Hindawi
Complexity
Volume 2019, Article ID 1439415, 14 pages
https://doi.org/10.1155/2019/1439415

Research Article

NOESIS: A Framework for Complex Network Data Analysis

Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero

Department of Computer Science and Artificial Intelligence & Research Center for Information and Communications Technologies (CITIC), University of Granada, Granada, Spain

Correspondence should be addressed to Fernando Berzal; berzal@acm.org

Received 28 June 2019; Accepted 9 September 2019; Published 31 October 2019

Academic Editor: Giulio Cimini

Network data mining has attracted a lot of attention since a large number of real-world problems have to deal with complex network data. In this paper, we present NOESIS, an open-source framework for network-based data mining. NOESIS features a large number of techniques and methods for the analysis of structural network properties, network visualization, community detection, link scoring, and link prediction. The proposed framework has been designed following solid design principles and exploits parallel computing using structured parallel programming. NOESIS also provides a stand-alone graphical user interface allowing the use of advanced software analysis techniques to users without prior programming experience. This framework is available under a BSD open-source software license.

1. Introduction

Data mining, an interdisciplinary subfield of computer science, studies the process of extracting valuable information from data by discovering patterns or relationships. Data mining includes classification, regression, clustering, or anomaly detection, among other tasks [1]. A large number of tools and techniques are available for tabular data, where all data examples can be represented as tuples in a relation and they share the same set of attributes. However, many problems involve dealing with relational data, where instances are explicitly related through semantic relations. Classic data mining techniques designed for tabular data have severe limitations when we try to fully exploit the relationships available in relational data. The scientific community has focused on the development of data mining techniques for relational data, leading to the availability of a large collection of techniques for the analysis of the structural properties of networks [2], their pleasant visualization [3], the detection of existing communities [4], and scoring existing or potential links to rank existing ones or predict their existence [5, 6]. Network data mining involves different problems. Some examples include the prediction of previously unknown protein interactions in protein-protein interaction networks [7], the prediction of collaborations and tendencies in coauthorship networks [8], or ranking the most relevant websites according to a user query [9].

Different software tools for analyzing relational data have been developed, according to their main goal and the type of user they are directed to. Several tools provide their functionality to end users through closed graphical user interfaces, leading to improved usability but also neglecting the possibility of using the provided techniques in ways unforeseen by their software developers and integrating them in other software projects. Other tools were designed as software libraries that can be used in different software projects, providing more functionality at the cost of limiting their usage to users with prior programming experience.

Most existing frameworks are focused towards a specific user community or task, as shown in Table 1. For example, Graphviz [10] and Cytoscape [11] are two popular alternatives for network visualization. On the contrary, igraph [12], NetworkX [13], and SNAP [14] are mainly focused on providing reusable collections of algorithms for network manipulation and analysis. Pajek [15], NodeXL [16], Gephi [17], and UCINET [18] are some of the most widely used tools for social network analysis (SNA) (a more comprehensive and up-to-date list of available software tools for network analysis can be found at Wikipedia: https://en.wikipedia.org/wiki/Social_network_analysis_software).

In this paper, we introduce NOESIS (Network-Oriented Exploration, Simulation, and Induction System), a software framework released under a permissive BSD open-source license for analyzing and mining complex networks. NOESIS is built on top of a structured parallel programming suite of design patterns, providing a large number of network mining techniques that are able to exploit multiple processing cores available in current microprocessors for a more efficient computation. Our framework is fully written in Java, which means it is portable across different hardware and software platforms, provided they include a Java virtual machine. In addition, we provide an API binding for Python, which enables the use of NOESIS from the Python scripting language, often used by data scientists.

Our paper is structured as follows: In Section 2, we describe and discuss the NOESIS architecture and design. In Sections 3 and 4, the network analysis and network mining techniques available in NOESIS are presented. In Section 5, NOESIS performance is compared to that of other popular network analysis tools. Finally, our project's current status and future directions are described in Section 6.

2. Design of the NOESIS Framework

NOESIS has been designed to be an easily extensible framework whose architecture provides the basis for the implementation of network data mining techniques. In order to achieve this, NOESIS is designed around abstract interfaces and a set of core classes that provide essential functions, which allows the implementation of different features as independent components with strong cohesion and loose coupling. NOESIS components are designed to be maintainable and reusable, yet highly efficient.

2.1. System Architecture. The NOESIS framework architecture and its core subsystems are displayed in Figure 1. These subsystems are described below.

The lowest-level component is the hardware abstraction layer (HAL), which provides support for the execution of algorithms in a parallel environment and hides implementation details and much of the underlying technical complexity. This component provides different building blocks for implementing well-studied parallel programming design patterns, such as MapReduce [19]. For example, we would just write *result = (double) Parallel.reduce(index ->x [index] * y[index], ADD, 0, SIZE-1)* to compute the dot product of two vectors in parallel. The HAL not only implements structured parallel programming design patterns but also is responsible for task scheduling and parallel execution. It allows the adjustment of parallel execution parameters, including the task scheduling algorithm.

The reflective kernel is at the core of NOESIS and provides its main features. The reflective kernel provides the base models (data structures) and tasks (algorithms) needed to perform network data mining, as well as the corresponding metaobjects and metamodels, which can be manipulated at run time. It is the underlying layer that supports a large collection of network analysis algorithms and data mining techniques, which are described in the following section. Different types of networks are dealt with using a unified interface, allowing us to choose the particular implementation that is the most adequate for the spatial and computational requirements of each application. Algorithms provided by this subsystem are built on top of the HAL building blocks, allowing the parallelized execution of algorithms whenever possible.

The data access layer (DAL) provides a unified interface to access external data sources. This subsystem allows reading and writing networks in different file formats, providing implementations for some of the most important standardized network file formats. This module also enables the development of data access components for other kinds of data sources, such as network streaming.

Finally, an application generator is used to build a complete graphical user interface following a model-driven software development (MDSD) approach. This component provides a user-friendly interface that allows users without programming skills to use most of the NOESIS framework features.

2.2. Core Classes. The core classes and interfaces shown in Figure 2 provide the foundation for the implementation of different types of networks with specific spatial and computational requirements. Basic network operations include adding and removing nodes, adding and removing links, or querying a node neighborhood. More complex operations are provided through specialized components.

An object-oriented purist might be surprised that nodes and links do not explicitly appear as Node and Link classes among the NOESIS core classes. This apparent violation of OO design principles has a technical reason—the performance degradation of garbage-collected programming languages (such as Java, C#, or Python) due to increased memory fragmentation. When dealing with large networks, the presence of millions of individual objects in the heap would significantly increase memory fragmentation and degrade the performance of the garbage collector. Therefore, nodes and links are not considered to be core classes in NOESIS, although they can be certainly used when needed (as lightweight facades to the underlying network data structure encapsulated within particular Network subclasses).

NOESIS supports networks with attributes in both their nodes and their links. These attributes are defined according to predefined data models, including categorical and numerical values, among others.

2.3. Supported Data Formats. Different file formats have been proposed for network datasets. Some data formats are more space efficient, whereas others are more easily parseable.

NOESIS supports reading and writing network datasets using the most common data formats. For example, the GDF

Table 1: Feature comparison of some popular network analysis software packages.

Tool name	Software license	Software platform	User interface	Programming library	Parallelization support	Extensibility
UCINET	Commercial	Pascal	✓			
Pajek	Closed	Delphi	✓			
Cytoscape	LGPL	Java/JavaScript	✓			✓
NodeXL	MPL	Excel (.NET)	✓			
Gephi	GPL	Java	✓	✓		
igraph	GPL	C/Python/R		✓		✓
NetworkX	BSD	Python	✓		✓	
Graphviz	EPL	C	✓			
SNAP	BSD	C++/Python	✓	✓	✓	
NOESIS	BSD	Java/Python	✓	✓	✓	✓

GPL: GNU Public License; LGPL: GNU Lesser General Public License; MPL: Microsoft Public License; BSD: Berkeley Software Distribution; EPL: Eclipse Public License.

