RESEARCH ARTICLE OPEN ACCESS

Representing Non-Relational Databases with Darwinian Networks

Paulo Roberto Martins de Andrade, Alex Volnei Teixeira
University of Regina Department of Computer Science Regina, SK, S4S 0A2, Canada
Pontificia Universidade Catolica do Parana Curitiba, PR, 80215-901, Brazil

ABSTRACT
The Darwinian networks (DNs) are first introduced by Dr Butz [1] to simplify and clarify how to work with Bayesian networks (BNs). DNs can unify modeling and reasoning tasks into a single platform using the graphical manipulation of the probability tables that takes on a biological feel. From this view of the DNs, we propose a graphical library to represent and depict non-relational databases using DNs. Because of the growing of this kind of databases, we need even more tools to help in the management work, and the DNs can help with these tasks.
***Index Terms*:** No SQL, Darwinian Networks, Non-Relational Database, Java-script, Software engineering

I. INTRODUCTION

The manner in which we communicate, exchange information and create content changed much over the years. We live in an era where web applications have revolutionized the world in many ways, and the trend is that this growth enables the creation of many new applications. The large volume of data generated by these web applications, together with the new form of user interaction (dynamic, efficient and intuitive), the scalability on demand and the need for a high degree of availability, has fostered the emergence of new paradigms and technologies. We can cite the Non-Relational databases (NoSQL) and the Darwinian Networks (DNs) as a result of this demand. Non-Relational database [2, 3], also known as NoSQL or Not Only SQL database, is a technology designed to support cloud applications requirements and to overcome the scale, performance, data model and the limitations of relational databases. Relational databases [4] have restrictions in scalability, requiring vertical distribution of servers. A possible downside of this issue is that as more data is acquired, the servers require more working memory and hard disk space. NoSQL has a large horizontal distribution facility. In practice, more data does not necessarily imply higher performance to be taken by the server. Large companies use NoSQL in information technology, including Google, which uses small and medium-sized computers for data distribution. Furthermore, NoSQL databases are error tolerant. Darwinian networks (DNs) [5, 6] are a probabilistic framework that can be used to simplify working with Bayesian networks (BNs) [7]. In DNs, a conditional probability table (CPT) is represented as a population. Graphically, a solid circle containing smaller circles filled or not represents a population is. The inner circles are called traits. In DNs, traits represent variables from the problem domain being modeled. Traits can be combative or docile, depending on where the variable is in a CPT, LHS or RHS, respectively. We can define a DN as a multiset of populations. We depict a DN as a dashed circle around its populations. We can manipulate populations in a DN using operations, including merge, replication, and natural selection. With these operations, DNs can represent inference in BNs. In this paper, we propose a graphical library to represent and depict non-relational databases using DNs. Our idea is to create a useful tool to help in the management of this kind of data and its representation. The library will be an interactive space with a non-relational database available through a graphical interface. Users will be able to create and edit DNs by using the interface. Manipulations on DNs are then translated by the library to the correspondent non-relational database operation. By using the DN operations available, users will be able to perform inference graphically. We organized the rest of the paper as follows. Section II presents the Non-Relational database and its main characteristics. Section III shows details about the Darwinian Networks. In section IV we present the created library and its use. Finally, Section V offers some concluding remarks and final thoughts.

II. RELATIONAL AND NON-RELATIONAL DATABASES

The relational database has emerged as a successor to the hierarchical network models. The relational model became the pattern for most Database Management Systems (DBMS). This model brought up the Normalization process which its goal is to apply a series of steps with certain rules on the database table to ensure proper design of

