg \leftarrow mean(sample)$ 
27:    $std \leftarrow standard\_deviation(sample)$ 

```

consider different initialization techniques, described below.

Single smart solution: A single “smart” solution is inserted into the initial population, that is generated by taking the first k nodes obtaining the highest centrality scores. We tested the following centrality metrics:

- betweenness: measures how often the node is on the shortest path among any two nodes in the graph;
- closeness: corresponds to the mean length of the shortest path between the node and all the other nodes in the graph;
- degree: the number of out-going links of the node;
- eigenvector: nodes neighboring with few highly-connected nodes score higher;
- katz: a variant of the eigenvector metric, where the distance between two nodes is measured by considering the number of possible paths among them, instead of only the shortest path.

Multiple smart solutions: A percentage of the initial population is initialized according to one of the following strategies:

- degree random: the probability of a node to be inserted into a solution is proportional to its degree;

- degree random ranked: the probability of a node to be inserted into a solution is proportional to its ranking position w.r.t. the nodes' degree;
- community degree: the general idea is that the network might have *community structure* and a good solution should contain the most influential nodes from the largest communities. Here, we use the Louvain algorithm [28] to perform community detection. The solutions are then built as follows: first, we select a community, with probability proportional to its size. Then, we select the node within the selected community, with probability proportional to its degree.

When using the initialization with both single and multiple smart solutions, the rest of the population is initialized randomly.

4.3 Graph-aware mutations

We attempt to accelerate the search process by introducing some knowledge about the node properties and graph structure into the mutation operator. In particular, we considered four different mutation schemes, described below.

Global mutation methods: We mutate the selected node to another node randomly chosen from the graph. Table 1 recapitulates the node selection techniques we use with the global mutation methods.

Local mutation methods: The idea is to perform a *local search* on the graph, i.e., to apply a mutation that preserves a certain proximity to the current solution (i.e., the seed set). In this case the node to be mutated is selected randomly, while the new node is chosen according to one of the criteria listed in Table 2.

Combination of multiple mutations: We have observed that some mutations are more effective than others when used in different evolution phases and when applied to networks with different degree distributions. Here, we model the dynamic selection of which mutation to use within a pool of available mutations as a non-stationary *Multi Armed Bandit* (MAB) problem. To solve the MAB problem, we adopt the Upper Confidence Bound (UCB1) algorithm [1], which promotes actions (in our case, mutations) with larger reward uncertainty. More precisely, to select the next action, the UCB1 algorithm maximizes the following expression: $Q(a) + exploration_weight \cdot \sqrt{(2 \log t) / N_t(a)}$, where $Q(a)$ is the cumulative reward of the last n times the action a was selected (n is a sliding window size), $N_t(a)$ corresponds to the number of times a was chosen, and t is the counter of the total action selections. To shrink the learning phase, we adopted an exponentially decaying exploration weight $exploration_weight(g) = g^{-3}$, where g is the generation counter. Note that the sliding window is used since the problem is non-stationary, i.e., the reward of a certain mutation may change during the evolutionary process. This mechanism should then produce a fair trade-off between exploration and exploitation along the search.

4.4 Node filtering

The search space of the IM problem is combinatorial, therefore most existing algorithms scale poorly with the graph size. We investigate two ways to reduce the search space by filtering the nodes in the

Table 1: Selection techniques used with global mutations.

Mutation	Description
Global random	The node to be mutated is selected randomly.
Global low degree	The mutation probability of a nodes is inversely proportional to its degree.
Global low spread	The mutation probability of a node is inversely proportional to its influence spread value.
Global low additional spread	The mutation probability of a node is inversely proportional to the increase in influence spread it produces when added to the seed set excluding it.

Table 2: Selection techniques used with local mutations.

Mutation	Description
Local neighbors random	The new node is selected randomly among the neighbors of the node to be mutated.
Local neighbors second degree	The new node is selected among the neighbors of the node to be mutated, with probability proportional to its degree.
Local neighbors approximated spread	The new node is selected among the neighbors of the node to be mutated, with probability proportional to its approximated spread.
Local embeddings random	The new node is selected randomly among the nodes closest to the node to be mutated, according to its corresponding node2vec [14] embeddings (i.e., a continuous feature representation of the graph's nodes, trained to predict the probability of nodes being neighbors).

graph before applying the EA.

Min degree nodes: Nodes with low degree have a very little chance of influencing the others, so it is very unlikely that they would be part of the optimal solution. Here, we filter the n nodes with degree larger than a given threshold.