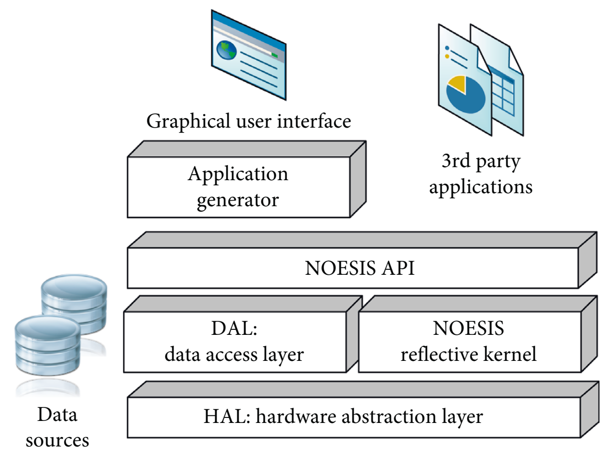


Figure 1: NOESIS framework architecture and its core subsystems.

file format is a CSV-like format used by some software tools such as GUESS and Gephi. It supports attributes in both nodes and links. Another supported file format is GML, which stands for Graph Modeling Language. GML is a hierarchical ASCII-based file format. GraphML is another hierarchical file format based on XML, the ubiquitous eXtensible Markup Language developed by the W3C.

Other file formats are supported by NOESIS, such as the Pajek file format, which is similar to GDF, or the file format of the datasets from the Stanford Network Analysis Platform (SNAP) [20].

2.4. Graphical User Interface. In order to allow users without programming knowledge to use most of the NOESIS features, a lightweight easy-to-use graphical user interface is included with the standard NOESIS framework distribution. The NOESIS GUI allows loading of nontechnical end users, visualizing them, and analyzing their own network datasets by applying all the techniques provided with NOESIS.

Some screenshots of this GUI are shown in Figure 3. A canvas is used to display the network in every moment. The network can be manipulated by clicking or dragging nodes. At the top of the window, a menu gives access to different options and data mining algorithms. The *Network* menu allows loading a network from an external source and exporting the results using different file formats, as well as creating images of the current network visualization as both raster graphics (a bitmap in the PNG or JPEG file format) and vector graphics (in the SVG format). The *View* menu allows the customization of the network appearance by setting specific layout algorithms and custom visualization styles. In addition, this menu allows binding the visual properties of nodes and links to their attributes. The *Data* menu allows the exploration of attributes for each node and link. Finally, the *Analysis* menu gives access to most of the techniques that will be described in the following sections.

3. Network Analysis Tools

NOESIS is designed to ease the implementation of network analysis tools. It also includes reusable implementations of a large collection of popular network-related techniques, from graph visualization [3] and common graph algorithms to network structural properties [21] and network formation models [22]. The network analysis tools included in NOESIS and the modules that implement them are introduced in this section.

3.1. Network Models. NOESIS implements a number of popular random network generation models, which are described by probability distributions or random processes. Such models have been found to be useful in the study and understanding of certain properties or behaviors observed in real-world networks. Some examples of these models are shown in Figure 4.

Among the models included in NOESIS, the Erdös–Rényi model [23] is one of the simplest ones. The Gilbert model [24] is similar to the Erdös–Rényi model, but a probability of existence is given for links instead. The anchored network model is also similar to the two previous models, with the advantage of reducing the occurrence of isolated nodes, but at the cost of being less than perfectly random. The connected random model is a variation of the anchored model that avoids isolated nodes.

Other models included in NOESIS exhibit specific properties often found in real-world networks. For example, the Watts–Strogatz model [25] generates networks with small-world properties, that is, low diameter and high

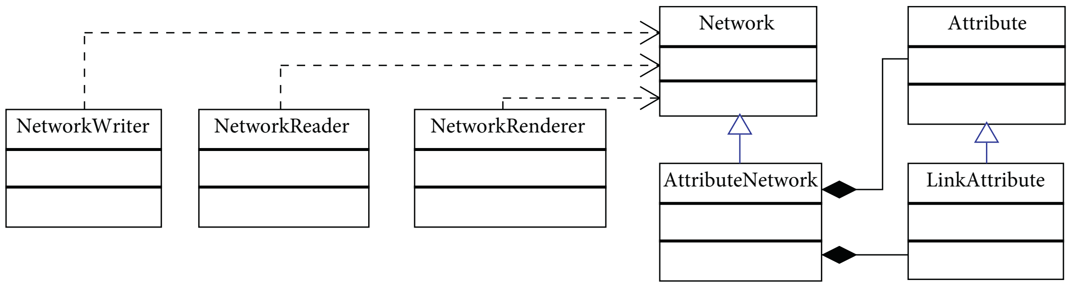
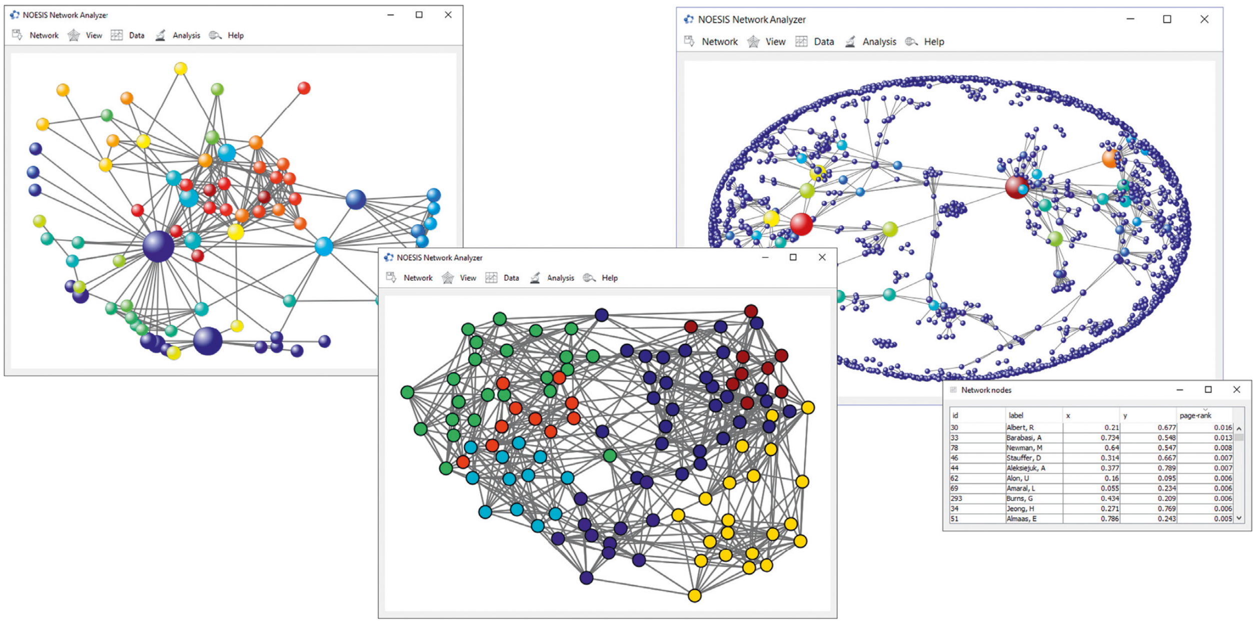


FIGURE 2: UML class diagram depicting some of the NOESIS core classes and interfaces.



id	label	x	y	page-rank
30	Albert, R	0.21	0.677	0.016
33	Barabasi, A	0.734	0.548	0.013
78	Newman, M	0.64	0.547	0.008
46	Stauffer, D	0.314	0.667	0.007
44	Aleksiejuk, A	0.377	0.789	0.007
62	Alon, U	0.16	0.095	0.006
69	Amaral, L	0.055	0.234	0.006
293	Burns, G	0.434	0.209	0.006
34	Jeong, H	0.271	0.769	0.006
51	Almaas, E	0.786	0.243	0.005

FIGURE 3: Different screenshots of the NOESIS graphical user interface.

