Industry and Applications Paper

Weaving Enterprise Knowledge Graphs: The Case of Company Ownership Graphs*

Paolo Atzeni
Università Roma Tre

Luigi Bellomarini
Banca d'Italia

Michela Iezzi
Banca d'Italia

Emanuel Sallinger
TU Wien and
University of Oxford

Adriano Vlad
Università Roma Tre and
University of Oxford

ABSTRACT

Motivated by our experience in building the Enterprise Knowledge Graph of Italian companies for the Central Bank of Italy, in this paper we present an in-depth case analysis of company ownership graphs, graphs having company ownership as a central concept. In particular, we study and introduce three industrially relevant problems related to such graphs: company control, asset eligibility and detection of personal links. We formally characterize the problems and present Vada-Link, a framework based on state-of-the-art approaches for knowledge representation and reasoning. With our methodology and system, we solve the problems at hand in a scalable, model-independent and generalizable way. We illustrate the favourable architectural properties of Vada-Link and give experimental evaluation of the approach.

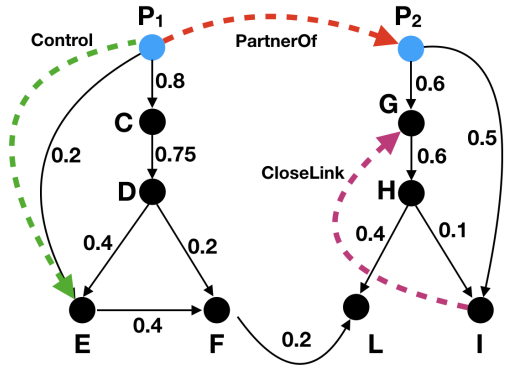

Figure 1: Sample excerpt of a company ownership KG.

1 INTRODUCTION

This paper is motivated by our experience in building the *Enterprise Knowledge Graph* of Italian companies for Banca d'Italia, the central bank of Italy.

Company ownership graphs are central objects in corporate economics [9, 19, 23, 36] and are of high importance for central banks, financial authorities and national statistical offices, to solve relevant problems in different areas: *banking supervision*, *creditworthiness evaluation*, *anti-money laundering*, *insurance fraud detection*, *economic and statistical research* and many more. As shown in Figure 1, in such graphs, *ownership* is the core concept: nodes are companies and persons (black resp. blue nodes), and ownership edges (black solid links) are labelled with the fraction of shares that a company or person x owns of a company y. Company graphs are helpful in many situations.

One first important problem that can be solved with such graphs is *company control* (also effectively formalized in the context of logic programming [18]), which amounts to deciding whether a company x controls a company y, that is, x can push decisions through in y having the vote majority. Consider the graph in Figure 1: P_1 controls C, D (via C), E (since it controls D, which owns 40% of E and P_1 directly owns 20% of it), and F (via E and D). Similarly, P_2 controls all its descendants except for L. Apparently, P_1 exerts no control on L either.

A second particularly representative application of company graphs is in the context of *collateral eligibility* (also known as *asset eligibility* or *close link*) problem, which consists of estimating the risk to grant a specific loan to a company x that is backed by collateral issued by another company y. According to European Central Bank regulations [1], company y cannot act as a guarantor for x if it is too "close" to it in terms of ownership; the regulation gives a detailed definition for this concept of *closely-linked entity*, which includes: "the two companies must not be owned by a common third party entity, which owns more than 20% of both". With respect to Figure 1, we see that, for example, G and I are closely linked since P_2 owns more than 20% of both.[1]

Besides financial relationships, *personal or family connections* enable much broader use of such company graphs: detecting *family businesses* or studying the real dispersion of control [21] are just two such applications. In our example in Figure 1, knowing that P_1 and P_2 have personal connections –e.g., are married– allows to deduce that, in fact, P_1 and P_2 together control L. Likely, they act as a single center of interest: L is in fact a family business, with control in the hands of a single family, with P_1 and P_2 together controlling 60% of it. Similarly, although D and G do not strictly fulfil the definition of close link, as P_1 and P_2 have a personal connection, there is very low risk differentiation between them and so it is reasonable to prevent G from acting as a guarantor for D or vice versa.

In all the above settings, and indeed in many more, it is our experience that the links representing relevant relationships in the financial realm are not immediately available in data stores. For instance, there are no enterprise graphs readily providing company control relationships, close links or family connections. Reasons fall mostly into four categories: (i) such links represent non-trivial relationships, whose calculation is complex and is not

*The views and opinions expressed in this paper are those of the authors and do not necessarily reflect the official policy or position of Banca d'Italia. This work is supported by the EPSRC grant EP/M025268/1, the Vienna Science and Technology Fund (WWTF) grant VRG18-013, and the EC grant 809965.

[1] Actually, here we are forcing the concept of close links to include individuals.

typically done while building the enterprise data stores, for example in the ETL (Extract, Transform, Load) jobs; (ii) the enterprise data stores are mainly relational and neglect specific connections because they are not easily navigable without a graph-based technology; (iii) edges between entities in different data stores are ignored because enterprise databases follow a siloed approach; (iv) edges are considered less trustworthy (lower data quality) than the entities they connect, and so are skipped.

Contribution. In this paper, we focus on the industrial context of the ownership graph of Banca d'Italia and provide the following contributions:

- We provide a compact but formal characterization of the problems of company control, asset eligibility and detection of personal connections over ownership graphs. We argue that the three problems belong to a much broader class, which we name *knowledge graph augmentation*, (KG augmentation). By KG augmentation, we wish to characterize a special case of *link prediction* [29], the key problem of predicting hidden links in network structures, where the emphasis is on the need for a careful combination of the extensional data (existing nodes and edges) and the available domain knowledge.
- We present Vada-Link, a framework for the solution of KG augmentation problems, leveraging state-of-the-art methodologies for *logic-based reasoning* [14, 24], which provide a very good balance between computational complexity of the reasoning task, being PTIME in data complexity, and expressive power, thanks to logic-based *Knowledge Representation and Reasoning* (KRR) languages such as Vadalog [14]. In fact, Vadalog captures full *Datalog* [18, 27] and is able to express SPARQL queries under set semantics and the entailment regime for OWL 2 QL [24].
- From the structured enterprise data stores, we build a *Property Graph* [3] (PG) –a graph with labelled nodes and edges– and model KG augmentation problems as *reasoning tasks* on a *Knowledge Graphs* (KG), based on such PG. In particular, here a KG is a data model combining an *extensional component*, that is, the data from the enterprise data stores, with a formal representation of the domain of interest, the *intensional component*, expressed in some KRR language. KGs typically represent domains that are well suited for being modeled as *property graphs* complemented with rules that express domain knowledge.
- We argue that our approach is general and can be used to augment KGs with new links that can be deterministically or heuristically deduced from a combination of extensional data and domain knowledge. In particular, we motivate that our approach is *schema independent*, as it is not coupled to a specific graph structure (i.e., specific types for nodes, edges and their properties), but relies on meta-level concepts describing graph constructs. At the same time, our solution is *problem aware*, as specific reasoning steps adopt *polymorphic behaviour*, depending on the specific problem. Moreover, our approach is *data model independent*, in the sense that the extensional component can be based on diverse data sources.
- With our KG-based approach, we build a *company knowledge graph*, where the PG of the ownership graph represents the extensional component of the KG and the hidden links, object of the above problems, are obtained by combining a logic-based intensional definition of their specification, i.e., the intensional component of the KG, with the extensional component. Our solution provides an approach to KG augmentation that shows at the same time high accuracy and efficiency, achieved by *multi-level clustering* of the search space, with techniques inspired by record linkage experience [20]. In particular, in the KG intensional component, we combine highly selective feature-based clustering with *neighborhood-preserving embeddings*: the former technique avoids quadratic blow-up of the search space, while the latter enhances accuracy by recognizing graph-based similarities of the entities.
- We exploit the framework to build our company knowledge graph. We motivate that our solution has very favourable properties: it is scalable, since it uses a tractable logic fragment, Vadalog [14] and reduces the search space via multi-level clustering; it is understandable, IT-independent and modifiable thanks to the adoption of a fully declarative approach; Vada-Link decisions are explainable and unambiguous, as the semantics of Vadalog is based on that of Datalog, well known in the database community [2].
- We provide extensive experimental evaluation of Vada-Link.

