applied sciences

MDPI

Article

A Combinatory Framework for Link Prediction in Complex Networks

Paraskevas Dimitriou and Vasileios Karyotis *

Department of Informatics, Ionian University, 49100 Corfu, Greece; c20dimi@ionio.gr
* Correspondence: karyotis@ionio.gr; Tel.: +30-266-108-7768

Abstract: Link prediction is a very important field in network science with various emerging algorithms, the goal of which is to estimate the presence or absence of an edge in the network. Depending on the type of network, different link prediction algorithms can be applied, being less or more effective in the relevant scenarios. In this work, we develop a novel framework that attempts to compose the best features of link prediction algorithms when applied to a network, in order to have even more reliable predictions, especially in topologies emerging in Industrial Internet of Things (IIoT) environments. According to the proposed framework, we first apply appropriate link prediction algorithms that we have chosen for an analyzed network (basic algorithms). Each basic algorithm gives us a numerical estimate for each missing edge in the network. We store the results of each basic algorithm in appropriate structures. Then we provide them as input to a developed genetic algorithm. The genetic algorithm evaluates the results of the basic algorithms for each missing edge of the network. At each missing edge of the network and from generation to generation, it composes the estimates of the basic algorithms regarding each edge and produces a new optimized estimate. This optimization results in a vector of weights where each weight corresponds to the effectiveness of the prediction for each of the basic algorithms we have employed. With these weights, we build a new enhanced predictor tool, which can obtain new optimized estimates for each missing edge in the network. The enhanced predictor tool applies to each missing edge the basic algorithms, normalizes the basic algorithms' estimates, and, using the weights of the estimates derived from the genetic algorithm, returns a new estimate of whether or not an edge will be added in the future. According to the results of our experiments on several types of networks with five well-known link prediction algorithms, we show that the new enhanced predictor tool yields in every case better predictions than each individual algorithm, therefore providing an accuracy-targeting alternative in the existing state of the art.

Keywords: genetic algorithms; link prediction; strongly connected components; predictor tool; optimization; area under curve; precision

check for updates

Citation: Dimitriou, P.; Karyotis, V. A Combinatory Framework for Link Prediction in Complex Networks. *Appl. Sci.* **2023**, *13*, 9685. https://doi.org/10.3390/app13179685

Academic Editors: Dimitris Mourtzis and Stefan Schönig

Received: 16 June 2023
Revised: 25 July 2023
Accepted: 25 August 2023
Published: 27 August 2023

Copyright: © 2023 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

A network [1] consists of nodes and edges connecting pairs of nodes, representing/simulating many systems of the real world, such as biological, railway, social, and others [2]. In these networks, nodes represent various entities such as people, cities, bacteria, industrial machines, sensors, actuators, etc. The edges of the networks represent interactions between two nodes—entities, such as interpersonal relationships in the case of a social network, roads or railways or airways in the case of an air traffic network, communication between machines, sensors, and actuators in an industrial environment, etc.

A very important aspect concerning the various types of networks is link prediction, namely the task of estimating the likelihood of unobserved connections to be observed in the future in a network [3,4]. Similarly, link prediction may refer to predicting the probability of connection between two nodes in a network that has not yet exhibited the corresponding edge, by using known information about the network nodes and network structure [5].

Over the last 25 years, many works have addressed the link prediction problem [6]. Link prediction is applicable in many types of networks. For biological networks such as protein interaction networks, metabolic networks, and food webs, link prediction can greatly reduce experimental costs [7]; for social networking sites, link prediction plays an important role in improving user loyalty and disease prediction [8]. The link prediction algorithm can be applied to solve the classification problem in partially marked networks, such as distinguishing the research field of scientific publications [9]. In an industrial environment comprising an Internet of Things network [10] with numerous interacting sensors, actuators, and machines, link prediction can reveal required communications and critical feedback loops.

For different types of networks, more than one link prediction algorithm has been proposed, being more or less effective. When one applies two or more link prediction algorithms to a network, it is very likely that different estimation results will be obtained for the occurrence or absence of some edges in the future. Each algorithm evaluates the information it may have about the nodes of a network differently and often reaches different conclusions because it makes different estimates. It would be ideal to be able to have as many as possible of the correct estimates of two or more algorithms, i.e., to compose their results and optimize their predictions. In this way, their outcomes can be used in a complementary fashion.

In this work, we tried to take advantage of the abundance of algorithms that exist in the field of link prediction [6]. We propose a framework with which we can compose two or more link prediction algorithms, among the many that exist in the literature. With the composition of the selected link prediction algorithms, a new enhanced predictor tool is built which gives better predictions than those of the individual algorithms.

According to the framework we propose, in order to have the best possible link prediction, we adapt the enhanced predictor tool to a specific network type each time. This has the effect of taking into account the characteristics of the network and its attributes, which greatly contributes to having the most reliable link prediction possible.

The adaptation is carried out initially by selecting two or more suitable algorithms for link prediction for the specific type of network under analysis. For example, we will use different link prediction algorithms for social networks and different ones for electrical networks or industrial networks. The algorithms we have chosen, which we call basic algorithms, are applied each time to the selected network, after we have removed some of its original edges, in order to create ground truth topologies using the results of each basic algorithm (a value ranking for each missing edge in the network) via a genetic algorithm (GA) [11–13] we have developed. Then a vector of weights is produced, which contains the weight of the link prediction value that we give to each basic algorithm. The new enhanced predictor tool normalizes and weights the obtained values for each edge according to the weight vector and returns a new link prediction value. As we show in the evaluation tests, this value is more reliable than the value given by the best of the selected basic algorithms for each specific network type.

To test and evaluate the framework and the newly enhanced predictor tool, we have performed several experiments on different types of networks. For basic algorithms, we have chosen five well-known ones from the literature: `Resource Allocation Index`, `Adamic-Adar Index`, `Common Neighbor Centrality`, `Jaccard Coefficient`, and `Preferential Attachment` [14–21].

The remaining sections of the parer are organized as follows. In Section 2, we present information on the basic prediction algorithms we use in the paper, while in Section 3 we describe the approach we follow in order to create the proposed framework. In Section 4, we report in detail the steps we follow to build the new enhanced predictor tool, while in Section 5, we provide information about the genetic algorithm we have implemented and its various components. In Section 6, we provide analysis for the theoretical foundation of our approach, in Section 7, we describe the way in which we evaluate the proposed framework and the new enhanced predictor tool, and in Section 8, we provide the results

from the performed experiments in various networks. Finally, in Section 9, we discuss our work cumulatively, and in Section 10, we provide directions for future work.

2. Related Work

As mentioned before, the problem of link prediction is a well-known problem and there are many proposed algorithms in the literature. In this section, we present five of the most well-known and effective link prediction algorithms. Each of them relies on a different metric regarding the properties of network nodes and/or links to make link predictions, such as the information exchanged between the network nodes, the common neighbors, and the degree of the network nodes. From the experiments performed in Section 8, we observe that the yielded results are in some cases quite different, even for the same network. Our effort is in composing these results to obtain more reliable predictions.

`Resource Allocation Index` [14]: The Resource Allocation Index expresses the fraction of information that a network node can send to another node, through their common neighbors. The Resource Allocation Index of nodes X and Y is defined as

$$res_alloc(X,Y) = \sum_{u \in N(X) \cap N(Y)} \frac{1}{|N(u)|}, \tag{1}$$

where $N(u)$ denotes the neighborhood of node u.

According to (1), the Resource Allocation Index of nodes X, Y is the sum over the inverse of the node degree of all common neighbors of X and Y, $(N(X) \cap N(Y))$. One may conclude that the more common neighbors that nodes X and Y have, the higher the value of this index will be. The value of the index grows even more if the common neighbors of X and Y have a small degree, i.e., they have few edges to other nodes.

`Adamic-Adar Index` [15]: The Adamic–Adar Index is very similar to the Resource Allocation Index. The only difference is that rather than considering the degree of the common neighbors, the log of this degree is considered. The Adamic–Adar Index of nodes X and Y is defined as

$$adamic_adar(X,Y) = \sum_{u \in N(X) \cap N(Y)} \frac{1}{log(|N(u)|)}. \tag{2}$$

`Common Neighbor Centrality` [16–18]: It computes the Common Neighbor and Centrality-based Parameterized Algorithm's (CCPA) score of all node pairs in the common neighborhood. The Common Neighbor and Centrality-based Parameterized Algorithm's score of nodes X and Y is defined as

$$CCPA(X,Y) = a * (|N(X) \cap N(Y)|) + (1-a) * \frac{N}{d_{XY}}, \tag{3}$$

where a is a parameter that varies in the range $[0,1]$, N denotes the total number of nodes in the network, and d_{XY} denotes the shortest distance between X and Y. This algorithm is based on two vital properties of nodes, namely the number of common neighbors and their centrality [2].

`Jaccard Coefficient` [19,20]: The Jaccard Coefficient takes into account the number of common neighbors of two nodes, normalizing it by the total number of their cumulative neighbors. Thus, the Jaccard coefficient of nodes X and Y is defined as

$$jaccard_coeff(X,Y) = \frac{|N(X) \cap N(Y)|}{|N(X) \cup N(Y)|}. \tag{4}$$

`Preferential Attachment` [20,21]: According to the Preferential Attachment rule, the nodes that have a very high degree are more likely to attract more neighbors in the future. The Preferential Attachment rule for nodes X and Y is defined as

$$pref_attach(X,Y) = |N(X)| * |N(Y)|. \tag{5}$$

The intuition behind this measure is that if one looks at a pair of nodes and they both have a very high degree, they are more likely to be connected to each other in the future.

The above approaches for link prediction are a small indicative subset of the vast relevant research. There are many different link prediction algorithms, owing to the different application objectives and underlying network topologies. There are link prediction algorithms for PPI [22] (protein–protein interaction) networks [23–26], for social networks [3,15,27], for recommendation systems [28], etc.

Our framework incorporates the results of the above link prediction algorithms and also employs a genetic algorithm approach. It differs from previous works in that it combines their inputs through the genetic algorithm to determine importance weights for the outcomes of the basic algorithms according to the underlying topology, thus dynamically adapting their contribution the the final prediction result. Genetic algorithms [11–13] are meta-heuristic stochastic algorithms that simulate the biological process of evolution. With genetic algorithms, we try, among other things, to address the difficulty in optimizing problems that have a huge number of states and are otherwise difficult to solve by conventional methods or algorithms [13,29].

A genetic algorithm for a particular problem consists of the following elements:

- A genetic representation of possible solutions to the problem.
- Generating an initial population of possible solutions.
- An objective (fitness) function that plays the role of the environment, ranking solutions based on their suitability.
- Genetic operators that convert offspring composition.
- Values for various parameters used by the genetic algorithm (population size, chances of applying genetic operators, etc.).

In the following, we explain in detail our framework and the specific design of the developed genetic algorithm.