these tables [8]. Furthermore, this model adopted a language for the manipulation and query of these data, the SQL (Structured Query Language) which was created by IBM and inspired by the relational algebra. In 1982, the American National Standard Institute (ANSI) defined the SQL as the official standard language for the relational environment [9]. For a long time, the relational database has been the most widely used database type in companies that have a high volume of data to be stored [2, 3]. Thinking that this quantity of data tends to grow every time, you begin to see those relational databases have certain limiting factors, especially when we refer to a scale system [10]. We can mention the case of Facebook, which reached the level of petabytes (in 2011 this data volume exceeded 30 petabytes, and less than a year before the volume was 20 petabytes). This case is a real example of how this data growth has expanded rapidly [11]. For these types of organizations, the use of relational DBMSs has been very problematic and not as efficient. Because of this limitations, we have the emergence of new alternative models for the database environment that can fill this gap. One example of these models that has gained enough strength and space is the NoSQL("Not Only SQL") [12]. the NoSQL is a generic term that defines a non-relational database. This model came with the proposal to meet, to organize, and to manage large volumes of data, looking for high performance and availability [13].

A. A new paradigm: the NoSQL

Database Thinking in solving various problems of relational models, NoSQL designers promoted an alternative with high storage, speed, and great availability. They were seeking to get rid of certain rules and structures that guide the Relational Model. With this breach in the relational model, we won performance and more flexible database systems for the various features that are unique to each company. This flexibility has become critical to meet the high scalability requirements needed to manage large amounts of data, as well as to ensure high availability, a key feature for Web 2.0 applications. The proposal of NoSQL database is not extinguishing the Relational Model, but use it in cases where it is needed more flexibility in structuring database [13]. This movement is very rooted in the open source field and although there are many databases in this category. The movement started to gain more strength when some companies, considered as technology giants, began to use their proprietary implementations. We can mention Google, which since 2004 invests in BigTable. We also have the Cassandra, developed by Facebook to handle the massive flow of information. In 2010, Cassandra proved to be a consolidated database and is now used by Twitter, which used MySQL before [14]. NoSQL databases have some important characteristics that make them so different from relational databases like Horizontal scalability; Schema-free or flexible schema; Native support for replication; API simple to access the database.

B. Main differences between DBMSs relational and NoSQL

When we think about the possibility of using a NoSQL database instead of a relational model, we need to take a few questions into consideration, such as escalation, consistency, and availability of data. To better understand all those differences, Table I exemplifies concisely these questions. We also can summarize the both kinds of databases (Relational Model and NoSQL) and their main characteristics as follows.

Table I: Comparative analysis of relational database and NoSQL

	Relational Database	NoSQL Database
Escalation	It is important to remember that it is possible to do the scaling in the relational model. However, it is very complex due its structured nature (predefined schema).	It does not have a predefined schema. It makes this model flexible, which facilitates the transparent inclusion of other elements.
Consistency	In this regard, the Relational Model shown strong. Their consistency rules are very strict in regard to consistency.	It may be performed in the model: only have the guarantee that if there is no update in the data, all access to items return the last value.
Availability	For failing to work efficiently with the distribution of data, Relational Model ends up not supporting a very large demand for information.	It has a high level of data distribution, allowing it to make a huge flow of requests for data, with the advantage of the system becomes unavailable for the shortest possible time.

- **Relational Databases**

– What is: based on that all data are stored in tables, by the concept of an entity and relationship. The data are separated in a unique form, trying to reduce the maximum redundancy, because the information is created by all the data, which are the relationships between the tables that make this service.
– Features: tables, defined schema, hierarchy, minimal redundancy, and entity relationship, normal forms, ACID transactions (atomicity, consistency, isolation, durability). – When to use: local, financial, corporate; information security; data consistency.

- **NoSQL Databases**

– What it is: an alternative solution to relational databases, have a high scalability and performance.
– Features: records, schema-free, fault tolerance, scalability, clustering, mapreduce, sharding.
– When to use: cloud systems, social analysis, high scalability, performance in the query / write replication.

