Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 363–371
DOI: 10.15439/2022F78
ISSN 2300-5963 ACSIS, Vol. 30

Boosting a Genetic Algorithm with Graph Neural Networks for Multi-Hop Influence Maximization in Social Networks

Camilo Chacón Sartori
Artificial Intelligence Research Institute (IIIA-CSIC)
Campus of the UAB, Bellaterra, Spain
Email: cchacon@iiia.csic.es

Christian Blum
Artificial Intelligence Research Institute (IIIA-CSIC)
Campus of the UAB, Bellaterra, Spain
Email: christian.blum@iiia.csic.es

***Abstract*—In this paper we solve a variant of the multi-hop influence maximization problem in social networks by means of a hybrid algorithm that combines a biased random key genetic algorithm with a graph neural network. Hereby, the predictions of the graph neural network are used with the biased random key genetic algorithm for a more accurate translation of individuals into valid solutions to the tackled problem. The obtained results show that the hybrid algorithm is able to outperform both the biased random key genetic algorithm and the graph neural network when used as standalone techniques. In other words, we were able to show that an integration of both techniques leads to a better algorithm.**

I. Introduction

Social networks form part of our daily lives. Whether a person is an artist or an engineer, a student or an academic, young or old, or a spartan troll, the person most likely is embedded in one or more social networks. People use social networks to communicate through audiovisual media or text messages, to help others, or even to attack them. In other words, people *influence* other people, either in a positive or in a negative way. Note that the world's leading technology companies invest huge amounts of money in advertising in social networks. For any social network it is essential to be aware of the transmission of information and its impact.

The identification of a group of users who can influence as many people as possible is a problem called influence maximization (IM). This problem was defined by Kempe et al. in 2003 [1]. By solving the IM problem, a set of adequate people can be identified, for example, for spreading the news about a certain product on a social network. In this way, the dissemination of the product can be supported and eventually maximized. The IM problem has been studied, for example, in the context of dealing with emerging negative opinions [2], social advertising [3], and influence maximization on Twitter for marketing campaigns [4]. Also, different variations of the IM problem have been studied. One example concerns the case of social networks that have a diversity of communities, which implies that there are different types of users who can influence in different ways [5]. Another example is the one of trying to maximize influence in time-evolving social networks [6].

A social network can be viewed as a (directed) graph in which the users are the nodes and user interactions are modeled by arcs. Furthermore, the propagation of influence is often simulated by models such as the independent cascade (IC) model and the linear threshold (LT) model, which might be deterministic or probabilistic. In any case, both models consider one-hop coverage, that is, if a person is covered depends exclusively on its direct neighbors. Although it is common to consider one-hop coverage models in IM problems, the interaction in social networks may also be of multi-hop nature [7].

In this paper we tackle a variant of the IM problem which can be seen as a variant of the classical minimum dominating set problem (MDSP) in a directed graph $G = (V, A)$. In the MDSP, the task is to identify a set of nodes $U \subseteq V$ of minimum cardinality such that for each node $v \in V$ the following holds: either (1) $v \in U$, or (2) it exists at least one node $v' \in U$ such that $(v', v) \in A$, where (v', v) is the directed arc from v' to v. In other words, the classical MDSP considers one-hop coverage. In contrast, in the problem tackled in this paper—known as the multi-hop influence maximization problem (k-dDSP)—d-hop coverage is considered. More specifically, in the k-dDSP the task is to find a set $U \subseteq V$ of cardinality k such that the set $C_U \subseteq V$ of nodes that are covered (or influenced) by U is of maximal cardinality. Hereby, a node v forms part of C_U—that is, v is said to be covered (or influenced) by U—if there exists a node $v' \in U$ such that the shortest directed path from v' to v consists of at most d arcs. As an example, consider Figure 1, where $k = 2$ and the two nodes with a purple color form part of set U, that is, $U = \{v_4, v_5\}$. In case $d = 1$ it holds that $C_U = \{v_4, v_5, v_3, v_6, v_7\}$ because v_3, v_6 and v_7 are in 1-hop distance from a node in U. In other words, the objective function value of U is 5. In case $d = 2$, $C_U = \{v_4, v_5, v_3, v_6, v_7, v_2, v_8, v_{11}\}$ and the objective function value of U would be 8. Finally, in case $d = 3$, $C_U = V$ and the objective function value would be 11.

For a large-scale graph, such as a large social network, exact solutions to the k-dDSP are costly to compute. Therefore, researchers have focused on heuristic and on machine learning

