A Scalable and Distributed Actor-Based Version of the Node2Vec Algorithm

Gianfranco Lombardo
Department of Engineering and Architecture
University of Parma
Parma, Italy
gianfranco.lombardo@unipr.it

Agostino Poggi
Department of Engineering and Architecture
University of Parma
Parma, Italy
agostino.poggi@unipr.it

***Abstract*—The analysis of systems that can be modeled as networks of interacting entities is becoming often more important in different research fields. The application of machine learning algorithms, like prediction tasks over nodes and edges, requires a manually feature extraction or a learning task to extract them automatically (embedding techniques). Several approaches have been proposed in the last years and the most promising one is represented by the Node2Vec algorithm. However, common limitations of graph embedding techniques are related to memory requirements and to the time complexity. In this paper, we propose a scalable and distributed version of this algorithm called ActorNode2vec, developed on an actor-based architecture that allows to overcome these kind of constraints. We demonstrate the efficacy of this approach with a real large network by analyzing the sensitivity of two algorithm's parameters (walk length and number of walks) with a comparison with the original algorithm. Results shows an average reduction between the 65% and the 82% in terms of the required computational times.**

***Index Terms*—Network science, embedding, graph embedding, node2vec, actodes, distributed systems, data mining, complex systems, random walk,actor model**

I. Introduction

Real systems are often characterized by heterogeneous or similar entities that interact with each other. These interactions, with their characteristics and the possible feature of the entities, are the key factors to discover the dynamics of these systems. These types of systems are pervasive in various disciplines such as sociology, health science, computer science, physics and finance. For this reason, the study of the, so-called, complex systems is gaining increasing interest in various research fields. The analysis of complex systems involves often the use of networks or graphs to model the behavior of the system. The basic idea is describing the entities in form of nodes and their interactions and dynamics with (un)directed edges. For decades the study of graph-data has been limited to analysis of the network topology with structural metrics that are capable of extracting connectivity patterns among the system entities. More recently, with the development of machine learning techniques, it is emerged the idea of taking advantages from this kind of structures, also to perform prediction tasks and knowledge discovery. For example in [1] and [2] the authors analyze a Facebook group of patients to extract new knowledge about their emotional state and disease temporal pattern by modeling them in two attributed networks: an interaction network and a friendship network. In [3] the authors analyze the same community in order to extract the giant component without using topology information with a bio-inspired algorithm. In [4] the authors uses a temporal network to model the USA financial market in order to discover correlations among the dynamics of stocks' cluster and to predict economic crises. In [5] the authors modeled the interaction between proteins as a network, with the aim of automatic predicting a correct label for each protein describing its functionalities. In light of this, this formulation of the complex systems enables analysis that keep in consideration not only the features of the entities (nodes), but also the relationships that get established among them. For this reason, the application of machine learning techniques to structured data, such as node classification task and link prediction, represents one of the most challenge in Data Mining and Network Science. Traditionally these tasks have been performed using statistical model with a manual extraction of features based on the network topology (e.g degree, clustering coefficient, modularity). During the last years, thanks to the adoption of neural networks in other data mining contexts, several works have highlighted the importance of performing previously a Representation Learning task to extract automatically features from networks that can preserve local and global information from the graph topology. The most promising approaches are represented by the embedding techniques, with the aim of learning nodes and edges representations in a n-dimensional euclidean space. To the best of our knowledge, the State of Art in this field is represented by the Node2Vec algorithm [6]. However, the main limitations of graph embedding techniques are related to memory constraints and the increase of computational times that make sometimes infeasible the application of this techniques on real and large networks. In this paper, we propose a scalable version of the Node2Vec algorithm based on a distributed actor-based framework: ActorNode2Vec. This version of the algorithm, has been developed on ActoDeS [7], a framework for the development of large concurrent and distributed systems. We demonstrate the efficacy of this approach with a real large network (Enron network [8]) by analyzing the sensitivity of two algorithm's parameters (walk length and number of walks). Results show a significant improvement in terms of reduction of computational times and

the overcome of the memory constraints in the case of a real large network.