III. DARWINIAN NETWORKS

Darwinian networks (DNs) [5, 6] were proposed to simplify working with Bayesian networks (BNs) [7]. Rather than modeling the variables in a problem domain, DNs represent the probability tables in the model. The graphical manipulation of the tables then takes on a biological feel, where a CPT P(X/Y) is viewed as the novel representation of a population p(C, D) using both combative traits C (coloured clear) and docile traits D (coloured dark).

A. Definitions

A *trait*, denoted t, is a named element. A trait is depicte by a circle along with its name t. A named circle can b either black or white. A white circle depicts a *combative* trai denoted t_c. A black circle depicts a *docile* trait, denoted t_d.

A *population*, denoted $p(C, D)$, contains a non-empty s CD of traits, where C and D are disjoint, C is exclusive combative, and D is exclusively docile.

A population is depicted by a closed curve around its trait For example, Figure 1 (i) shows two populations, includir $p(b, a)$, short for $p(\{b\}, \{a\})$, illustrated with a closed curv around the (white) combative trait b and the (black) docile tra a.

A *Darwinian network* (DN), denoted $\mathcal{D}$, is a finite, multis of populations. A DN $\mathcal{D}$ is depicted by a dashed close curve around its populations. For example, Figure 1 (i)-(i respectively depicts four DNs $\mathcal{D}_1$ = { p(a),p(b,a)}, $\mathcal{D}_2$ {p(b,a)}, $\mathcal{D}_3$ = { p(b),p(a,b)}, and $\mathcal{D}_4$ = { p(a,b)}, whe $p(C, \emptyset)$ is succinctly written $p(C)$.

All combative traits in a given DN $\mathcal{D}$ are defined as $T_c(\mathcal{D})$ $= \{t_c \mid t_c \in C$, for at least one $p(C, D) \in \mathcal{D}\}$. All docile traits in $\mathcal{D}$, denoted $T_d(\mathcal{D})$, are defined similarly. For exampl considering DN $\mathcal{D}$ in Figure 1 (i), then $T_c(\mathcal{D}_1) = \{a_c, b_c\}$. addition, $T_d(\mathcal{D}_2) = \{a_d\}$.

Figure 1: Unique DN representations $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$, respectively.

B. Operations

Docilization of a DN $\mathcal{D}$ adds $p(\emptyset, D)$ to $\mathcal{D}$, for every population $p(C, D)$ in $\mathcal{D}$ with $|D| > 1$.

To *delete* a population $p(C, D)$ from a DN $\mathcal{D}$ is to remove all occurrences of it from $\mathcal{D}$.

Two populations *merge* together as follows: for each trait t appearing in either population, if t is combative in exactly one of the two populations, then t is combative in the merged population; otherwise, t is docile. The merge of two populations $p(a)$ and $p(b, a)$ in Figure 2 (i) can be readily modeled based upon the following colour combinations:

Note that Figure 2 (ii) shows the merge of two populations p(ab) and p(b), resulting on the population p(a; b). Natural selection removes recursively all barren populations from a DN D with respect to another DN D .

Replication of a population $p(C, D)$ gives $p(C, D)$, as well as any set of populations $p(C', D)$, where $C' \subset C$. The replication of population $p(ab, c)$ can be represented in Figure 3. Here, two steps are used. First, population $p(ab, c)$ in Figure 3 (i) makes an imperfect copy of itself, as depicted in Figure 3 (ii). Second, the original population $p(ab, c)$ is removed, as illustrated in Figure 3 (iii).

Figure 2: (i) Representing the merge of populations $p(a)$ and $p(b, a)$. (ii) Representing the merge of populations $p(ab)$ and $p(b)$.

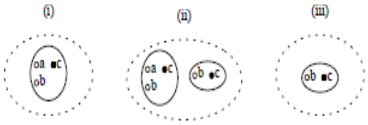

Figure 3: Representing the removing of trait a in population $p(ab, c)$.

C. Modeling

In DNs, how populations 'adapt' to the deletion of other populations corresponds precisely with testing independencies in BNs.