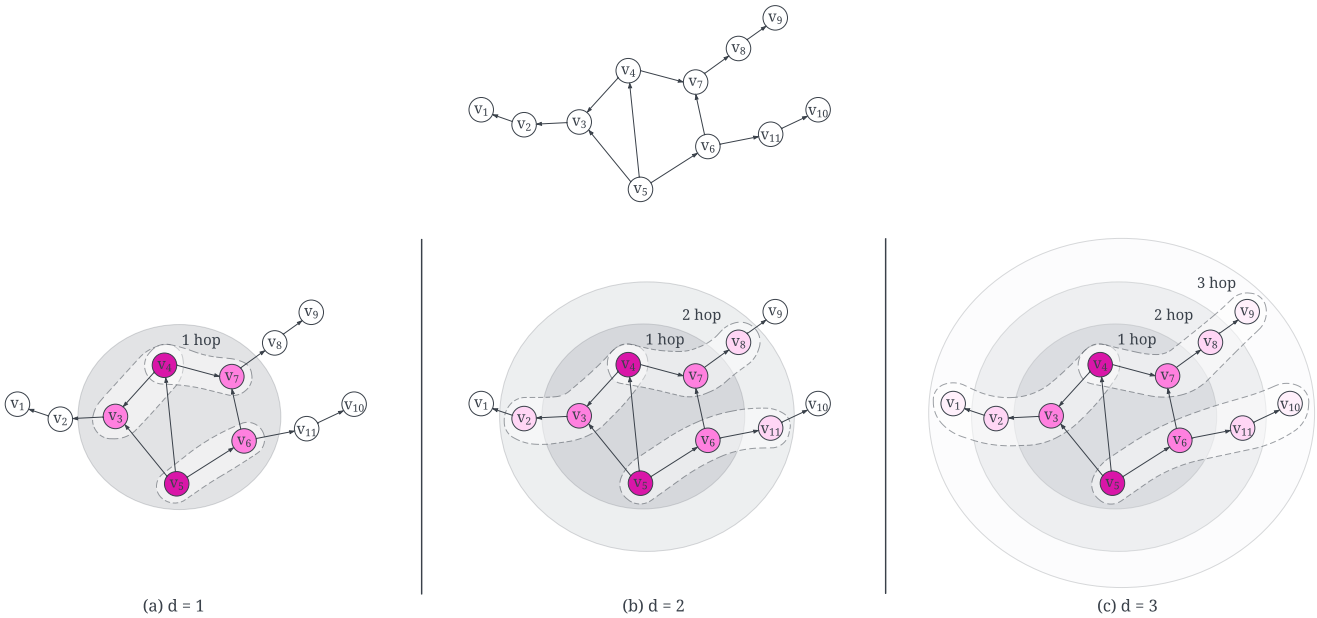

Fig. 1. **Multi-hop influence process**. Given is a directed graph with 11 nodes and 12 arcs (top). Let us assume the k-dDSP is solved with $k = 2$. The two purple nodes (v_4 and v_5) form part of the example solution U. If $d = 1$ (bottom left) then nodes $\{v_3, v_7, v_6\}$ are 1-hop covered by U. If $d = 2$ (bottom center) then nodes $\{v_2, v_3, v_7, v_8, v_6, v_{11}\}$ are 2-hop covered by U. Finally, if $d = 3$ (bottom right), then all remaining nodes of the graph are 3-hop covered by U.

techniques for solving this problem (see Section II). In this paper, we present a novel hybrid algorithm to solve k-dDSP. This algorithm is obtained through an integration of (1) a new graph neural network (GNN) called graph inverse attention network (GRAT) that incorporates the influence of the neighbourhood into the feature embedding of each node [8], and (2) a biased random key genetic algorithm (BRKGA) [9]. More specifically, the information provided by the GNN is used in the BRKGA as greedy information. Our algorithm is evaluated on real-world networks with up to 500.000 nodes and 1 million arcs. Experimental results prove that our hybrid method is at least as good as the two individual techniques in most cases. Therefore, our method is another example for the successful integration of a GNN framework with a metaheuristic to boost the metaheuristics' performance.

The article is organized as follows. Section II introduces prior and related work. In Section III, we provide a more technical description of the k-dDSP. In Section IV, we present our hybrid approach. In Section V, we compare and analyze our hybrid algorithm on real-world data sets. Finally, Section VI concludes the work with some discussions on the utilized type of hybridization.

II. Related Works

As mentioned before, most of the works on influence maximization (IM) in social networks make use of the independent cascade (IC) and the linear threshold (LT) diffusion models for calculating the influence of solutions. Chen et al. [10], [11] present a fined-tuned heuristic for generating scalable solutions to the IM problem and an improved runtime in comparison to previous approaches. Jung et al. [12] provide an even faster heuristic for the application to large graphs. Goyal et al. [13] added Monte Carlo simulation in order to obtain an improved heuristic.

Multi-hop influence is a way of measuring the influence of a group of people (set of nodes) in IM problems that has been studied recently. Nguyen et al. [14] proposed a heuristic which takes into account the probability of each node in the network for contributing to a high influence spread. However, the proposed algorithm is only evaluated on small graphs of less than 30.000 nodes. With respect to large graphs, Nguyen et al. [15] presented an alternative heuristic that consists of three phases: pre-optimization to reduce the size of the graph, a construction phase to build a k-dominating set, and post-optimization by removing redundant nodes from the set. This algorithm improves in speed over the one from [16]. However, by doing so it sacrifices performance for a reduction of computation time.

Recently, machine learning techniques have entered the scene to solve combinatorial problems [17]. An early example is S2V-DQN, which is a general reinforcement learning (RL) framework for combinatorial optimization problems in graphs proposed by Khalil et al. [18]. It uses graph neural networks (GNNs) for graph embedding with partial solutions and a *deep Q-network* (DQN) for node selection. This framework is more and more used by the community working with learning techniques for combinatorial optimization.

In the same line, FastCover is a very recent unsupervised

learning framework for solving the k-dDSP by Ni et al. [8]. It uses a multi-layer GNN known as *graph reversed attention network* (GRAT) for generating for each node of a given graph a probability to belong to the optimal solution. The output of FASTCOVER are the k nodes with the highest probability. The authors of [8] show that FASTCOVER outperforms the existing heuristics.

III. PROBLEM DEFINITION

Many optimization problem in social networks can be formalized by modeling the social network as a directed graph $G = (V, A)$, where V is the set of nodes and A is the set of directed arcs present in the graph. This is also the case for the multi-hop influence maximization problem tackled in this paper, denoted by k-dDSP.

The most important concept in this context is the one of the *influence* $I_d(u) \subseteq V$ of a node $u \in V$, which depends on two things:

1) Parameter $d \geq 1$, which is part of the problem input.
2) A distance measure $dist(u, v)$ between nodes. In the context of this paper, $dist(u, v)$ is defined as the length—in terms of the number or arcs—of the shortest directed path from u to v in G.

With this we can provide the definition of $I_d(u)$ as follows:

$$I_d(u) := \{v \mid dist(u, v) \leq d\} \tag{1}$$

In other words, $I_d(u)$ is the set of all nodes of G that can be reached from u by means of a directed path with at most d arcs. We say that u influences (or covers) all nodes from $I_d(u)$. This definition can naturally be extended to sets of nodes in the following way:

$$I_d(U) := \bigcup_{u \in U} I_d(u) \quad \forall U \subseteq V \tag{2}$$

That is, $I_d(U)$ is the set of all nodes of G that are influenced by at least one node from U.

