A scalable and distributed actor-based version of the Node2Vec algorithm

Gianfranco Lombardo, Agostino Poggi

20TH WORKSHOP “FROM OBJECTS TO AGENTS”

Parma, June 26th-28th, 2019
Complex systems

- Different kinds of **real-world systems** are often characterized by heterogeneous entities that interact with each other.
- To represent systems’ behavior as a combination of its entities and their relationships, **these systems are modeled as networks (or graphs)**.

- Some examples:
  - Social Networks
  - Recommender systems
  - Fraud systems detection
  - Financial market
  - Biological and genetic networks
Enhance Network Analysis with Machine Learning

- Taking in consideration the relationships among entities, and not only their own domain feature, can improve ML tasks on complex systems.
- The combination between Machine Learning and Network analysis enables to other more specific prediction tasks:
  
  - **Node classification**
    - Node labeling - Missing data
    - Community detection
    - Role detection
  
  - **Link prediction**
    - Recommendations systems
    - Missing links
    - Friends suggestion on Social Network
Representation learning: Embedding techniques

- To train ML models we need an euclidean form of our dataset
- However, networks are not described by an euclidean space!
- The solution is performing a **Representation Learning task** of nodes and edges before feeding the models, in order to learn an euclidean representation of the entire network

![Diagram of network embedding](image-url)
Shallow embedding with Neural Networks

- One of the most promising ways of generating an embedding from generic data is based on the use of Neural networks.
- The main reasons are:
  - Taking advantages from the automatic feature extraction of NNs.
  - Well-known models: typically based on auto-encoders.
  - Shallow and unsupervised embedding:
    - Training a NN for a prediction task.
    - Extract the internal representation of data in terms of hidden layers’ weights.

- E.g: Skip-gram Model for word embedding (Word2Vec)
  - Auto-encoder with a single hidden layer.

![Diagram of a neural network showing input, hidden, and output layers with connections between nodes.](image-url)
Node2Vec: Node embedding algorithm

- Developed by the Stanford SNAP laboratory

- State of Art in terms of node embedding quality

- Based on the Skip-gram model and inspired by the well-known **Word2Vec algorithm**
  - The basic idea is to build a text corpora from the network
    - Words are represented by networks nodes
    - Sentences are random walks that visit each node of the network
    - Each node \( \text{(word)} \) is embedded by considering other nodes in its context
    - The context is a fixed window on the random walks \( \text{(sentences)} \)

![Diagram of Node2Vec algorithm](image)
Node2Vec: Main steps of the algorithm

- In order to extract a set of “sentences” the network has to be random explored
- Node2Vec uses a combination of **Depth-First-Search** (DFS) and **Breadth-First-Search** (BFS) for the exploration
- This combination is obtained by associating a set of probabilities to each edge following a **second-order Markov Chain**

- Node2Vec can be summarized in three main steps:
  - Probabilities computation
  - Random walks generation
  - Embedding with Skip-gram model
Node2Vec: Hyperparameters

- **Dimension**: embedding space dimensions
- **N° of walks (nw)**: Number of walks to be generated for each node
- **Walks length (wl)**: Fixed length for each walk
- **Return parameter (p)**: Controls the likelihood of immediately revisiting a node in the walk
- **In-Out parameter (q)**: If q >1 more BFS exploration, else more DFS exploration
- **Window size**: Size of the window that take in consideration the context of each node in its walks
Node2Vec: Probabilities computation

- Probabilities are computed following a second-order Markov chain.
- For each source node $t$ it is considered its second-order neighborhood.
- The next edge to be traversed is chosen considering the probabilities distribution in equation:
  - Where $\omega$ is the edge weight, $d_{tx}$ is the geodesic distance and $\alpha$ is the bias function to combine DFS and BFS.

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

$$\alpha_{pq}(t,x) = \begin{cases} 
\frac{1}{P} & \text{if } d_{tx} = 0 \\
1 & \text{if } d_{tx} = 1 \\
\frac{1}{Q} & \text{if } d_{tx} = 2
\end{cases}$$
Node2Vec: Random walks generation and embedding

- Once a set of probabilities have been computed for each node’s neighborhood, walks are generated by sampling the probability distribution using the **Alias method technique**
  - Alias method: Technique to sample discrete distribution based on a biased coin to be flipped

- The embedding phase is performed using the skip-gram model where the hidden layer has a number of neurons equal to the embedding dimension parameter
Node2Vec: Computational limits

- Node2Vec uses a multi-threading solution only for the random walk generation but not for the probabilities computation.