Best spread nodes: When considering large-scale networks, it is very likely that the optimal seed set would be made of nodes placed far enough from each other, to avoid activating the same nodes. According to this assumption, it is highly probable that the nodes in the optimal solution are isolated among them. In other words, their activations do not overlap and the resulting spread is very close to the sum of the spreads of each of the nodes calculated separately. To test this assumption, we first compute the spread generated by each node, separately, and then filter the best n nodes in terms of mean spread. This technique requires the spread measures of any two nodes to be comparable with each other, in order to determine which one is bigger. Furthermore, this technique is computationally expensive since it requires to perform MC simulations for *all the nodes* in the network, before running the EA. However, deciding *a priori* the required number of MC simulations needed to have a statistically significant comparison of the spread produced by all nodes (averaged across multiple MC simulations, as seen earlier) is not possible. Therefore, we use the following procedure that increases iteratively the number of MC simulations, until a needed precision is reached:

- (1) Initialize a node set \mathcal{N} with all the nodes in the graph, and set the maximum error rate max_error_rate to a given value.

- (2) Perform MC sampling of the spread of each node in \mathcal{N} with a number of MC simulations (which might be different for each node) needed to ensure a maximum error rate max_error_rate on the mean spread. The error of the mean is estimated by calculating the confidence interval of the Student’s T distribution.
- (3) Set \mathcal{N} to the best n nodes in terms of mean spread, plus all the nodes in the graph which are not statistically comparable with the node with the lowest spread among the best n nodes. Decrease max_error_rate and go to step 2.

This loop is repeated until all the best n nodes are comparable with the others. However, the sample size required to compare any pair of nodes would be too costly, thus we stop the loop as soon as the number of best nodes falls in a given range $[l, u]$. More specifically, as soon as the algorithm finds a number of best nodes l , and the number of nodes incomparable with those l nodes is lower than $u - l$, it stops. When the algorithm terminates, it returns the l best nodes, plus all the nodes which cannot be compared with them.

Table 3: Real-world datasets specifications.

Dataset	Graph size	Node degree				
		Avg	Std	Min	Max	Median
Wiki-vote	7.115 nodes 103.689 edges	14.57	42.28	0	893	2
Amazon	262.111 nodes 1.234.877 edges	4.71	0.95	0	5	5
CA-GrQc	5.242 nodes 14.496 edges	5.52	7.92	1	81	3

Table 4: Synthetic Barabási-Albert datasets specifications. The n_edges parameter indicates the number of edges added from a new node to the existing nodes.

Graph size	Node degree				
	Avg	Std	Min	Max	Median
1.000 nodes $n_edges=1$	1.99	3.51	1	75	1
1.000 nodes $n_edges=3$	5.98	7.37	3	99	4
1.000 nodes $n_edges=5$	9.95	10.57	2	114	7
1.000 nodes $n_edges=7$	13.90	12.80	7	137	10
1.000 nodes $n_edges=9$	17.84	15.65	9	148	12
1.000 nodes $n_edges=11$	21.76	19.25	11	211	15
10.000 nodes $n_edges=1$	1.99	4.13	1	243	1
10.000 nodes $n_edges=3$	5.99	8.96	3	293	4
10.000 nodes $n_edges=5$	9.99	13.45	2	365	7
10.000 nodes $n_edges=7$	13.99	17.00	7	446	10
10.000 nodes $n_edges=9$	17.98	21.25	9	470	12
10.000 nodes $n_edges=11$	21.98	25.97	11	695	15

5 EXPERIMENTS

For the experiments, we used three real-world graphs taken from the SNAP repository [19], namely Wiki-Vote, Amazon and CA-GrQc. In Wiki-Vote the edges represent who-voted-whom information in elections for promoting adminship. The Amazon dataset contains co-purchased products relations. The smallest dataset, CA-GrQc, is a network of collaboration in the General Relativity and

Quantum Cosmology fields. Table 3 contains a detailed description of each of the aforementioned graphs. For testing the function approximations, we also used 6 synthetic datasets generated by the Barabási-Albert model [2], detailed in Table 4. These were created using the networkx library [17] with a *random_seed* parameter set to 0. Except for the function approximations experiments, where we set $p = 0.1$ in the IC model and $max_hop = 2$, in all the other experiments we set $p = 0.01$ and $max_hop = 3$. Each of the four proposed modifications presented above (fitness function approximations, smart initialization, graph-aware mutations, and node filtering) was tested separately. Every experimental condition was repeated for 10 independent runs (in the next figures, we provide mean values \pm std. dev.)¹.