II. Graph embedding

Combining machine learning and graph analysis techniques to work with networks is challenging since decades and two main general approaches have been proposed in literature:

- **Ad-hoc learning models for graph**: Building new learning models that can take directly a network as input and that are specialized in prediction tasks on graph-data. The most important advances in this case are represented by the Graph Neural Network Model [9] that makes a mapping of the network topology as units of a Recurrent Neural Network and Graph Convolutional Network [10] that takes the adjacency matrix and nodes specific features as input to learn prediction tasks by using different kinds of convolution operations in the graph domain.
- **Embedding techniques for non-euclidean data**: The idea of finding a latent vector representation that is able to capture the key features of the data is common in different Data mining and Machine Learning tasks (e.g Natural Language Processing, Computer Vision, Dimensionality Reduction). The main reason of this approach lies on the need of use the well-known machine learning models (e.g SVM, decision trees, regression models and neural networks) that requires an euclidean feature representation to work with non-euclidean heterogeneous data, such as text, graph-data, collection of images, database records. In this paper we will refer to this approach.

Graph embedding techniques can be divided in two general sectors: ***Factorization based Methods*** and ***Random walk based methods***. Algorithms of the first case obtain the embedding by factorizing specific matrix of the graph, like the Adjacency matrix, Laplacian matrix or the node transition probability matrix. Approaches to factorize the representative matrix vary based on the matrix properties. If the considered matrix is positive semi-definite, e.g. the Laplacian matrix, one can use eigenvalue decomposition. In other cases it is possible to factorize using gradient descent methods. Some examples of this kind of embedding algorithms are LLE [11] which preserves first-order proximity and LINE [12] which preserves also the second-order proximity in the graph. Instead, algorithms of the second case obtain a graph embedding by generating walks on the edges randomly and processing them using different learning models to extract and preserve key feature of the graph. The main advantages of random walks are that are useful to approximate many properties in the graph, including node centrality and similarity and the easy generation of them. This approach is gaining interest for its simplicity and thanks to the adoption of the Skip-gram model [13] as learning model, largely used in text embedding yet. The key innovation of this approach is optimizing the node embeddings so that nodes have similar representations if they tend to co-occur on short random walks over the graph. To the best of our knowledge the best algorithm that outperforms the State of Art in graph embedding is Node2Vec [6] that is a random walk based algorithm.

III. The Node2Vec algorithm

The Node2Vec algorithm aims to learn a vectorial representation of nodes (edges) in a graph by optimizing a neighborhood preserving objective. It extends the previous node embedding algorithm DeepWalk [14] and it is inspired to the State of Art word embedding algorithm Word2Vec [15]. From the last one, Node2Vec retrieves the Skip-gram model [13] and the idea of learning embeddings by analyzing the relationships among the entities in the form of sequences and the use of a shallow neural network model. Infact, Word2Vec learns word embeddings by analyzing the context of each word in a corpus. It predicts the current word from a sliding window of surrounding context words. The key idea of applying this model in the graph domain is to build something similar to a text corpus (a set of word sequences) by performing random walks on the graph. The result can be thought as a set of word sentences where words are substituted with nodes that compose the walk. The innovation of Node2Vec with respect to DeepWalk is in the method that is used to build biased random walks on the network. In the previous one infact, random walking is obtained by a uniform random sampling over the linked nodes, while Node2Vec combine two different strategies for the network exploration: Depth-First-Search (DFS) and Breadth-First-Search (BFS), see Figure 3. It uses also a second-order Markov chain and the Alias sampling to select the next node to be visited in the walk. The main parameters of the algorithm are presented in Table I, while the main steps of the algorithm are the following:

1) Probabilities computation over edges with a second-order Markov chain.
2) Alias sampling and random walks generation.
3) Embedding with the Neural network model.

TABLE I
Main parameters of Node2Vec

Parameter	Description
Dimension	Dimension of the embedding space
Number of walks	N° of walks to be generated per each node
Walks length	The desired length for each walk
P (Return parameter)	Controls the likelihood of immediately revisiting a node in the walk. If high more DFS, else more BFS exploration
Q (In-out parameter)	If Q higher than 1 more BFS else more DFS exploration

A. Probabilities computation

In order to feed the neural network for the embedding step, Node2Vec has to perform a fixed number of random walks starting from each node, with a fixed length. These walks are based on a second-order Markov chain that takes in consideration the parameters P and Q to mix the two different search strategies introduced previously. To achieve this result, the first duty of the algorithm is to generate all of the required probabilities over the edges for the next step of random walks

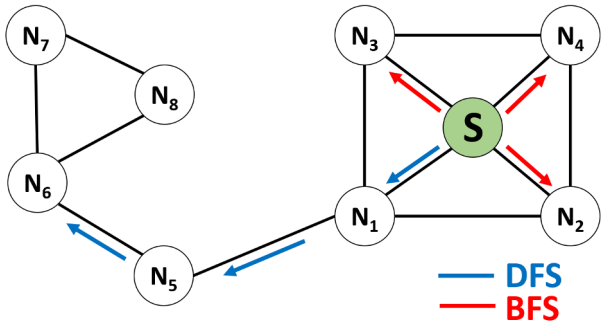

Fig. 1. BFS and DFS search strategies from node S.