Valid solutions to the k-dDSP are all sets $U \subseteq V$ such that $|U| \leq k$, that is, any valid solution may consists of at most k nodes. The goal of the k-dDSP is to find a valid solution $U^* \subseteq V$ such that $|I_d(U^*)| \geq |I_d(U)|$ for all valid solutions U to the problem. In other words, the objective function value of a valid solution U is $|I_d(U)|$. In technical terms,

$$\begin{aligned} \max_{U \subseteq V} \quad & |I_d(U)| \\ \text{s.t.} \quad & |U| \leq k \end{aligned} \tag{3}$$

Finally, note that the k-dDSP was proven to be NP-hard in [8], [19].

IV. METHODOLOGY

In this section, we present a novel hybrid algorithm that emerges from the integration between a BRKGA and a GNN framework for solving the k-dDSP in social networks. To begin, we briefly introduce both methods individually. Then we present the developed hybridization strategy.

Algorithm 1 The pseudo-code of BRKGA

Require: a directed graph $G = (V, E)$
Ensure: values for params. p_{size}, p_e, p_m, $prob_{elite}$, $seed$
1: $P \leftarrow$ GENERATEINITIALPOPULATION$(p_{size}, seed)$
2: EVALUATE(P) ▷ dependent part (greedy)
3: **while** computation time limit not reached **do**
4: $P_e \leftarrow$ ELITESOLUTIONS(P, p_e)
5: $P_m \leftarrow$ MUTANTS(P, p_m)
6: $P_c \leftarrow$ CROSSOVER$(P, p_e, prob_{elite})$
7: EVALUATE$(P_m \cup P_c)$ ▷ dependent part (greedy)
8: $P \leftarrow P_e \cup P_m \cup P_c$
9: **end while**
10: **return** Best solution in P

A. Biased Random Key Genetic Algorithm

We implemented a Biased Random Key Genetic Algorithm (BRKGA), which is a well-known GA variant for combinatorial optimization. In general, a BRKGA is problem-independent because it works with populations of individuals that are vectors of real numbers (random keys). The problem-dependent part of each BRKGA deals with the way in which individuals are translated into solutions to the tackled problem. The problem-independent pseudo-code of BRKGA is provided in Algorithm 1.

In the following, we first describe the independent or generic part of the algorithm. It starts by invoking function GenerateInitialPopulation(p_{size}, $seed$), which generates a population P formed by p_{size} individuals. In case $seed = 0$, all p_{size} individuals are randomly generated. Hereby, each individual $\pi \in P$ is a vector of length $|V|$, where V is the set of nodes from the input graph. For this purpose, the value at position i of π, denoted by $\pi(i)$, is chosen uniformly at random from $[0, 1]$, for all $i = 1, \ldots, |V|$. In case $seed = 1$, only $p_{size} - 1$ individuals are randomly generated. The last individual is obtained by defining $\pi(i) := 0.5$ for all $i = 1, \ldots, |V|$. Next, the individuals from the initial population are evaluated. This means, each individual $\pi \in P$ is transformed into a valid solution U_π to the k-dDSP, and the value $f(\pi)$ of π is defined as follows: $f(\pi) := |U_\pi|$. The transformation of individuals to valid solutions is discussed below.

Then, at each iteration of the algorithm, the operations to be performed are as follows. First, the best $\max\{\lfloor p_e \cdot p_{size} \rfloor, 1\}$ individuals are copied from P to P_e in function EliteSolutions(P, p_e). Second, a set of $\max\{\lfloor p_m \cdot p_{size} \rfloor, 1\}$ so-called mutants are generated and stored in P_m. These mutants are random individuals generated in the same way as the random individuals from the initial population. Finally, a set of $p_{size} - |P_e| - |P_m|$ individuals are generated by crossover in function Crossover$(P, p_e, prob_{elite})$ and stored in P_c.

Each such individual is generated as follows: (1) an elite parent π_1 is chosen uniformly at random from P_e, (2) a second parent π_2 is chosen uniformly at random from $P \setminus P_e$, and (3) an offspring individual π_{off} is generated on the basis of π_1 and

π_2 and stored in P_c. In the context of the crossover operator, value $\pi_{off}(i)$ is set to $\pi_1(i)$ with probability $prob_{elite}$, and to $\pi_2(i)$ otherwise. After generating all new offspring in P_m and P_c, these new individuals are evaluated in function Evaluate(); see line 7. Note that the individuals in P_e are already evaluated. Finally, the population of the next generation is determined to be the union of P_e with P_m and P_c.

The evaluation of an individual (see lines 2 and 7 of Algorithm 1) is the problem-dependent part of our BRKGA algorithm. The function that evaluates an individual is often called the *decoder*. In our case, we make use of a simple greedy heuristic which is based on the intuition that nodes with a higher degree (number of neighbors) are more likely to have a high influence than nodes with a lower degree. Hereby, the set of neighbors $N(v_i)$ of a node $v_i \in V$ is defined as follows: $N(v_i) := \{v_j \in V \mid (v_i, v_j) \in A\}$, that is, neighbors of v_i are only those nodes that can be reached via a directed arc from v_i. The greedy value $\phi(v_i)$ of each $v_i \in V$ is defined as follows:

$$\phi(v_i) := |N(v_i)| \cdot \pi(i) \tag{4}$$

In other words, the greedy value of a node v_i is obtained by multiplying the degree of v_i with the numerical value found at position i of the individual to be translated into a solution. Subsequently, solution U_π is obtained by adding the k nodes with the highest greedy values.

Note that, in Section IV-C, greedy function ϕ will be modified in order to obtain a hybrid algorithm.

B. Graph Neural Network Framework

The general objective of a graph neural network (GNN) [20]–[22] is to automatically find patterns in data. In contrast to more classical deep learning techniques, GNNs directly work on graphs. Therefore, they can be used to make predictions about nodes, arcs, or subgraphs without the need for unnecessary transformations of the graph. The crucial idea of GNNs is to iteratively update so-called node representations by combining the representations of a nodes' neighbors with its own representation. Given a graph $G = (V, A)$, $H^l \in \mathbb{R}^{|V| \times C}$ are node attribute matrices, one for each layer $l \in \{0, 1, \ldots, L\}$ of the GNN. Note that C is hereby the number of chosen features. Each line in such a matrix is a representation for the respective node. The final goal of a GNN is to learn competent node representations in these matrices.