Let $\mathcal{P}_X$, $\mathcal{P}_Y$, and $\mathcal{P}_Z$ be pairwise disjoint subsets of populations in a DN $\mathcal{D}$ and let DN $\mathcal{D}' = p(C)$, where $C = T_c(\mathcal{P}_X\mathcal{P}_Y\mathcal{P}_Z)$. The test *adaptation* of $\mathcal{P}_X$ and $\mathcal{P}_Z$ given $\mathcal{P}_Y$, denoted $A(\mathcal{P}_X, \mathcal{P}_Y, \mathcal{P}_Z)$, in $\mathcal{D}$ with four simple steps: (i) let natural selection act on $\mathcal{D}$ with respect to $\mathcal{D}'$, giving $\mathcal{D}^s$; (ii) construct the docilization of $\mathcal{D}^s$, giving $\mathcal{D}^s_m$; (iii) delete $p(C, D)$ from $\mathcal{D}^s_m$, for each $p(C, D)$ in $\mathcal{P}_Y$; and, (iv) after recursively merging populations sharing a common trait, if there exists a population containing both a combative trait in $T_c(\mathcal{P}_X)$ and a combative trait in $T_c(\mathcal{P}_Z)$, then $A(\mathcal{P}_X, \mathcal{P}_Y, \mathcal{P}_Z)$ fails; otherwise, $A(\mathcal{P}_X, \mathcal{P}_Y, \mathcal{P}_Z)$ succeeds.

D. Inference

The *evolution* of a DN $\mathcal{D}$ into a DN $\mathcal{D}'$ occurs by natural selection removing recursively all barren, independent, and spent populations, merging existing populations, and replicating to form new populations. For example, in Figure 4, consider one explanation of the evolution of $\mathcal{D}$ in (i) into $\mathcal{D}' = p(e, b)$ in (xiii). Natural selection removes barren populations $p(g, ef)$ and $p(f, a)$, yielding (ii). Next, natural selection removes independent population $p(a)$, giving (iii), and evident population $p(b, a)$, yielding (iv). Then, $p(c, h)$ and $p(e, cd)$ merge to form $p(ce, dh)$ in (v). Replication gives (vi). The rest of the example involves natural selection (vii), merge (viii), replication (ix), natural selection (x), merge (xi), replication (xii), and natural selection (xiii), leaving $\mathcal{D}'$ with population $p(e, b)$.

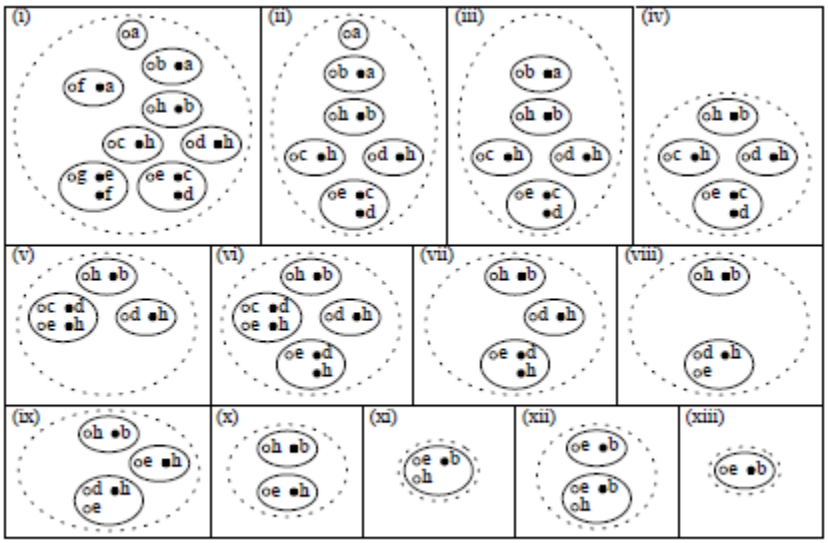

Figure 4: Graphically representing a DN evolution.