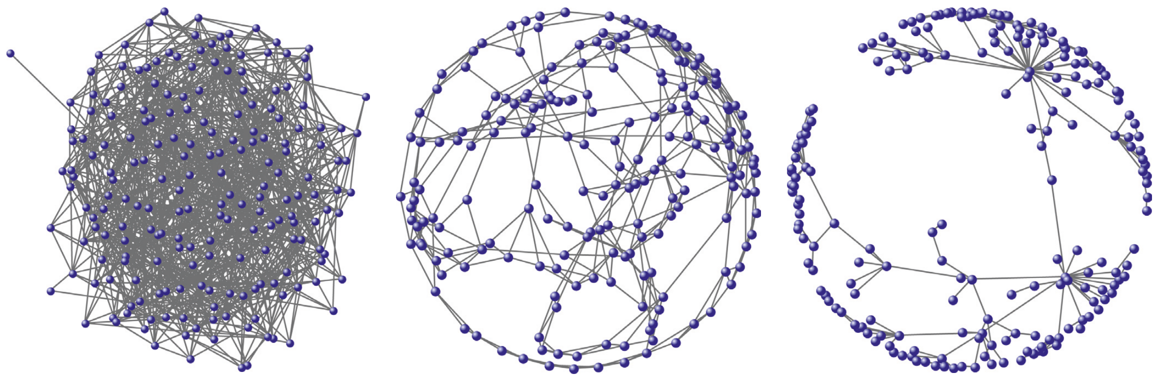

(a) (b) (c)

FIGURE 4: Random networks generated using the Erdös–Rényi model (a), the Watts–Strogatz model (b), and the Barabási–Albert model (c).

clustering. This model starts by creating a ring lattice with a given number of nodes and a given mean degree, where each node is connected to its nearest neighbors on both sides. In the following steps, each link is rewired to a new target node with a given probability, avoiding self-loops and link duplication.

Despite that the small-world properties exhibited by networks generated by the Watts–Strogatz model are closer to properties of real-world networks than those generated by models based on the Erdös–Rényi approach, they still lack some important properties observed in real networks. The Barabási–Albert model [26] is another well-known model that generates networks whose node degree distribution follows a power law, which leads to scale-free networks. This model is driven by a preferential attachment process, where new nodes are added and connected to existing nodes with a probability proportional to their current degree. Another model with very similar properties to the Barabási–Albert model is Price's citation model [27].

In addition to random network models, a number of regular network models are included in NOESIS. These models generate synthetic networks that are useful in the process of testing new algorithms. The regular network models include complete networks, where all nodes are interconnected; star networks, where all nodes are connected to a single hub node; ring networks, where each node is connected to its closest two neighbors along a ring; tandem networks, like the ring model but without closing the loop; mesh networks, where nodes are arranged in rows and columns and connected only to their adjacent nodes; toruses; meshes, where nodes in the extremes of the mesh are connected; hypercubes; binary trees; and isolates, a network without links.

3.2. Network Structural Properties. Network structural properties allow the quantification of features or behaviors present in the network. They can be used, for instance, to measure network robustness or reveal important nodes and links. NOESIS considers three types of structural properties: node properties, node pair properties (for pairs both with and without links among them), and global properties.

NOESIS provides a large number of techniques for analyzing network structural properties. Many structural properties can be computed for nodes. For example, in-degree and out-degree indicate the number of incoming and outgoing links, respectively. Related to the node degree, two techniques to measure node degree assortativity have been included: biased [28] and unbiased [29] node degree assortativity. Node assortativity is a score between −1 and 1 that measures the degree correlation between pairs of connected nodes. The clustering coefficient can also be computed for nodes. The clustering coefficient of a node is the fraction of its neighbors that are also connected among them.

Reachability scores are centrality measures that allow the analysis of how easy it is to reach a node from other nodes. The eccentricity of a node is defined as the maximum distance to any other node [30]. The closeness, however, is the inverse of the sum of the distance from a given node to all other nodes [31]. An adjusted closeness value that normalizes the closeness according to the number of reachable nodes can also be used. Inversely to closeness, the average path length is defined as the mean distance of all shortest paths to any other node. Decay is yet another reachability score, computed as the summation of a delta factor powered by the path length to any other node [22]. It is interesting to note that, with a delta factor close to 0, the measure becomes the degree of the node, whereas with a delta close to 1, the measure becomes the size of the component the node is located at. A normalized decay score is also available.

Betweenness, as reachability, is another way to measure node centrality. Betweenness, also known as Freeman's betweenness, is a score computed as the count of shortest paths the node is involved in [32]. Since this score ranges from $2n-1$ to $n^2-(n-1)$ for the number of nodes n in strongly connected networks, a normalized variant is typically used.

Influence algorithms provide a different perspective on node centrality. These techniques measure the power of each node to affect others. The most popular influence algorithm is PageRank [9] since it is used by the Google search engine. PageRank computes a probability distribution based on the likelihood of reaching a node starting from any other node. The algorithm works by iteratively updating node probability based on direct neighbor probabilities, which leads to convergence if the network satisfies certain properties. A similar algorithm is HITS [33], which stands for hyperlink-induced topic search. It follows an iterative approach, as PageRank, but computes two scores per node: the hub, which is a score related to how many nodes a particular node links, and the authority, which is a score related to how many hubs link a particular node. Both scores are connected by an iterative updating process: authority is updated according to the hub scores of nodes connected by incoming links, and hub is updated according to authority scores of nodes connected by outgoing links. Eigenvector centrality is another iterative method closely related to PageRank, where nodes are assigned a centrality score based on the summation of the centrality of their neighbor nodes. Katz centrality considers all possible paths but penalizes long ones using a given damping factor [34]. Diffusion centrality [35] is another influence algorithm based on Katz centrality. The main difference is that while Katz considers infinite-length paths, diffusion centrality considers only paths of a given limited length.

In the following Java example, we show how to load a network from a data file and compute its structural properties using NOESIS, in particular its PageRank scores:

```
FileReader fileReader = new FileReader
("karate.gml");
NetworkReader reader = new GMLNetworkReader
(fileReader);
Network network = reader.read();
PageRank task = new PageRank(network);
NodeScore score = task.call();
```

We also show how to implement this example using the NOESIS API for Python:

```
ns = Noesis()
network_reader = ns.create_network_reader
("GML")
network = network_reader.read("karate.gml")
pagerank_scorer = ns.create_node_scorer
("PageRank")
scores = pagerank_scorer.compute(network)
ns.end()
```

Apart from the aforementioned centrality measures, NOESIS also provides ready-to-use implementations of percolation centrality [36] and cross-clique connectivity [37], ensuring that all mainstream centrality measures are provided by our software platform. Beyond common centrality measures, NOESIS can compute a robustness coefficient [38] to analyze the structural robustness of complex networks, which is specifically useful for evaluating scenarios of targeted attacks.

Different structural properties for links can also be computed by NOESIS, for example, link betweenness, which is the count of shortest paths the link is involved in, or link rays, which is the number of possible paths between two nodes that cross a given link. Some of these properties are used by different network data mining algorithms.

3.3. Network Visualization Techniques. Humans are still better than machines at recognizing certain patterns when analyzing data in a visual way. Network visualization is a complex task since networks tend to be huge, with thousands of nodes and links. NOESIS enables the visualization of networks by providing the functionality needed to render the network and export the resulting visualization using different image file formats.

NOESIS provides different automatic graph layout techniques, such as the well-known Fruchterman–Reingold [39] and Kamada–Kawai [40] force-based layout algorithms (see Figure 5). Force-based layout algorithms assign forces among pairs of nodes and solve the system to reach an equilibrium point, which usually leads to an aesthetic visualization.

Hierarchical layouts [3], which arrange nodes in layers trying to minimize edge crossing, are also included. Different radial layout algorithms are included as well [42]. These layouts are similar to the hierarchical ones but arrange nodes in concentric circles. Several regular layouts are provided too since they are common for visualizing regular networks, such as meshes or stars.