3. System Model

We consider a network as a graph $G = (V, E)$, where V is the node set and E is the edge set. For two nodes $u, v \in V$, $e(u, v) \in E$ represents a link between nodes u and v. A link prediction algorithm over G attempts to determine whether a link $e(u, v)$ will exist between nodes u and v, in the future, given that it is not present at the moment.

We considered networks as undirected graphs, where the nodes are represented by integers and the edges connecting them are tuples of these node numbers. The employed networks ranged between 600 and 2500 nodes and edge count varied between 5500 and 20,000. Larger networks can be analyzed straightforwardly. We consider the processing time less critical and target a more accurate predictor for missing edge estimations.

We propose a framework that concerns the composition of various link prediction algorithms in order to obtain the best possible result from this composition. According to our approach, we rely on two or more well-known and basic algorithms that perform link prediction and have good results, that is, they make good estimates for edges that are missing from a network and will probably be added to it in the future. These estimates are combined and a new estimate results from their composition. The new estimate is essentially a more enhanced evaluation (rank) of how much a particular missing edge in a network could be added to it in the future. With the enhanced estimation, we attempt to promote the best possible estimation from each basic algorithm. This idea follows the principle that there is no link prediction algorithm which is the best for every type of network, but there are link prediction algorithms which, depending on the type of network to which they are applied [30–32], return good or bad results or estimates.

In our approach for link prediction, we have used five well-known algorithms. These are `Resource Allocation Index`, `Adamic-Adar Index`, `Common Neighbor Centrality`, `Jaccard Coefficient`, and `Preferential Attachment` [14–21]. Section 2 described the operation of each one. The use of these algorithms is not limiting. We could have used any

other link prediction algorithm as a basic algorithm. The only limitation is that the algorithm can return a numeric prediction estimate for some currently missing network link.

The network on which we apply the basic algorithms is derived from the original network as follows: In a random but controlled manner for the consistency of our network, we remove a number of edges from the original network. The number of edges we remove each time depends on the size of the network on which we will apply the basic algorithms. We usually remove a sufficient number of edges (over 1000) to have a sufficient number of edges that will allow the basic algorithms to perform satisfactorily. This results in two sets of edges. The first set of edges is the network that is the remaining part of the network and is essentially the `train set`, while the second set is the set of removed edges that serves the role of the `test set`.

In order to be able to have enhanced estimates (in relation to the estimates of the basic algorithms) for the missing edges of a network, we should first apply and obtain the results of the basic algorithms we have chosen to the train set. We store the obtained results, which consist of the prediction estimates of each edge missing, which results from the estimations of the basic algorithms. In these evaluation lists storing the computed rankings, each edge missing from the network is accompanied by the algorithm's estimate of how likely it is to appear in the future. Then we sort the contents of the evaluation lists in descending order of estimation, so that the edges that each basic algorithm estimates are most likely to be missing from the network are found at the beginning of each evaluation list. The evaluated edges present in these lists are not only the edges we have previously removed (and are present in the test set) but all the edges that the nodes of the network we are working with can potentially have, except of course those already present in the training set. This means that depending on the number of nodes in the network, these evaluation lists can contain from tens of thousands to millions of missing edges. However, in practice, we do not need all of them but only a small part of them. In our experiments, we only have 15,000 potential missing edges in each evaluation list of basic algorithms. Since we have sorted the lists of evaluation results (every list contains only 15,000 potential missing edges, and we do not need more) and using the `test set` that contains the edges that we have actually removed from the original network, we can now evaluate the performance of each of the basic algorithms that we have chosen to apply in the given network. To achieve this, we check the list of evaluations corresponding to each algorithm from its beginning to the position corresponding to the number of edges we have removed, i.e., the number of edges in the `test set`. These initial edges of each evaluation list define new sets of edges. The performance of each basic algorithm results from the number of edges of the intersection of the `test set` with each set separately from those that have been created.

The framework for link prediction that we propose is based on the fact that the various basic algorithms that it uses, depending on their mode of operation, do not always have the same results. That is, they do not always make the same estimates for the edges that are missing from a network. In addition to the edges on which they agree that there is a high probability of their occurrence in the future, there are also edges on which their estimates differ. Our attempt is to enhance the estimates for some edges that in the normal operation of the basic algorithms would be in lower positions than others even though in the network they had a higher probability of appearing in the future than the estimate given to them by a basic algorithm.

To be able to have these enhanced estimates of the future occurrence of some missing edges, we first normalize the results of each of the basic algorithms by storing them in respective lists. Then, we use a `genetic algorithm` [11–13] with which we try to find the weight that the evaluation of each algorithm will have in the composition we attempt. In the `genetic algorithm`, we give as input the list of edges that we have removed from the graph (`test set`) as well as the lists of estimates of the various basic algorithms for each edge that is missing from the graph (evaluation lists). We also give the number of the `population` as well as the number of `generations`. The `genetic algorithm` after its execution returns a list of weights. Each weight corresponds to a basic algorithm from those

we have used and which expresses the weight of its estimate in the enhanced estimate we will construct.

Having this `weights list`, we can now make our own assessment (prediction) of how likely it is that an edge is missing from our original graph (or added in the near future). The procedure we follow is to take the estimate of each basic algorithm for some missing edge, normalize these estimates, and then compose a new estimate (on any scale we want; here we have used the scale 0–1,000,000) based on `genetic algorithm` output weights. In the tests found in the following sections, we experimentally prove that in networks where the algorithms selected in this paper work well, the enhanced estimate we create in this way for a network is more reliable than the estimate returned by the best basic algorithm that we have applied for this network.

4. Proposed Framework

The framework we propose for the construction of a new enhanced predictor tool for link prediction takes as input the edges of an undirected graph, which represents the network on which we will create the new enhanced predictor tool. These edges are given in the form of a delimited file, where in each line of the file there are two nodes represented by integers and separated from each other by a comma (i.e., csv) or some other symbol. Here we must point out that the new enhanced predictor tool we propose is only suitable for each specific network and its features and not for every type of network (after all, such a thing according to the literature would not be possible). However, for any type of network, for which we have at our disposal at least two link prediction algorithms that have a satisfactory performance but differ to some extent from their results, we can create a new enhanced predictor tool that will yield better estimations than both. In Figure 1, we show a flowchart of our proposed framework.

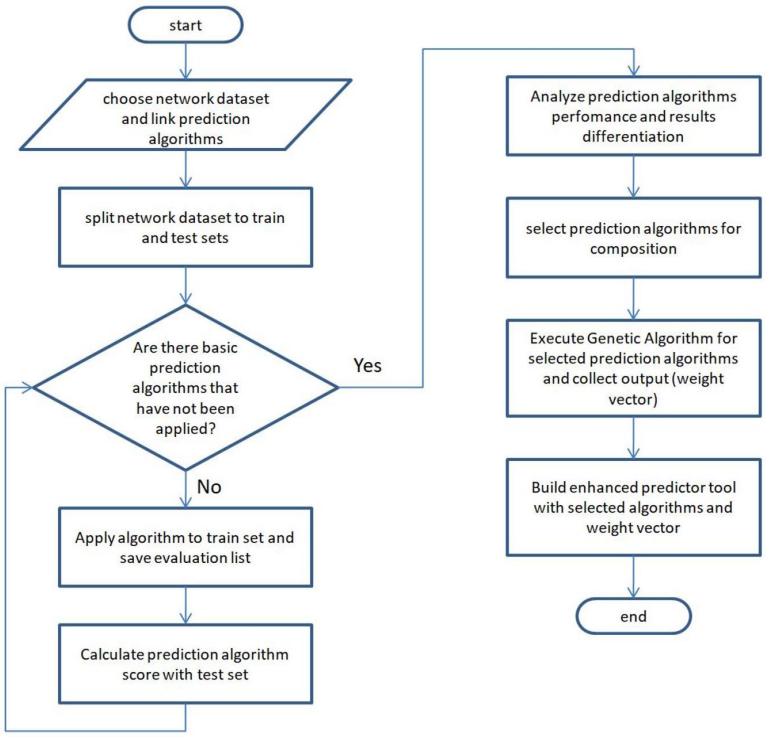


Figure 1. Flowchart of the proposed approach.

4.1. Creation of Train Set and Test Set

We begin with the creation of our train and test dataset. The train set is obtained by pseudo-randomly removing a number of edges, typically 5% to 30% of the initial set, depending on the size of the network we are looking at to obtain clearer results. When

removing the edges, we are careful to maintain the graph in a single connected component. The train set continues to be a graph with one component, missing some edges from the original one. The test set is the set of edges we have removed. With the test set we will be able to measure the performance of each algorithm we use as well as our enhanced predictor tool we will create.

4.2. Evaluation of Basic Algorithms

We apply each basic algorithm from those we have chosen for all edges that could connect two nodes other than the existing ones, i.e., for all currently missing edges. For each of these edges, each basic algorithm returns its estimate. We store all estimates for each basic algorithm in appropriate structures (evaluation lists) together with the edges with which they are concerned. These evaluation lists are sorted in descending order of rating. This has the result that in the first positions of the evaluation lists, we have the edges which are more likely to be missing from the graph (or to be added in the future) according the evaluation of each basic algorithm.

The evaluation lists contain all the edges that the specific graph could have, and as a result, they are quite large, depending on the number of nodes that the graph contains. However, practically speaking, we need a small percentage of them. In our experiments, we have used the first 15,000 edges, while the edges that are actually missing from the graph, i.e., the ones we have previously removed, are between 500 and 1500.

After creating evaluation lists, we can easily measure the effectiveness of each of the basic algorithms on the given network. We count from the first position of each evaluation list and up to the position corresponding to the number of edges we have removed from the original graph. This is the size of the test set. For example, if we have removed 500 edges from the graph, then we check the first 500 positions of the evaluation list of one of the basic algorithms we use and compare them with the edges found in the test set to find the intersection of the two sets. The number of edges of the intersection shows us the efficiency of the algorithm in the network we are considering.

In this step, it is also important to investigate whether the missing edge estimates of each basic algorithm are different. We are interested in the edges found by the various basic algorithms being as different as possible so that the evaluation list of the new enhanced predictor tool we will construct contains edges that are indeed missing from two or more basic algorithms' evaluation lists. Therefore, in an easy way, we find by basic algorithm pairs how much the evaluation lists of the basic algorithms differ from each other. In the final stage, we will prefer to compose basic algorithms that differ as much as possible in their evaluation lists so that the new algorithm that will emerge composes the individual evaluation list in the best way.

4.3. Selection of Basic Algorithms—Creation of Participation Weights

In this step, we choose the basic algorithms, the evaluation lists we will use, in order to find the weight each of them will have in the new enhanced predictor tool. We feed the evaluation lists of the selected algorithms as input to the genetic algorithm that we have developed and which returns us a weight list in which each position corresponds to the percentage (weight) that the estimate of the corresponding basic algorithm will have in the new enhanced predictor tool. As we have mentioned before, we try to choose basic algorithms that differ as much as possible in their evaluation lists. The right choice helps the genetic algorithm quite a bit in yielding better results.