KG augmentation. In Vada-Link, we structure our link prediction tasks into two of subgoals. First, a *clustering task*, where we aim at grouping the nodes into candidate clusters in order to limit the search space of connected nodes. This allows to overcome the need to compare a quadratic number of nodes, which would make the approach infeasible. Clustering is performed with a combination deterministic rule-based choices and embedding techniques, e.g., *node2vec* [26], which allows to exploit not only the textual or numeric features of edges, but also their structural properties, such as: specific shared neighbours (along with their properties, etc.), topological role nodes and so on.

Clearly, clustering requires a preliminary *feature engineering phase*, needed to individuate the most informative features, which is however out of the scope of this paper. The second subgoal is a *multi-class classification problem*, where we aim at assigning each pair of nodes a link class, if any.

Related work. Different research areas and data management problems can be seen as related to Vada-Link. Link prediction is the problem of discovering links between nodes in network structures: it is relevant in social network analysis [29] and a variety of approaches have been recently proposed, e.g. [39]. In this work, we refer to a different setting, which we called Knowledge Graph augmentation: it is the problem of inferring connections that are necessarily present on the basis of a combination of extensional knowledge and domain experience. In this sense, we borrow techniques that are common in link prediction, e.g. node2vec [26] for clustering purposes, but we do not use them to make linking decisions, which are taken only via logic-based reasoning. In the context of KGs, there have been some proposals for link prediction approaches [28, 32]. However, unlike our approach, the emphasis is still on guessing new connections that cannot be deduced from existing data.

In this paper, for our goals, we borrow from the vast experience of the database community in *record linkage* [20], which is a different yet closely related problem. It consists in deciding whether two records without a common database identifier actually refer to the same real-world entity. Here, the problem is different as we are deciding whether two different real-world entities have some relationship and, if so, its type. Nevertheless, we inherit ideas for search space reduction, namely *blocking* and feature-based *probabilistic record linkage*, adapting them in the context of knowledge graphs. In doing so, we also rely in our experience in formulating and solving data science and data integration problems in a declarative context and in Vadalog in particular [11]. Finally, the general theoretical setting, including

the definition of basic data models (e.g., property graphs), is also shared with the *graph querying* community [15].

Overview. The remainder of the paper is organized as follows. In Section 2 we describe our industrial setting of the company ownership graph at Banca d'Italia. In Section 3, we introduce the background of Vadalog-based KGs, graph embeddings as well as the foundations of our approach. In Section 4 we present the details of Vada-Link. In Section 5 we illustrate the architecture of the system, while Section 6 shows experimental evaluation. We draw our conclusions in Section 7.

2 INDUSTRIAL SETTING: AUGMENTING OWNERSHIP GRAPHS

A central bank needs data about companies to pursue a number of core institutional goals. In particular, the Bank of Italy owns the database of Italian companies, provided by the Italian Chambers of Commerce. Although the database is extremely rich and comprehensive, we shall see that many of the problems of interest that cannot easily be solved using traditional query languages can be concisely formulated as reasoning tasks on knowledge graphs. We show how to build such a company knowledge graph, in particular to identify hidden links between the involved entities in various scenarios. Such problems can be easily represented as a KG augmentation problem: given an input graph, we seek new edges that improve the connectivity of the graph to gain the ability to find new and undiscovered patterns. Let us start with a high-level description of the database.

The Italian Company Database. The database provided by the Italian Chambers of Commerce contains several features for each company such as legal name, address, incorporation date, legal form, shareholders and so on. A shareholder can be either a person or company. For persons, we find the associated personal data, such as first name, surname, date and place of birth, sex, home address, and so on. Detailed shareholding structure is also available: for each shareholder the database contains the actual share as well as the type of legal right associated to each share (ownership, bare ownership and so on).

The database at our disposal contains data from 2005 to 2018. If we see the database as a graph, where companies and persons are nodes and shareholding is represented by edges, on average, for each year the graph has $4.059M$ nodes and $3.960M$ edges. There are $4.058M$ Strongly Connected Components (SCC), composed on average of one node, and more than $600K$ Weakly Connected Components (WCC), composed on average of 6 nodes, resulting in an high level of fragmentation. Interestingly, the largest SCC has only 15 nodes, while the largest WCC has more than one million nodes. The *average in-* and *out-degree* of each node is ≈ 1 and that the *average clustering coefficient* is ≈ 0.0084, very low when compared to the number of nodes and edges. Furthermore, it is interesting to observe that the *maximum in-degree* of a node is more than $5K$ and the *maximum out-degree* is more than $28K$ nodes. We also observe a high number of self-loops, almost $3K$, i.e., companies that own shares of themselves, which could be referred to the buy-back phenomenon [4]. The resulting graph shows a scale-free network structure, as most real-world networks [7, 22, 34]: the degree distribution follows a power-law and there are several nodes in the network that act as hubs.

Use Cases. Let us introduce three relevant problems in the context of this database considering Figure 2.

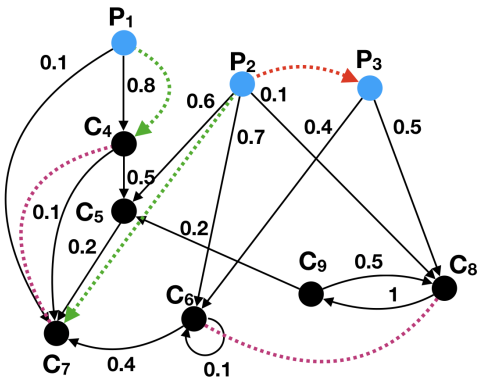

Figure 2: Italian Company Graph. Red-dashed edges represent personal connections, green-dashed edges represent control links, magenta-dashed edges are close-links.

(1) Does P_2 control C_7? More generally, we want to understand if a company or a person exerts control, through the majority of voting rights, on another company, i.e., the *Company Control* problem.
(2) Are companies C_6 and C_8 closely related? More generally, we want to understand whether there exists a link between two companies based on high overlap of shares, i.e., the *close link* problem.
(3) Are there personal/family links between the persons in the company graph? More generally, we wish to *detect personal connections* and label them, on the basis of the kind of relationship.

Solving the third problem allows us to see the first two under a new light: are there a groups of people (e.g., of the same family) in control of a certain company? Are two companies closely related because of the personal ties between their shareholders?

Towards a formalization of the above problems, we propose a graph-based representation of the company database, for which we need some preliminary notions.

Definition 2.1. A (regular) *Property Graph* (PG) is a tuple of the form $G = (N, E, \rho, \lambda, \sigma)$, where:

- N is a finite set of nodes;
- E (disjoint from N) is a finite set of edges;
- *incidence function* $\rho : E \rightarrow N^n$ is a total function that associates each edge in E with an n-tuple of nodes from N –we will consider $n = 2$ from hereinafter;
- the *labelling function* $\lambda : (N \cup E) \rightarrow \mathbf{L}$ is a partial function that associates nodes/edges with a label from a set $\mathbf{L}$;
- $\sigma : (N \cup E) \times \mathbf{P} \rightarrow \mathbf{V}$ is a partial function that associates nodes/edges with properties from $\mathbf{P}$ to a value from a set $\mathbf{V}$ for each property.

The provided company database can be represented in terms of a property graph, as follows.

Definition 2.2. A *Company Graph* is a property graph $G = (N, E, \rho, \lambda, \sigma)$, such that:

- N contains companies and persons;
- E are shareholding edges, from companies to companies or persons to companies;
- the labeling set $\mathbf{L}$ is defined as $\{C, P, S\}$, where C stands for a Company node and P stands for a Person node; S represents a Shareholding edge;

- each node n has an identifier $x \in \mathbf{P}$, with value $\sigma(n, x)$, and a set of properties (or features) $f_1, \ldots, f_m$, with values $\sigma(n, f_1), \ldots, \sigma(n, f_m)$;
- each edge e has share amount w with value $\sigma(e, w) \in (0, 1] \subseteq \mathbb{R}$.