IV. THE GRAPHICAL LIBRARY

In this section, we will describe how a Darwinian network (DN) can be modeled in a non-relational database. Furthermore, we also discuss how evolution can be saved as collections in the same non-relational database. By explaining that, we are formalizing the product of this paper: a simple javascript library for representing DNs in a non-relational database with an intuitive and graphical interface for the user. To ensure that the product was well-developed and controlled during the entire process, we used the PM5 approach to manage this project [15].

A. Database Description

The system is formed by a non-relational database server and a web interface to manipulate the database. The server runs a Node.js javascript platform, forming the backend part of the library, which also includes handling client's requests and managing the database. The web interface is also written in javascript and communicate with the browser by offering up-to-date visualization of the database. For drawing DNs in the browser, the web interface uses Scalable Vector Graphics (SVG), which is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. The first important concept in the library is the representation of a trait. By convention, a trait will be represented as one character or a single string. For example, a trait a is represented by character 'a'. The second concept that we used in this project was the convention of unique identification codes for each collection, called an id. The id is a randomly generated 32 characters long code. Moreover, the id is unique for each population, rule enforced by the non relational database server. A population is represented by a collection in the database. The first field of the

collection is the population id. The second field is dnId, that is the id for the DN from where this population is contained. Next, the population has two array fields: combative and docile. They both hold a sequence of traits (characters). For the visualization of this population, we also needed to save the population's position on the screen.

We can achieve that by adding two more fields: posX and posY, saving the population's horizontal and vertical positions, respectively. Lastly, we keep a field createdAt saving when the population was created. Follows one example of population p(a; b) as held in the database:

```
showspaces
{
    "_id" : "axhAbdRWsimsLZL2C",
    "dnId" : "XLYHsqTjjpBJwHKHb",
    "combative" : [
        "a"
    ],
    "docile" : [
        "b"
    ],
    "posX" : 77.5,
    "posY" : 116.28125,
    "createdAt" : ISODate("2016-08-03T23:23:26.418+0000")
}
```

A DN is a collection created in order to be referenced by populations. Its first field is the unique id. Follows a field called name, which is a user's input to identify a DN in specific. DNs are used for modeling a problem domain and they are also used for inference. Thus, we needed a field to distinguish between these two situations. That one is a Boolean field isEvlotion which holds a false if the DN is used in modeling and true if it is used in inference. A DN collection also has a field createdAt to save user's creation date. Bellow is a document from a DN collection:

```
showspaces
{
    "_id" : "XLYHsqTjjpBJwHKHb",
    "name" : "ESBN",
    "isEvolution" : false,
    "createdAt" : ISODate("2016-08-03T23:23:16.562+0000")
}
```

The last thing the library needs to represent in the database is evolution. The first field is its unique id. Next, we have a field name, used to save a familiar identification for the user. The field most important in this collection is the dns one. Here, we have an array type of field with an ordered sequence of DN's ids. These ids correspond to DNs with the field isEvolution being true. Thus, the evolution collection can save a sequence of DNs, which is exactly the meaning of the inference process in DNs. Lastly, createdAt has similar functionality here. This is one possible evolution document from the evolution collection:

```
showspaces
{
    "_id" : "aWnyikzrzHSEbbBdN",
    "name" : "Applying VE on ESBN",
    "dns" : [
        "C3JyTwKXKuybXGjA4",
        "cRoEyNuhpznWMAFbq",
        "CxJWuiSeCTbvoodTX",
        "D3bwy6zi67QPDK2MF",
        "KKsL7hKEAyMCy9GBL"
    ],
    "createdAt" : ISODate("2016-08-03T23:27:58.663+0000")
}
```

B. Web Interface

The web interface is composed of screens where the user can create, edit, and remove DNs and evolutions. DNs are created independently from evolutions. That is, the user can define a DN and save it on a screen. Later, if the user wants to perform evolution with a certain DN, the user can select it from the DNs list and start the population manipulations. Notice that whenever the user starts evolution a copy of the original DN is made, meaning that all population manipulations are not applied to the defined DN, but to the copy. Users can create DN, edit, and remove DNs at the DNs list screen, as shown in Figures 5 and 6.