5.1 Fitness function approximations

The first part of the experimentation was aimed at verifying which fitness function approximation works best in terms of runtime and quality of results. To do that, we compared the runtime and spread (mean and std. dev. across multiple simulations) computed by MC sampling and the two approximations, i.e., 2-hop spread and MC max-hop ($max_hop = 2$), on the different datasets above, under both the IC and WC models. For each dataset and model, the three methods were executed on 100 randomly generated seed sets, the same for all the methods. The Pearson correlation on the spread values calculated by the proposed approximations and MC sampling was then calculated.

Generally, we observed that both approximations in some cases had an important gain in runtime and high spread correlations w.r.t. MC sampling. The correlation of the spread values is high for both approximations (> 0.9) for the WC model, and for the IC model with low p values ($p \leq 0.1$). On the other hand, we observed a dependency of the runtime on the size and connectivity of the graph, the size of the seed set k , and, of course, the number of simulations used for the MC methods. Figure 1 shows the runtime of the three methods on six selected datasets as the number of MC simulations varies (WC model, seed set size $k = 5$). Here we can observe the linear growth of the runtime of the MC methods w.r.t the number of simulations. We can also notice that in some cases the 2-hop spread is slower not only when the number of MC simulations is low (somehow an obvious result), but also when the dataset size grows. Another thing which can be noticed is that MC max-hop is particularly useful as the dataset connectivity increases. See the Barabási-Albert graphs in the figure, where MC max-hop obtains a lower performance gain in the top row (lower connectivity) than in the bottom row (higher connectivity). We observed this trend in all our experiments: more connected graphs yield a longer influence spread process since usually more nodes get influenced, so a truncation of this process up to some level (as in MC max-hop) can lead to larger runtime gains.

We also observed that the runtime has, obviously, a linear relationship with the seed set size k . Figure 2 reports the runtime as a function of k (WC model, number of MC simulations set to 100). Here we can observe that the 2-hop spread may be convenient only if the graph size is kept small, while for the real-world cases it might be the slowest spread evaluation method.

¹Our code is available at: <https://github.com/katerynak/Influence-Maximization>.

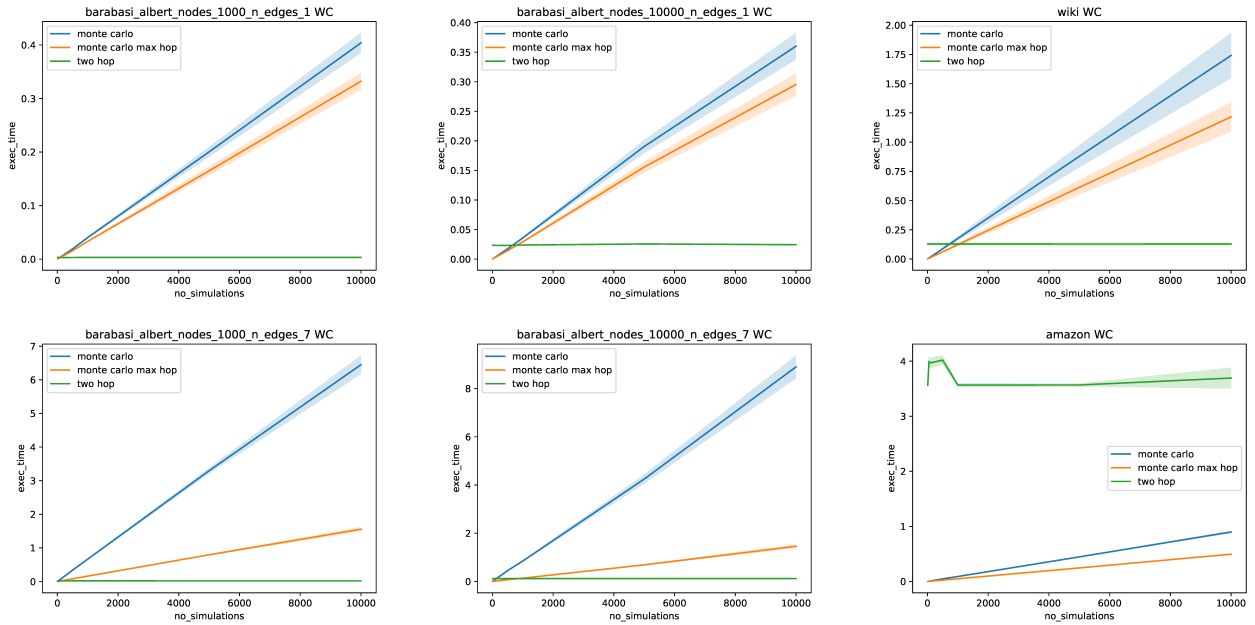


Figure 1: Spread function runtime as a function of the number of MC simulations.