Figure 2 shows an example of the Company Graph. Let us introduce the definitions for the industrial use cases of our interest.

The Company Control Problem. Let us start with that of company control, with an effective formulation presented in the context of logic programming [18].

Definition 2.3. A company (or a person) x *controls* a company y, if: (i) x directly owns more than 50% of y; or, (ii) x controls a set of companies that jointly (i.e., summing the share amounts), and possibly together with x, own more than 50% of y.

Example 2.4. Referring to Figure 2: P_1 controls C_4 by means of a direct 80% edge; P_2 controls C_7, via C_5 and C_6. Green-dashed edges represent the resulting control links.

The Close Link Problem. For the second use case, we need to define the notion of *accumulated ownership* of a company or person x over a company y.

Definition 2.5. Let G be a Company Graph and let ϕ_{xy} be finite the set of all the s simple paths from x to y. Let ϕ_{ij}^m, with $1 \leq m \leq s$, be each of such paths.

The *accumulated ownership* $\Phi_G(x, y)$ of x over y is:

$$\Phi_G(x, y) = \sum_{\phi_{xy}^m \in \phi_{xy}} W(\phi_{xy}^m) \tag{1}$$

where:

$$W(\phi_{xy}^m) = \prod_{e \in \phi_{xy}^m} \sigma(e, w). \tag{2}$$

Starting from Definition 2.5, we can now define *Close Links*.

Definition 2.6. Given a Company Graph G, there is a *Close Link* relationship between a pair of companies x and y for a threshold $t \in (0, 1] \subseteq \mathbb{R}$ if: (i) $\Phi_G(x, y) \geq t$; or, (ii) $\Phi_G(y, x) \geq t$; or, (iii) there exists a person or company z s.t. $\Phi_G(z, x) \geq t$, $\Phi_G(z, y) \geq t$.

Example 2.7. Let us refer to Figure 2 and consider $t = 0.2$. We have that P_3 owns 40% of C_6 and 50% of C_8, therefore they are in close link relationship by Definition 2.6-(iii). Also, since $\Phi_G(C_4, C_7) = 0.2$, it follows that C_4 and C_7 are in close link relationships by Definition 2.6-(i).

Detecting Personal Connections. As we have seen, detecting personal connections enables insightful analyses, including the possibility to achieve more comprehensive view of company control and close links. In particular, we are interested in family relationships of various degrees, i.e. "PartnerOf", "SiblingOf", and so on. Many different models to predict family links exist. We adopt a simple one and, as common in these cases, we define the presence of a family link between x and y with a multi-feature Bayesian classifier. For a given feature f_i, we compute the conditional probability $p_i = P(L_{xy} \mid d(f_i^x, f_i^y) < T_f)$ of having a link L_{xy}, given that some distance between the feature values is under a given threshold for that feature (e.g., Levenshtein distance between two strings "name" of person). The probability p_i and threshold T_f can be estimated by observing $P(d(f_i^x, f_i^y) < T_f \mid L_{xy})$ from training data and given the a priori likelihood of $P(d(f_i^x, f_i^y) < T_f)$ and $P(L)$. Then, we combine all the conditional probabilities p_i into p by applying *Graham Combination* [25]:

$$p = \frac{\prod_{i=1}^n p_i}{\prod_{i=1}^n p_i + \prod_{i=1}^n (1 - p_i)} \tag{3}$$

Finally, a family link exists whenever $p > T$, where T is an established confidence threshold.

Knowing family connections, we can extend Definitions 2.3 and 2.6 to a more general setting. Let us define a family F as a set of persons directly or indirectly connected by personal links.

Definition 2.8. A family F *controls* a company y if: (i) a person $x \in F$ controls y according to Definition 2.3; or, (ii) F controls a (possibly empty) set of companies that jointly, and possibly together with direct ownerships of $\{x_1, \ldots, x_n\} \subseteq F$ (i.e., summing the share amounts), directly own more than 50% of y.

Definition 2.9. Given a Company Graph G, there is a *Close Link* relation between a pair of companies x and y for a threshold $t \in (0, 1] \subseteq \mathbb{R}$ if: (i) Definition 2.6 holds; or, (ii) there exist two persons z_1 and z_2 ($z_1 \neq z_2$) s.t. $z_1, z_2 \in F$, where F is a family, and $\Phi_G(z_1, x) \geq t$ and $\Phi_G(z_2, y) \geq t$.

Example 2.10. Referring to Figure 2, the red-dashed edge between P_2 and P_3 represent "PartnerOf" relationship. For example, P_2 and P_3 are in the same family. Neither P_2 nor P_3 control C_8 singularly. Yet, they have a jointly ownership of 60%; so, their family controls C_8. Also, C_5 and C_8 are closely linked, via P_2 that owns the 60% of C_5 and P_3 that owns 50% of C_8.

The major limitation to answer the illustrated business cases is the translation of all the above definitions into computationally infeasible algorithms due to the dimension of the graph and the complexity of the problems. For example, the close link definition resorts to the ownership definition, and not only: with regard to the third condition of Definition 2.6, the so-called third-party requires to look for a comparison of all existing ownership of companies linked to the third-party entity and a check of the fixed threshold passing. This involves finding all simple paths between pairs of nodes, a paradigmatic computationally hard problem in complexity theory, technically in the #P class [38]. Finding family connections is also challenging, requiring exhaustive comparison of all pairs of persons. Therefore, the intensional part of the company knowledge graph is urgently needed.

3 RELATIONAL GRAPH REPRESENTATION

In order to present the full detail of our KG-based solutions to the settings illustrated in Section 2, let us introduce some basic notions and see how we use them to model and reason on company graphs.

Knowledge Graphs (KG). Along the lines given in [10], we define a KG as a semi-structured data model composed of three components: (i) a *ground extensional component* (or simply extensional component), that is, a set of relational constructs for schema and data, which can be effectively modeled as a *property graph*; (ii) an *intensional component*, that is, a set of *inference rules* over the constructs of the ground extensional component; (iii) a *derived extensional component* that can be produced as the result of the application of the inference rules over the ground extensional component (with the so-called "reasoning" process).

In order to fulfil (i), we adopt a mapping of PG constructs into relational constructs. Let us introduce the background for both and describe our mapping.

Relational Foundations. Let **C**, **N**, and **V** be disjoint countably infinite sets of *constants*, *(labeled) nulls* and (regular) *variables*, respectively. A *(relational) schema* **S** is a finite set of relation symbols (or predicates) with associated arity. A *term* is either a constant or variable. An *atom* over **S** is an expression of the form $R(\bar{v})$, where $R \in \mathbf{S}$ is of arity $n > 0$ and $\bar{v}$ is an n-tuple of terms. A *database instance* (or simply *database*) over **S** associates to each relation symbol in **S** a relation of the respective arity over the domain of constants and nulls. The members of relations are called *tuples*. By some abuse of notations, we sometimes use the terms tuple and fact interchangeably.

Relational Representation for PGs. We map constructs of PGs into relational terms as follows. L-labelled nodes $n \in N$ (with $L_n \in \mathbf{L}$) are represented by facts $L(\hat{c}_x, c_f^1, c_f^2, \ldots, c_f^n)$ of predicate L, where for each property (feature) $f_i \in \mathbf{P}$, we have a constant term c_f^i of L of value $\sigma(n, f_i)$. Note that here we assume a total ordering of property names, so we can map them into positional atom terms and no ambiguity arises (a non-positional perspective could be easily adopted as well). We also assume every node has an identifier x, whose value $\hat{c}_x = \sigma(n, x)$ identifies its facts.

We map each L_e-labelled edge $e \in E$, into facts of predicate L_e: $L_e(\hat{c}_x^1, \ldots, \hat{c}_x^k, f_1, \ldots, f_m)$, where for each argument i of the incidence function ρ, there is a constant term $\hat{c}_x^i$ of L_e with value $\sigma(n, x)$, where $n = \rho(e)[i]$ and x is n identifier, and for each feature $f_i \in \mathbf{P}$ of e there is a constant c_f^i of L_e with value $\sigma(e, f_i)$.