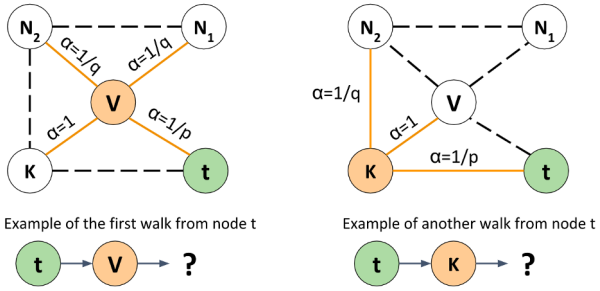

Fig. 2. Example of the calculus of probabilities over edges with a second-order Markov chain

generation. Formally, given a source node **t**, that can be also an intermediate step for other walks, a 2nd order walk has to be generated by considering the 2nd order neighborhood of node t. Let c0 = t and c1 = v, where v is a first order neighbor of node t. The next edge to be traversed is chosen by the following distribution:

$$P(C_i = x | C_{i-1} = v) = \begin{cases} \frac{\omega_{vx} \cdot \alpha_{pq}(t,x)}{Z} & if (v,x) \, \epsilon \, E \\ 0 & otherwise \end{cases} \quad (1)$$

where ω_{vx} is the edge weight, Z is the normalizing constant and $\alpha_{pq}(t,x)$ is the search bias function. The bias function α is defined with the aim of joining BFS and DFS strategies:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{P} & if \; d_{tx} = 0 \\ 1 & if \; d_{tx} = 1 \\ \frac{1}{Q} & if \; d_{tx} = 2 \end{cases} \quad (2)$$

Where d_{tx} is the geodesic distance between the starting node t and the considered node x. An example of this step is presented in Figure 2. At the end of this phase, each edge of the networks has a set of values that describe the probabilities for the edge to be traversed during a random walk, depending on the source node. This computational step represents the first bottleneck of the algorithm in terms of memory and required computational times because it depends on the dimension of the network (nodes and edges number) and on its density. How to overcome the emerging limits of this calculus, while analyzing a large network, will be further explained in the V section.

B. Generation of the random walks

Once that a set of probabilities is associated to each edge depending on the second-order neighborhood of each node, Node2Vec is ready to perform effectively the random walks based on the Number of walks and Walks length parameters. To sample the edges to be traversed from the probability distribution obtained in the previous step, the algorithm uses the Alias method [16]. This method enables the algorithm to sample efficiently from the discrete probability distribution by building and consulting two tables: the probabilities table and the alias table. To take a decision the algorithm uses a system based on a biased coin to be flipped. At the end of this phase, random walks get collected in a structure that plays the role of a sentences corpus in text mining, where sentences are composed by an ordered sequence of nodes that have been traversed by each walk.

C. Embedding with Neural networks

The neural network model used by Node2Vec is exactly the same of Word2Vec [15]. The basic idea is to train a simple neural network with a single hidden layer to predict the most probably word that follows the input one, but then the algorithm does not use this model for the task we trained it on. Instead, the goal is actually just to learn the weights of the hidden layer that will represent our euclidean representation of the input. The training set is composed by words pairs (x,y) where x is the input word and y is a nearby word of x in a fixed window in each sentence of the corpus. In the case of Node2Vec the operation is the same but the algorithm uses node pairs and the corpus is composed by the previous calculated random walks.

IV. ActoDeS Overview

ActoDeS (Actor Development System) is a Java software framework for the development of concurrent and distributed applications [17], [18] that has been experimented in different application domains (see, for example, [17], [19], [20], [21]). This software framework allows the definition of applications where the computation is distributed on some concurrent objects (from here called actors), derived from the actor model [22] and are implemented on the top of some preexisted Java software libraries and solutions for supporting concurrency and distribution. These actors interact by exchanging asynchronous messages and, after their creation, can change several times their behaviors until they kill themselves. Each behavior has the main duty of processing the incoming messages that match some message patterns. Therefore, if an unexpected message arrives, then the actor maintains it in its mailbox until a behavior will be able to process it, or a behavior kills the actor. The processing of the incoming messages is performed through some message handlers. In response to a message, an actor uses the associated handler for: sending other messages, creating new actors, changing its local state or its behavior,