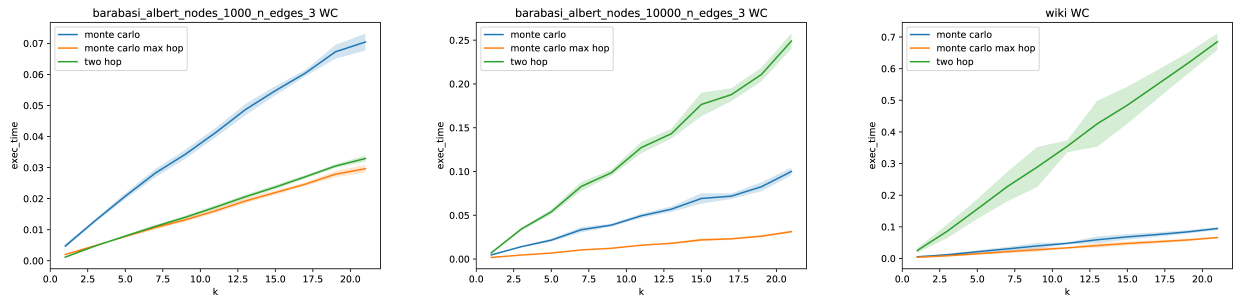


Figure 2: Spread function runtime as a function of the seed set size k .

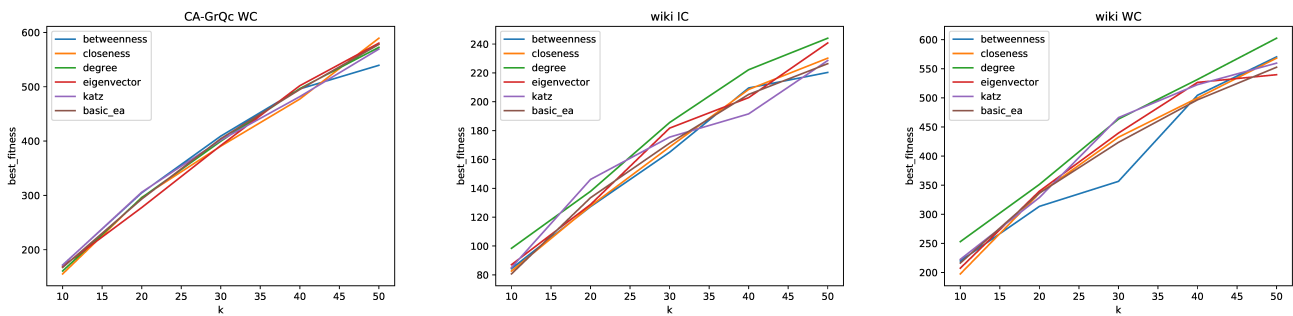


Figure 3: Single smart solution initialization techniques compared.

5.2 Smart initialization

We tested smart initialization only on the three real-world datasets. Overall, the results indicate that smart initialization is most useful

in the case of datasets that contain single, highly connected nodes.

Single smart solution: We were not able to run complete experiments for the Amazon dataset, given their cost in memory and/or runtime (we did not consider centrality metrics with computation time > 12 hours). As shown in Figure 3, the *degree* metric is a clear winner for the Wiki-Vote dataset, which is reasonable considering its degree distribution. CA-GrQc experiments did not present a clear difference between various metrics under the IC model, while when considering the WC model the *betweenness* metric obtained slightly better results w.r.t. the other metrics. The only metric we could compute for the Amazon dataset was the *degree* metric, but this did not bring any improvement to the basic EA (results not shown for brevity).

Multiple smart solutions: Due to computing resource limitations, we could not run these experiments for different values of k . We used instead a fixed seed set size, $k = 10$, and a percentage of smart solutions of 50%. From Table 5, it can be seen that there is not a clear winner. On the other hand the *degree random* strategies are to be preferred because of the much lower computation time requirement w.r.t. the *community degree* strategy.

Table 5: Multiple smart solutions initialization techniques compared (for $k = 10$).

Dataset	Model	Best fitness (averaged across 10 runs)			
		Degree random	Comm. degree	Degree random ranked	No smart. init.