Observe that node and edge labels operate at schema level and map into predicate names. Properties and identifiers are at instance level and define term values for facts representing nodes and edges. With this premises given, in our setting the extensional component of a KG is therefore a database instance representing the PG by means of the mapping described above.

Example 3.1. The extensional component of a KG based on the PG in Figure 1 has the following relational representation.
Company(C), Company(D), Company(E), Company(F), Company(G), Company(H), Company(I), Company(L), Person(P_1), Person(P_2), Own($P_1, C, 0.8$), Own($C, D, 0.75$), Own($D, E, 0.4$), Own($D, F, 0.2$), Own($E, F, 0.4$), Own($P_1, E, 0.2$), Own($P_2, G, 0.6$), Own($G, H, 0.6$), Own($H, L, 0.4$), Own($H, I, 0.1$), Own($P_2, I, 0.5$), Own($F, L, 0.2$).

Inference Rules. We describe the intensional components of KGs with logical inference rules in the Vadalog language. At the core of Vadalog, there is Warded Datalog$^{\pm}$ [12, 14] a language part of the Datalog$^{\pm}$ family [17]. Datalog$^{\pm}$ languages consists of *existential rules*, which generalize Datalog rules with existential quantification in rule heads, so that a rule is a first-order sentence of the form: $\forall \bar{x} \forall \bar{y} (\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z}))$, where φ (the *body*) and ψ (the *head*) are conjunctions of atoms with constants and variables. For brevity, we write this existential rule as $\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$ and replace $\wedge$ with comma to denote conjunction of atoms. The semantics of such rule is intuitively as follows: for each fact $\varphi(\bar{t}, \bar{t}')$ that occurs in an instance I, then there exists a tuple $\bar{t}''$ of constants and nulls such that the facts $\psi(\bar{t}, \bar{t}'')$ are also in I. More formally, the semantics of a set of existential rules Σ over a database D, denoted $\Sigma(D)$, is defined via the well-known *chase procedure* [2]: new facts are added to D by the chase (possibly involving null values to satisfy existentially quantified variables) until $\Sigma(D)$ satisfies all the existential rules of Σ.

Example 3.2. The following rules define intensional edges linking persons with companies they are influential on (e.g., in the sense of control on company decisions). By Rule (1) a person x affects a company c she owns; her spouse also affects the company by Rule (2). Rules (3) and (4) generate Spouse edges, having a validity interval from t_1 to t_2, from Married edges.

(1) Person(x), Own(x, c, v) $\rightarrow$ Influence(x, c).
(2) Own(x, c, v), Spouse(x, y, t_1, t_2) $\rightarrow$ Influence(y, c).
(3) Married(x, y) $\rightarrow \exists t_1, t_2$ Spouse(x, y, t_1, t_2).
(4) Spouse(x, y, t_1, t_2) $\rightarrow$ Spouse(y, x, t_1, t_2).

4 THE VADA-LINK FRAMEWORK

In this section we focus on Vada-Link and describe in detail our technique for KG augmentation.

Algorithm 1 Basic KG augmentation algorithm.
Input: $G = (N, E, \rho, \lambda, \sigma)$, C: link classes
Output: $U = (N, E', \rho', \lambda', \sigma)$

```
1:  U ← G
2:  changed ← true
3:  while changed do
4:      changed ← false
5:      K ← GraphEmbedClust(U)
6:      for K ∈ K do                                   ▷ First level
7:          B ← GenerateBlocks(K)
8:          for B(N_B, E_B, ρ_B, λ_B, σ_B) ∈ B do      ▷ Second level
9:              for p_1, p_2 ∈ N_B, c ∈ C do
10:                 if Candidate(p_1, p_2, c) and e ∉ E_B then
11:                     add e to E_B
12:                     set ρ_B(e) = (p_1, p_2)
13:                     set λ_B(e, TYPE) = C.
14:                     changed ← true
15: return U
```

Algorithm 1 presents a high-level overview of the approach. We take as input a property graph G and return a property graph U, which is obtained by adding the predicted edges. In our industrial case introduced in Section 2, the property graph is a company graph and the predicted links can be a control relationship, a close link relationship, or a family link.

The overall approach consists in a double level of clustering (which we will also call *blocking*) of the graph: after the first grouping, possible links between nodes are searched only within a single cluster in order to limit the number of needed comparisons. We start from a clustering **K** (line 5) performed by the function *GraphEmbedClust*. We compute a node embedding of the whole graph U: nodes are mapped into multi-dimensional vectors, whose distance reflects the similarity of the nodes, evaluated on the basis of both their features and role in the graph topology. Then each cluster K is in turn partitioned into a more specific clustering **B** (line 7) by the function *GenerateBlocks*. The search for nodes to be linked is then performed within each cluster B and for each type $c \in C$ of links by *Candidate* (line 10). If an edge is to be created for the considered nodes, it is labeled with the proper type (lines 11-13). Once all clusters B in **B** have been used to enrich U with new edges, first-level clustering via graph embedding is recursively applied (line 5). The algorithm proceeds until no changes occur. Finally, U is returned.

The Knowledge Graph. We implemented the core KG augmentation logic of Vada-Link described in Algorithm 1 with a Vadalog KG, where G is the ground extensional component and the

intensional component is defined by three set of rules, encoding the prediction logic for the edges to be added to G to obtain U. The first set of rules (Algorithm 2) is the *input mapping*: they take as input the relational representation of one specific PG, as defined in Section 3 (for example that of the company graph) and transform it into higher-level concepts: generic nodes, generic edges and types and properties for those nodes and edges. The second set of rules (Algorithm 3) is the actual *link prediction logic*: it contains the core reasoning process giving rise to new edges; it operates on generic nodes and edges. Finally, the third set of rules (Algorithm 4) is the output mapping: it transforms the high-level generic links that have been created by the link prediction logic into their relational representation in the PG.

Let us now analyse the details of the three sets of rules.

Algorithm 2 Input mapping for the company PG.

$$\text{(1) Company(name, addr, inc. date, leg. form, \ldots)},$$
$$z = \#\text{sk}_c(\text{name}) \rightarrow$$
$$\text{Node}(z, \text{name}, \text{addr}, \text{inc. date}, \text{leg. form}), \text{NodeType}(z, \text{Comp}).$$
$$\text{(2) Person(name, birth, addr, \ldots)},$$
$$z = \#\text{sk}_p(\text{name}) \rightarrow$$
$$\text{Node}(z, \text{name}, \text{birth}, \text{addr}, \ldots), \text{NodeType}(z, \text{Person}).$$
$$\text{(3) Own}(x, y, \text{amount}, \text{right}, \ldots), \text{right} = \text{pers.share} \rightarrow$$
$$\exists z\ \text{Link}(z, \#\text{sk}_p(x), \#\text{sk}_c(y), \text{amount}, \text{right}, \ldots),$$
$$\text{EdgeType}(z, \text{Shareholding}).$$
$$\text{(4) Own}(x, y, \text{amount}, \text{right}, \ldots), \text{right} = \text{comp.share} \rightarrow$$
$$\exists z\ \text{Link}(z, \#\text{sk}_c(x), \#\text{sk}_c(y), \text{amount}, \text{right}, \ldots),$$
$$\text{EdgeType}(z, \text{Shareholding}).$$

The ground extensional component is modeled by means of three atoms: `Company`, `Person` and `Own`. `Company` and `Person` hold the basic features for companies and persons, respectively –the feature names are self explanatory– and are identified by `name`; `Own` connects a person/company with name x to a company with name y, when x is a shareholder of y and contains the respective features, such as `amount` and type (`right`) of share. Clearly, here we are assuming that `name` is a valid unique identifier for the sake of simplicity, while in practice fiscal code or other more appropriate codes are used. Rule (1) upgrades companies into generic `Nodes` of `NodeType` "Company" (quotes are omitted in the rules for readability). Similarly, Rule (2) upgrades persons. For simplicity of presentation and without loss of generality, here we assume that atoms are variadic (denoting extra terms with "…") so as to support an arbitrary number of features; coherence is guaranteed by the adoption of a positional perspective in our relational representation of PGs and by nodes and edges being typed. Clearly, we could replace variadic nodes and edges with Feature(x, f, v) atoms, explicitly representing a feature f of value v for node x.