4.4. The New Enhanced Predictor Tool

Once the weight list becomes available, we know the weight of each of the basic algorithms. The resulting new enhanced predictor tool takes the following steps for each edge missing from the network:

- It applies each basic algorithm separately to obtain their estimates for some edge.
- It normalizes these estimates to some scale.

- With the normalized values and with the list of weights, it composes a new estimate resulting from the inner product of the two vectors, i.e., the normalized values and the weights.

As we show in the experiments that follow, the new estimate is more reliable than the best estimate of each of the basic algorithms.

5. Genetic Algorithm

An important part of our Framework is the developed genetic algorithm [11–13]. As we have mentioned before, after analyzing the network with the basic algorithms, we create and use the evaluation lists that contain, for each edge that does not exist, the edge in tuple format and the estimation of the basic algorithm, and we give these evaluation lists as input to our genetic algorithm. We also give the truly missing edges (evaluation set) in list form as well as population count and generation count. Then we run the genetic algorithm, which returns the list of weights with which each basic algorithm will participate in the new enhanced predictor tool. Below we describe in detail the operation of the proposed genetic algorithm.

5.1. Population Encoding

The population to which we will apply the various `genetic procedures` consists of a list, which contains `chromosomes` and whose size is given as an input parameter in the genetic algorithm. `Chromosomes` [11–13] are objects of the `Chromosome class` that we have implemented to which we have given properties and functionality so that they store the necessary information and also return some of its parts in a useful form. A `chromosome` is associated with the set of basic algorithms we have chosen to run for some type of network. That is, it contains useful information for each basic algorithm, such as the percentage (weight) with which it participates in the specific `chromosome`.

Specifically, if we denote by n the number of algorithms with which each chromosome is connected, then it consists of the following:

- A weight list containing n integers in binary form. The numbers in this list are modified during the genetic processes (`crossover–mutation`) according to the theory of genetic algorithms [11–13]. From this list, we then obtain the result in percentage form (weights).
- A variable that contains the score which shows the robustness of the `chromosome`, i.e., how capable it is for `reproduction`. The score is the number of correct estimates made by the weights of the n basic algorithms on the specific `chromosome`.
- For error control purposes, it also contains the result of executing the n basic algorithms, i.e., an evaluation list consisting of edges and estimates on some scale that we have defined of the composition of the n basic algorithms according to the instructions of the `chromosome` (n weight percentages).

When created, each chromosome is initialized in a pseudo-random fashion. On initialization, n integer numbers in $[0, MAX_NUM]$ are listed, where $MAX_NUM = 2^{BITS_NUM}$ and $BITS_NUM$ is the number of bits in binary form.

Each `chromosome` has the functionality to return us the content of the list of natural numbers in normalized percentage form. The best `chromosome` of the population is the one with the highest score, and the percentages (weights) it contains show the importance of the various basic algorithms in order to obtain the best results.

5.2. Genetic Algorithm Initialization

When creating the genetic algorithm as an object, the input data are first stored so that we can use it in the genetic algorithm's various phases. Appropriate structures are also created for better data management. The data stored are the number of individuals in the population, the number of generations, and the list of edges that have actually been removed from the graph. In addition to these, the algorithm is given as an argument, as we

have mentioned, the evaluation lists of the various basic algorithms that we use each time. These evaluation lists are in a list that contains them. This list is managed by the genetic algorithm in a special way. First, a function is called which, for each evaluation list of a basic algorithm, normalizes its result to some scale that we have chosen (we have preferred the results to be converted to the interval 0–1,000,000 so that we can better see the differences of the various estimates that have been made). Then from the individual evaluation lists, we create a newly composed evaluation list, each position of which contains the edge as well as the normalized prediction values of each basic algorithm. This is a very important action in our genetic algorithm because, based on this composed evaluation list, the percentages (weights) will be created and the robustness of each chromosome of the population will be calculated.

The last step of initializing the genetic algorithm is to create the population, i.e., the `chromosomes` that will take part in the various genetic processes. Indeed, the genetic algorithm generates a list of `chromosomes` whose count we have previously taken as input. As we mentioned before, during its creation, a `chromosome` is initialized with random values, in the list where the weights of each algorithm are represented. However, each `chromosome` should still be evaluated, i.e., we should have a certain value for its robustness. This value is the score and during the creation of the `chromosome` it is 0.

At this stage of initialization and as soon as the new population is created, the genetic algorithm calls for each `chromosome` a special function which calculates and stores the score on the `chromosome`. The evaluation, i.e., calculating the score of a `chromosome`, is described in the next section.

5.3. Chromosome Evaluation

This process aims to calculate and store the fitness of each `chromosome`; sometimes it is also known as a `fitness function` according to the theory of genetic algorithms [11–13]. First, we take from the `chromosome` we want to evaluate the list with the weight percentages of each basic algorithm. Then based on these percentages, we iterate through the list containing all edges and base algorithm estimates (evaluation list), in normalized form, of how likely they are to be added to the graph in the future. For each edge encountered, we generate the new estimation coefficient and add the result to a new evaluation list. We then sort the new evaluation list of synthesized estimates in descending sort order. Suppose that the edges that are really missing and that we have stored in a separate list of the genetic algorithm are n. We check the first n positions of the new evaluation list and find how many missing edges those first n positions actually contain. This is also the evaluation of the `chromosome`.

5.4. Selection

In the selection process [11–13], we perform the following procedure. We store in a variable the sum of all the scores of the population's `chromosomes`. Then for each `chromosome`, we compare the quotient of dividing the robustness of the `chromosome` by the total robustness of the population, with a random value in the interval $[0, 1]$ that we have generated. If this random value is less than the division quotient, we return a copy of the `chromosome`.

5.5. Crossover

Crossover [11–13] is a very important genetic process, according to the theory of genetic algorithms. From the total population, a large part of it goes through the crossover process (the probability we have used in the experiments is usually 0.9). With crossover, two individuals of the population exchange genetic material. In our genetic algorithm, we use the `arithmetical crossover` [12,13] variant suggested by the literature. Specifically, during the crossing process between two chromosomes, we first find the largest number found in the weight list from one chromosome, letting $t1$ and $t2$ be the corresponding number of the other chromosome. Then we change these numbers according to the following formula:

$$t1 = int(a * t1 + (1 - a) * t2), \tag{6}$$

$$t2 = int(a * t2 + (1 - a) * t1), \tag{7}$$

where a is a small number equal to 0.01.

5.6. Mutation

Mutation [11–13] is also a very important genetic process according to the theory of genetic algorithms. During this process, we randomly change some or all of the numbers found in the weight list of a chromosome by making an integer division by 2. With this process, we try to prevent the genetic algorithm from becoming trapped in `local extrema`. We expect a small part of the population in each generation to undergo mutation, about 15%, according to the settings we have given to the genetic algorithm.

5.7. Reproduction

Our genetic algorithm at each step, i.e., at each generation, passes the population of that generation through the genetic processes we described above. During the application of these genetic processes, new chromosomes are produced in place of the old ones, that is, in each generation, we have a new population to manage. From the population of each generation, the genetic algorithm finds the most robust chromosomes (i.e., those with the highest score) and either does or does not store them in a list that always contains the most robust chromosomes. These may belong to different generations, but they all have the same score, i.e., the highest that has existed during reproduction. At the end of the reproduction, it returns to us the lists of weight percentages present in the chromosomes as well as their averages. These percentages are essentially the use of the basic algorithm's assessment weight that the new enhanced predictor tool we have created will carry out on each edge it wants to evaluate.

6. Theoretical Analysis

In the literature, there are several link prediction algorithms [14–21] which, depending on the type of network applied, exhibit different performance. We use appropriate metrics to evaluate these results. One of the most well-known and effective metrics is Precision [32], which is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

where TP stands for True Positive and FP stands for False Positive. The values of this metric range between 0 and 1. It is clear that the higher the Precision value a prediction algorithm gives, the more suitable it is for the network to which it is applied.

From the Precision we can define the top K predictive rate [32]. The top K predictive rate is the percentage of correctly classified positive samples among the top K instances in the ranking produced by a link predictor P. We denote the top K predictive rate as TPR_K, where K is a definable threshold. In the following, when we refer to Precision, we will mean the TPR_K with a well-defined K. To find the Precision of a predictor P in a network, we work as follows.

First, we randomly remove K edges from the network and store them in a set, denoted as $removed_E$. The K threshold comes from a percentage of the number of edges in the network, usually more than 1000 and less than half the number of edges in the network so as not to spoil the original network's structure. In the experiments we have performed, the number of edges of the networks we have used is over 5000 and we removed a small percentage of them (between 5% and 30%). If $G = (V, E)$ is the initial graph, then the resulting graph is

$$G' = (V, E'), E' = E - removed_E. \tag{9}$$

Let E'' be the set of edges where they do not belong to G', that is,

$$E'' = (u, v) : u, v \in V, (u, v) \notin E', \tag{10}$$

and let $Eval_Set = P(E'')$, where $P(E'')$ is a tuple $(u, v, value) : (u, v) \in E''$ and $value$ the P evaluation value for the edge (u, v), where $value \in \mathbb{R}$.

Thus, if we have $P_1, P_2, \ldots, P_n$, we can order them according their TPR_K, where $P_i > P_j$ if $TPR_{Ki} > TPR_{Kj}$, for $0 < i, j \leq n$.

Let *Eval_Set_sorted* be the *Eval_Set* in descending order by *value*. From the definition of TPR_K, it is clear that the more edges that are actually missing in the first K edges of an *Eval_Set_sorted*, the better the Predictor P that generated it. Therefore, if we want to have a worthwhile Predictor that will make good predictions, this Predictor should give *Eval_Set_sorted*, where its first K elements contain as many missing edges as possible.

But we can also think the other way around. We can construct in an organized way, and using the *Eval_Set_sorted* sets of Predictors that we already have, a new *Eval_Set_sorted* set that contains in its first K elements more edges that are truly missing and thus a larger TPR_K than all the rest of the *Eval_Set_sorted* sets. We can construct a better *Eval_Set_sorted* if we improve every value of the tuples of which it consists. The improvement will come from the corresponding values of the *Eval_Set_sorted* sets for each missing network edge.

This problem is an optimization problem and can be formulated as follows. If we have $P_1, P_2, \ldots, P_n$ Predictor algorithms which we have applied to G' and these Predictors have constructed n *Eval_Set_sorted* sets, is there a weight vector $w^T = [w_1, w_2, \ldots, w_n], w_i \in [0, 1]$ such that for every missing edge value of these sets, a new missing edge value is created and all these new values and corresponding missing edges result in a *Eval_Set_sorted* with larger TPR_K? The new *Eval_Set_sorted* will consist of tuples $(u, v, value) : (u, v) \in E''$ and

$$value = \sum_{i=1}^{n} w_i * value_i, \tag{11}$$

where $w_i \in w^T = [w_1, w_2, \ldots, w_n]$ and $value_i \in normalize[value_1, value_2, \ldots, value_n]$. The $normalize[value_1, value_2, \ldots, value_n]$ is a vector that contains the unique edge values of all *Eval_Set_sorted* sets, and these values have been normalized.