Wiki-Vote	WC	226.24	223.35	230.65	220.75
Wiki-Vote	IC	93.7	93.21	94.04	87.92
Ca-GrQc	WC	147.38	147.36	146.91	146.13
Ca-GrQc	IC	18.64	18.59	18.59	18.67
Amazon	WC	66.55	65.99	67.97	67.87
Amazon	IC	10.79	10.83	10.80	10.82

5.3 Graph-aware mutations

Also in this case we tested only the Wiki-vote, CA-GrQc and Amazon graphs. We omit for brevity the detailed results, but we report below our main findings.

Global mutations: The *global low degree* mutation obtained slightly better results on the graphs presenting a low number of large degree nodes, such as Wiki-Vote and CA-GrQc, while for the Amazon dataset it led to worse results. The runtime of these methods were in general comparable, except the *global low additional spread* method whose runtime grows linearly with the seed set size.

Local mutations: We report the results of the local mutations in Figure 4. For this specific experiment, we trained node2vec on the Amazon dataset and fine-tuned its parameters. As expected then, the *local embeddings random* mutation performed better on the Amazon dataset, while its results for CA-GrQc and Wiki-Vote were way worse than the other mutations: this shows that local embeddings may work well, but at a cost of an expensive and time-consuming tuning process. On the other hand, the *local neighbors second degree* mutation was at least as good as the *global random* mutation for all the datasets, while the *local neighbors approximated*

spread and the *local neighbors random* mutation worked well on CA-GrQc and Wiki-Vote, but not on the Amazon dataset.

Combination of multiple mutations: We tested the UCB1 algorithm with all the mutations in Tables 1-2 and a sliding window size of 100. We noted that while the improvement w.r.t. the basic EA was not very significant, it was however positive for all graphs, which confirms that the dynamic selection of mutations adapts well to different cases (results not shown for brevity).

5.4 Node filtering

Filtering was tested only on the real-world graphs. Below we report our findings.

Min degree nodes: Min-degree values of 1, 2, 3 and 4 were tested. We obtained a visible improvement w.r.t. the basic EA only on the Wiki-Vote dataset, while in the other cases the high std. dev. did not permit to define a clear winner.

Best spread nodes: We used MC max-hop with $max_hop = 2$ to approximate the spread of each single node; max_error_rate was set initially to 0.8 and subsequently decreased of 0.1 at each iteration. We set $l = 10^9$ and $u = 10^{11}$, in order to have the same search space size regardless of k . From Figure 5, we can see that this method significantly improved the basic EA (up to 40%).

6 CONCLUSIONS

We improved a basic EA applied in previous research to the IM problem with various graph-aware enhancements aimed at reducing the algorithm runtime. Using MC max-hop instead of MC sampling to evaluate the influence spread permitted a significant computational time saving, and allowed the possibility to conduct a variety of experiments. The most important progress was achieved by limiting the search space of the EA by means of node filtering, a method which selected the most promising nodes in terms of information spread before running the evolutionary search: the limited number of node combinations allowed in fact the EA to better scale with the increasing graph size. Moreover, the combination of several local and global graph-aware mutations permitted the EA to adapt to graphs with different structure or connectivity.

There are, on the other hand, some limitations in the proposed mechanisms. For instance, MC max-hop may be not a good choice in case of networks with scarce connectivity or under the IC model with high probability values. Furthermore, node filtering is valid only under the assumption that the most influential nodes are distant, in the sense that they do not influence the same nodes. So this method may perform worse on small-scale networks where the optimal seed set may include nodes with common spread influence. Additionally, the runtime of the metrics which involve the computation of the influence spread, as well as the degree-based mutations, is proportional to the fitness of the solution. So, the runtime of the algorithm is proportional to the graph connectivity and a large number of fitter candidates found during the search may imply an increasingly high computation cost.