Both Rules (1) and (2) use functions denoted by #sk, namely *Skolem functors*, to generate node identifiers. Skolem functors are often used in variants of Datalog with OID invention, where identifiers need to be generated [5, 16] with specific properties: (i) *determinism*: repeated applications of the same functor on the same argument yield the same OID –this will be useful to generate edges; (ii) *injectivity*: there are no distinct domain elements yielding the same OID –for instance, no different companies will be assigned the same OID; (iii) *disjoint range*: different Skolem functors cannot produce the same OID –in case a company and a person have the same name, $\#\text{sk}_c$ and $\#\text{sk}_p$ will produce different OIDs. Vadalog supports Skolem functors in this form.

Rule (3) and Rule (4) upgrade ownership into generic `Links` with an `EdgeType` depending on the type of ownership: `pers.share` for persons holding companies, `comp.share`, in the case of a company owning a company. We adopt existential quantification to generate OIDs for `Links` and Skolem functors to obtain the OIDs of the `Nodes` associated to names x and y.

Observe that the application order of the rules is irrelevant: thanks to determinism of Skolem functors, `Links` can be generated even before the respective `Nodes` for companies and persons.

Algorithm 3 Vadalog KG augmentation logic.

$$\text{(1) Node}(x, f_1^x, \ldots, f_n^x), \text{Link}(e, v, w, f_1^e, \ldots, f_m^e),$$
$$\text{NodeType}(x, t_n), \text{EdgeType}(e, t_e),$$
$$b_1 = \#\text{GraphEmbedClust}(f_1^x, \ldots, f_n^x, f_1^e, \ldots, f_m^e, t_n, t_e, \langle e \rangle),$$
$$b_2 = \#\text{GenerateBlocks}(f_1^x, \ldots, f_n^x, t_n) \rightarrow \text{Block}(b_1, b_2, x)$$
$$\text{(2) Node}(x, f_1^x, \ldots, f_n^x), \text{Node}(y, f_1^y, \ldots, f_n^y),$$
$$\text{NodeType}(x, t_n), \text{NodeType}(y, t_n), x \neq y,$$
$$\text{Block}(b_1, b_2, x), \text{Block}(b_1, b_2, y), \text{LinkClass(t)},$$
$$\text{Candidate}(x, y, t) \rightarrow \exists z\ \text{Link}(z, x, y, \ldots), \text{EdgeType}(z, t).$$

Algorithm 3 represents the core prediction logic of Vada-Link. For every node x, Rule (1) considers all the edges e of the graph and positions x into a two-level nested clustering structure represented by the atom `Block`, where b_1 and b_2 are the clustering levels. The first clustering is established by applying the function #GraphEmbedClust, which wraps a function call to a specific clustering algorithm based on node embedding (details in Section 4.1). It takes as input the features of x, all the edges e of the graph along with their features, the respective types t_n and t_e and returns the identifier b_1 of the first-level cluster. The second-level clustering is determined by applying the function #GenerateBlocks, whose resulting cluster identifier b_2 only depends on the node properties and type.

Some discussion of the use of functions in our solution is needed. #GenerateBlocks is a *fact-level function* (details in Section 4.2) and its semantics is quite straightforward: for a specific binding of the function arguments that is part of its domain, a value for b_1 is produced. The function is in some sense polymorphic: depending on the type t_n of the involved nodes, a specific semantics is applied to decide the target cluster on the basis of the node features. For example, if the node represents a person, a specific algorithm may rely on last names or addresses; in case of companies, the industrial sector may be relevant, and so on.

#GraphEmbedClust is a *monotonic aggregation function*. Aggregation functions are adopted in various settings making use of logical formalism and the need for a careful definition arises in all of them, especially in the presence of model based semantics [31]. Among the various types of aggregation that Vadalog features [13], in our solution we adopt the monotonic one [37], which respects monotonicity w.r.t. to set containment. Intuitively, aggregation is provided in the form stateful fact-level functions, which memorize the current aggregate value; subsequent invocations of a function then yield updated values for the aggregate so as that the "final value" is the actually desired aggregate value. Thanks to monotonicity, such final value can be easily identified as the minimum/maximum value. In order to decide the first-level clustering for node x, the function #GraphEmbedClust takes as input the node features, an edge e with its features, and the node and edge types t_n and t_e; for a given node x, whenever the function is activated for a new edge e (notation $\langle e \rangle$ denotes that e is such an aggregation contributor), a more accurate clustering

is possible and a new, greater value for b_1 is returned. In fact, #GraphEmbedClust wraps the invocation of a node2vec primitive (some details in Section 4.1), whose precision depends on the portion of the graph that is available to it.

For every second-level cluster defined by a `Block` fact, Rule (2) exhaustively considers all the pairs of nodes x and y and for every possible `LinkClass` t (wrt our case many exist: Control, CloseLink, ParentOf, PartnerOf, etc.); the `Candidate` predicate (details in Section 4.3) is used decide whether a `Link` from x to y must be produced or not. If the case, a new t-typed edge is created. Observe that Rule (2) compares only the pairs of nodes in the same sub-cluster (identified by b_1 and b_2).

Algorithm 4 Output mapping for the company PG.

(1) $\text{Link}(z, x, y), \text{EdgeType}(z, \text{Control}) \rightarrow \text{Control}(x, y)$.
(2) $\text{Link}(z, x, y), \text{EdgeType}(z, \text{CloseLink}) \rightarrow \text{CloseLink}(x, y)$.
(3) $\text{Link}(z, x, y), \text{EdgeType}(z, \text{ParentOf}) \rightarrow \text{ParentOf}(x, y)$.
(4) $\text{Link}(z, x, y), \text{EdgeType}(z, \text{PartnerOf}) \rightarrow \text{PartnerOf}(x, y)$.

Algorithm 4 is the output mapping, actually transforming the predicted edges back into the PG language. With reference to our industrial case, Rules (1) and (2) generate `Control` and `CloseLink` links, respectively. Rules (3) and (4) exemplify possible family links, and many more exist.

4.1 Clustering with Node Embeddings

Embeddings are mappings of real-world objects into high dimensional real-valued vectors that guarantee specific, e.g. geometric, properties reflecting the semantic relationships between the objects. Typically, embeddings are based on some similarity or neighbourhood notion, like in word embeddings [6]. The function #GraphEmbedClust in Rule (1) of Algorithm 3 implements a graph embedding, specifically a *node embedding*, that is, mappings of graph nodes into vectors so that network node neighbourhood is preserved. Specific algorithms learn node embeddings with different random walk strategies, resulting in the optimization of different measures as a consequence. In the function we adopt *node2vec* [26], a particularly interesting embedding which optimizes both network vicinity and network role of a node. We map graph nodes into vectors with 128 dimensions, in such a way that their distance preserves feature-based node similarity as well as neighbourhood, e.g., the so-called "homophily", that is, nodes having the same friend nodes are considered similar. We also apply a preliminary dimensionality reduction step based on T-SNE (*t-distributed stochastic neighbor embedding spectral clustering*) [35]. The #GraphEmbedClust is *polymorphic* in the sense that depending on the types t_n and t_e of involved nodes and edges, it adopts different embedding strategies, which highlight the topological peculiarity of the problem. For instance, it is indeed common that persons having largely overlapping groups of family members are in turn connected by a family relationship; conversely, companies in hold of overlapping sets of shares of other companies, tend to be part of the same group.

Clustering based on graph embeddings shows to be particularly useful in the industrial cases at hand, since it includes in the same clusters candidates that would be far in terms of their descriptive features. This goes beyond the textual or numeric features of companies and persons. Moreover, the interplay between different types of links is also interesting: for example, it is our experience that people in hold of overlapping sets of shares tend to be family members; vice versa, companies owned by overlapping sets of people tend to be part of the same group. The combination of quantitative features and structural graph property is also noteworthy. People owning certain patterns of shares of the same company tend to have family connections.