Thus, we want to calculate $OPT(w) = w^\star$ where $w^\star$ is the optimized Predictor contribution vector for calculating every new edge evaluation value in order to give us an optimized *Eval_Set_sorted* with the optimum TPR_K. To be able to calculate a $w^\star$, we have used a genetic algorithm with very good results, as can be seen in the experiments described in the rest of the paper.

With $w^\star$, we can construct an improved Predictor which is a composition of the basic Predictors $P_1, P_2, \ldots, P_n$ but gives optimized *Eval_Set_sorted* with optimum TPR_K compared to the *Eval_Set_sorted* of basic Predictors $P_1, P_2, \ldots, P_n$. Since we have shown that the quality (in terms of Precision) of an Predictor depends on its *Eval_Set_sorted*, and since this Predictor produces the optimal *Eval_Set_sorted*, then this Predictor is optimal.

Summarizing so far, for every missing edge $e = (u, v), u, v \in V$ and $e \in E''$ if $P_i(e)$ is a basic Predictor missing edge value, the new enhanced Predictor evaluates this missing edge as

$$value = \sum_{i=1}^{n} w_i^\star * P_i(e), \tag{12}$$

where $w_i^\star \in w^\star$, the optimized Predictor contribution vector. The enhanced Predictor *Eval_Set_sorted* consist of tuples $(u, v, value) : (u, v) \in E''$, $value \in \mathbb{R}$ and has optimum TPR_K.

7. Evaluation Process

To be able to evaluate the effectiveness of the proposed approach, we applied our model to various graphs. These graphs come either from publicly available real networks or from networks that we constructed with the help of Python libraries such as `networkx` https://networkx.org/ (accessed on 5 April 2023). We performed enough tests on several kinds of networks to have as clear conclusions as possible.

The method we have used to test our model on different types of networks and draw useful conclusions that we report below consists of the following steps.

7.1. Dataset Selection or Creation

The datasets we select or create always have the form of tuples. Each tuple consists of two nodes and represents an edge of the graph. Nodes are represented by natural numbers.

7.2. Create Train_Set, Omissible_Edges_Trainset

From the original dataset and with the help of Python software we have developed, we randomly remove some edges. For each edge we remove, we check that the graph remains `strongly connected`. The set obtained after controlled edge removal is the train_set. The omissible_edge_trainset is the set consisting of the edges we removed before. With these sets of edges, we can train our model.

7.3. Create Test_Set, Omissible_Edges_Testset

In a similar way as in the previous subsection, we also create these sets, which we will need in the evaluation of our model.

7.4. Network Structure Analysis

With the help of Python libraries as well as code we have written, we can have various properties of the graph that describe its structure, in order to draw conclusions about the type of graph on which we will evaluate our model. The properties we find each time are detailed below.

7.4.1. Transitivity

The transitivity T of a graph is based on the relative number of triangles in the graph, compared to the total number of connected triples of nodes:

$$T = \frac{3 \times \text{number of triangles in the network}}{\text{number of connected triples of nodes in the network}}. \quad (13)$$

The factor of three in the number accounts for the fact that each triangle contributes to three different connected triples in the graph, one centered at each node of the triangle.

With this definition:

$$0 \leq T \leq 1 \text{ and } T = 1,$$

if the network contains all possible edges [33].

7.4.2. Average Clustering (Clustering Coefficient)

The clustering coefficient of an undirected graph is a measure of the number of triangles in a graph. The clustering coefficient of a graph is based on a local clustering coefficient for each node:

$$T = \frac{\text{number of triangles connected to node } i}{\text{number of triples centered around node } i}, \quad (14)$$

where a triple centered around node i is a set of two edges connected to node i.

The clustering coefficient for the whole graph is the average of the local values C_i:

$$C = \frac{1}{n} \sum_{n=1}^{n} C_i, \quad (15)$$

where n is the number of nodes in the network [34].

7.4.3. Average Shortest Path Length

The mean path length is the average of the shortest path length, averaged over all pairs of nodes. For an undirected graph of n nodes, the mean path length is

$$\ell = \frac{1}{n(n-1)} \sum_{i \neq j} d_{ij}, \tag{16}$$

where d_{ij} is the length of the shortest path between nodes i and j and the sum is over all pairs of distinct nodes [35].

7.4.4. Network Diameter

The diameter of a graph is the length of the shortest path between the most distanced nodes.

7.4.5. Network Efficiency (Average Global Efficiency)

The efficiency of a pair of nodes in a graph is the multiplicative inverse of the shortest path distance between the nodes. The average global efficiency of a graph is the average efficiency of all pairs of nodes [36].

7.4.6. Average Local Efficiency

The efficiency of a pair of nodes in a graph is the multiplicative inverse of the shortest path distance between the nodes. The local efficiency of a node in the graph is the average global efficiency of the sub graph induced by the neighbors of the node. The average local efficiency is the average of the local efficiencies of each node [36].

7.4.7. Mean Degree

Mean degree is simply the average number of edges per node in the graph.

7.4.8. Degree Pearson Correlation Coefficient

Compute degree assortativity of graph. Assortativity measures the similarity of connections in the graph with respect to the node degree [37,38].

The results of the measurements of all the above properties are reported in each test and refer to the graphs on which the tests are performed. Also, for each graph that participates in a test, we list its graphical representation as well as histograms for the degrees of its nodes.

7.5. Evaluation of Basic Algorithms

In each graph corresponding to the sets of edges `train_set` and `test_set`, we apply the five basic link prediction algorithms we use. The application and evaluation of each basic algorithm for each graph is carried out with code that we have written. In a graph and for each edge missing from it, we apply a basic algorithm, which returns for this edge its estimate, which is a real number. We store all estimates for all edges (along with the edge for each estimate) in an evaluation list and then sort this evaluation list in descending order of the basic algorithm's estimate. Then we use the `omissible_edge_set` corresponding to the graph we are considering. The evaluation of the basic algorithm we implemented results from how many edges in the first `len(omissible_edge_set)` or |`omissible_edge_set`| positions of the evaluation list are actually contained in the `omissible_edge_set`. The more edges the evaluation list contains than those that are really missing in its first `len(omissible_edge_set)` positions, the more effective is the estimation that the algorithm gives us for an edge that does not exist in the graph.

For example, if 500 edges are missing (i.e., we have previously removed) from a graph and a basic algorithm manages to have 350 of them in the first 500 positions of its evaluation list, then we can conclude how correct of estimates it can make for edges that do not exist in the graph and may be added in the future. In our tests, this score is the point of comparison of the effectiveness of the basic algorithms we use and of the new enhanced predictor tool of the framework we propose and results from their composition. As a consequence of all the above, the higher the score of an algorithm, the more effective it is in the specific network graph.

From the above method of evaluating an algorithm, the value of a well-known metric evaluation, which is Precision [32], is derived. Precision is defined as the ratio of the true positive item to the sum of true positive and false positive selected items:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}. \quad (17)$$

Corresponding to the previous ones, the true positives are the edges that the algorithm found and are really missing from the graph, while the denominator corresponds to the number of edges that we have actually removed from the graph before.

Another standard that we have measured and reported in our experiments is the Area Under the Curve (AUC) [32]. For link prediction, AUC can be interpreted as the probability that a randomly selected missing edge gives a higher score than a randomly selected nonexistent edge. In this work, at each time we randomly pick an existing edge and a non-existing edge to compare their scores, if among n independent comparisons, there are n_1 times that the existing edge has a higher score and n_2 times that they have the same or lower score, the AUC value is

$$AUC = \frac{n_1 + 0.5n_2}{n}. \quad (18)$$

If all the scores are from independent and identical distributions, then the AUC should be approximately 0.5. Therefore, the degree to which AUC exceeds 0.5 indicates that link prediction is much better than random selection. In our work, we have taken AUC measurements with $n = 1000$.

7.6. Selection of Basic Algorithms

The new enhanced predictor tool of the framework we propose is based on the composition of different basic algorithms to obtain the best information from them in order to obtain better results. So we are interested in composing algorithms that are as diverse as possible in their evaluation lists, that is, the edges that they find to be truly missing and contained to evaluation lists are as diverse as possible. If for example two basic algorithms have in their evaluation lists the same edges that are really missing, then we cannot expect anything better from their composition. For this reason, we have implemented code that finds the number of edges that are truly missing and are different in the evaluation lists of the basic algorithms we use every time. In other words, the smaller the intersection of the sets of truly missing edges found by two or more basic algorithms in their evaluation lists, the better the results that can be given by their composition attempted by our framework.

7.7. Building the Model

After we have chosen the basic algorithms that will participate in the new enhanced predictor tool, we can now build it. How to create the new enhanced predictor tool has been described in a previous section. The new enhanced link predictor tool that has been created consists of the estimation weights of each basic algorithm that participates in it.

7.8. Enhanced Predictor Tool Score

For calculating the score of the new enhanced predictor tool, we have developed a code which takes as input the weights of the basic algorithms participating in this enhanced predictor tool and returns the score achieved by the new enhanced predictor tool. Calculating the score is achieved in a similar way to the way we use to calculate the score of each basic algorithm. Specifically, the framework we have implemented creates the list of estimates of the new enhanced predictor tool. The estimates of the new enhanced predictor tool are now based on the weights of the basic algorithms we have used in it. It then sorts the list of estimates of the new enhanced predictor tool in descending order of every estimate. Finally, it calculates how many edges from the ones we have removed are in the first positions of the evaluation list, i.e., the score. The number of these positions is the number of edges we have previously removed from the graph we are considering. When we have the score of our enhanced predictor tool as well as the individual scores of the basic link prediction algorithms we use, we can make comparisons to see the effectiveness of our enhanced predictor tool in relation to the basic algorithms.

8. Evaluation

In this section, we present the tests we have performed on various network datasets to evaluate the proposed framework and its enhanced predictor tool. We have found these datasets from various sources on the internet or created them ourselves. We tried to evaluate our framework and its enhanced predictor tool on many types of networks in order to draw useful conclusions, which we report in the next section. The organization of tests in this document is as follows. First, we state the type of network (graph) to which we will apply our framework and its enhanced predictor tool. For each network, we give useful information about its properties that describe it as well as useful graphs and graphical visualization. Then we report the different sets (sub graphs) we have created for our tests as well as the evaluation (score) in these sets of every basic algorithm we use. These sets are subsets of the main network we are considering. For convenience, we use abbreviations in the names of the basic algorithms. These are Common Neighbor Centrality as cnc, Resource Allocation Index as res_alloc, Adamic–Adar Index as adamic_adar, Jaccard Coefficient as Jaccard, and Preferential Attachment as pref_attach. Finally, we give comparable data for the tests we have performed with our framework and its enhanced predictor tool.