There is still further research to do on graph-aware mutations, which would probably work better with a small number of pre-filtered nodes. In particular, the success of the node2vec embeddings mutation (although with some hyper-parameter search) suggests

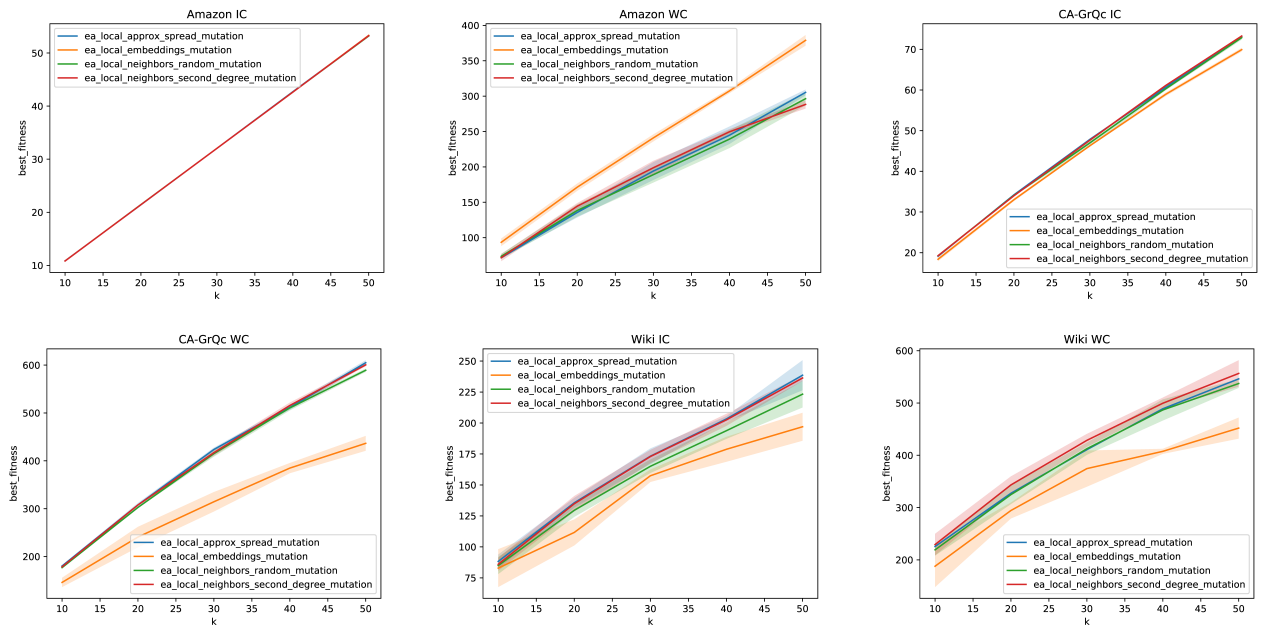


Figure 4: Local mutations compared.

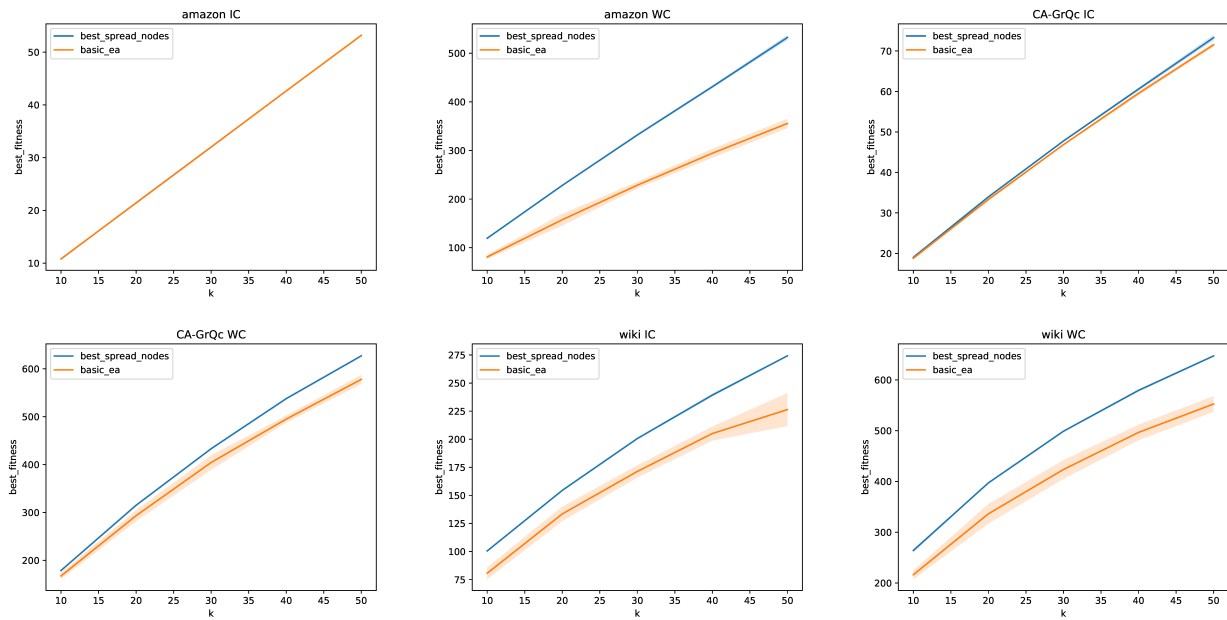


Figure 5: Best spread node filtering compared.