4.2 Generating Blocks

#GenerateBlocks in Rule (1) of Algorithm 3 reduces the search space for link candidates by sub-clustering. In particular, the function takes as input a vector of features of a node x (a person or a company in our case), belonging to first-level cluster b_1 and returns the identifier b_2 of a second-level cluster. Note that a careful choice the features, *feature engineering phase*, is fundamental: in Vada-Link, we modularize out this highly domain dependent aspect into the specific implementations of #GenerateBlocks, so that the overall solution is general and ad-hoc tuning is possible whenever new business domains arise.

Vada-Link provides different pluggable implementations for various domains. In most of the cases sub-clustering can be determined on the basis of well-known hashing or partitioning techniques. We support intuitive declarative specifications of auxiliary functions in Vadalog as follows:

$$\text{Hash}(h, f_1, \ldots, f_n), \text{Features}(f_1, \ldots, f_n) \rightarrow \text{Ans}(h).$$

Here, the `Feature` atom binds to the vector of features taken as input by #GenerateBlocks; with a join with the `Hash` atom, the functionally dependent hash value h is returned. The above rule is clearly complemented by the set of facts defining the underlying `hash` relation. Conventionally the `Ans` atom denotes the function return value. Alternative implementations could be based on Skolem functors as follows:

$$h = \#\text{Sk}_h(f_1, \ldots, f_n), \text{Features}(f_1, \ldots, f_n) \rightarrow \text{Ans}(h).$$

4.3 Generating Matching Candidates

The possible candidates to be linked are matched by the polymorphic `Candidate` predicate, which has different implementations, depending on type of link to be predicted. Let us show how this is applied in our cases.

Algorithm 5 `Candidate` predicate for company control.

(1) $\text{Node}(x, f_1, \ldots, f_n), \text{NodeType}(x, \text{Company}) \rightarrow$
$\text{Candidate}(x, x, \text{Control})$.
(2) $\text{Candidate}(x, z, \text{Control}), \text{Link}(u, z, y, w)$
$\text{EdgeType}(\text{u}, \text{Shareholding}), \mathit{msum}(\text{w}, \langle \text{z} \rangle) > 0.5$
$\rightarrow \text{Candidate}(x, y, \text{Control})$.

The Vadalog rules in Algorithm 5 define candidate companies to be linked by a control relationship, according to Definition 2.3. Such relationship holds for a company on itself (Rule (1)); then, whenever a company x controls a set of companies z that jointly own more than 50% of a company y, then x controls y.

Algorithm 6 defines companies that are in close link relationships (with threshold T) according to Definition 2.6. Rule (1) and (2) calculate accumulated ownership for companies x and y, according to Definition 2.5. Rule (3) gives the base case for close links: accumulated ownership greater than or equal to 20% is a close link. Close links are symmetric, by Rule (4). Finally, if a company z owns a significant share of both x and y, they are a close link by Rule (5).

Algorithm 6 `Candidate` predicate for close links.

(1) Link(z, x, y, w), EdgeType(z, Shareholding) $\rightarrow$
AccOwn(x, y, w).
(2) Link(u, x, z, w_1), EdgeType(u, Shareholding),
AccOwn(z, y, w_2), $v = msum(w_1 \cdot w_2, \langle z \rangle)$
$\rightarrow$ AccOwn(x, y, v).
(3) AccOwn(x, y, w), $w \geq T \rightarrow$ Candidate(x, y, CloseLink).
(4) Candidate(y, x, CloseLink) $\rightarrow$ Candidate(x, y, CloseLink).
(5) AccOwn(z, x, w_1), AccOwn(z, y, w_2), $w_1 \geq T, w_1 \geq T$,
$\rightarrow$ Candidate(x, y, CloseLink).

Algorithm 7 `Candidate` predicate for PartnerOf.

Node($x, f_1^x \ldots f_n^x$), Node($y, f_1^y \ldots f_n^y$), NodeType(x, Person),
NodeType(y, Person), #LinkProbability($f_1^x \ldots f_n^x, f_1^y \ldots f_n^y$) >
T $\rightarrow$ Candidate(x, y, PartnerOf).

Algorithm 7 defines a pair of candidate persons that have a family connection. In the example, the function #LinkProbability implements Equation 3 of Section 2. Here we consider the "PartnerOf" relationship, but similar algorithms are valid for all personal connection types.

We are now able to extend company control and close link detection to support the presence of families. For the sake of space, we omit here Vadalog rules generating "Family" nodes F and "Family" links connecting persons to their family.

Algorithm 8 `Candidate` predicate for family control.

(1) Node($F, f_1^F, \ldots, f_n^F$), NodeType(F, Family),
Link(z, x, F), EdgeType(z, Family), Link(v, x, y),
EdgeType(v, Control) $\rightarrow$ Candidate(F, y, Control).
(2) Candidate(F, x, Control), Link(z, x, y, w),
EdgeType(z, Shareholding), $msum$(w, $\langle$x$\rangle$) > 0.5
$\rightarrow$ Candidate(F, y, Control).
(3) Link(u, i, F), EdgeType(u, Family),
Link(z, i, y, w), EdgeType(z, Shareholding),
$msum(w, \langle i \rangle) > 0.5 \rightarrow$ Candidate(F, y, Control).

Rules (1) Algorithm 8 implement condition (i) of Definition 2.8, while Rules (2) and (3) implement condition (ii) by accounting for the contribution on y of both the companies x controlled by F and the direct ownership of members of F. Technically, the two monotonic summations of Rules (2) and (3) contribute to the same total, one for each $\langle F, y \rangle$ pair.

Algorithm 9 `Candidate` predicate for family close link.

(1) Link(z, i, F), EdgeType(z, Family),
Link(k, j, F), EdgeType(k, Family), $i \neq j$,
AccOwn(i, x, v), v $\geq$ 0.2, AccOwn(j, y, w), w $\geq$ 0.2,
$\rightarrow$ Candidate(x, y, CloseLink).

Finally, Algorithm 9 extends Algorithm 6 and implements part (ii) of Definition 2.9.

4.4 Discussion

We conclude the section with some informal arguments about termination, correctness and properties of our approach. I.e., Algorithms 2, 3 and 4 terminate and, in particular, Algorithm 3 correctly adds the required edges as defined in Algorithm 1.

Termination. As we have touched on in Section 3, the semantics of a set of existential rules Σ is given by the chase procedure, where new facts are added to database D (the extensional component), until $\Sigma(D)$ satisfies all the rules. We argue that in our case $\Sigma(D)$ is always finite.

- Algorithms 2 and 4 are non-recursive, therefore each rule adds a finite number of facts to $\Sigma(D)$, because of the finiteness of the extensional component.
- Algorithm 3 is recursive, as `Links` generated by Rule (2) appear in the body of Rule (1) –edges recursively improve the embeddings. The number of `Links` that can be generated by Rule (2) is finite and, in the worst case, it amounts to $|N|^2 \times C$, where N are the PG nodes and C is the number of possible link types. Therefore, Rule (2) produces a finite number of facts in $\Sigma(D)$, up to renaming of link identifiers z. Technically, the Vadalog chase procedure applies isomorphism check to prevent the generation of redundant facts. Therefore, Rule (1) produces a finite number of clusterings $\langle b_1, b_2 \rangle$, since it can fire in the worst case for every single edge in E plus all the ones introduced by Rule (2). Therefore Algorithm 3 always terminates.
- Special care must be paid in the specific polymorphic implementations of the `Candidate` predicate. Observe that in our settings, Algorithms presented in Section 4.3, in the worst case enumerate all the graph paths, and so always terminate.

Correctness. Let us show that each element (e.g., a company or a person) is correctly assigned to a single cluster and pairwise comparison is correctly performed inside each of them and for all possible link classes, as defined in Algorithm 1.

- The nested clustering is produced by the joint use of functions #GraphEmbedClust and #GenerateBlocks within Rule (1) of Algorithm 3. They are both applied to each node and the generated identifiers, b_1 and b_2, appear as terms the head of Rule (1) as well as the node x. As a consequence, because of set semantics, every node x is assigned to a unique pair $\langle b_1, b_2 \rangle$.
- In the body of Rule (2) of Algorithm 3, the two `Node` atoms operate on all pairs of nodes $\langle x, y \rangle$ such that x and y share the clustering configuration $\langle b_1, b_2 \rangle$. Therefore, no other elements are involved in the comparison performed by the `Candidate` predicate. Moreover, Rule (2) fires for each possible class t of `LinkClasses` and so, eventually, all possible triples $\langle x, y, t \rangle$, with $\langle x, t \rangle$ in the same cluster are evaluated.