In order to adapt/train the representations to be useful for a specific task, there are two actions that are successively performed at each GNN layer: (1) *Aggregate*, which aggregates all the information from the neighbors of each node, and (2) *Combine*, which updates the node representations by combining the aggregated information from the neighbors with the current node representation. Based on this, the general framework of a GNN can be specified as follows:

$$a_v^l = \textsc{Aggregate}^l\{H_u^{l-1} \ : \ u \in N(v)\}$$
$$H_v^l = \textsc{Combine}^l\{H_v^{l-1}, a_v^l\}$$

where $N(v)$ is the set of neighbors of node v. H^K is the node representation matrix for each layer. Once the training process finishes, the final representations can be used for making predictions.

A GNN can be trained, for example, in order to make predictions about the probability of each node to belong to the optimal solution of the k-dDSP. In fact, as mentioned already in the section on related work, such a GNN approach was recently presented in [8]. This GNN—called FastCover (FC)—is an unsupervised GNN framework. FC can be characterized as follows: (1) the features of all nodes are embedded as vector spaces, and the direction of each arc is reversed, (2) a multi-layer GNN known as *graph reversed attention network* (GRAT) assigns each node to its value within $[0, 1]$, and (3) the representations of the GNN are optimized in the training stage through a differentiable loss function over all nodes scores.

The GRAT layer is the heart of FC. In particular, in contrast to a standard *graph attention network* (GAT) [23], the so-called attention mechanism is integrated at the origin nodes instead of the destination nodes. The central idea is that the nodes with more influence are likely to receive a stronger reward. This means a higher probability of getting a potential score.

C. The Hybrid BRKGA Algorithm

Our hybrid algorithm—henceforth called BRKGA+FC—starts with two offline steps. Given a network in which the k-dDSP must be solved, first, all node probabilities are extracted from the trained FC model; note that the prior training process is described in the next section. This probability is denoted by $p_i \in (0, 1]$ for each $v \in V$. Then, the original greedy function $\phi()$ from Eq. 4 is replaced by the following one that incorporates the node probabilities extracted from FC:

$$\phi_{FC}(v_i) := |N(v_i)| \cdot \pi(i) \cdot p_i \quad \forall v_i \in V \tag{5}$$

The hypothesis is that good/correct predictions will bias the algorithm towards the area in the search space in which an optimal solution is located, or, at least, solutions of very high quality. Moreover, we expect the probabilities obtained from FC to undo the bias introduced by the degree of a node, which might sometimes be misleading. The integration process is also shown in Figure 2.

V. Experimental Evaluation

This section is divided into three parts. First, we will describe the preparation of the data for training and evaluation, and the parameter tuning procedure. Then, the experimental setting and the numerical results of three algorithms will be presented (FC, BRKGA, and BRKGA+FC). In this context note that FC can, of course, be used as a standalone technique by simply adding the k nodes with highest probabilities to the solution. Finally, we will analyse the algorithms graphically by means of so-called search trajectory networks.

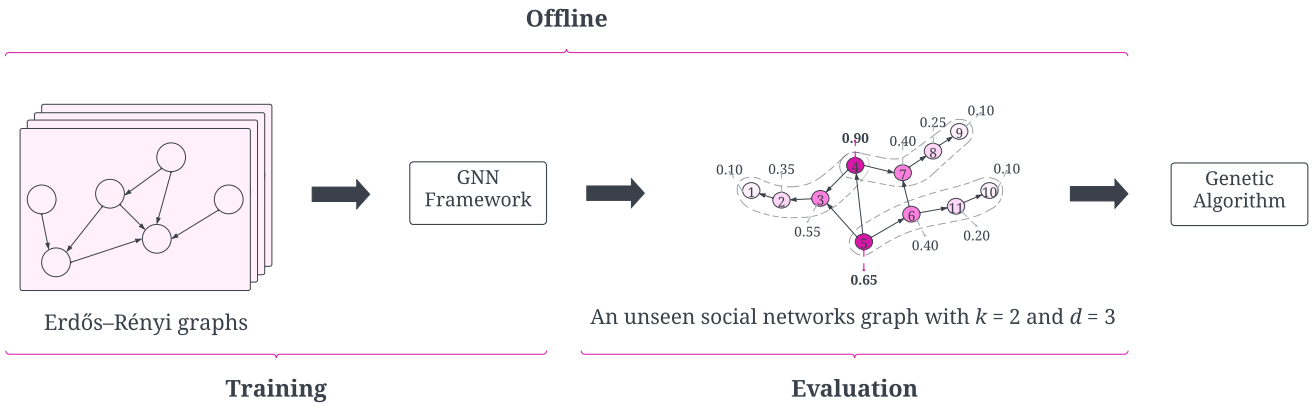

Fig. 2. **Hybridization Process.** The integration of BRKGA with FC starts with two offline steps concerning FC as follows. The training phase begins by using 15 random graphs (Erdős–Rényi). This provides us with a trained version of FC (called GNN Framework in the graphic). Then, the social network in which the k-dDSP is to be solved is presented to FC, which returns probabilities for all nodes of the network to belong to the optimal solution. Finally, the final phase consists of integrating these probabilities into the BRKGA (called Genetic Algorithm in the graphic).

TABLE I
TUNING CONFIGURATION. FINAL PARAMETER SETTING FOR BRKGA AND BRKGA+FC (FOR $k \in \{32, 64, 128\}$)

		BRKGA			**BRKGA+FC**		
		k			k		
Parameters	**Tuning domain**	**32**	**64**	**128**	**32**	**64**	**128**
P_{size}	[50, 250]	113	162	132	183	198	137
P_e	[0.1, 2.0]	0.17	0.24	0.25	0.19	0.22	0.2
P_m	[0.3, 5.0]	0.27	0.22	0.14	0.3	0.21	0.21
$prob_{elite}$	[0.01, 0.1]	0.6	0.59	0.58	0.57	0.67	0.67
$seed$	{0, 1}	0	0	0	1	1	1

A. Data Preparation and Tuning Process

We decided to execute experiments for three different values of k, that is, $k \in \{32, 64, 128\}$. For this reason we trained 3 different FC models, one for each value of k. Figure 3 illustrates that each model uses the fixed-parameter $d = 1$. In other words, the same FC model is used for applications of FC and BRKGA+FC for all $d \in \{1, 2, 3\}$. This was done for reducing the computational burden. Nevertheless, in the analysis of the final results we will see that this had some influence on the quality of the node probabilities extracted from the FC models, that is, these probabilities seem to loose accuracy with a growing value of d.