NOESIS allows tuning the network visualization look and feel. The visual properties of nodes and links can be customized, including color, size, and borders. In addition, visual properties can be bound to static or dynamic properties of the network. For example, node sizes can be bound to a specific centrality score, allowing the visual display of quantitative information.

NOESIS provides an interactive graphical user interface that can be used to explore modest-sized networks and test the results different methods provide using an easy-to-use drag-and-drop interface. This complementary tool can be useful for analyzing small datasets and also for teaching courses on the subject. The current version of NOESIS includes a JavaFX network renderer that makes use of hardware acceleration, the same kind of technology employed by other network analysis tools with a strong focus on network visualization, such as Gephi or Cytoscape. The NOESIS JavaFX-based network renderer offers the same performance gains that can be obtained by other tools that also make use of hardware acceleration (e.g., Gephi). It should be noted, however, that the complementary UI tool included within the NOESIS distribution was not designed to handle large networks since it is often impractical to visualize networks with thousands or millions of nodes and edges in a million-pixel computer screen. NOESIS offers Gephi-like performance, and for visually navigating truly large networks, specialized solutions, such as Cytoscape, are recommended. In such situations, for network data analysis, the programmatic NOESIS API should be used instead of the interactive exploration tool.

4. Network Data Mining Techniques

Network data mining techniques exist for both unsupervised and supervised settings. NOESIS includes a wide array of community detection methods [4] and link prediction techniques [43]. These algorithms are briefly described below.

4.1. Community Detection. Community detection can be defined as the task of finding groups of densely connected nodes. A wide range of community detection algorithms have been proposed, exhibiting different pros and cons. NOESIS features different families of community detection techniques and implements more than ten popular community detection algorithms. Some examples of these community detection techniques are shown in Figure 6. The included algorithms, their time complexity, and their bibliographic references are shown in Table 2.

NOESIS provides hierarchical clustering algorithms. Agglomerative hierarchical clustering treats each node as a cluster and then iteratively merges clusters until all nodes are in the same cluster [56]. Different strategies for the selection of clusters to merge have been implemented, including the single-link method [45], which selects the two clusters with the smallest minimum pairwise distance; the complete-link method [46], which selects the two clusters with the smallest maximum pairwise distance; and the average-link method [47], which selects the two clusters with the smallest average pairwise distance.

Modularity-based techniques are also available in our framework. Modularity is a score that measures the strength of particular division into modules of a given network. Modularity-based techniques search for communities by attempting to maximize their modularity scores [57]. Different greedy strategies, including fast greedy [48] and multistep greedy [49], are available. These greedy algorithms merge pairs of clusters that maximize the resulting modularity, until all possible merges would reduce the network modularity.

Partitional clustering is another common approach. Partitioning clustering decomposes the network and performs an iterative relocation of nodes between clusters. For example, Kernighan–Lin bipartitioning [50] starts with an arbitrary partition into two clusters. Then, nodes are iteratively exchanged between both clusters to minimize the number of links between them. This approach can be applied multiple times to subdivide the obtained clusters. K-means community detection [51] is an application of the traditional K-means clustering algorithm to networks and another prominent example of partitioning community detection.

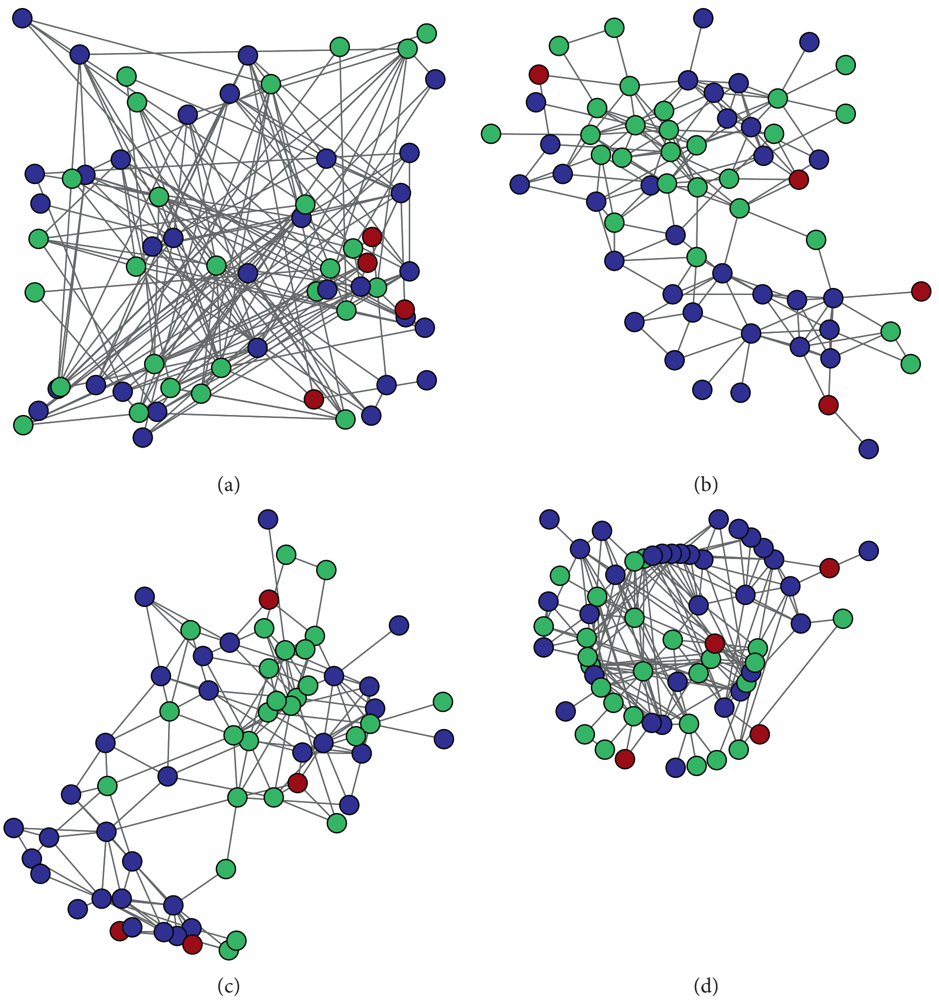


Figure 5: The same dolphin social network [41] represented using different network visualization algorithms: random layout (a); Kamada–Kawai layout (b); Fruchterman–Reingold layout (c); circular layout using the average path length (d).

Spectral community detection [56] is another family of community detection techniques included in NOESIS. These techniques use the Laplacian representation of the network, which is a network representation computed by subtracting the adjacency matrix of the network from a diagonal matrix where each diagonal element is equal to the degree of the corresponding node. Then, the eigenvectors of the Laplacian representation of the network are computed. NOESIS includes the ratio cut (EIG1) algorithm [52], Jordan and Weiss NG (KNSC1) algorithm [53], and spectral K-means [54].

The BigClam overlapping community detector is also available in NOESIS [55]. In this algorithm, each node has a profile, which consists in a score between 0 and 1 for each cluster that is proportional to the likelihood of the node belonging to that cluster. Also, a score between pairs of nodes is defined yielding values proportional to their clustering assignment overlap. The algorithm iteratively optimizes each node profile to maximize the value between connected nodes and minimize the value among unconnected nodes.