8.1. EML Network Dataset

We have downloaded the EML dataset from Noesis open framework https://noesis.ikor.org/datasets/link-prediction (accessed on 5 March 2023). EML is a network of individuals who shared emails. In Table 1, we show the EML network properties. In Figure 2, we show the visualization of the network as well as histograms of the degrees of its nodes.

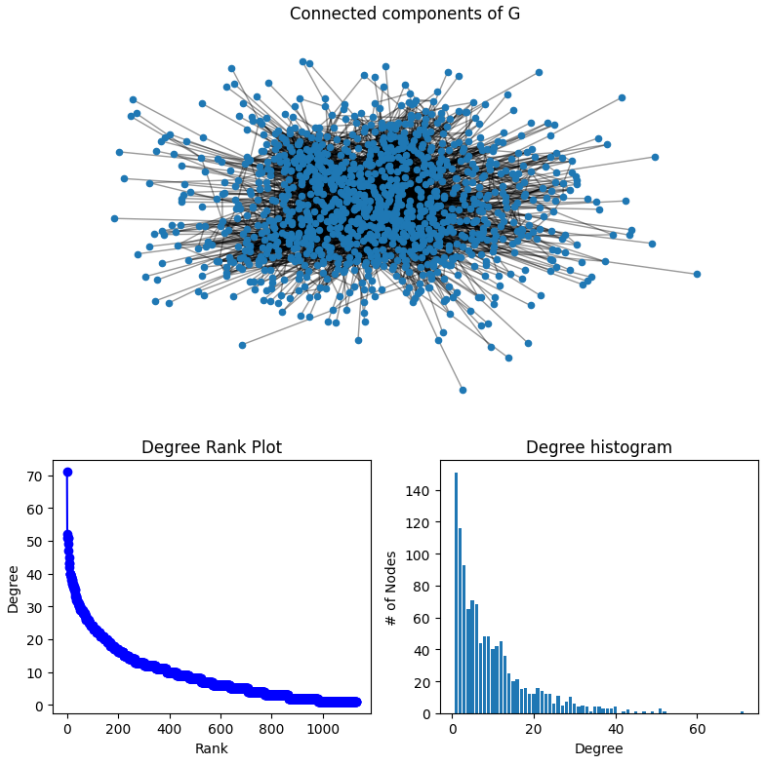


Figure 2. EML visualization and histograms. Here we present a visualization of EML network, as well as the distribution of its node degrees in the form of a histogram. From the visualization and histograms, we can conclude that the network consists of nodes with relatively few links to each other. Source: https://github.com/parisdim/A-Combinatory-Framework-for-Link-Prediction-in-Complex-Networks---Datasets (accessed on 5 August 2023).

Table 1. Noesis EML network properties.

EML Network Properties	
Nodes	1133
Edges	5451
Transitivity	0.166
Average clustering	0.220
Average shortest path length	3.606
Network diameter	8
Network efficiency	0.300
The average local efficiency of the network	0.312
Mean degree	9.622
Degree Pearson Correlation coefficient	0.078

From the EML network, we created three sub networks. Each of them contains a subset of EML edges, while the edges we have removed (omissible edges) are stored in a separate set. We call these sets Subgraph_0, Subgraph_1, Subgraph_2. In Table 2, we show the edges that we have removed from each set as well as the score of the five basic algorithms that we have applied to each of these sets. We remind you that each score of an algorithm corresponds to the number of edges it has found and are really missing from each set. These edges are at the top of the evaluation list of each algorithm. For example, in the Subgraph_0 network, we have removed 1582 edges. The common neighbor centrality (cnc) algorithm has a score of 297. This means that it has managed to have, in the first 1582 positions of its evaluation list, 297 edges of those we have removed from the original network. We recall that each position in an algorithm's evaluation list contains a missing edge as well as the algorithm's estimate of how likely it is to be added in the future. These lists are sorted in descending order of algorithm evaluation. Therefore, at the top of the evaluation list are edges that are more likely to be added in the future, according to the algorithm we use. From the scores in the table, we can observe that the five basic algorithms do not make such good estimates in the network we are examining, because they find a small subset of the edges that are actually missing.

To build and evaluate our enhanced predictor tool, we use one of Subgraph_0, Subgraph_1, or Subgraph_2 as a training set, while we use the remaining two as test sets for our new enhanced predictor tool. With the training set, we create the vector of weights that gives us the participation of each basic algorithm that we use in the final estimation of our enhanced predictor tool. In the test sets, we apply our enhanced predictor tool estimate to each missing edge so that we have a new estimate for each of them.

In Table 3, we show four comparative tests. In each test, we compare the performance of the evaluation derived from the composition of specific basic algorithms according to our enhanced predictor tool as well as from a basic algorithm that had the best performance in the sets participating in the test. Each test consists of four columns. In the first column, we report the training set and the test sets. In the next two columns, we report the score of the two estimators, and in the fourth column, we show how much one estimate differs from the other, i.e., how many more edges than the ones that are really missing did the test winner manage to find.

In Test 1, we have built our enhanced predictor tool using all five basic algorithms and compare its results with the results of Adamic–Adar, which has the best results of the five basic algorithms we use. Our enhanced predictor tool was trained with the Subgraph_0 set and managed to find 341 true missing edges compared to 334 found by Adamic–Adar. We see that our model finds more edges than the best of the basic algorithms, although not significantly more. Of course, since we are in training mode, we expect to find better results because our enhanced predictor tool is guided by the genetic algorithm to optimize.

Indeed, when applying our model and Adamic–Adar to the other two sets, we see that the results of our model are slightly better or the same as Adamic–Adar.

Table 2. Scores of 5 basic link prediction algorithms.

EML Subgraphs Basic Algorithms Score/AUC/Precision									
Algorithm	*Subgraph_0 (omissibles: 1582)*			*Subgraph_1 (omissibles: 1518)*			*Subgraph_2 (omissibles: 1516)*		
cnc	297	0.919	0.187	265	0.923	0.174	275	0.916	0.181
pref-attach	65	0.895	0.041	58	0.902	0.038	64	0.899	0.042
res-alloc	294	0.814	0.185	278	0.826	0.183	282	0.819	0.186
adamic-adar	334	0.824	0.211	305	0.827	0.200	320	0.834	0.211
Jaccard	153	0.819	0.096	146	0.823	0.096	140	0.843	0.092

Table 3. Comparable tests in EML network.

EML Comparable Tests							
Test 1: Composition of adam_adar - cnc - res_alloc - Jacc - pref_att vs. adam_adar							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (train set)	341	0.947	0.215	334	0.824	0.211	+7 more
Subgraph_1 (test set)	310	0.948	0.204	305	0.827	0.200	+5 more
Subgraph_2 (test set)	320	0.944	0.211	320	0.834	0.211	+0 more
Test 2: Composition of res_alloc - cnc vs. res_alloc							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (train set)	344	0.926	0.217	297	0.814	0.187	+47 more
Subgraph_1 (test set)	311	0.925	0.204	278	0.826	0.183	+33 more
Subgraph_2 (test set)	320	0.926	0.211	282	0.819	0.186	+38 more
Test 3: Composition of adam_adar - cnc - res_alloc - Jacc - pref_att vs. adam_adar							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (test set)	334	0.948	0.211	334	0.824	0.211	+0 more
Subgraph_1 (train set)	314	0.945	0.206	305	0.827	0.200	+9 more
Subgraph_2 (test set)	323	0.939	0.213	320	0.834	0.211	+3 more
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (test set)	332	0.909	0.209	297	0.814	0.187	+35 more
Subgraph_1 (train set)	312	0.929	0.205	278	0.826	0.183	+34 more
Subgraph_2 (test set)	321	0.921	0.211	282	0.819	0.186	+39 more

In Test 2, we have created our enhanced predictor tool using only two basic algorithms, Resource Allocation algorithm (res_alloc) and Common Neighbor Centrality algorithm (cnc). The comparison is made with the results of Resource Allocation algorithm, which are better than the results of Common Neighbor Centrality algorithm. Our enhanced predictor tool has been trained again with the Subgraph_0 set. We can see that our enhanced predictor tool gives significantly better results than those of Resource Allocation algorithm both in the training set and in the test sets. In particular, it finds 47 additional edges than those found by Resource Allocation algorithm in the training set as well as 26 and 34 additional edges, respectively, in the test sets.

This is similar to the previous results we have in the following two Tests 3 and 4, where for the training set we use Subgraph_1.

But why is it that when we use all five basic algorithms in our enhanced predictor tool we obtain worse results, i.e., we find fewer extra edges than are actually missing, than when we use two only basic algorithms?

The explanation has to do with the way these basic algorithms work. Each basic algorithm gives some edges missing from the graph, higher evaluation values than the remaining edges. That is, it considers it more likely that some edges are missing than others, which is the point. If the basic algorithm is suitable for a network, then it usually displays in the first positions of its evaluation list edges that are really missing from the graph we are

examining. The more suitable the algorithm is for the network, the more these edges will be. However, from one position in the evaluation list onwards, edges appear which, although they have high estimate values, will not be added to the graph in the near future. This makes sense because the basic algorithm makes predictions according to the logic with which it was written. These predictions cannot be perfect precisely because they are predictions.

The framework and the enhanced predictor tool we propose tries to combine as best as possible the separate results of each basic algorithm through its evaluation list. So we are interested in each basic algorithm, in addition to the common edges it will have in its evaluation list with the evaluation lists of the other basic algorithms, to also contain different edges so that they have enough probabilities to appear in the evaluation list of our enhanced predictor tool that composes them. And this is actually accomplished when we compose in the enhanced predictor tool only two basic algorithms in Tests 2 and 4. When there are many basic algorithms for composition, then together with the correct selections of the edges, the incorrect selections which obtain a high score multiply. This results in our enhanced predictor tool's evaluation list containing more wrong basic algorithm edge choices than we would like.

Ideally, we want to use basic algorithms that make good predictions for the network they are applied to, that are as diverse as possible in their evaluation list, but that are also few in number. If, for example, all the algorithms made almost the same predictions about the missing edges of a network, then composing them would not have a better result. If these basic algorithms were not efficient in the network, then their composition would not have a better result either (perhaps something better than a single algorithm, but not satisfactory). We try to choose two or three basic algorithms, if they exist, for our enhanced predictor tool for even better results.

8.2. HMT Network Dataset

We have downloaded the HMT dataset from Noesis open framework https://noesis.ikor.org/datasets/link-prediction (accessed on 5 April 2023). HMT is a social network of individuals. In Table 4, we show the HTM network properties. In Figure 3, we show the visualization of the network as well as histograms of the degrees of its nodes.