setting a timeout within receiving a new message and finally killing itself. In particular, when a timeout fires, the actor sends automatically a timeout message to itself and the corresponding message handler is executed. Depending on the complexity of the application and on the availability of computing and communication resources, an application can be distributed on one or more actor spaces. Each actor space corresponds to a Java virtual machine and so a system distributed on some actor spaces can be deployed on one or more computational nodes. An actor space acts as container for a set of actors and provides them the necessary services for their execution. In particular, an actor space performs its tasks through three main run-time components, (i.e., controller, dispatcher and registry) and through two run-time actors (i.e., the executor and the service provider). In particular, the controller manages the execution of an actor space. In particular, it configures and starts the run-time components and the actors of the application, and manages its activities until the end of its execution by using different communication technologies (the current release of the software supports ActiveMQ, MINA RabbitMQ, RMI, and ZeroMQ). The registry is a run-time component that supports actor creation and message passing. In fact, it creates the references of new actors and supports the delivery of those messages that come from remote actors, by mapping each remote reference onto the local reference of the final destination of the message. The executor actor manages the execution of the actors of an actor space and in some cases creates its initial set of actors. Finally, the service provider actor provides am extensible set of services to the other actors of the application that allows them to perform new kinds of actions (e.g., to broadcast or multi-cast a message and to move from an actor space to another one). The development of a standalone or distributed application consists in the definition of the behaviors assumed by its actors and in the definition of few configuration parameters that allow the selection of the more appropriate implementation of the actors and of the other run-time components of the application. This solution allows the optimization of some or others execution attributes of an application (e.g., performance, reliability and scalability). Moreover, the deployment of an application on a distributed architecture is simplified because an actor can move to another computational node and can transparently communicate with remote actors.

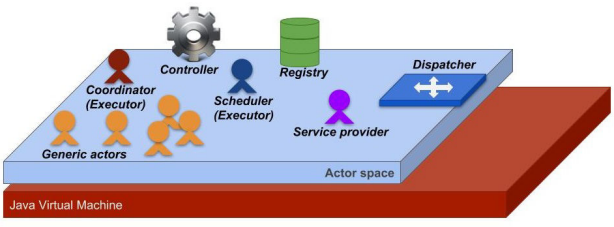

Fig. 3. The actor space structure in ActoDeS.

V. ActorNode2Vec

In this section, we present our contribution in the development of a distributed actor-based version of the Node2Vec algorithm, ActoNode2Vec, with the aim of improving performance in terms of the required computational times and scalability. In order to achieve this result we started by analyzing the three different phases of the algorithm in order to detect the main bottlenecks of the algorithm.

A. Computational limits of Node2Vec

Although Node2Vec is presented as a scalable algorithm, the original formulation and implementation involve the use of a multi-threading solution only for the second phase of the random walk generation (workers parameter) to reduce the computational times. However experimenting Node2Vec with large networks it is immediately clear that the first phase of the probabilities computation is the most critical point of the architecture. This issue is due to the construction of the required probability distribution presented in equation 1. The use of a second-order Markov chain considering the second-order neighborhood of each node, inevitably has the consequence of leading to a heavy dependency on the number of nodes and in particular on the network density. Increasing the dimension of the network (number of nodes and edges) in fact, leads to two main critical issues:

- **Memory requirements become computationally unfeasible**: Memory availability represents often a critical point in large networks mining. Requirements become infeasible on a single calculus node when the probability distribution construction requires to store a vector of probabilities on each edge depending on the second order Markov chain.
- **The explosion of algorithm's time complexity**: Required time to construct the probability distribution increases depending on the dimension of the graph and on random walks parameters (e.g number of walks and walk length)

In light of this, this phase of the algorithm represents a first considerable bottleneck of the algorithm. Our proposal is to overcome this issue using an actor-based architecture that enables a reformulation of this step. The key idea is to distribute the computation of the probabilities distribution in equation 1 using several specialized actors that take the burden of this computation each one for a subset of actors without the need of a preliminary ordering of nodes in the graph.

B. The actor-based architecture

In order to implement an actor-based solution, we have defined different actor behaviors and elements that compose the software architecture:

- **Remote Launcher**: It represents the algorithm initialization point, it has the duty of deciding how many and which computational nodes have to be involved in the execution of the algorithm.