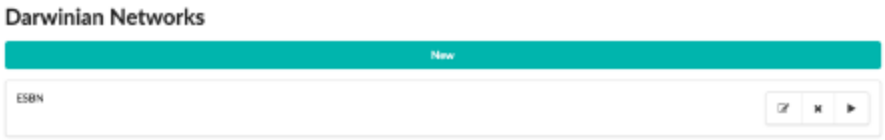

Figure 5: DN list screen: users can create, edit, and remove DNs.

In order to add and remove populations, the edit screen, depicted in Figure 7 offers two simple tools: a form for adding populations at the top and the option of double clicking a population for removing it. The user can list combative and docile trait by separating them with a comma. The interface parses the input for convenience. At the evolution list, users edit and resume an evolution but can not create one. Evolutions can only be created from the DN list screen. In Figure 8, the evolution list screen is shown. Whenever an evolution is removed, all corresponding DNs are also removed. But notice that the original DN is never modified, only the copy. Next, users can perform evolution by manipulating populations in the evolution screen, as illustrated in Figure 9. Populations can replicate and merge. To replicate, it is required a double click in a population, which triggers a pop up window asking which trait is going to be removed in the replica. In order to merge population, users can drag one above the other.

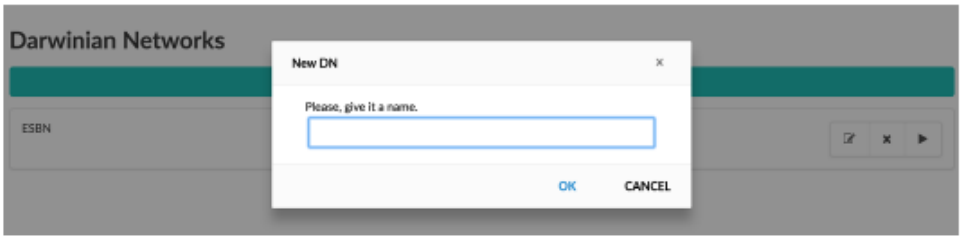

Figure 6: Creating a DN: users provide a name for the DN

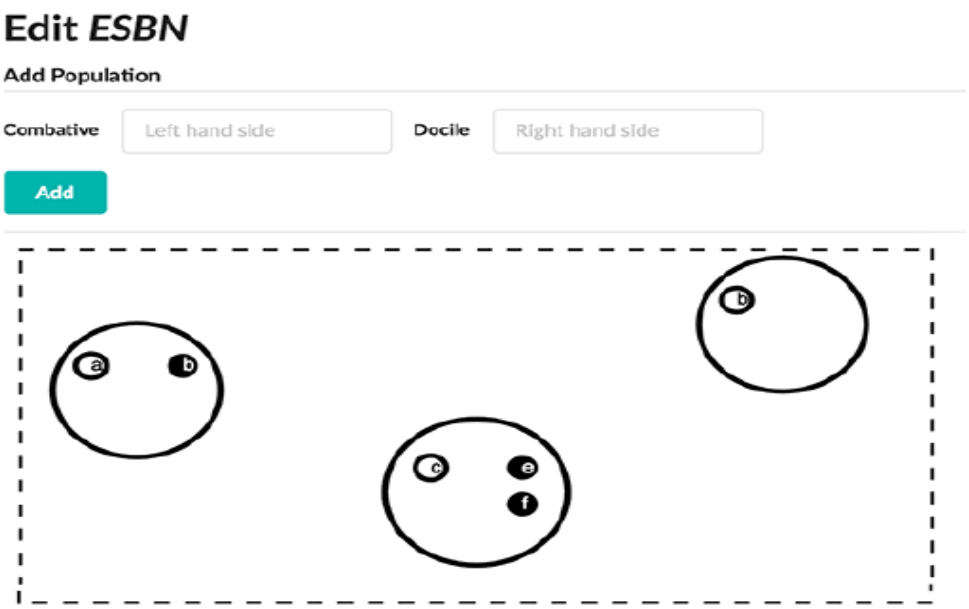

Figure 7: Editing a DN by adding and removing populations.

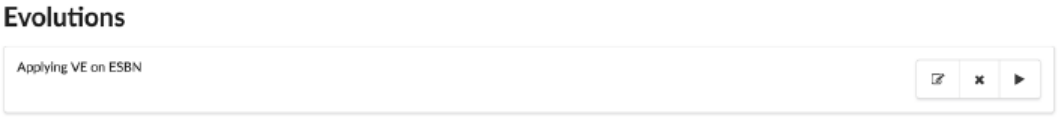

Figure 8: List of evolutions: users can edit names and resume evolutions.

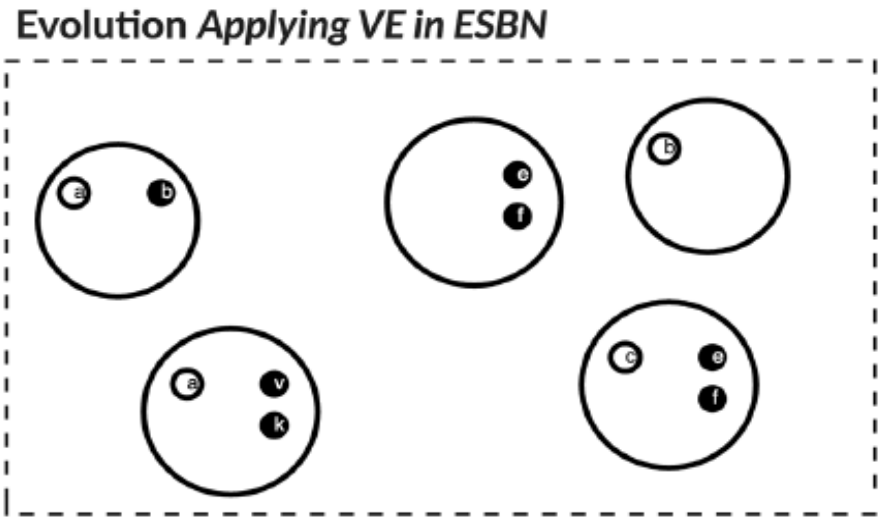

Figure 9: Users manipulate population in order to replicate and merge them when doing evolution.