In a similar way as before, from the HMT network we have created three sub-networks. Each of these has been derived by removing random edges from the original network each time. Table 5 shows the number of edges removed as well as the performances of the basic algorithms we use. In Table 6, we show four benchmark tests by which we compare the performance of our enhanced predictor tool with the performance of the best basic algorithm participating in it each time. In all benchmark tests, we see that our enhanced predictor tool's predictions are better than those of the best participating basic algorithm. In Test 4, however, we can observe that our enhanced predictor tool has impressive performances. This led us to further investigate the operation of the basic algorithms in the network in which we apply them. By writing some more lines of code, we tried to see where their predictions, contained in their evaluation lists, differed. Finally, we found that the pair of cnc and Jaccard basic algorithms differs more than the other pairs of basic algorithms in its predictions. Specifically, the number of different edges predicted by the two algorithms to be truly missing from the training network is about 300, while the common edge predictions are about 200. In the remaining pair combinations, the number of different edges they predict is below 150. For example, between basic algorithms Adamic–Adar and res_alloc, their common predictions are about 540 edges, while the different ones are only 116. From this example, we see that our enhanced predictor tool can combine different algorithms in an efficient way to obtain significantly better results.

Table 4. Noesis HMT network properties.

HMT Network Properties	
Nodes	2426
Edges	16,630
Transitivity	0.229
Average clustering	0.540
Average shortest path length	3.589
Network diameter	10
Network efficiency	0.306
The average local efficiency of the network	0.664
Mean degree	16.097
Degree Pearson Correlation coefficient	0.023

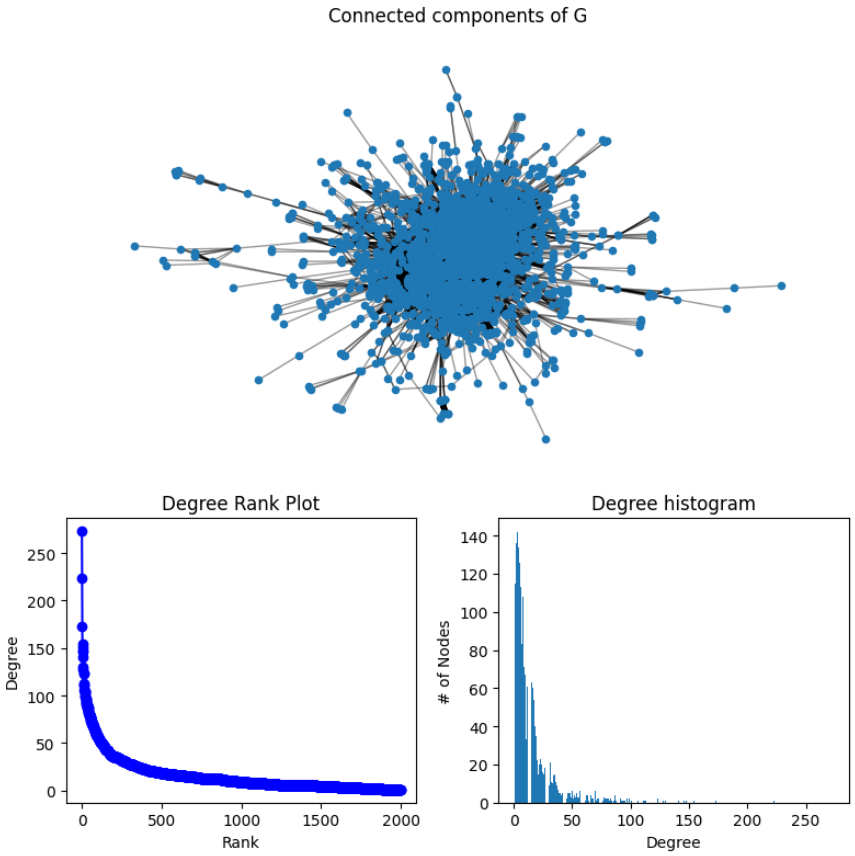


Figure 3. HMT visualization and histograms. Here we present a visualization of EML network, as well as the distribution of its node degrees in the form of a histogram. From the visualization and histograms we can conclude that the network consists of nodes with relatively few links to each other. Source: https://github.com/parisdim/A-Combinatory-Framework-for-Link-Prediction-in-Complex-Networks---Datasets (accessed on 5 August 2023).

Table 5. Scores of 5 basic link prediction algorithms.

HMT Subgraphs Basic Algorithms Score/AUC/Precision									
Algorithm	*Subgraph_0 (omissibles: 2256)*			*Subgraph_1 (omissibles: 1639)*			*Subgraph_2 (omissibles: 1659)*		
cnc	497	0.978	0.220	320	0.982	0.195	322	0.983	0.194
pref-attach	127	0.926	0.056	81	0.926	0.049	73	0.906	0.044
res-alloc	826	0.980	0.366	518	0.974	0.316	561	0.977	0.338
adamic-adar	650	0.963	0.288	424	0.977	0.258	414	0.978	0.249
Jaccard	487	0.967	0.215	332	0.973	0.202	352	0.971	0.212

Table 6. Comparable tests in HMT network.

HMT Comparable Tests							
Test 1: Composition of res_alloc - Jaccard - adamic_adar vs. res_alloc							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (train set)	857	0.979	0.379	826	0.980	0.366	+31 more
Subgraph_1 (test set)	545	0.976	0.332	518	0.974	0.316	+27 more
Subgraph_2 (test set)	584	0.984	0.352	561	0.977	0.338	+23 more
Test 2: Composition of res_alloc - cnc vs. res_alloc							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (train set)	838	0.982	0.371	826	0.980	0.366	+12 more
Subgraph_1 (test set)	521	0.976	0.317	518	0.974	0.316	+3 more
Subgraph_2 (test set)	561	0.985	0.338	561	0.977	0.338	+0 more
Test 3: Composition of res_alloc - Jaccard vs. res_alloc							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (train set)	849	0.966	0.376	826	0.980	0.366	+13 more
Subgraph_1 (test set)	540	0.985	0.329	518	0.974	0.316	+22 more
Subgraph_2 (test set)	579	0.980	0.349	561	0.977	0.338	+18 more
Test 4: Composition of Jaccard - cnc vs. Jaccard							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (train set)	680	0.981	0.301	487	0.967	0.215	+193 more
Subgraph_1 (test set)	444	0.982	0.270	332	0.973	0.202	+112 more
Subgraph_2 (test set)	450	0.981	0.271	352	0.971	0.212	+98 more

8.3. Random Geometric Graph

With the help of Python's `NetworkX` library by calling the function `random_geometric_graph(620, 0.2)`, we have created a random geometric graph which has the characteristics we mention in Table 7. The first argument in the function is the number of nodes and the second the distance threshold value. In Figure 4, we show the visualization of the network as well as histograms of the degrees of its nodes.

Table 7. Random geometric network properties.

Random Geometric Graph Properties	
Nodes	620
Edges	20,351
Transitivity	0.626
Average clustering	0.658
Average shortest path length	3.240
Network diameter	8
Network efficiency	0.394

Table 7. *Cont.*

Random Geometric Graph Properties	
The average local efficiency of the network	0.829
Mean degree	65.648
Degree Pearson Correlation coefficient	0.573

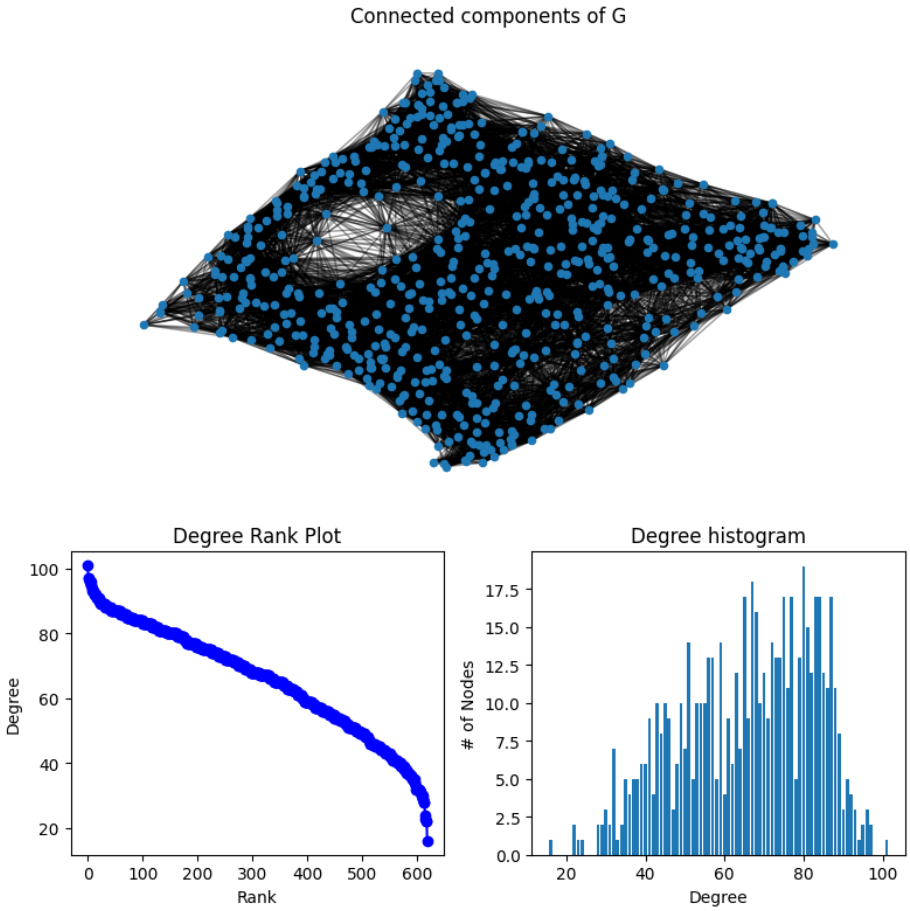


Figure 4. Random geometric visualization and histograms. Here we present a visualization of random geometric network, as well as the distribution of its node degrees in the form of a histogram. From the visualization and histograms, we can conclude that the network consists of nodes with links to each other uniformly distributed. We also conclude that this network is more dense than EML and HMT. Source: https://github.com/parisdim/A-Combinatory-Framework-for-Link-Prediction-in-Complex-Networks---Datasets (accessed on 5 August 2023).

From the random geometric network, we have created six sub-networks. Again, each of these has been derived by removing random edges from the original random geometric network each time. Table 8 shows the number of edges removed as well as the performances of the basic algorithms we use. From Table 8, we can observe that the basic algorithms other than pref_attach have appreciable results. However, from the table of comparative results, Table 9, we can see that our proposed enhanced predictor tool manages to improve the already good performance of the basic algorithms. In all tests, it manages to make better predictions, finding more edges that are actually missing from the network than the best of the basic algorithms find in that network. The good results of our enhanced predictor tool are maintained even if we change the training set, which shows that these results are not random at all.

Table 8. Scores of 5 basic link prediction algorithms.