A broader consideration of the overall correctness and completeness of the approach applied to the specific problems over company ownership graphs is necessary. All the implementations of the `Candidate` predicate we presented in Section 4.3 are deterministic, in the sense that they apply a priori conditions that encode the domain knowledge and produce a linking decision. Clearly, `Candidate` may be implemented as a statistical model, but still, this decision remains deterministic. The specific configuration of the clustering mechanism, i.e., the specific implementations of #GraphEmbedClust and #GenerateBlocks, determines which pairs of nodes are considered by `Candidate` and, as we will see in Section 6, this is the main key to achieving scalability. Hence, if two nodes x and y are supposed to be connected by a t-typed edge e and the recursive clustering mechanism always assigns them to different blocks, the final result will not be complete and miss e. It is the responsibility of the data engineer to strike an acceptable balance between completeness and granularity of clustering from case to case.

Finally, observe that the correctness of the links predicted by Vada-Link depends on the correctness/statistical robustness of the single implementations of `Candidate`, which is problem dependent and out of the scope of this paper.

Complexity. In the average case, clustering allows for linear behaviour, as we experimentally show in Section 6. In the worst case, i.e., all nodes are assigned to the same cluster, the approach performs $|N|^2 \times C$ comparisons, where N are the nodes in the extensional components and C the number of possible link types. However, with highly dense graphs, complexity is dominated by that of #GraphEmbedClust since node2vec has quadratic complexity in the graph branching factor [33]. Also, the complexity of the specific implementations of `Candidate` must be taken into consideration from case to case, as it could be dominating, for example with problems that require path enumeration. Full complexity characterization of company graph problems is beyond the scope of this paper. However, we point out that the complexity of `Candidate` can be controlled by a careful choice of the sub-fragment of Vadalog language. In fact, if the task is described in *Warded Datalog*$^{\pm}$, the fragment at the core of the Vadalog language, there is the formal guarantee of polynomial complexity [12].

Properties of the approach. Our approach is *schema and model independent*. It is schema independent in the sense that Vada-Link is able to perform KG augmentation regardless of the specific input PGs. This independence is achieved by means of a preliminary "promotion" of the relational representation of the extensional component into a generic graph model (Algorithm 2), with abstract constructs (nodes, edges, features, types). The link prediction logic is then applied within the generic graph, with specific polymorphic implementations of `Candidate` for each problem, encoded in Vadalog in terms of these high-level constructs. In this sense, the approach is also *problem aware*. Finally, the generated links are mapped again into the specific graph schema with Algorithm 4.

The approach is model independent in the sense that the extensional component can originate from heterogeneous data sources, even based on different data models (relational, object oriented, XML, NoSQL models), under the condition it can be imported into the relational representation of PGs described in Section 3.

Finally, it is worth remarking that our prediction technique is based on a kind of *reinforcement principle* because the positively predicted edges in turn help new predictions. In fact, the first-level clustering in Algorithm 3 is gradually improved with new edges being considered from both the ground ones and those generated by Rule (2).

5 THE ARCHITECTURE OF VADA-LINK

The development of enterprise applications using KGs is still a largely unexplored area and design methodologies as well as architectural patterns are gradually emerging. In [10], we proposed a set of architectural principles for the design of KG-based applications; in the following, we briefly motivate how those principles are satisfied by Vada-Link architecture.

- *Assign the immediately known, original information to the ground extensional component of the KG.* In fact, Algorithm 2 embodies the construction of a property graph exactly holding the information available in the company database. Besides the simple mapping, we clearly perform data cleaning and quality enhancement steps, whose details are omitted in this work.
- *For the schema design of the extensional component, adopt well-known conceptual modeling techniques.* In Vada-Link, we provide a relational representation of the extensional data (Section 3) that respects consolidated design and data normalization practices: relevant entities, persons, ownerships and companies in the case at hand, reflect into standalone relations, incorporating identifiers as well as the specific features as attributes.
- *Use extensional rules to model sophisticated and reasoning intensive business rules.* We represent the logic needed to generate the links in the form of declarative specifications. More concretely, with respect to the company KG, we consider multiple kinds of links (close links, company control, family relationships) and for each of them propose a Vadalog program representing it.
- *Keep business logic in the applications (do not let it drift into the KG intensional component).* We carefully separate the business logic of the client applications, that is, the software components using our KG for internal purposes (e.g., economic research, anti-money laundering, etc.) from any logic needed to generate the links. The Vadalog rules represent only the latter, whereas the application business logic resides within the application components. We choose not to implement each polymorphic variant of generation of clusters and evaluation of matching candidates (Sections 4.2 and 4.3) in a dedicated software module. First, such module would be highly coupled to data and hardly explainable or modifiable; more in general, the advantages of declarative approaches in the KG realm are largely acknowledged [30]. In our case, the adoption of logic-based rules appears particularly effective under three perspectives. *Understandability*: it is our experience that business users approach and appreciate human-readable rules instead of pure code; *modifiability*: given by the combination of high abstraction level and compactness of code (20-30 lines of Vadalog rules against 1K+ lines of Python code for the three cases at hand); *IT independence*: avoiding the strong coupling to a specific programming language.

Figure 3 shows the full functional architecture of Vada-Link. Its goal is building a KG from an existing source database. To this aim, data fetched from the RDBMS are enriched with features and extensions from external sources, with common ETL jobs.

The enriched dataset is then used as input to build the extensional component of the KG, that is, the property graph. Our *graph-building pipeline* takes as input an arbitrary database schema and maps it into the relational representation for PGs described in Section 3. The property graph is stored into a *Neo4J* server. The set of Vadalog rules in Algorithms 2, 3 and 4 are stored into a dedicated repository and executed by Vadalog. Enterprise applications interact with the KG via a *reasoning API*. Full details about Vadalog architecture are in [14].

6 EXPERIMENTS

In this section, we provide an experimental evaluation our approach to KG augmentation. The goal of the section is to highlight the scalability and accuracy of the overall approach in Vada-Link and not focussing on the specific Vadalog implementations of the three problems. In fact, out of the three KG augmentation problems we have presented, for the goal of this section we concentrate on the detection of family connections, which is at the same time straightforward but helpful to stress the system. Specific performance tests on complex reasoning tasks, including company control, can be found in [12, 14].

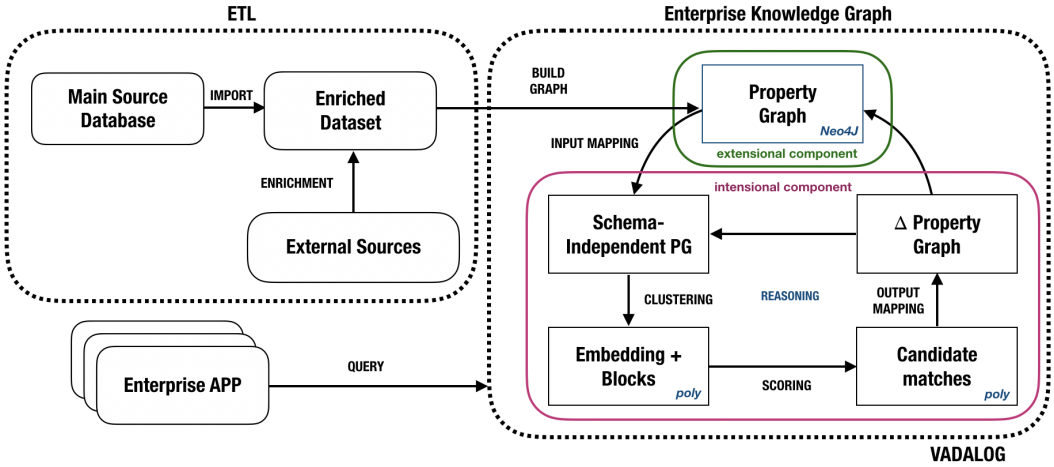

Figure 3: The functional architecture of Vada-Link.