that this might be a promising direction to explore more in depth, also because differently from the other tested mutations this kind of mutation does not need node-specific expensive computations.

REFERENCES

- [1] Auer, Peter and Cesa-Bianchi, Nicolo and Fischer, Paul. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.
- [2] Barabási, Albert-László and Albert, Réka. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512.
- [3] Bucur, Doina and Iacca, Giovanni. 2016. Influence Maximization in Social Networks with Genetic Algorithms. In *European Conference on the Applications of Evolutionary Computation (LNCS, Vol. 9597)*. Springer, Berlin, Heidelberg, 379–392.
- [4] Bucur, Doina and Iacca, Giovanni and Marcelli, Andrea and Squillero, Giovanni and Tonda, Alberto. 2017. Multi-objective Evolutionary Algorithms for Influence Maximization in Social Networks. In *European Conference on the Applications of Evolutionary Computation (LNCS, Vol. 10199)*. Springer, Cham, 221–233.
- [5] Bucur, Doina and Iacca, Giovanni and Marcelli, Andrea and Squillero, Giovanni and Tonda, Alberto. 2018. Evaluating Surrogate Models for Multi-Objective Influence Maximization in Social Networks. In *Genetic and Evolutionary Computation Conference Companion (GECCO)*. ACM, New York, NY, USA, 1258–1265.
- [6] Caraffini, Fabio and Iacca, Giovanni and Neri, Ferrante and Picinali, Lorenzo and Mininno, Ernesto. 2013. A CMA-ES super-fit scheme for the re-sampled inheritance search. In *Congress on Evolutionary Computation*. IEEE, Piscataway, NJ, USA, 1123–1130.
- [7] Caraffini, Fabio and Neri, Ferrante and Cheng, Jixiang and Zhang, Gexiang and Picinali, Lorenzo and Iacca, Giovanni and Mininno, Ernesto. 2013. Super-fit multi-criteria adaptive differential evolution. In *Congress on Evolutionary Computation*. IEEE, Piscataway, NJ, USA, 1678–1685.
- [8] Christian Borgs and Michael Brautbar and Jennifer Chayes and Brendan Lucier. 2014. Maximizing Social Influence in Nearly Optimal Time. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. ACM, New York, NY, USA, 946–957.
- [9] da Silva, Arthur Rodrigues and Rodrigues, Rodrigo Ferreira and da Fonseca Vieira, Vinicius and Xavier, Carolina Ribeiro. 2018. Influence Maximization in Network by Genetic Algorithm on Linear Threshold Model. In *International Conference on Computational Science and its Applications (ICCSA)*. Springer, Cham, 96–109.
- [10] García-Nájera, Abel and Zapotecas-Martínez, Saúl and Bernal-Jaquez, Roberto. 2020. Selection Schemes Analysis in Genetic Algorithms for the Maximum Influence Problem. In *Mexican International Conference on Artificial Intelligence (MICAI)*. Springer, Cham, 211–222.
- [11] Gong, Maoguo and Song, Chao and Duan, Chao and Ma, Lijia and Shen, Bo. 2016. An Efficient Memetic Algorithm for Influence Maximization in Social Networks. *IEEE Computational Intelligence Magazine* 11, 3 (2016), 22–33.
- [12] Gong, Maoguo and Yan, Jianan and Shen, Bo and Lijia, Ma and Cai, Qing. 2016. Influence Maximization in Social Networks Based on Discrete Particle Swarm Optimization. *Information Sciences* 367 (2016), 600–614.
- [13] Goyal, Amit and Lu, Wei and Lakshmanan, Laks V.S. 2011. CELF++: Optimizing the Greedy Algorithm for Influence Maximization in Social Networks. In *International Conference Companion on World Wide Web (Hyderabad, India) (WWW)*. ACM, New York, NY, USA, 47–48.
- [14] Grover, Aditya and Leskovec, Jure. 2016. Node2vec: Scalable Feature Learning for Networks. In *International Conference on Knowledge Discovery and Data Mining (San Francisco, California, USA) (KDD)*. ACM, New York, NY, USA, 855–864.
- [15] Guo, Jian-bin and Chen, Fu-zan and Li, Min-qiang. 2019. A Multi-objective Optimization Approach for Influence Maximization in Social Networks. In *International Conference on Industrial Engineering and Engineering Management (IEEM)*. Springer, Singapore, 706–715.