Random Geometric Subgraphs Basic Algorithms Score/AUC/Precision									
Algorithm	*Subgraph_0 (omissibles: 989)*			*Subgraph_1 (omissibles: 1016)*			*Subgraph_2 (omissibles: 1037)*		
cnc	697	0.9965	0.704	693	0.9965	0.682	724	0.9980	0.698
pref-attach	38	0.7995	0.038	45	0.8055	0.044	29	0.7930	0.027
res-alloc	738	1.0000	0.746	756	0.9985	0.744	773	0.9995	0.745
adamic-adar	732	1.0000	0.740	721	0.9990	0.709	756	0.9985	0.729
Jaccard	735	0.9995	0.743	740	0.9995	0.728	770	0.9985	0.742
Algorithm	*Subgraph_0 (omissibles: 1076)*			*Subgraph_1 (omissibles: 1024)*			*Subgraph_2 (omissibles: 1009)*		
cnc	741	0.9965	0.688	704	0.9965	0.687	689	0.9970	0.682
pref-attach	33	0.8085	0.030	35	0.7855	0.034	28	0.8010	0.027
res-alloc	771	1.0000	0.716	753	0.9965	0.735	732	0.9965	0.725
adamic-adar	775	0.9980	0.720	737	0.9980	0.719	720	0.9995	0.713
Jaccard	753	0.9990	0.699	751	0.9985	0.733	721	1.0000	0.714

Table 9. Comparable tests in random geometric network.

Random Geometric Network Comparable Tests							
Test 1: Composition of Jaccard - cnc vs. Jaccard							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (train set)	772	0.9985	0.780	735	0.9995	0.743	+37 more
Subgraph_1 (test set)	775	0.9975	0.762	740	0.9995	0.728	+35 more
Subgraph_2 (test set)	813	0.9990	0.783	770	0.9985	0.742	+43 more
Subgraph_3 (test set)	803	0.9985	0.746	753	0.9990	0.699	+50 more
Subgraph_4 (test set)	787	0.9980	0.768	751	0.9985	0.733	+36 more
Subgraph_5 (test set)	759	0.9990	0.752	721	1.0000	0.714	+38 more
Test 2: Composition of res_alloc - adamic_adar - Jaccard vs. res_alloc							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (test set)	774	1.0000	0.782	738	1.0000	0.746	+36 more
Subgraph_1 (train set)	774	1.0000	0.761	756	0.9985	0.744	+18 more
Random Geometric Network Comparable Tests							
Subgraph_2 (test set)	805	0.9995	0.776	773	0.9995	0.745	+32 more
Subgraph_3 (test set)	796	0.9985	0.739	771	1.0000	0.716	+25 more
Subgraph_4 (test set)	784	0.9995	0.765	753	0.9965	0.735	+31 more
Subgraph_5 (test set)	754	1.0000	0.747	732	0.9965	0.725	+22 more
Test 3: Composition of res_alloc - adamic_adar - Jaccard vs. res_alloc							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (train set)	774	1.0000	0.782	738	1.0000	0.746	+36 more
Subgraph_1 (test set)	771	0.9985	0.758	756	0.9985	0.744	+15 more
Subgraph_2 (test set)	804	0.9990	0.775	773	0.9995	0.745	+31 more
Subgraph_3 (test set)	797	0.9985	0.740	771	1.0000	0.716	+26 more
Subgraph_4 (test set)	784	1.0000	0.765	753	0.9965	0.735	+31 more
Subgraph_5 (test set)	754	0.9980	0.747	732	0.9965	0.725	+22 more
Test 4: Composition of Jacc - cnc - res_alloc - adam_adar - pref_attach vs. res_alloc							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (test set)	772	0.9980	0.780	738	1.0000	0.746	+34 more
Subgraph_1 (train set)	773	0.9970	0.760	756	0.9985	0.744	+17 more
Subgraph_2 (test set)	802	1.0000	0.773	773	0.9995	0.745	+29 more
Subgraph_3 (test set)	802	0.9995	0.745	771	1.0000	0.716	+31 more
Subgraph_4 (test set)	783	0.9980	0.764	753	0.9965	0.735	+30 more
Subgraph_5 (test set)	757	0.9985	0.750	732	0.9965	0.725	+25 more

8.4. Random Graph

With the help of Python's `NetworkX` library, by calling the function `erdos_renyi_graph(620, 0.015)`, we have created a random graph which has the characteristics we mention in Table 10. The first argument in the function is the number of nodes and the second is the probability for edge creation. In Figure 5, we show the visualization of the network as well as histograms of the degrees of its nodes.

Table 10. Random network properties.

Random Graph Properties	
Nodes	620
Edges	2950
Transitivity	0.016
Average clustering	0.017
Average shortest path length	3.094
Network diameter	5
Network efficiency	0.344
The average local efficiency of the network	0.018
Mean degree	9.516
Degree Pearson Correlation coefficient	−0.013

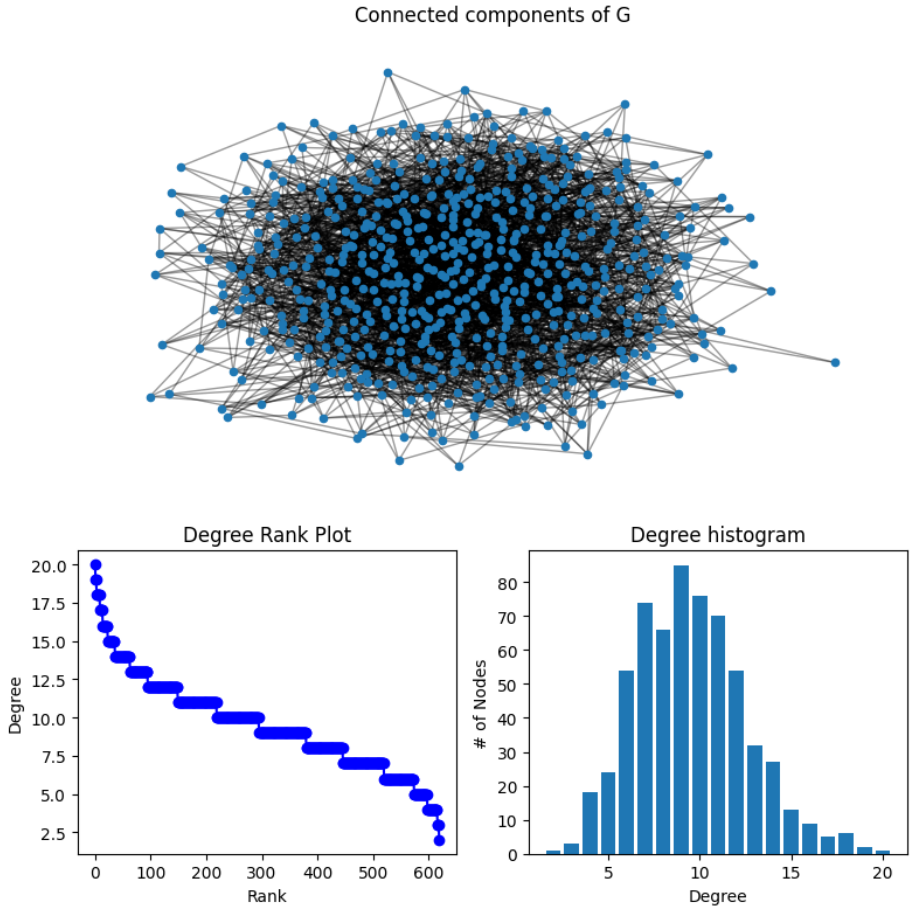


Figure 5. Random graph visualization and histograms. Here we present a visualization of pure random network, as well as the distribution of its node degrees in the form of a histogram. From the visualization and histograms, we can conclude that the network consists of nodes with links to each other uniformly distributed. We also conclude that this network is more dense than EML and HMT. Source: https://github.com/parisdim/A-Combinatory-Framework-for-Link-Prediction-in-Complex-Networks---Datasets (accessed on 5 August 2023).

From the pure random network we have created three sub networks. Like before, each of these has been derived by removing random edges from the original pure random network each time. Table 11 shows the number of edges removed as well as the performances

of the basic algorithms we use. From Table 11, we notice that all the basic algorithms have failed in their predictions and have a very low score. This is something we expected, because the construction of a random graph is unpredictable, i.e., it does not follow any rules but only chance. The non-deterministic construction of random graphs means that no algorithm can predict whether or not an edge will exist in the future. Consequently, no deterministic algorithm can be applied to these graphs, and therefore, neither can the enhanced predictor tool we propose which is a composition of deterministic algorithms. As we have mentioned before, in order for the proposed enhanced predictor tool to perform well, the basic algorithms it composes should also give remarkable results in the network in which we apply them. In fact, in Table 12, the scores are so low that they do not allow other considerations of exploiting these basic algorithms in this type of graph.

Table 11. Scores of 5 basic link prediction algorithms.

Random Subgraphs Basic Algorithms Score/AUC/Precision									
Algorithm	*Subgraph_0 (omissibles: 539)*			*Subgraph_1 (omissibles: 581)*			*Subgraph_2 (omissibles: 556)*		
cnc	0	0.644	0.000	6	0.660	0.010	2	0.648	0.003
pref-attach	4	0.736	0.007	1	0.741	0.001	0	0.736	0.000
res-alloc	3	0.540	0.005	3	0.552	0.000	3	0.548	0.005
adamic-adar	2	0.538	0.003	5	0.537	0.008	2	0.545	0.003
Jaccard	2	0.537	0.003	4	0.539	0.006	1	0.544	0.001

Table 12. Comparable tests in HMT network.

HMT Comparable Tests							
Test 1: Composition of Jacc - cnc - res_alloc - adam_adar - pref_attach vs. best							
Subgraph	*Predictor Score /AUC/Precision*			*Best Basic Algorithm Score /AUC/Precision*			*Difference*
Subgraph_0 (test set)	3	0.746	0.005	4	0.736	0.007	−1 less
Subgraph_1 (test set)	3	0.751	0.005	6	0.660	0.010	−3 less
Subgraph_2 (train set)	3	0.760	0.005	3	0.548	0.005	+0 more

9. Discussion

The framework we propose can be applied to any type of network. The prediction tool produced by the proposed framework is proper only for the specific network at a given time, while the predictions it gives are usually better than those of the basic algorithms it composes.

From the proposed framework, we can create the new enhanced prediction tool by carrying out the following steps:

1. Creation of train set and test set: From the edges of the network to which we will apply link prediction, we create two subsets of the train and test sets. The train set is the network on which we will apply the various basic algorithms and the new enhanced prediction tool we will build in order to predict how likely it is that some of the missing edges will be added in the future. With the test set, we can obtain how correct the predictions of each basic algorithm are in order to have a measure of comparison.
2. Evaluation of Basic Algorithms in the network—analysis of results: With the help of the train set, we create for each basic algorithm the list which contains all the edges that are not present in the train set together with the prediction value given by the algorithm for whether to add each missing edge from the train set in the future. This list is called the evaluation list and contains tuples with the missing edges and their values. The evaluation lists are sorted in descending order by the prediction value of each tuple. From the evaluation list of each basic algorithm and the test set, we find how many edges of those that are really missing from the train set the basic algorithm manages to find. Also, from the evaluation lists of the basic algorithms, we can see

how different their results are, that is, how many different edges that are really missing from the train set they find.