The three FC models (for each value of k) were trained as follows. First, we used 15 Erdős–Rényi graphs [24] with 4000 nodes each, similar to what is presented by the authors of [8]. After the training phase, the probabilities for all 19 social networks used later for the final experimental evaluation are extracted (for each value of $d \in \{1, 2, 3\}$) and stored in text files.

In order to ensure a fair experimental evaluation, both BRKGA and BRKGA+FC were tuned for each value of k using 10 test graphs. In particular, we used Erdős–Rényi graphs[1] with $n = 25.000$ nodes and an arc probability of $p = 10/n$. The tuning was done using a well-known tool called `irace` [25]. The considered parameter domains together with their finally chosen values are provided in Table I. Note that the number of nodes (25.000) of the test graphs corresponds to approximately the average number of nodes in the networks used for the final experimental evaluation (presented in Section V-B). The size of the tuning graphs is reasonable because the population size parameter in BRKGA is highly dependent on the size of the graphs. In the case of FC, we do not modify the parameters and the configuration as described in [8].

Note that the training phase of FC and the parameter tuning procedure for BRKGA and BRKGA+FC were performed with random graphs to maintain generality.

B. Experimental Evaluation

In this subsection, we apply all three approaches—FC, BRKGA and BRKGA+FC—to 19 real-world social networks from the SNAP library [26]. Each of these networks is a directed, unweighed graph. The sizes of these graphs are provided in Table II (columns $|V|$ and $|A|$).

We use three different values of $k \in \{32, 64, 128\}$. Also for d, the multi-hop influence parameter, we used three different values: $d \in \{1, 2, 3\}$. Note that $k = 64$ was used for the experimental evaluation on social networks of FC [8]. In order to provide a broader experimentation and analysis we also considered two additional values of k: a smaller one (32) and a larger one (128). The reason to not consider values of d greater than 3 is that we were not able to observe substantial differences to the case $d = 3$.

[1]Instances can be downloaded from the repository https://github.com/camilochs/genetic-algorithm-with-gnn

TABLE II
NUMERICAL RESULTS OBTAINED BY FC, THE BRKGA, AND OUR HYBRID ALGORITHM BRKGA+FC ON 19 WELL-KNOWN SOCIAL NETWORKS. FOR EACH NETWORK THE ALGORITHMS WERE APPLIED FOR $d \in \{1, 2, 3\}$ AND $k \in \{32, 64, 128\}$. FOR $k = 32$ BRKGA+FC WINS IN 73% OF THE CASES; FOR $k = 64$ IN 71%; AND FOR $k = 128$ IN 66%.