In the following example, we show how to load a network from a data file and detect communities with the KNSC1 algorithm using NOESIS:

```
FileReader fileReader = new FileReader
("mynetwork.net");
NetworkReader reader = new PajekNetwork
Reader(fileReader);
Network network = reader.read();
CommunityDetector task = new NJWCommunity
Detector(network);
Matrix results = task.call();
```

The same example can also be coded in Python using the NOESIS API for Python:

```
ns = Noesis()
network_reader = ns.create_network_reader
("Pajek")
network = network_reader.read("mynetwork.
net")
```

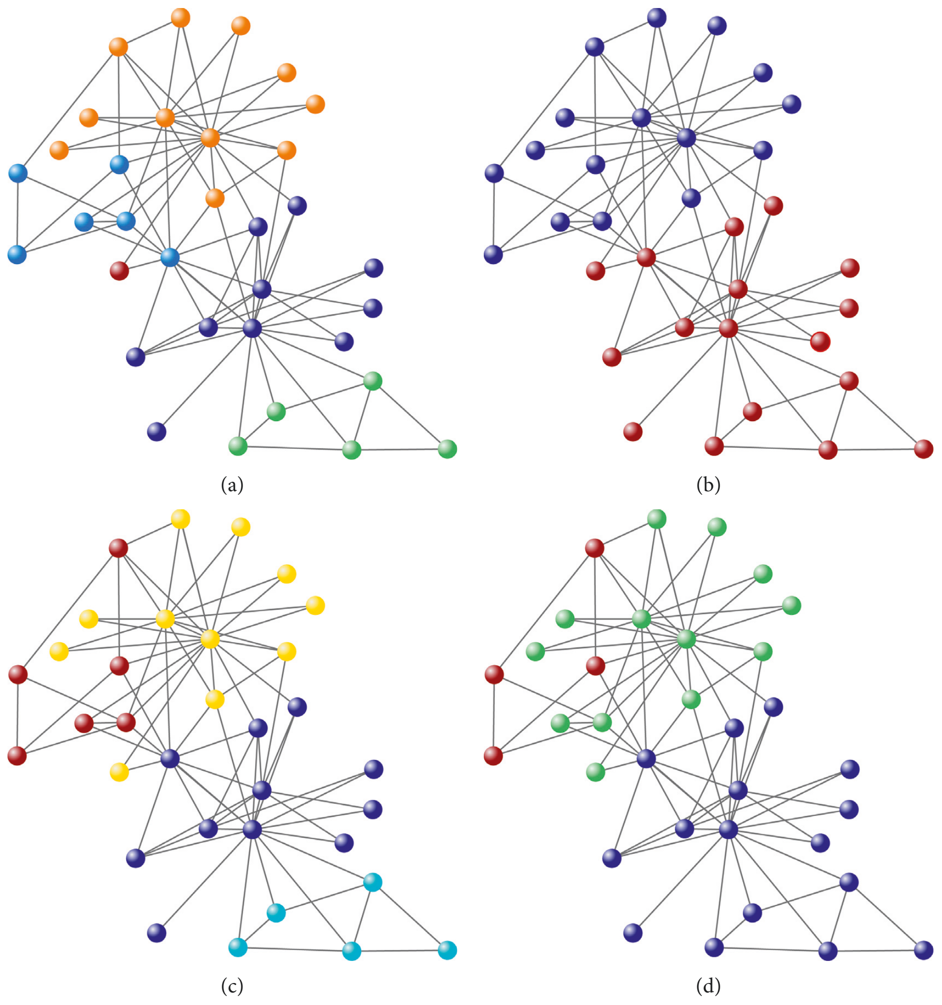


Figure 6: Different community detection methods applied to Zachary's karate club network [44]: fast greedy partitioning (a); Kernighan–Lin bipartitioning (b); average-link hierarchical partitioning (c); complete-link hierarchical partitioning (d).

Table 2: Computational time complexity and bibliographic references for the community detection techniques provided by NOESIS.

Type	Name	Complexity	Reference
Hierarchical	Single link (SLINK)	$O(v^2)$	[45]
	Complete link (CLINK)	$O(v^2 \log v)$	[46]
	Average link (UPGMA)	$O(v^2 \log v)$	[47]
Modularity	Fast greedy	$O(kvd \log v)$	[48]
	Multistep greedy	$O(kvd \log v)$	[49]
Partitional	Kernighan–Lin bipartitioning	$O(v^2 \log v)$	[50]
	K-means	$O(kvd)$	[51]
Spectral	Ratio cut (EIG1)	$O(v^3)$	[52]
	Jordan–Weiss NG (KNSC1)	$O(v^3)$	[53]
	Spectral K-means	$O(v^3)$	[54]
Overlapping	BigClam	$O(v^2)$	[55]

In the time complexity analysis, v is the number of nodes in the network, d is the maximum node degree, and k is the desired number of clusters.

```
community_detector = ns.create_community_
detector("NJW")
communities = community_detector.compute
(network)
ns.end()
```

4.2. Link Scoring and Prediction. Link scoring and link prediction are two closely related tasks. On the one hand, link scoring aims to compute a value or weight for a link according to a specific criterion. Most link scoring techniques obtain this value by considering the overlap or relationship between the neighborhood of the nodes at both ends of the link. On the other hand, link prediction computes a value, weight, or probability proportional to the likelihood of the existence of a certain link according to a given model of link formation.

The NOESIS framework provides a large collection of methods for link scoring and link prediction, from local methods, which only consider the direct neighborhood of nodes, to global methods, which consider the whole network topology. Some examples are shown in Figure 7. As the amount of information considered is increased, the computational and spatial complexity of the techniques also increases. The link scoring and prediction methods available in NOESIS are shown in Table 3.

Among local methods, the most basic technique is the common neighbors score [59], which is equal to the number of shared neighbors between a pair of nodes. Most techniques are variations of the common neighbors score. For example, the Adamic–Adar score [60] is the sum of one divided by the logarithm of the degree of each shared node. The resource-allocation index [61] follows the same expression but directly considers the degree instead of the logarithm of the degree. The adaptive degree penalization score [62] also follows the same approach but automatically determines an adequate degree penalization by considering properties of the network topology. Other local measures consider the number of shared neighbors but normalize their value according to certain criteria. For example, the Jaccard score [63] normalizes the number of shared neighbors by the total number of neighbors. The local Leicht–Holme–Newman score [64] normalizes the count of shared neighbors by the product of both neighborhoods' sizes. The Salton score [65] also normalizes similarly but this time using the square root of the product of both node degrees. The Sorensen score [66] considers twice the count of shared neighbors normalized by the sum of both neighbors' sizes. The hub promoted and hub depressed scores [67] normalize the count of shared neighbors by the minimum and the maximum of both node degrees, respectively. The preferential attachment score [68] only considers the product of both node degrees.

Global link scoring and prediction methods are more complex than local methods. For example, the Katz score [34] sums the influence of all possible paths between two nodes, incrementally penalizing paths by their length according to a given damping factor. The global Leicht–Holme–Newman score [64] is quite similar to the Katz score but resorts to the dominant eigenvalue to compute the final result.

Random walk techniques simulate a Markov chain of randomly selected nodes [69]. The idea is that starting from a seed node and randomly moving through links, we can obtain a probability vector where each element corresponds to the probability of reaching each node. The classical random walk iteratively multiplies the probability vector by the transition matrix, which is the row-normalized version of the adjacency matrix, until convergence. An interesting variant is the random walk with restart [70], which models the possibility of returning to the seed node with a given probability. Flow propagation is another variant of random walk [71], where the transition matrix is computed by performing both row and column normalization of the adjacency matrix.

Some spectral techniques are also available in NOESIS. Spectral techniques, as we mentioned when discussing community detection methods, are based on the Laplacian matrix. The pseudoinverse Laplacian score [72] is the inner product of the rows of the corresponding pair of nodes from the Laplacian matrix. Another spectral technique is the average commute time [72], which is defined as the average number of steps that a random walker starting from a particular node takes to reach another node for the first time and go back to the initial node. Despite that it models a random walk process, it is considered to be a spectral technique because it is usually computed in terms of the Laplacian matrix. Given the Laplacian matrix, it can be computed as the diagonal element of the starting node plus the diagonal element of the ending node, minus two times the element located in the row of the first node and the column of the second one.