3. Selection of basic Algorithms—creating a list of participation rates (weights): In this step, we select the basic algorithms that we will compose to create the new enhanced predictor tool. We prefer to compose basic algorithms that have as many different evaluation lists as possible but also find several edges that are missing from the train set. After the choice of basic algorithms we make, we give as input to the genetic algorithm we have developed the evaluation lists of the basic algorithms we have chosen and the test set that contains the edges that are actually missing from the train set. We also give other parameters to run the genetic algorithm, such as population size. After its execution, the genetic algorithm returns the vector of weights with which we will build the new enhanced predictor tool.
4. Building and Operation of the new Enhanced Predictor Tool: Since we have the vector of weights of the basic algorithms we selected in the previous step, we can now apply the composition of their predictions to the network we are analyzing. The new enhanced predictor tool gives each edge that is missing from the network a new prediction value that results from the composition of the prediction values of the basic algorithms.

To test the effectiveness of the proposed framework and the new enhanced predictor tool, we performed several tests on different types of networks. The results of these tests were quite remarkable. In almost all tests, the new enhanced predictor tool performed better than the basic algorithms it composes. Because the new enhanced predictor tool is not a new link prediction algorithm but a composition of existing basic algorithms, it makes no sense to apply it to networks where the basic algorithms it composes do not work well. But when we apply it to networks in which we obtain good prediction results from the basic algorithms which it composes, then the new enhanced predictor tool always gives better prediction results than them.

10. Conclusions and Future Work

Several algorithms have been proposed for link prediction. Many of these algorithms have quite appreciable results in different types of networks, e.g., small-world, scale-free, etc. In this work, we proposed a framework with which we can create an enhanced predictor tool that makes more meaningful predictions in some applied network topologies. This tool composes different basic algorithms that have good prediction performance in specific types of networks applied. The evaluation results showed that this composition leads to even better link prediction results. In the future, we will further explore the possibilities offered by the proposed framework. Specifically, we will test the synthesis of basic algorithms in different fields, such as clustering, community detection, multiplex and complex networks analysis, machine learning, etc.

Author Contributions: P.D. conceived the framework of the paper, developed the proposed approach, contributed to the analysis, and provided the results of the paper. V.K. contributed to the finalization of the framework and the analysis and aided in the processing of the results. All authors contributed to the writing, revision, and proofreading of this paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets of the network topologies analyzed in this study can be found here: https://github.com/parisdim/A-Combinatory-Framework-for-Link-Prediction-in-Complex-Networks---Datasets (accessed on 5 August 2023).

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

Abbreviations

The following abbreviations are used in this manuscript:

IIoT	Industrial Internet of Things
GA	Genetic Algorithm
PPI	Protein–Protein Interaction

References

1. Newman, M. *Networks: An Introduction*; Oxford University Press: Oxford, UK, 2010.
2. Karyotis, V.; Stai, E.; Papavassiliou, P. *Evolutionary Dynamics of Complex Communications Networks*, 1st ed.; CRC Press: Boca Raton, FL, USA, 2013.
3. Liben-Nowell, D.; Kleinberg, J. The link prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.* **2007**, *58*, 1019–1031.
4. Lu, L.; Zhou, T. Link prediction in complex networks: A survey. *Phys. Stat. Mech. Its Appl.* **2011**, *390*, 1150–1170.
5. Getoor, L.; Diehl, C.P. Link mining: A survey. *ACM SIGKDD Explor. Newsl.* **2005**, *7*, 3–12.
6. Wang, H.; Zichun, L. Seven-Layer Model in Complex Networks Link Prediction: A Survey. *Sensors* **2020**, *20*, 6560. [CrossRef] [PubMed]
7. Abbas, K.; Abbasi, A.; Dong, S.; Niu, L.; Yu, L.; Chen, B.; Cai, S.-M.; Hasan, Q. Application of network link prediction in drug discovery. *BMC Bioinform.* **2021**, *22*, 187. [CrossRef]
8. Buket, K.; Mustafa, P. Unsupervised link prediction in evolving abnormal medical parameter networks. *Int. J. Mach. Learn. Cybern.* 2016, *7*, 725–757.
9. Shouwei, L.; Wang, Y. Research on Knowledge Transfer on Multilayer Networks Based on Link Prediction Algorithm. *J. Phys. Conf. Ser.* **2022**, *2224*, 012015. [CrossRef]
10. In, L.; Kyoochum, L. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Bus. Horizons* **2015**, *58*, 431–440.
11. Sastry, K.; Goldberg, D.E.; Kendall, G. Genetic Algorithms. In *Search Methodologies*, 2nd ed.; Burke, E.K., Kendall, G., Eds.; Springer Books; Springer: Berlin/Heidelberg, Germany, 2014; Chapter 4, pp. 93–117.
12. Mitchell, M. *An Introduction to Genetic Algorithms*; The MIT Press: Cambridge, MA, USA, 1996.
13. Lykothanasis, S. *Genetic Algorithms and Applications*; Hellenic Open University: Patras, Greece, 2001.
14. Zhou, T.; Lü, L.; Zhang, Y.C. Predicting missing links via local information. *Eur. Phys. J. B* **2009**, *71*, 623–630. [CrossRef]
15. Adamic, L.E.; Adar, E. Friends and neighbors on the web. *Soc. Netw.* **2003**, *25*, 211–230.
16. Zhao, P.; Aggarwal, C.; He, G. Link prediction in graph streams. In Proceedings of the 32nd IEEE International Conference on Data Engineering, Helsinki, Finland, 16–20 May 2016; pp. 553–564.
17. Yao, L.; Wang, L.; Pan, L. Yao, K. Link prediction based on common-neighbors for dynamic social network. *Procedia Comput. Sci.* **2016**, *83*, 82–89.
18. Ahmad, I.; Akhtar, M.U.; Noor, S.; Shahnaz, A. Missing Link Prediction using Common Neighbor and Centrality based Parameterized Algorithm. *Sci. Rep.* **2020**, *10*, 364. [CrossRef] [PubMed]
19. Jaccard, P. Étude Comparative de la Distribution Florale Dans une Portion des Alpes et des Jura. *Bull. Soc. Vaudoise Des Sci. Nat.* **1901**, *37*, 547–579.
20. Liben-Nowell, D.; Kleinberg, J. The Link Prediction Problem for Social Networks. 2004. Available online: http://www.cs.cornell.edu/home/kleinber/link-pred.pdf (accessed on 3 July 2023).
21. Mitzenmacher, M. A brief history of generative models for power law and lognormal distributions. *Internet Math.* **2004**, *1*, 226–251. [CrossRef]
22. Rasti, S.; Vogiatzis, C. A survey of computational methods in protein–protein interaction networks. *Ann. Oper. Res.* **2019**, *276*, 35–87.
23. Lei, C.; Ruan, J. A novel link prediction algorithm for reconstructing protein-protein interaction networks by topological similarity. *Bioinformatics* **2012**, *29*, 355–364. [PubMed]
24. Franceschini, A.; Szklarczyk, D.; Kuhn, M.; Simonovic, M.; Roth, A.; Lin, J.; Minguez, P.; Bork, P.; Von Mering, C.; Jensen, L.J. String v9.1: Protein-protein interaction networks, with increased coverage and integration. *Nucl. Acids Res.* **2012**, *41*, D808–D815.
25. Szklarczyk, D.; Franceschini, A.; Wyder, S.; Forslund, K.; Heller, D.; Huerta-Cepas, J.; Simonovic, M.; Roth, A.; Santos, A.; Tsafou, K.P.; et al. String v10: Protein-protein interaction networks, integrated over the tree of life. *Nucl. Acids Res.* **2015**, *43*, D447–D452.
26. Szklarczyk, D.; Morris, J.H.; Cook, H.; Kuhn, M.; Wyder, S.; Simonovic, M.; Santos, A.; Doncheva, N.T.; Roth, A.; Bork, P.; et al. The string database in 2017: Quality-controlled protein–protein association networks, made broadly accessible. *Nucl. Acids Res.* **2017**, *45*, D362–D368. [CrossRef]
27. Bonchi, F.; Castillo, C.; Gionis, A.; Jaimes, A. Social network analysis and mining for business applications. *ACM Trans. Intell Syst. Technol.* **2011**, *2*, 22.
28. Chen, H.; Li, X.; Huang, Z. Link prediction approach to collaborative filtering. In Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries, Denver, CO, USA, 7–11 June 2005; pp. 141–142.
29. Georgopoulos, E.; Lykothanasis, S. *Introduction in Genetic Algorithms*; Technological Educational Institute of Patras: Patras, Greece, 1999. Available online: http://edu.eap.gr/pli/pli31/docs/GAs_introduction.pdf (accessed on 4 May 2023).

30. Zhang, M.; Chen, Y. Link Prediction Based on Graph Neural Networks. In Proeedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montreal, QC, Canada, 3–8 December 2018. Available online: https://proceedings.neurips.cc/paper/2018/file/53f0d7c537d99b3824f0f99d62ea2428-Paper.pdf (accessed on 2 April 2023).
31. Toth, C.; Helic, D.; Geiger, B. Synwalk: Community Detection via Random Walkmodelling. *Data Min. Knowl. Discov.* **2022**, *36*, 739–780. [CrossRef]
32. Yang, Y.; Lichtenwalter, R.N.; Chawla, N.V. Evaluating link prediction methods. *Knowl. Inf. Syst.* **2015**, *45*, 751–782. [CrossRef]
33. Nykamp, D.Q. Definition of the Transitivity of a Graph. From Math Insight. 2023. Available online: http://mathinsight.org/definition/transitivity_graph (accessed on 2 April 2023).
34. Nykamp, D.Q. Mean Path Length Definition. From Math Insight. 2023. Available online: http://mathinsight.org/definition/clustering_coefficient (accessed on 2 April 2023).
35. Nykamp, D.Q. Clustering Coefficient Definition. From Math Insight. 2023. Available online: http://mathinsight.org/definition/network_mean_path_length (accessed on 2 April 2023).
36. Marchiori, L.; Marchiori, V.; Marchiori, M. Efficient behavior of small-world networks. *Phys. Rev. Lett.* **2001**, *87*, 198701. [CrossRef]
37. Newman, M.E.J. Mixing patterns in networks. *Phys. Rev. E* **2003**, *67*, 026126. [CrossRef]
38. Foster, J.G.; Grassberger, D.V.; Paczuski, P.; Edge, M. Direction and the structure of networks. *Proc. Natl. Acad. Sci. USA* **2010**, *107*, 10815–10820. [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.