				$k = 32$			$k = 64$			$k = 128$		
Instance	$\|V\|$	$\|E\|$	**Distance**	**FC**	**BRKGA**	**BRKGA+FC**	**FC**	**BRKGA**	**BRKGA+FC**	**FC**	**BRKGA**	**BRKGA+FC**
			1	2338	2464.13	**2469.13**	2865	2948.67	**2948.90**	3313	3340.17	**3372.33**
advogato	6551	51332	2	4069	**4139.83**	4132.30	4153	4206.83	**4207.77**	4220	**4266.97**	4251.13
			3	4268	**4279.67**	4275.47	4275	**4281.80**	4280.00	4277	**4301.07**	4284.00
			1	8556	8566.80	**8570.70**	**9045**	8981.40	9002.83	**9650**	9537.60	9626.47
anybeat	12645	67053	2	11104	11177.10	**11205.63**	11209	11300.53	**11305.83**	11384	11371.47	**11400.77**
			3	11507	11527.17	11526.00	11515	**11531.00**	11530.33	**11556**	11542.27	11546.87
			1	1266	1714.33	**1808.67**	1954	2483.03	**2640.27**	3023	3448.00	**3711.63**
brightkite	56739	212945	2	4018	4160.63	**4671.50**	5444	5088.90	**5910.40**	6795	6075.13	**6891.07**
			3	6094	5699.57	**6535.13**	7530	6349.90	**7614.10**	**8650**	7178.47	8189.63
			1	8522	10860.00	**10864.83**	12431	**15793.53**	15792.90	19483	21044.43	**21170.37**
delicious	536108	1365961	2	21119	22341.80	**22481.57**	26018	26909.07	**26995.33**	32248	32811.27	**33414.13**
			3	32000	33041.13	**33175.07**	36112	36039.43	**36328.03**	40309	40418.77	**41722.63**
			1	1093	**1503.83**	1482.30	2117	**2649.93**	2637.90	3950	4557.10	**4565.73**
douban	154908	327162	2	4147	**6809.23**	6743.50	6801	9516.80	**9594.53**	10583	12950.27	**13093.53**
			3	11686	13988.10	**14448.00**	15938	17548.57	**17866.60**	20720	21277.43	**22368.23**
			1	1532	1753.27	**1774.70**	2198	2333.10	**2413.17**	3019	3000.47	**3170.93**
epinions	26588	100120	2	3645	3711.43	**3853.17**	4271	4086.47	**4416.07**	**4904**	4549.13	4897.77
			3	4500	4487.73	**4634.20**	4948	4661.37	**5002.83**	**5430**	4973.10	5334.60
			1	1998	3296.13	**3384.73**	3415	4869.60	**5122.30**	5553	6976.07	**7403.00**
gowalla	196591	950327	2	7509	9723.47	**10754.63**	10734	12534.07	**13628.10**	15386	14888.50	**17251.67**
			3	14247	14913.00	**16657.80**	18418	17386.73	**18966.73**	**23692**	19064.10	22800.10
			1	17498	**18077.00**	17896.00	22138	**22496.93**	22167.90	23543	**23628.00**	23567.00
gplus	23628	39242	2	21277	**23077.20**	22726.63	23200	**23562.93**	23172.73	**23628**	**23628.00**	**23628.00**
			3	21636	**23271.37**	22884.60	23271	**23559.80**	23169.00	**23628**	**23628.00**	**23628.00**
			1	8778	9041.33	**9047.27**	11232	11719.57	**11722.57**	14749	15058.97	**15128.27**
loc-brightkite	58228	214078	2	37295	38212.10	**38267.47**	40161	**41258.13**	41190.97	42929	43777.03	**43827.50**
			3	52335	52645.00	**52744.00**	53272	**53600.17**	53469.27	53783	54123.80	**54134.60**
			1	7162	**8087.00**	**8087.00**	11362	11999.33	**12003.93**	17046	17514.80	**17524.33**
sign-Slashdot081106	77350	516575	2	37521	42221.13	**42352.03**	44291	47782.33	**47827.97**	47747	**51839.47**	51796.63
			3	59456	**60393.67**	60367.30	60709	**61148.47**	61148.07	61333	61683.10	**61701.40**
			1	7232	8127.87	**8128.00**	11385	12094.27	**12108.50**	16949	17592.80	**17613.43**
sign-Slashdot090216	81867	545671	2	39841	43723.57	43781.93	46021	49774.13	**49832.23**	49661	**54244.47**	54141.53
			3	62964	63840.83	63817.00	64209	64710.97	**64723.77**	64912	65375.13	**65399.37**
			1	7182	**8129.00**	**8129.00**	11421	**12129.03**	12126.33	17010	**17642.67**	17641.80
sign-Slashdot090221	82140	549202	2	39220	43869.37	**43982.53**	46410	**49972.23**	49968.17	49917	**54408.47**	54334.93
			3	62958	**64062.17**	64036.97	64473	64935.37	**64953.90**	65145	65589.77	**65598.23**
			1	3455	**3479.00**	**3479.00**	4010	4038.17	**4040.97**	4615	4595.70	**4617.97**
sign-bitcoinotc	5881	35592	2	5568	5631.97	**5632.60**	5645	**5715.37**	5715.20	5761	5761.83	**5781.17**
			3	5814	5838.00	**5838.03**	5834	5839.00	**5839.10**	**5844**	5842.00	**5844.00**
			1	17765	18690.03	**18693.50**	22933	23569.77	**23609.43**	28969	29052.87	**29284.87**
sign-epinions	131828	841372	2	56849	59372.80	**59411.70**	60208	62238.23	**62288.90**	63070	64153.33	**64309.37**
			3	70410	**70739.73**	70721.93	70829	71021.30	**71024.17**	71188	71245.93	**71309.97**
			1	3419	3722.70	**3791.37**	4683	5020.93	**5083.63**	6304	6515.80	**6683.03**
slashdot	70068	358647	2	7849	8302.53	**8958.23**	9534	9605.07	**9771.83**	11083	10802.73	**11486.17**
			3	10537	10527.87	**11223.53**	11699	11166.57	**11810.70**	12648	11964.73	**12660.03**
			1	7229	**8100.00**	**8100.00**	11460	12021.30	**12051.57**	17068	17514.53	**17515.70**
slashdot-zoo	79120	515581	2	37664	41741.40	**41848.30**	43850	**47347.50**	47346.50	47629	**51361.57**	51334.70
			3	58974	**59818.57**	59805.10	60176	60600.87	**60604.67**	60797	61135.87	**61140.97**
			1	32088	32320.00	**32320.00**	35567	35891.10	**35908.27**	39245	39137.20	**39258.97**
themarker	69413	1644849	2	51299	52117.30	**52127.60**	52100	52654.87	**52665.13**	52785	53016.10	**53093.40**
			3	53506	53598.40	53593.03	53575	53665.10	**53666.67**	53729	53754.67	**53758.90**
			1	14063	14548.57	**14549.53**	26981	27755.17	**27760.93**	48870	**51240.20**	50531.97
twitter-follows	404719	713319	2	52319	88442.93	**88620.80**	83380	117605.90	**117692.57**	85371	**151365.57**	135258.13
			3	171902	**207181.27**	204973.27	203499	213702.93	**214154.77**	212524	219099.80	**221819.07**
			1	2146	2167.00	**2176.70**	2227	2265.63	**2268.63**	2306	**2367.70**	2366.47
wiki-elec	7118	107071	2	2328	2354.73	**2355.10**	2341	**2390.00**	2388.03	2366	**2454.50**	2427.53
			3	2331	2357.10	**2357.27**	2344	2389.53	**2389.67**	2366	**2452.23**	2426.47

As FC is a deterministic approach (at least for what concerns the use of the model after training), it was applied exactly once to each of the 19 networks, for each combination of k and d. On the contrary, both BRKGA and BRKGA+FC were applied 30 times to each network and combination of k and d. As a computation time limit of BRKGA and BRKGA+FC we used 900 CPU seconds for each run. All experiments were performed on machines with an Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz. The FC framework uses Python 3 and our implementations of BRKGA and BRKGA+FC were coded in C++.

In Table II we compare the results of FC with the average results of BRKGA and BRKGA+FC over 30 runs. The following observations can be made:

- Generally, both BRKGA and BRKGA+FC outperform FC which, in turn, was shown in [8] to outperform the existing heuristics for solving the k-dDSP. The only exceptions are 1 case with $k = 64$ and 9 cases with $k = 128$. This suggests that the performance of FC (when compared to the BRKGA versions) improves with an increasing value of k.
- Even though BRKGA generally outperforms FC, the

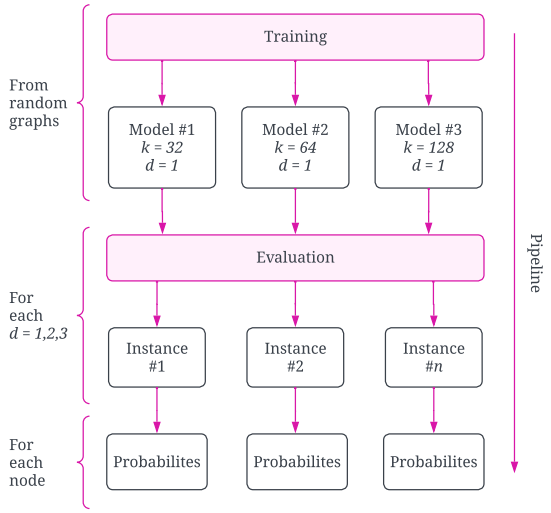

Fig. 3. **Data preparation and pipeline.** The pipeline starts by training three FC models, one for each $k \in \{32, 64, 128\}$. Random graphs (Erdős–Rényi) were used for this purpose. Next, the evaluation of the FC models is performed for each of the 19 instances (social networks), for each value of parameter $d \in \{1, 2, 3\}$. Finally, the obtained probabilities (FC output) are exported and stored in text files.

hybrid algorithm BRKGA+FC generally benefits from the use of the probability information extracted from FC for the translation of individuals into solutions. This advantage of BRKGA+FC over BRKGA is greatest for the smallest value of d (that is, $d = 1$). In this case BRKGA+FC outperforms BRKGA in 73% of all cases.