- On large and dense networks, probabilities computation become a critical point because of the use of the second-order Markov Chain (heavy dependency on the number of edges and nodes):
  - Memory requirements become unfeasible: Need of storing vector of probabilities on each edge
  - Explosion of algorithm’s time complexity: Increasing the dimension of the network or the random walks parameters ($nw$ and $wl$), the construction of the probability distribution can require several hours or days.
Our proposal: ActorNode2Vec

- The goal is to overcome the main limits of the original algorithm with an actor-based architecture
- Probabilities computation can be distributed on different machines
- The actor solution provides also scalability and good performances for time complexity

- Developed using the ActoDeS framework (Actor Development System)
  - Concurrent and distributed application development
  - Use of the actor model
  - Actors interact each other by exchanging asynchronous messages and changing their behavior
ActorNode2Vec: Architecture

Components:
- **Remote Launcher**: It initializes the architecture deciding which computational nodes have to be involved
- **ActoDeS Broker**: Software component that initializes communication among actor spaces

Actors’ behaviors:
- **Node2Vec Initiator**: Set hyper-parameters, defines a primary actor space and manages the initialization of the algorithm
- **Actorspace Initiator (AI)**: It initializes a generic secondary actor space and generate the next actors for the probabilities computation
- **Coordinator**: It manages the execution of the algorithm once the initialization steps are finished
- **Probabilities Manager (PM)**: It represents a reformulation of the probabilities computation step of Node2Vec. Each actor with this behavior has the duty of computing probabilities only for a specific subset of nodes
ActorNode2Vec: Initialization steps

Starting step and actor spaces creation

Initialization step of ActorNode2Vec
ActorNode2Vec: Probabilities computation

- The Node2Vec initiator actor changes its behavior in “Coordinator” after the initialization.
- It receives all the probabilities manager actors references.
- After that it communicates these references to all probabilities managers.
  - In this way actors know each other and can select different subsets of nodes for the probabilities computation.
- PMs computes a set of probabilities and the necessary items for the following use of the Alias method.
  - At the end they send probabilities to the Coordinator.
Random walks generation and embedding

- Once all of the probabilities have been computed the Coordinator begins to sample the distribution with the Alias method (precomputed by the probabilities Managers)

- After that it begins to train the Skip-gram model to extract the embedding

- The embedding step has been implemented using Java libraries instead of using the Python Gensim implementation of Node2Vec
Experiments specifications

- We evaluated ActorNode2Vec in terms of required time to embed a large network
- Memory issues are overcome by the scalability of the actor-based solution

- Computer cluster with 4 Linux nodes
  - 1 Intel(R) Xeon(R) 2.10 GHz with 8 cores and 64 GB of RAM
  - 3 Genuine Intel(R) T1300 - 1.66 GHz with 4 cores and 16 GB of RAM

- We experimented the algorithm on the Enron email dataset
  - 37,000 emails generated by the past employees of the Enron Corporation
  - Gold dataset and network in literature
  - Each email address is modeled as a node
  - Email exchanges are modeled as undirected edges (about 184,000 edges)
Results

- Configuration of the hyper-parameters:
  - Dimension: 100
  - $P$ and $Q = 1$
  - Walk length = 10
  - Number of walks for each node = 10

<table>
<thead>
<tr>
<th></th>
<th>Probabilities and RWs generation</th>
<th>Embedding</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node2Vec</td>
<td>244.23 s</td>
<td>390.87 s</td>
<td>635.10 s</td>
</tr>
<tr>
<td>ActorNode2Vec</td>
<td><strong>58.3 s</strong></td>
<td>69.62 s</td>
<td><strong>127.92 s</strong></td>
</tr>
</tbody>
</table>

ActorNode2Vec requires the 80% time less than Node2Vec
Results with dimension @ 300

- Configuration of the hyper-parameters:
  - **Dimension:** 300
  - P and Q = 1
  - Walk length = 10
  - Number of walks for each node = 10

<table>
<thead>
<tr>
<th></th>
<th>Probabilities and RWs generation</th>
<th>Embedding</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node2Vec</td>
<td>481.22 s</td>
<td>745.16 s</td>
<td>1226.38 s</td>
</tr>
<tr>
<td>ActorNode2Vec</td>
<td>60.28 s</td>
<td>823.75 s</td>
<td><strong>884.03 s</strong></td>
</tr>
</tbody>
</table>

ActorNode2Vec requires the 28% time less than Node2Vec
Sensitivity analysis of walk length parameter

- Using the same initial configuration of the hyper-parameters
- Changing the walk length in a range between 1 and 50
- Each configuration has been tested 3 times taking the average required time
- **ActorNode2Vec requires on average the 65% less time**
Sensitivity analysis of number of walks parameter

- Using the same initial configuration of the hyper-parameters
- Changing the number of walks in a range between 1 and 50
- Each configuration has been tested 3 times taking the average required time
- **ActorNode2Vec requires on average the 82% less time**
Conclusions

- Results show a significant reduction in required times with ActorNode2Vec using a moderate large network.
- Memory issues are resolved thanks to the scalability of the actor solution.
- However, the embedding phase remains another bottleneck of the algorithm but not with a hard dependency on the network dimension.
Future developments

- Enhance scalability by using the probabilities managers to distribute also the random walk generation
- Use of DeepLearning4J libraries to implement the Skip-gram model
- Use of different network exploration strategies
- Development of an incremental way to do the embedding
- Development of tool for ActoDeS to simplify the design of applications