- [16] H. T. Nguyen and M. T. Thai and T. N. Dinh. 2017. A Billion-Scale Approximation Algorithm for Maximizing Benefit in Viral Marketing. *IEEE/ACM Transactions on Networking* 25, 4 (2017), 2419–2429.
- [17] Hagberg, Aric and Swart, Pieter and Chult, Daniel. 2008. Exploring Network Structure, Dynamics, and Function Using NetworkX. In *Python in Science Conference*. SciPy, Pasadena, CA USA, 11–15.
- [18] Iacca, Giovanni and Mallipeddi, Rammohan and Mininno, Ernesto and Neri, Ferrante and Suganthan, Pannuthurai Nagaratnam. 2011. Super-fit and population size reduction in compact differential evolution. In *Workshop on Memetic Computing*. IEEE, Piscataway, NJ, USA, 1–8.
- [19] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. <http://snap.stanford.edu/data>.
- [20] Kempe, David and Kleinberg, Jon and Tardos, Éva. 2003. Maximizing the Spread of Influence Through a Social Network. In *International Conference on Knowledge Discovery and Data Mining (Washington, D.C.) (KDD)*. ACM, New York, NY, USA, 137–146.
- [21] Krömer, Pavel and Nowaková, Jana. 2017. Guided Genetic Algorithm for the Influence Maximization Problem. In *International Computing and Combinatorics Conference (COCOON)*. Springer, Cham, 630–641.
- [22] Lee, Jong-Ryul and Chung, Chin-Wan. 2014. A Fast Approximation for Influence Maximization in Large Social Networks. In *International Conference on World Wide Web (Seoul, Korea) (WWW)*. ACM, New York, NY, USA, 1157–1162.
- [23] Leskovec, Jure and Krause, Andreas and Guestrin, Carlos and Faloutsos, Christos and Faloutsos, Christos and VanBriessen, Jeanne and Gance, Natalie. 2007. Cost-effective Outbreak Detection in Networks. In *International Conference on Knowledge Discovery and Data Mining (San Jose, California, USA) (KDD)*. ACM, New York, NY, USA, 420–429.
- [24] Michalak, Krzysztof. 2018. Informed Mutation Operator Using Machine Learning for Optimization in Epidemics Prevention. In *Genetic and Evolutionary Computation Conference (Kyoto, Japan) (GECCO)*. ACM, New York, NY, USA, 1294–1301.
- [25] Rodrigues, Rodrigo and Silva, Arthur and Vieira, Vinicius and Xavier, Carolina. 2018. Optimization of the Choice of Individuals to Be Immunized Through the Genetic Algorithm in the SIR Model. In *International Conference on Computational Science and its Applications (ICCSA)*. Springer, Cham, 62–75.
- [26] Tang, Youze and Shi, Yanchen and Xiao, Xiaokui. 2015. Influence Maximization in Near-Linear Time: A Martingale Approach. In *International Conference on Management of Data (Melbourne, Victoria, Australia) (SIGMOD)*. ACM, New York, NY, USA, 1539–1554.
- [27] Tang, Youze and Xiao, Xiaokui and Shi, Yanchen. 2014. Influence Maximization: Near-Optimal Time Complexity Meets Practical Efficiency. In *International Conference on Management of Data (Snowbird, Utah, USA) (SIGMOD)*. ACM, New York, NY, USA, 75–86.
- [28] Vincent D Blondel and Jean-Loup Guillaume and Renaud Lambiotte and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (2008), P10008.
- [29] Weskida, Michal and Michalski, Radosław. 2016. Evolutionary Algorithm for Seed Selection in Social Influence Process. In *International Conference on Advances in Social Networks Analysis and Mining (Davis, California) (ASONAM)*. IEEE, Piscataway, NJ, USA, 1189–1196.
- [30] Weskida, Michal and Michalski, Radosław. 2018. Finding Influentials in Social Networks using Evolutionary Algorithm. *Journal of Computational Science* 31 (2018), 77–85.
- [31] Xavier, Carolina and Vieira, Vinicius and Evsukoff, Alexandre. 2016. Populational Algorithm for Influence Maximization. In *International Conference on Computational Science and its Applications (ICCSA)*. Springer, Cham, 346–357.
- [32] Zhang, Kaiqi and Du, Haifeng and Feldman, Marcus. 2017. Maximizing influence in a social network: Improved results using a genetic algorithm. *Physica A: Statistical Mechanics and its Applications* 478 (2017), 20–30.