- The worst performance of BRKGA+FC is obtained for $k = 32$ and $d = 3$ (47% of superiority). This may be due to two possible reasons: (1) FC might find it difficult to detect pattern for rather small values of k; (2) all our FC models were trained for $d = 1$, which might suggest that our results could be improved by specifically training FC for each value of d.

Summarizing, we can say that making use of information from the GNN framework FC within our BRKGA clearly improves the algorithm.

C. Analysis

There are cases when our hybrid algorithm does not perform as expected. This is the case when the results are similar to the ones of BRKGA, or when they are even worse than the ones of BRKGA. In an attempt to analyse such cases we used the Search Trajectory Networks (STNs) tool from [27], which allows to visualize the trajectories of algorithms in the search space. Moreover, it lets us compare the behavior of more than one metaheuristic. For this analysis we chose three network corresponding to three different cases as outlined in Figure 4. The obtained graphics allow to make the following observations.

1) Figure 4 (a). This is a case in which the hybrid algorithm BRKGA+FC does not perform well in comparison to BRKGA. We can see in the graphic that both algorithms are clearly focused on different areas of the search space. In particular, BRKGA is attracted by a certain area of the search space. Nevertheless, the best solution found (red dot), even though it belongs to this part of the search space, it is not close to the area of attraction (see the two larger grey triangles). One hypothesis is that the probabilities provided by the graph neural network framework (FC) for this instance are rather misleading.
2) Figure 4 (b). In this case, the performance of both algorithms is comparable. Again, the two algorithm version are focused on different areas of the search space. This time there is a minimal overlap between two of the algorithm trajectories (see the light gray dot in the middle of the graphic). Interestingly, even though both algorithms find a best solution of the same quality, these two solutions are clearly different to each other (see the two red dots).

However, as mentioned before, in a majorty of cases BRKGA+FC outperforms BRKGA. Such a case is visualized in the graphic of Figure 4 (c). It can be observed that the trajectory of BRKGA+FC is more bounded and, therefore, it is not dispersed in the search space as it occurs for BRKGA. Moreover, the best solution is found in the area of the search space that attracts BRKGA+FC. This means that, in this case, the information provided by the FC is very useful.

VI. Conclusion and Future Work

In this work we have devised a hybrid algorithm combining a biased random key genetic algorithm with a graph neural network called FastCover. This was done in the context of an NP-hard combinatorial optimization problem dealing with the maximization of influence spreading in social networks. In particular, our hybrid algorithm makes use of the recommendations provided by FastCover (in the form of probabilities) for translating individuals to valid solutions to the tackled problem. The results have shown that, in a majority of the cases, our hybrid algorithm outperforms both its individual algorithmic components: the biased random key genetic algorithm and FastCover. The experimental evaluation of our approaches was done in the context of 19 real-world social networks.

One opportunity to advance this type of hybridization is to address other problems using a similar integration methodology, especially taking the recent progress of graph representation learning into account.

Acknowledgment

This paper was supported by grant PID2019-104156GB-I00 funded by MCIN/AEI/10.13039/501100011033.

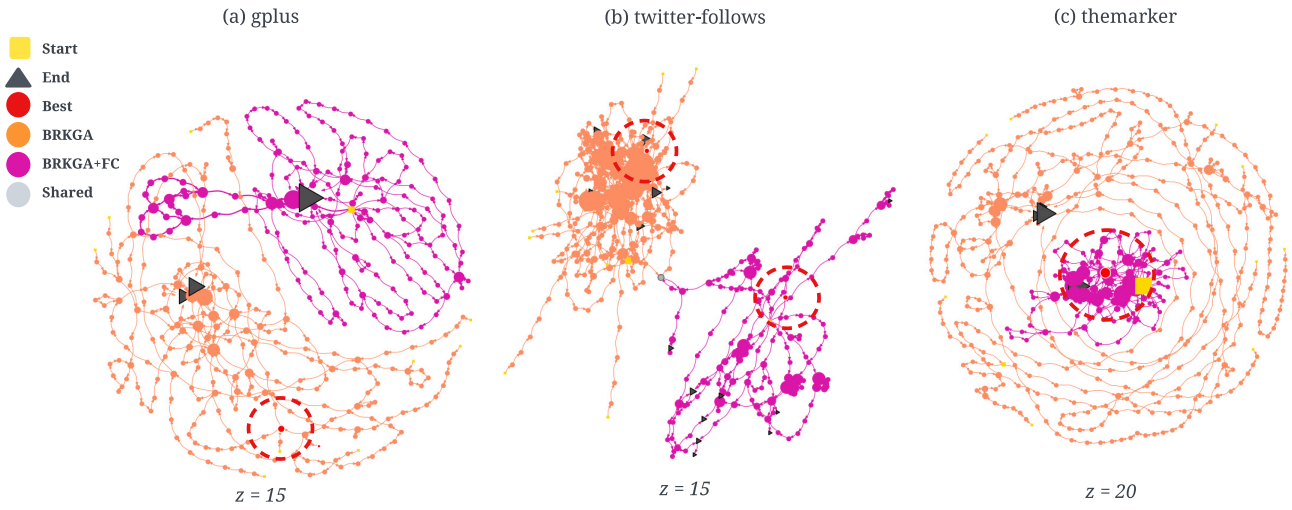

Fig. 4. **Search trajectory analysis concerning BRKGA and BRKGA+FC.** The three graphics show 10 executions (trajectories) of BRKGA (orange) and BRKGA+FC (pink) for three instances (gplus, twitter-follows, and themarker). The value of z indicates the degree of search space partitioning used to generate the graphics (see [27]). Yellow squares indicate the start of trajectories, while gray triangles indicate their ends. Also, light gray circles indicate that both algorithms passed through this location of the search space, while red circles indicate the best solutions found. (a) A case in which BRKGA is able to outperform BRKGA+FC (gplus). (b) A case in which BRKGA and BRKGA+FC achieve similar results (twitter-follows). (c) A case in which BRKGA+FC outperforms BRKGA (thermarker). For each graphic we used the force-directed layout based on physical analogies, not relying on any assumptions about the structure of the networks.