- **ActoDeS Broker**: It is a software component implemented in ActoDeS that has the duty of initializing communications among the actor spaces.
- **Node2Vec Initiator**: This behavior has the duty of initializing the algorithm. It requires some starting messages from the remote launcher to set the Node2Vec parameters and the input. network to embed. The actor space that contains the unique actor with this behavior is defined as the primary actor space.
- **Actorspace Initiator (AI)** : It has the duty of initializing the necessary actors for the algorithm in the actor space that do not contain any Node2Vec Initiator (secondary actor spaces). In detail it has the burden of creating the actors that are responsible for the probabilities computation.
- **Coordinator**: This behavior represents the core behavior of the algorithm. Once the initialization operations are terminated, it has the duty of managing all of the actors in the architecture and of managing the entire graph embedding process.
- **Probabilities Manager (PM)** This behavior represents the reformulation of the Probabilities computation in the original Node2Vec algorithm. Each actor that assumes this behavior has the duty of computing probabilities only for a subset of nodes, considering their second-order neighborhood.

C. Starting and initialization steps

The execution of the algorithm is performed by the Remote Launcher (RL) introduced in the previous section. The RL represents a server application that has the duty of reading the input network from the memory, of choosing the embedding parameters and of initializing different ActoDeS actor spaces in the distributed network. It initializes the primary actor space, requiring the creation of the Node2Vec initiator on it and by sending to this actor the graph and the parameters. At the same time, the RL initializes an ActoDeS broker on a secondary actor space to establish the communication among the different actor spaces that build the architecture (Figure 4). After this step the RL main duties are concluded and it can wait a notification of the algorithm's termination. At the same time, the Node2Vec Initiator actor requests the creation of an Actorspace Initiator on each secondary actor space and sends to each one the entire graph to be embedded and the Node2Vec parameters. The previous ActoDeS broker become another Actorspace initiator for its actor space. These different Initiators generate several Probabilities Manager actors (PM) to prepare the architecture for the next steps. The number of actors that are created is a parameter, but the default value is equal to the number of available logical CPUs on the network node. Each PM actor receives the entire graph and the other parameters. This design choice is motivated by the next steps of the algorithm and in particular to do not have to order the nodes of the graph,that would make computational complexity exponential. Not only, this choice is due also to take advantages from the shared memory of the actors on a single node because operation on the graphs are read-only. (See Figure 5).

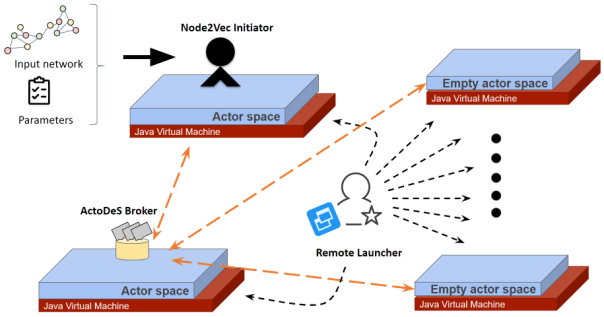

Fig. 4. Starting step and creation of the actor spaces

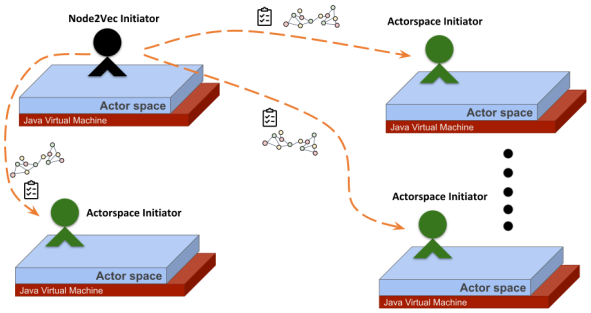

Fig. 5. Initialization step of ActorNode2Vec

D. Distributed and actor-based probabilities computation

This step represents the key difference between the original version of the algorithm and our proposal. In light of the limits presented previously, we propose a reformulation of this step where the required probability distribution is computed not on a single thread and neither with a multi-thread solution but dividing and distributing the entire computation. The simple multi-thread solution on a single computational node has been excluded because it does not permit to overcome the memory issue presented in section A in the case of a large network. We defined a specialized typology of actor's behavior (Probabilities Manager or PM) that has the duty of computing probabilities for a subset of the graph nodes. Once the initialization step is terminated, the Node2Vec Initiator changes its behavior into Coordinator. This last actor, as its name suggests, it is the real execution responsible of the following algorithms step. Its first action is to receive all of the PM's references from the Actorspace Initiators and then it kills the latters as they are no longer necessary. Secondly, the Coordinator sends all of the actors references to each PM actor. This operation is necessary to enable the PMs to understand which subset of nodes are under their responsibilities for the probabilities computation. The Probabilities Managers behavior will be explained in detail in the next section. Once the probabilities have been calculated for each second-order node

neighborhood by the PMs, they send them to the Coordinator that builds effectively the probability distribution for the next steps. Figure 6 describes this phase of the algorithm.