The random forest kernel score [73] is a global technique based on the concept of spanning tree, i.e., a connected undirected subnetwork with no cycles that includes all the nodes and some or all the links of the network. The matrix-tree theorem states that the number of spanning trees in the network is equal to any cofactor, which is a determinant obtained by removing the row and column of the given node, of an entry of its Laplacian representation. As a result of this, the inverse of the sum of the identity matrix and the Laplacian matrix gives us a matrix that can be interpreted as a measure of accessibility between pairs of nodes.

Using network data mining algorithms in NOESIS is simple. In the following code snippet, we show how to generate a Barabási—Albert preferential attachment network with 100 nodes and 10 links per node and then compute the resource-allocation score for each pair of nodes using NOESIS:

```
Network network = new BarabasiAlbertNet-
work(100, 10);
LinkPredictionScore method = new Resource
AllocationScore(network);
Matrix result = method.call();
```

In Python, the previous example would be implemented as follows:

```
ns = Noesis()
network = ns.create_network_from_model
("Barabasi-Albert", 100, 10)
predictor = ns.create_link_predictor
("ResourceAlloca-tion")
result = predictor.compute(network)
ns.end()
```

5. Performance Comparison

NOESIS is designed to manage large complex networks comprising thousands or even millions of nodes. It provides

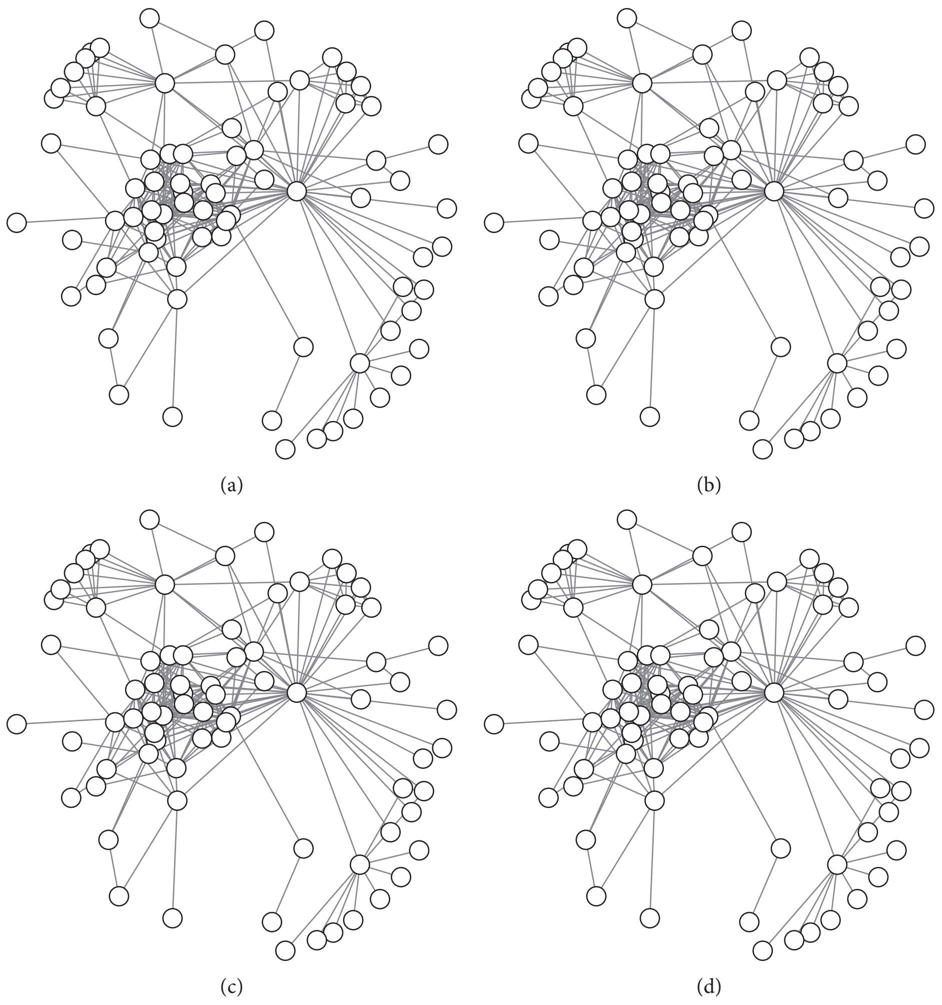


Figure 7: Different link scoring techniques applied to the Les Miserables coappearance network [58]: common neighbors (a); preferential attachment score (b); Sorensen score (c); Katz index (d). Link width in the figure is proportional to the link score.

Table 3: Computational time complexity and bibliographic references for the link scoring and prediction methods provided by NOESIS.

Type	Name	Complexity	Reference
Local	Common neighbors count	$O(vd^3)$	[59]
	Adamic–Adar score	$O(vd^3)$	[60]
	Resource-allocation index	$O(vd^3)$	[61]
	Adaptive degree penalization score	$O(vd^3)$	[62]
	Jaccard score	$O(vd^3)$	[63]
	Leicht–Holme–Newman score	$O(vd^3)$	[64]
	Salton score	$O(vd^3)$	[65]
	Sorensen score	$O(vd^3)$	[66]
	Hub promoted index	$O(vd^3)$	[67]
	Hub depressed index	$O(vd^3)$	[67]
	Preferential attachment score	$O(vd^2)$	[68]
Global	Katz index	$O(v^3)$	[34]
	Leicht–Holme–Newman score	$O(cv^2d)$	[64]
	Random walk	$O(cv^2d)$	[69]
	Random walk with restart	$O(cv^2d)$	[70]
	Flow propagation	$O(cv^2d)$	[71]
	Pseudoinverse Laplacian score	$O(v^3)$	[72]
	Average commute time score	$O(v^3)$	[72]
	Random forest kernel index	$O(v^3)$	[73]

In the time complexity analysis, v is the number of nodes in the network, d is the maximum node degree, and c refers to the number of iterations required by iterative global link prediction methods.

structures to efficiently represent and deal with relational data. While these features are often offered by other network analysis frameworks, they often lack the parallel computing features NOESIS includes.

In this section, we perform an empirical comparison of the performance of different popular network analysis frameworks for some common data analysis tasks. The igraph, SNAP, and NetworkX frameworks have been chosen for their comparison with NOESIS because they offer a programmatic API. Software developers can use them as libraries in their own software projects within the constraints imposed by their corresponding software licenses (virtually none in the BSD licenses of NetworkX, SNAP, and NOESIS). Closed-source tools, such as Pajek, offer no API, whereas other tools strongly focus on particular areas (e.g., Cytoscape on interactive network visualization). We restricted the comparison of NOESIS to those tools with the most similar use cases (i.e., igraph, SNAP, and NetworkX).

For NOESIS, different CPU scheduling techniques were tested: a work-stealing scheduler (WSS), a future scheduler (FS), and a conventional thread pool scheduler (TPS). Our experiments were performed in an Intel Core i7-3630QM processor (8 cores at 2.40 GHz) with 16 GB of RAM under Windows 10 (64 bits).

Three datasets were used in our experiments: a network extracted from Wikipedia with 27475 nodes and 85729 links (the Wikipedia dataset can be downloaded from http://spark-public.s3.amazonaws.com/sna/other/wikipedia.gml), a peer-to-peer Gnutella file-sharing network from August 2002 with 6301 nodes and 20777 edges [74], and a Facebook ego network with 4039 nodes and 88234 edges [75].

The experimentation carried out in this section consists of computing different computationally expensive network metrics that are commonly used in network mining: betweenness centrality (BC), link betweenness (LB), closeness (CN), and all shortest paths from every node using Dijkstra's algorithm (APSP). The APSP task could not be implemented using SNAP because of support limitations. We performed 5 runs for each measure and reported the average execution time required to complete the task for each software tool and network dataset.