REFERENCES

[1] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 137–146. [Online]. Available: https://doi.org/10.1145/956750.956769

[2] W. Chen, A. Collins, R. Cummings, T. Ke, Z. Liu, D. Rincon, X. Sun, Y. Wang, W. Wei, and Y. Yuan, *Influence Maximization in Social Networks When Negative Opinions May Emerge and Propagate*, 2011, pp. 379–390. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611972818.33

[3] S. Tang, "When social advertising meets viral marketing: Sequencing social advertisements for influence maximization," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11306

[4] Y. Mei, W. Zhao, and J. Yang, "Influence maximization on twitter: A mechanism for effective marketing campaign," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/7996805/

[5] J. Li, T. Cai, K. Deng, X. Wang, T. Sellis, and F. Xia, "Community-diversified influence maximization in social networks," *Information Systems*, vol. 92, p. 101522, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437920300326

[6] H. Zhuang, Y. Sun, J. Tang, J. Zhang, and X. Sun, "Influence maximization in dynamic social networks," in *2013 IEEE 13th International Conference on Data Mining*, 2013, pp. 1313–1318. [Online]. Available: https://ieeexplore.ieee.org/document/6729640

[7] A. Goyal, W. Lu, and L. V. Lakshmanan, "Simpath: An efficient algorithm for influence maximization under the linear threshold model," in *2011 IEEE 11th International Conference on Data Mining*, 2011, pp. 211–220. [Online]. Available: https://ieeexplore.ieee.org/document/6137225

[8] R. Ni, X. Li, F. Li, X. Gao, and G. Chen, "Fastcover: An unsupervised learning framework for multi-hop influence maximization in social networks," 2021. [Online]. Available: https://arxiv.org/abs/2111.00463

[9] J. F. Gonçalves and M. G. C. Resende, "Biased random-key genetic algorithms for combinatorial optimization," *Journal of Heuristics*, vol. 17, no. 5, pp. 487–525, Oct 2011. [Online]. Available: https://doi.org/10.1007/s10732-010-9143-1

[10] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 199–208. [Online]. Available: https://doi.org/10.1145/1557019.1557047

[11] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *2010 IEEE International Conference on Data Mining*, 2010, pp. 88–97. [Online]. Available: https://ieeexplore.ieee.org/document/5693962

[12] K. Jung, W. Heo, and W. Chen, "Irie: Scalable and robust influence maximization in social networks," in *2012 IEEE 12th International Conference on Data Mining*, 2012, pp. 918–923. [Online]. Available: https://ieeexplore.ieee.org/document/6413832

[13] A. Goyal, W. Lu, and L. V. Lakshmanan, "Celf++: Optimizing the greedy algorithm for influence maximization in social networks," in *Proceedings of the 20th International Conference Companion on World Wide Web*, ser. WWW '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 47–48. [Online]. Available: https://doi.org/10.1145/1963192.1963217

[14] D.-L. Nguyen, T.-H. Nguyen, T.-H. Do, and M. Yoo, "Probability-based multi-hop diffusion method for influence maximization in social networks," *Wireless Personal Communications*, vol. 93, no. 4, pp. 903–916, Apr 2017. [Online]. Available: https://doi.org/10.1007/s11277-016-3939-8

[15] M. H. Nguyen, M. H. Hà, D. N. Nguyen, and T. T. Tran, "Solving the k-dominating set problem on very large-scale networks," *Computational Social Networks*, vol. 7, no. 1, p. 4, Jul 2020. [Online]. Available: https://doi.org/10.1186/s40649-020-00078-5

[16] Y. Wang, S. Cai, J. Chen, and M. Yin, "A fast local search algorithm for minimum weight dominating set problem on massive graphs," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18. AAAI Press, 2018, p. 1514–1522. [Online]. Available: https://www.ijcai.org/proceedings/2018/210

[17] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," 2018. [Online]. Available: https://arxiv.org/abs/1811.06128

[18] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," 2017. [Online]. Available: https://arxiv.org/abs/1704.01665

[19] P. Basuchowdhuri and S. Majumder, "Finding influential nodes in social networks using minimum k-hop dominating set," in *Applied Algorithms*, P. Gupta and C. Zaroliagis, Eds. Cham: Springer International Publishing, 2014, pp. 137–151. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-04126-1_12

[20] L. Wu, P. Cui, J. Pei, and L. Zhao, Eds., *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer, Singapore, 2022. [Online]. Available: https://link.springer.com/book/10.1007/978-981-16-6054-2

[21] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, jan 2021. [Online]. Available: https://doi.org/10.1109%2Ftnnls.2020.2978386

[22] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" 2018. [Online]. Available: https://arxiv.org/abs/1810.00826

[23] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017. [Online]. Available: https://arxiv.org/abs/1710.10903

[24] P. Erdos and A. Renyi, "On the evolution of random graphs," *Publ. Math. Inst. Hungary. Acad. Sci.*, vol. 5, pp. 17–61, 1960. [Online]. Available: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.5943

[25] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214716015300270

[26] J. Leskovec and R. Sosic, "Snap: A general purpose network analysis and graph mining library," 2016. [Online]. Available: https://arxiv.org/abs/1606.07550

[27] G. Ochoa, K. M. Malan, and C. Blum, "Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics," *Applied Soft Computing*, vol. 109, p. 107492, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494621004154