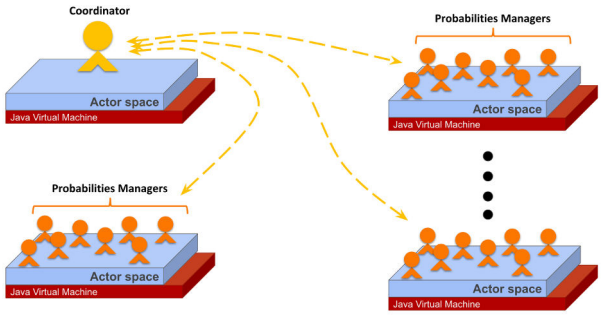

Fig. 6. Distributed and actor-based probabilities computation

E. Probabilities Managers behavior

Probabilities Managers have the duty of computing probabilities following the second-order Markov chain model presented in equations 1 and 2. Each one is responsible for a subset of the graph nodes. PMs know the references of each other in order to identify the number of actors involved in that calculus and to divide the nodes set. This choice permits to avoid nodes ordering that is an operation that grows exponentially with the number of nodes. In this way, each PM has the visibility of the entire graph to consider the neighborhood of each node and it is computationally efficient using the advantages of the shared memory among the actors on a single working node. The nodes set is divided by each PM using an ordered list of the other involved PMs references, so each actor is able to detect the subset under its responsibility and who are the responsible for the other nodes. Once probabilities have been computed by each actor using for the first-order neighborhood the edges weights and for the second-order the Markov chain model, they also prepare the probability distribution to be sampled with the alias method. The output of each PM in fact, are two data structures, one for the probabilities and one for the related alias as the method requires [16]. At the end each PM sends these two structures to the Coordinator, which has the duty of collecting and joining them in a single probability distribution. To summarize, Probabilities Managers compute probabilities for a subset of nodes and edges but they compute also the alias probabilities to improve the following sampling process.

F. Random walks generation and embedding phase

Once the Coordinator received all of the required probabilities, it builds the probability distribution to be sampled for the random walks generation. It implements the biased coin system to sample from the probabilities or from the alias data structures and generates the random walks from each node with a fixed length, depending on the algorithm's parameters. After that, the Coordinator collects all of the random walks in a corpus and feed the Neural Network model based on Word2Vec described previously. The embedding phase can represent the second bottleneck of the system in terms of time complexity, in particular depending on which technologies are used to implement the model. Some approaches to resolve this other issue are presented in the future developments section.

VI. Experimentation and Results

We experimented ActorNode2Vec with a well-known large network "Enron", to demonstrate the benefits of the actor-based solution and its efficiency in terms of time complexity. Memory issues are overcome by the scalability of the actor-based solution.

A. The Enron Email dataset

The Enron Email dataset is a large collection of over 600,000 emails generated by the past employees of the Enron Corporation and acquired by the Federal Energy Regulatory Commission during its investigation after the company bankruptcy in 2001. This collection has been processed for several scientific studies (e.g [8], [23]), analyzing discussion threads and the content of the email. These works have identified the most important component in about 37 thousands of email addresses. In particular in [23] they modeled this component as a graph by considering each email address as a node (36692 nodes), and the emails exchanges as undirected edges (183831 edges). Currently it is considered one of the most important standard network in Network Science.

B. Experiments setup

In order to compare Node2Vec and ActorNode2Vec with the Enron network we used a computer cluster with 4 linux nodes. The most performing one (Node 1) has been used to execute Node2Vec and The coordinator of ActorNode2Vec, the others to execute the Remote Launcher, the ActoDeS broker and the secondary actor spaces for the probabilities computation. Hardware specifications are the followings:

- **Node 1**: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz with 8 cores per socket, 2 threads for each core, 64 GB of RAM
- **Node 2**: Genuine Intel(R) CPU T1300 @ 1.66GHz, with 4 cores per socket, 1 threads for each core, 16 GB of RAM
- **Node 3**: Genuine Intel(R) CPU T1300 @ 1.66GHz, with 4 cores per socket, 1 threads for each core, 16 GB of RAM
- **Node 4**: Genuine Intel(R) CPU T1300 @ 1.66GHz, with 4 cores per socket, 1 threads for each core, 16 GB of RAM

C. Results

In order to compare the two versions of the algorithm, we choose an initial configuration of the parameters:

- Embedding dimensions : 100
- P : 1
- Q : 1
- Walk length: 10
- Numbers of walks: 10

TABLE II
REQUIRED TIMES WITH THE INITIAL PARAMETERS CONFIGURATION

	Probabilities and rand. walks	Embedding	Total
Node2Vec	244.23 s	390.87 s	635.10 s
ActorNode2Vec	58.3 s	69.62 s	127.92 s

TABLE III
REQUIRED TIMES WITH EMBEDDING DIMENSIONS @300

	Probabilities and rand. walks	Embedding	Total
Node2Vec	481.22 s	745.16 s	1226.38 s
ActorNode2Vec	60.28 s	823.75 s	884.03 s

In particular with parameters P and Q equal to one, we can assume that the DFS and BSF strategies are used uniformly in the random walks generation. This particular choice of parameters enable us also a comparison with Deep Walk algorithm where the two searching strategies are balanced. However P and Q do not affect the algorithm's performance. Results of this first comparison are in Table II. In this first experiment ActorNode2vec required about the 80 % less time to embed the Enron network. In this first case the improvements seem to be related both to a minus time in probabilities computation and in the embedding phase. We have further analyzed this case by increasing the number of embedding dimension to 300. In such way the embedding phase take several minutes in both versions of the algorithm because it represents the number of neurons in the hidden layer of the adopted Multilayer perceptron. It is also to report that training and inference of the statistical model could be more efficient in Python with the Gensim implementation of the neural network than in Java. Results of this case are in Table III. In fact, in this case ActorNode2Vec took about the 28 % less than Node2Vec, however the improvements are related only to a minus required time of the actor-based probabilities computation.

D. Sensitivity analysis of the parameters

In light of the results presented in the previous section, we experimented ActorNode2Vec and Node2Vec with the same initial configuration set of the parameters, but further analyzing the results by changing the number of walks and the walks length in a range between 1 and 50. Results concerning the walk length sensitivity are presented in Figure 7, while the others concerning the numbers of walks in Figure 8. These two analysis are necessary to understand how the actor-based solution of ActorNode2Vec improves the time complexity of the algorithm. Walk length and the number of walks are the main parameters that affect the probabilities computation phase in terms of required time. In both cases, it is possible to note that our solution requires less seconds to embed the large network of interest. Results have been obtained as an average of three executions of the two algorithms. ActorNode2Vec requires on average the 65 % less time by varying the Walk length and the 82 % less in the case of the number of walks.

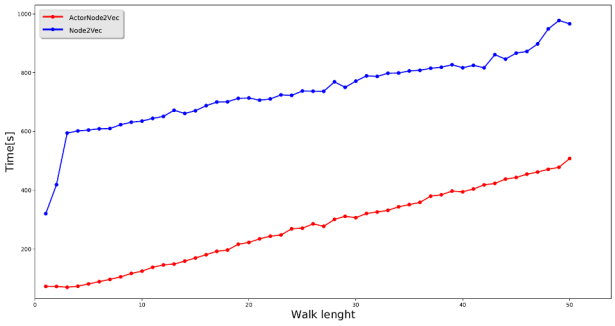

Fig. 7. Sensitivity analysis of the Walk length parameter

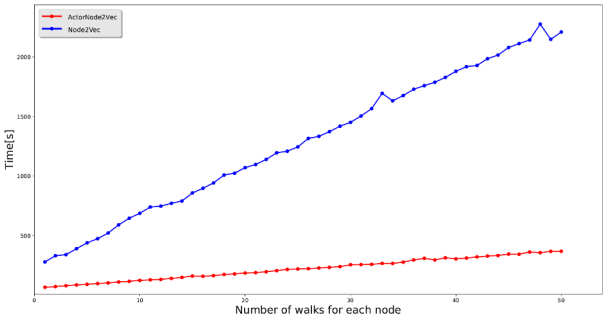

Fig. 8. Sensitivity analysis of the Number of walks parameter

VII. CONCLUSIONS AND FUTURE DEVELOPMENTS

In this paper, we propose a scalable and distributed version of the Node2Vec algorithm called ActorNode2vec, developed on an actor-based architecture that allows to overcome some limitations in terms of memory requirements and time complexity when applied to large networks. Results show an average reduction between the 65% and the 82% in terms of the required computational times, while memory issues are overcome with the scalability of the solution. Some future developments are related to the second bottleneck of the algorithm, that concerns with the time complexity of the Neural Network adopted for the embedding phase. For example an implementation with DeepLearning4j could improve required training times, as the choice of distributing also the training phase. Some other future developments are related to the use of the actors also for a distributed generation of the random walks and the use of different strategies for the network exploration. Finally, other future research activities will be dedicated to ActoDeS. We are working about the improvement of the support for knowledge and ontology based applications [**?**], [24], [25] and in the development of tool that has the aim of simplifying the design of applications [26], [27].