The average execution time for each task/dataset/tool triplet is shown in Table 4. It should be noted that NOESIS is consistently faster that well-known existing frameworks. The superior performance of the NOESIS framework can be attributed to different factors. First of all, NOESIS is written in pure Java, which can be orders of magnitude faster than the Python implementation of igraph and NetworkX. The NOESIS Java code is highly optimized taking into account the peculiarities of the Java virtual machine (e.g., trying to minimize memory fragmentation and using a properly tuned cost model for different operations). As a consequence, the NOESIS Java implementation is surprisingly faster than the C++ implementation of SNAP, which should be expected to be more efficient since highly optimized C++ applications are usually more efficient than their Java counterparts. NOESIS parallel programming design patterns let algorithm designers easily exploit the parallelism in multicore processors and multiprocessor systems. The combination of these features allows NOESIS to complete tasks associated with computing important network-related structural properties faster than some popular software packages, even orders of magnitude faster.

Table 4: Performance of different network analysis tools in different tasks over several network datasets (time is shown in milliseconds).

Dataset	Framework	BC	LB	CN	APSP
Wikipedia	NOESIS-WSS	14485	32748	23581	54374
	NOESIS-FS	13837	33082	23657	57299
	NOESIS-TPS	**10854**	**23424**	**23233**	**48005**
	igraph	43328	110681	73181	198568
	SNAP	464055	579435	184254	—
	NetworkX	2246487	2812940	409698	1598431
P2P Gnutella	NOESIS-WSS	576	1025	879	1971
	NOESIS-FS	499	1097	868	1804
	NOESIS-TPS	**331**	**717**	**865**	**1288**
	igraph	1576	2875	4046	8947
	SNAP	18506	21854	8949	—
	NetworkX	67631	81315	16719	60983
Facebook	NOESIS-WSS	2267	6778	5087	1778
	NOESIS-FS	2500	7937	4977	2140
	NOESIS-TPS	**1841**	**5339**	**4874**	**1443**
	igraph	4677	26962	5302	15640
	SNAP	15206	17604	15295	—
	NetworkX	244842	291605	120558	481848

The abbreviations WSS, FS, TPS, BC, LB, CN, and APSP stand for the work-stealing scheduler, future scheduler, thread pool scheduler, betweenness centrality, link betweenness, closeness, and all-pairs shortest paths, respectively. Best times for each dataset are shown in bold.

In our parallelization experiments, we implemented different CPU scheduling techniques. NOESIS is faster than existing tools regardless of which scheduler is employed. The NOESIS conventional thread pool scheduler, which implements a common solution to the parallelization of software with the help of a single thread pool, consistently obtains the best results in the batch of experiments we have performed. An alternative implementation using Java features is somewhat slower. Similar results were obtained using a work-stealing CPU scheduler [76]. In a work-stealing scheduler, each processor in a computer system has a queue of work items (computational tasks (threads)) to perform. When a processor runs out of work, it looks at the queues of other processors and "steals" their work items. As a result, work stealing distributes the scheduling work over idle processors, and as long as all processors have work to do, no scheduling overhead occurs. It is employed in the scheduler for the Cilk programming language [77], the Java fork/join framework [78], and the .NET Task Parallel Library [79]. For the tests we performed, which involved a heavy work load for a single multicore processor, the simplest strategy seemed to work better, yet typical workloads might be different and alternative CPU schedulers might be preferable.

6. Conclusion

In this paper, we have presented the NOESIS framework for complex network data mining. NOESIS provides a large collection of data analysis techniques for networks, in a much more efficient way than competing alternatives, since NOESIS exploits the parallelism available in existing hardware using structured parallel programming design patterns [80]. As shown in our experimentation, NOESIS is significantly faster than other libraries, even orders of magnitude faster, which is extremely important when dealing with massive networks.

Currently, the NOESIS project has achieved a mature status with more than thirty-five thousand lines of Java code, hundreds of classes, and dozens of packages. In addition, the production code is covered by automated unit tests to ensure the correctness of the implemented algorithms. NOESIS includes a custom library of reusable components with more than forty thousand lines, providing different general functionalities, such as customizable collections, structured parallel programming building blocks, mathematical routines, and a model-driven application generator on top of which the NOESIS graphical user interface is built. Since Python has become a very popular programming language for scientific computing, a complete API binding for Python is provided so that NOESIS can be used from Python.

NOESIS is one of the most complete and efficient "out-of-the-box" frameworks for analyzing and mining relational data. Our framework can accelerate the development of software that needs to analyse networks by providing efficient and scalable techniques that cover different aspects of network data mining. Our framework is open source and freely available from its official website, http://noesis.ikor.org, under a permissive Berkeley Software Distribution license.

Data Availability

The network datasets used in our performance evaluation are from publicly available studies, which have been cited in our paper. Network datasets can be obtained from the Stanford Large Network Dataset Collection (http://snap.stanford.edu/data).

Disclosure

The NOESIS framework has served as the foundation for the development of the data mining techniques that comprise the Ph.D. thesis of Gómez [81].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The NOESIS project was partially supported by the Spanish Ministry of Economy and the European Regional Development Fund (FEDER), under grant TIN2012–36951, and the Spanish Ministry of Education under the program "Ayudas para contratos predoctorales para la formación de doctores 2013" (predoctoral grant BES–2013–064699). The authors are grateful to Aarón Rosas, Francisco–Javier Gijón, and Julio–Omar Palacio for their contributions to the implementation of community detection methods in NOESIS.

References

[1] P.-N. Tan, M. Steinbach, V. Kumar et al., *Introduction to Data Mining*, Pearson, Addison Wesley, Boston, MA, USA, 2006.

[2] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. Hwang, "Complex networks: structure and dynamics," *Physics Reports*, vol. 424, no. 4-5, pp. 175–308, 2006.

[3] R. Tamassia, *Handbook of Graph Drawing and Visualization*, CRC Press, Boca Raton, FL, USA, 2013.

[4] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparative analysis," *Physical Review E*, vol. 80, no. 5, 2009.

[5] L. Getoor and C. P. Diehl, "Link mining," *ACM SIGKDD Explorations Newsletter*, vol. 7, no. 2, pp. 3–12, 2005.

[6] L. Lü and T. Zhou, "Link prediction in complex networks: a survey," *Physica A: Statistical Mechanics and Its Applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[7] V. Martínez, C. Cano, and A. Blanco, "ProphNet: a generic prioritization method through propagation of information," *BMC Bioinformatics*, vol. 15, no. S1, 2014.

[8] M. Pavlov and R. Ichise, "Finding experts by link prediction in co-authorship networks," *Finding Experts on the Web with Semantics Workshop*, vol. 290, pp. 42–55, 2007.

[9] L. Page, S. Brin, R. Motwani, and W. Terry, "The PageRank citation ranking: bringing order to the web," Technical Report, Stanford InfoLab, Stanford, CA, USA, 1999.

[10] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and W. Gordon, "Graphviz–open source graph drawing tools," in *Graph Drawing*, pp. 483-484, Springer, Berlin, Gemany, 2002.

[11] P. Shannon, A. Markiel, O. Owen et al., "Cytoscape: a software environment for integrated models of biomolecular interaction networks," *Genome Research*, vol. 13, no. 11, pp. 2498–2504, 2003.

[12] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal, Complex Systems*, vol. 1695, no. 5, pp. 1–9, 2006.

[13] D. A. Schult and P. Swart, "Exploring network structure, dynamics, and function using networkX," in *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, vol. 2008, pp. 11–16, Pasadena, CA, USA, August 2008.

[14] J. Leskovec and R. Sosič, "SNAP," *ACM Transactions on Intelligent Systems and Technology*, vol. 8, no. 1, pp. 1–20, 2016.

[15] V. Batagelj and A. Mrvar, "Pajek-program for large network analysis," *Connections*, vol. 21, no. 2, pp. 47–57, 1998.

[16] M. A. Smith, S. Ben, N. Milic-Frayling et al., "Analyzing (social media) networks with NodeXL," in *Proceedings of the Fourth International Conference on Communities and Technologies*, pp. 255–264, ACM, University Park, PA, USA, June 2009.

[17] M. Bastian, S. Heymann, M. Jacomy et al., "Gephi: an open source software for exploring and manipulating networks," *International AAAI Conference on Weblogs and Social Media*, vol. 8, pp. 361-362, 2009.