The web interface currently works in open session, that is all users have access to the same database. Future works will create individual sections, restricting users to their own databases. Future works also can also implement representing multiple DNs. Here, the idea would be to create a new collection for being an entity to where all DNs are contained. This approach is similar to how populations from a specific DN refer to the id of that DN.

V. CONCLUSION

We proposed a representation framework for Darwinian Networks with Non-Relational Databases, available at https: //dns-lab.herokuapp.com/. It is a javascript implementation of the basic operations for adaptation and evolution in DNs. The main advantage of the DN graphical library is to visualize the novel ideas and techniques of DNs. Thus, it is a great tool for teaching, quick prototyping with DNs and testing. Non-relational databases (NoSQL) are an alternative solution to relational databases, have a high scalability and performance. We have shown a comprehensive comparison between relational and non-relational databases. The motivations and basic concepts on the non-relational database were stated as well as examples to illustrate its main characteristics. Next, we studied how this database paradigm compares with relational database. We also provided examples to elucidate the advantages and disadvantages. By utilizing the unique features of NoSQL we developed an intuitive framework for visualizing working with DNs. For instance, users can create DN, edit, and remove DNs at the DNs list screen. Another salient feature is the DN manipulation. By the list of evolutions, users can edit names and resume evolutions. Users can manipulate population in order to replicate and merge them when doing evolution. This paper proposed a graphical library to represent and depict non-relational databases using DNs. In summary, we have established four main advantages of using DN graphical library:

- The library is an interactive space with a non-relational database available through a graphical interface. Users can create and edit DNs by using the web interface.
- Manipulations on DNs are then translated by the library to the correspondent non-relational database operation.
- By using the DN operations available, users can perform inference graphically.
- The framework is a great tool for learning DNs.
- and all source code are available free online on GitHub through the web page:

https://gitlab.com/jhonatanoliveira/dn-lab

With the framework, DNs can be applied as the simple and yet remarkably robust tool they are, allowing users to depict reasoning with DNs.

ACKNOWLEDGEMENTS

The CNPq (Conselho Nacional de Desenvolvimento Cient´ıfico e Tecnol´ogico - ”National Counsel of Technological and Scientific Development”) supports this work.

REFERENCES

[1]. C. J. Butz, “Introducing darwinian networks,” in Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, 2015, pp. 604–610.

[2]. C. Mohan, “History repeats itself: Sensible and nonsensql aspects of the nosql hoopla,” in Proceedings of the 16th International Conference on Extending Database Technology, 2013, pp. 11–16.

[3]. A. Lith and J. Mattsson, “Investigating storage solutions for large data-a comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data,” 2010.

[4]. D. Maier, Theory of Relational Databases. Computer Science Pr, 1983.

[5]. C. J. Butz, J. S. Oliveira, and A. E. dos Santos, “Darwinian networks,” in Proceedings of the Twenty-Eighth Canadian Artificial Intelligence Conference, 2015, pp. 16–29.

[6]. ——, “On Darwinian networks,” Computational Intelligence, 2015.

[7]. J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, 1988.

[8]. E. F. Cood, The Relational Model for Database Managment. Addison-Wesley Publishing Company, 1990.

[9]. IBM, “Relational database,” 2012, online; Available at http://www-03.ibm.com/ibm/history/ibm100/ us/en/icons/reldb/, Accessed 24-July-2016. [Online]. Available: http://www-03.ibm.com/ibm/history/ibm100/ us/en/icons/reldb/

[10]. Uniredes, “List of the most used databases in the would,” 2013, online; Available at http://uniredes.org/kb/?View= entry&EntryID=224, Accessed 24-July-2016. [Online]. Available: http://uniredes.org/kb/?View=entry&EntryID= 224

[11]. InfoQ, “Facebook - the biggest migration,” 2011, online; Available at https://www.infoq.com/br/news/ 2011/08/facebook-maior-migracao, Accessed 24-July- 2016. [Online]. Available: https://www.infoq.com/br/ news/2011/08/facebook-maior-migracao

[12]. N. Database, “List of nosql databases,” 2015, online; Available at http://nosql-database.org, Accessed 16-July- 2016. [Online]. Available: http://nosql-database.org/

[13]. MongoDB, “Mongodb for giant ideas,” 2016, online; Available at https://www.mongodb.com/, Accessed 25-July-2016. [Online]. Available: https://www.mongodb. com/

[14]. NoSQLDatabases, “How to create a consistent hasher using php,” 2010, online; Available at http://www. nosqldatabases.com/main/tag/consistent-hashing, Accessed 21-July-2016. [Online]. Available: http: //www.nosqldatabases.com/main/tag/consistent-hashing

[15]. P. R. M. Andrade, A. B. Albuquerque, O. F. Frota, and J. F. S. Filho, “Pm5: One approach to the management of it projects applied in the brazilian public sector,” in Proceedings of 13th International Conference on Software Engineering Research and Practice - SERP. WorldComp, 2015.