REFERENCES

[1] G. Lombardo, P. Fornacciari, M. Mordonini, L. Sani, and M. Tomaiuolo, "A combined approach for the analysis of support groups on facebook-the case of patients of hidradenitis suppurativa," *Multimedia Tools and Applications*, vol. 78, no. 3, pp. 3321–3339, 2019.

[2] G. Lombardo, A. Ferrari, P. Fornacciari, M. Mordonini, L. Sani, and M. Tomaiuolo, "Dynamics of emotions and relations in a facebook group of patients with hidradenitis suppurativa," in *International Conference on Smart Objects and Technologies for Social Good*, pp. 269–278, Springer, 2017.

[3] L. Sani, G. Lombardo, R. Pecori, P. Fornacciari, M. Mordonini, and S. Cagnoni, "Social relevance index for studying communities in a facebook group of patients," in *International Conference on the Applications of Evolutionary Computation*, pp. 125–140, Springer, 2018.

[4] A. Kocheturov, M. Batsyn, and P. M. Pardalos, "Dynamics of cluster structures in a financial market network," *Physica A: Statistical Mechanics and its Applications*, vol. 413, pp. 523–533, 2014.

[5] P. Radivojac, W. T. Clark, T. R. Oron, A. M. Schnoes, T. Wittkop, A. Sokolov, K. Graim, C. Funk, K. Verspoor, A. Ben-Hur, *et al.*, "A large-scale evaluation of computational protein function prediction," *Nature methods*, vol. 10, no. 3, p. 221, 2013.

[6] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, ACM, 2016.

[7] F. Bergenti, A. Poggi, and M. Tomaiuolo, "An actor based software framework for scalable applications," in *International Conference on Internet and Distributed Computing Systems*, pp. 26–35, Springer, 2014.

[8] B. Klimt and Y. Yang, "Introducing the enron corpus.," in *CEAS*, 2004.

[9] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[11] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[12] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077, International World Wide Web Conferences Steering Committee, 2015.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[14] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning

[23] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, ACM, 2014.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[16] A. J. Walker, "An efficient method for generating discrete random variables with general distributions," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 253–256, 1977.

[17] E. Franchi, A. Poggi, and M. Tomaiuolo, "Blogracy: A peer-to-peer social network," in *Censorship, Surveillance, and Privacy: Concepts, Methodologies, Tools, and Applications*, pp. 675–696, IGI Global, 2019.

[18] F. Bergenti, E. Iotti, A. Poggi, and M. Tomaiuolo, "Concurrent and distributed applications with actodes," in *MATEC Web of Conferences*, vol. 76, p. 04043, EDP Sciences, 2016.

[19] G. Angiani, P. Fornacciari, G. Lombardo, A. Poggi, and M. Tomaiuolo, "Actors based agent modelling and simulation," in *Highlights of Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection*, (Cham), pp. 443–455, Springer International Publishing, 2018.

[20] P. Fornacciari, M. Mordonini, A. Poggi, L. Sani, and M. Tomaiuolo, "A holistic system for troll detection on twitter," *Computers in Human Behavior*, vol. 89, pp. 258–268, 2018.

[21] G. Lombardo, P. Fornacciari, M. Mordonini, M. Tomaiuolo, and A. Poggi, "A multi-agent architecture for data analysis," *Future Internet*, vol. 11, no. 2, p. 49, 2019.

[22] G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*. Cambridge, MA, USA: MIT Press, 1986.

[24] F. Bergenti, A. Poggi, M. Tomaiuolo, and P. Turci, "An ontology support for semantic aware agents," in *Proc. Seventh International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2005@ AAMAS), Utrecht, The Netherlands*, 2005.

[25] A. Poggi, "Developing ontology based applications with o3l," *WSEAS Trans. on Computers*, vol. 8, no. 8, pp. 1286–1295, 2009.

[26] F. Bergenti and A. Poggi, "Exploiting uml in the design of multi-agent systems," in *International Workshop on Engineering Societies in the Agents World*, pp. 106–113, Springer, 2000.

[27] F. Bergenti and A. Poggi, "A development toolkit to realize autonomous and interoperable agents," in *Proceedings of the fifth international conference on Autonomous agents*, pp. 632–639, ACM, 2001.