[18] S. P. Borgatti, M. G. Everett, and L. C. Freeman, "UCINET for windows: software for social network analysis," Technical Report, Analytic Technologies, Lexington, KY, USA, 2002.

[19] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[20] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford Large Network Dataset Collection*, Stanford University, Stanford, CA, USA, 2014, http://snap.stanford.edu/data.

[21] M. E. J. Newman, *Networks: An Introduction*, Oxford University Press, Oxford, UK, 2010.

[22] M. O. Jackson, *Social and Economic Networks*, Princeton University Press, Princeton, NJ, USA, 2008.

[23] P. Erdős and A. Rényi, "On random graphs," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.

[24] E. N. Gilbert, "Random graphs," *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 1959.

[25] D. J. Watts and S. H. Strogatz, "Collective dynamics of "small-world" networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[26] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, no. 1, pp. 47–97, 2002.

[27] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003.

[28] M. Piraveenan, M. Prokopenko, and A. Y. Zomaya, "Local assortativeness in scale-free networks," *EPL (Europhysics Letters)*, vol. 84, no. 2, Article ID 28002, 2008.

[29] M. Piraveenan, M. Prokopenko, Y. Albert, and Zomaya, "Classifying complex networks using unbiased local assortativity," in *Proceedings of the Twelfth International Conference on the Synthesis and Simulation of Living Systems*, pp. 329–336, Odense, Denmark, August 2010.

[30] P. Hage and F. Harary, "Eccentricity and centrality in networks," *Social Networks*, vol. 17, no. 1, pp. 57–63, 1995.

[31] A. Bavelas, "Communication patterns in task-oriented groups," *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[32] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

[33] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, vol. 46, no. 5, pp. 604–632, 1999.

[34] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[35] C. Kang, C. Molinaro, S. Kraus, Y. Shavitt, and V. S. Subrahmanian, "Diffusion centrality in social networks," in *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, pp. 558–564, IEEE Computer Society, Istanbul Turkey, August 2012.

[36] M. Piraveenan, M. Prokopenko, and L. Hossain, "Percolation centrality: quantifying graph-theoretic impact of nodes during percolation in networks," *PLoS One*, vol. 8, no. 1, Article ID e53095, 2013.

[37] M. R. Faghani and U. T. Nguyen, "A study of XSS worm propagation and detection mechanisms in online social networks," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1815–1826, 2013.

[38] M. Piraveenan, G. Thedchanamoorthy, S. Uddin, and K. S. K. Chung, "Quantifying topological robustness of networks under sustained targeted attacks," *Social Network Analysis and Mining*, vol. 3, no. 4, pp. 939–952, 2013.

[39] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[40] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, no. 1, pp. 7–15, 1989.

[41] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, "The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations," *Behavioral Ecology and Sociobiology*, vol. 54, no. 4, pp. 396–405, 2003.

[42] G. J. Wills, "NicheWorks: interactive visualization of very large graphs," *Journal of Computational and Graphical Statistics*, vol. 8, no. 2, pp. 190–212, 1999.

[43] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[44] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, no. 4, pp. 452–473, 1977.

[45] R. Sibson, "Slink: an optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, 1973.

[46] D. Defays, "An efficient algorithm for a complete link method," *The Computer Journal*, vol. 20, no. 4, pp. 364–366, 1977.

[47] B. Liu, *Web Data Mining*, Springer, Berlin, Germany, 2 edition, 2011.

[48] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Physical Review E*, vol. 69, no. 6, 2004.

[49] P. Schuetz and A. Caflisch, "Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement," *Physical Review E*, vol. 77, no. 4, 2008.

[50] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[51] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, Oakland, CA, USA, January 1967.

[52] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 9, pp. 1074–1085, 1992.

[53] A. Y. Ng, M. I. Jordan, Y. Weiss et al., "On spectral clustering: analysis and an algorithm," *Advances in Neural Information Processing Systems*, vol. 2, pp. 849–856, 2002.

[54] J. Jianbo Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[55] J. Yang and J. Leskovec, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pp. 587–596, ACM, Rome, Italy, February 2013.

[56] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3–5, pp. 75–174, 2010.

[57] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, 2004.

[58] D. E. Knuth and S. GraphBase, *A Platform for Combinatorial Computing*, Addison-Wesley, Boston, MA, USA, 1st edition, 2009.

[59] M. E. J. Newman, "Clustering and preferential attachment in growing networks," *Physical Review E*, vol. 64, no. 2, 2001.

[60] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social Networks*, vol. 25, no. 3, pp. 211–230, 2003.

[61] T. Zhou, L. Lü, and Y.-C. Zhang, “Predicting missing links via local information,” *The European Physical Journal B*, vol. 71, no. 4, pp. 623–630, 2009.

[62] V. Martínez, F. Berzal, and J.-C. Cubero, “Adaptive degree penalization for link prediction,” *Journal of Computational Science*, vol. 13, no. 1–9, 2016.

[63] J. Paul, “Étude comparative de la distribution florale dans une portion des alpes et des jura,” *Bulletin de la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.

[64] E. A. Leicht, P. Holme, and M. E. J. Newman, “Vertex similarity in networks,” *Physical Review E*, vol. 73, no. 2, 2006.

[65] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, Inc., New York, NY, USA, 1986.

[66] T. Sørensen, “A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons,” *Biologiske Skrifter*, vol. 5, pp. 1–34, 1948.

[67] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási, “Hierarchical organization of modularity in metabolic networks,” *Science*, vol. 297, no. 5586, pp. 1551–1555, 2002.

[68] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[69] K. Pearson, “The problem of the random walk,” *Nature*, vol. 72, no. 1867, p. 342, 1905.

[70] H. Tong, C. Faloutsos, and J.-Y. Pan, “Fast random walk with restart and its applications,” in *Proceedings of the Sixth International Conference on Data Mining ICDM’06*, pp. 613–622, IEEE Computer Society, Hong Kong, China, December 2006.

[71] O. Vanunu and R. Sharan, “A propagation-based algorithm for inferring gene-disease assocations,” in *Proceedings of the German Conference on Bioinformatics*, pp. 54–52, Dresden, Germany, September 2008.

[72] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens, “Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 3, pp. 355–369, 2007.

[73] P. Chebotarev and E. Shamis, “Matrix-forest theorems,” 2006, https://arxiv.org/abs/math/0602575.

[74] M. Ripeanu, I. Foster, and A. Iamnitchi, “Mapping the gnutella network: properties of large-scale peer-to-peer systems and implications for system design,” 2002, https://arxiv.org/ftp/cs/papers/0209/0209028.pdf.

[75] J. Leskovec and J. J. Mcauley, “Learning to discover social circles in ego networks,” in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 539–547, Lake Tahoe, NV, USA, December 2012.

[76] R. D. Blumofe and C. E. Leiserson, “Scheduling multithreaded computations by work stealing,” *Journal of the ACM*, vol. 46, no. 5, pp. 720–748, 1999.

[77] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, “Cilk: an efficient multithreaded runtime system,” *Journal of Parallel and Distributed Computing*, vol. 37, no. 1, pp. 55–69, 1996.

[78] D. Lea, “A Java fork/join framework,” in *Proceedings of the ACM 2000 Conference on Java Grande*, pp. 36–43, ACM, San Francisco, CA, USA, June 2000.

[79] D. Leijen, W. Schulte, and S. Burckhardt, “The design of a task parallel library,” *ACM SIGPLAN Notices*, vol. 44, no. 10, pp. 227–242, 2009.

[80] M. D. McCool, A. D. Robison, and J. Reinders, *Structured Parallel Programming: Patterns for Efficient Computation*, Elsevier, Amsterdam, Netherlands, 2012.

[81] V. M. Gómez, *Supervised data mining in networks: link prediction and applications*, Ph.D. thesis, University of Granada, Granada, Spain, 2018.