We validate Vada-Link on both real-world and artificial data, showing that it exhibits good scalability and accuracy.

Datasets. The *real-world data* we use in the following experiments is the database of Italian companies of Banca d'Italia, as described in Section 2. For the *synthetic data*, we developed a graph generator. In particular, since company networks tend to be *scale-free networks* (see Section 2), we built different artificial graphs by adopting Barábasi algorithm [8] for the generation of scale-free networks, varying the number of nodes and the graph density. For each node, we randomly generated 6 features, out of distributions respecting their statistical properties.

Software and hardware configuration. We ran the experiments on a MacBook with 1.8 GHz Intel Core i5 and 4 GB 1600 MHz DDR3 memory. Vada-Link has been compiled with JDK 1.8.0_4 and clustering functions executed with Python 3.2.3.

6.1 Evaluating Scalability

We tested the scalability of Vada-Link in a number of settings, varying both the graph topology (e.g., number of nodes, density) and the data distribution, inducing different clustering structures.

Varying number of nodes. We investigate the impact of the number of nodes on the performance of Vada-Link. For the *real-world case*, we built 20 scenarios with subsets from the Italian company graph presented in Section 2, with ≈ 1-100k nodes representing persons. In order to stress the system even more with *synthetic data*, we built 6 artificial graphs with ≈ 1-10k nodes, having the same scale-free topology as the real-world graphs, but much higher density. We performed each experiment 10 times and averaged the elapsed times.

Results. The results for real-world data are reported in Figure 4(a). Vada-Link shows good scalability: execution time (blue line) grows slightly more than linearly with the number of nodes and remains under 20 seconds for 10k nodes. The trend is significantly far from quadratic growth, which we would expect with the naive approach consisting of exhaustive all-pairs comparison (red line). Figure 4(b) shows the results for synthetic data. Although the elapsed times are higher by one order of magnitude, which we explain with the highly dense topology of the generated artificial data, the trend is still linear and the approach effective.

Varying the number of clusters. We seek to observe the impact of the number of clusters on the overall execution time. To this end, we crafted a *real-world-like* experiment adopting the company graph presented in Section 2 and artificially tweaking the value of some features. As we have illustrated in Sections 4.1 and 4.2, our clustering technique is based on a recursive combination of node2vec (first-level clustering) and feature based blocking (second-level clustering); in particular, in Section 4.2 we have shown that the assignment to a second-level cluster is decided on the basis of a deterministic mapping –via hashing or Skolem functors– of a feature vector $f_1, \ldots, f_n$ into a cluster identifier. In this experiment, we alter the value of k of such n features in order to hijack the mapping into an increasing number of clusters of decreasing size and observe how this affects elapsed time. We extract values for the vector $f_1, \ldots, f_k$ from a discrete multivariate uniform distribution over the sample space $S_1, \ldots, S_k$. To induce more (fewer) and smaller (bigger) clusters, we restrict (expand) the cardinality of the domain of $S_1 \times \ldots \times S_k$. Specifically, we mapped $f_1, \ldots, f_n$ into ≈ 1-500 clusters.

Results. Figure 4(c) confirms that, as usual in clustering approaches, effective application of Vada-Link calls for a careful feature engineering phase: the selectivity of the features, i.e., the number of distinct values in the domain, and the cardinality of their domain affect the number and size of the clusters. For example, searching for the "siblingOf" relationship among people of the same last name and age range, would lead to clusters including thousands of persons, since certain last names are notably more common than others. Resorting to specific features, for example address vicinity or geographic area, could highly reduce the search space. Also, while the above considerations are somehow intrinsic and common to any clustering approach, in Vada-Link, the adoption of a hybrid node2vec/feature-based clustering combined within a recursive self-improving approach make it easier to strike a good balance between number of clusters (and hence elapsed time) and recall, as we shall see in Section 6.2.

Varying the density. In this experiment we investigate the impact of the density of the graph on the performance of Vada-Link. To this end, we built 4 artificial scenarios, *superdense*, *dense*, *normal*, *sparse*, corresponding to graphs of increasing density, and measured execution time for subset of these graphs of 1-1k nodes.

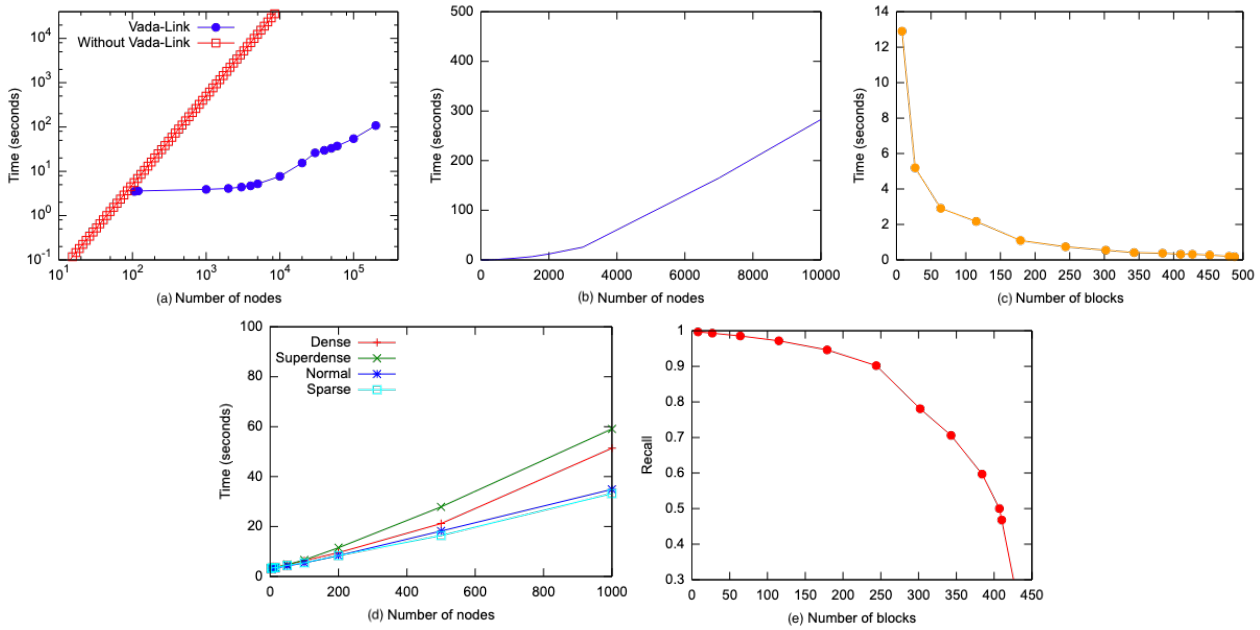

Figure 4: Analysis of VADA-LINK execution time, depending on: (a) number of nodes in the graph (real-world data from the graph of Italian companies); (b) number of nodes in the graph (synthetic data); (c) number and size of the clusters (real-world data); (d) graph density (synthetic data). Analysis of VADA-LINK recall, depending on: (e) number and size of the clusters (synthetic data).

Results. The results, shown in Figure 4(d), highlight that the performance of VADA-LINK is influenced by the density of the considered graph. In particular, we notice an increase of the elapsed times, especially significant for more than 500 nodes. For lower values, sparse, normal and dense show similar trends, whilst superdense is slower, with ≈30 seconds for 500 nodes. The trend is amplified for greater values, with superlinear growth for dense and superdense. While for second-level clustering, #GenerateBlocks in Algorithm 3 is not affected by node density by construction, since it only considers node features, both first-level clustering (#GraphEmbedClust) and the implementations of `Candidate` predicate are. Node2vec needs to process a number of random walks that grows with the density; nevertheless, once a density is fixed, it scales almost linearly with the number of nodes, as also experimentally shown in [26]. Also the behaviour of `Candidate` is highly dependent on the considered KG augmentation problem. For example, detection of family connections have good scalability w.r.t. density as evident in Figure 4(d). Company control and close link detections are more challenging (specific experiments in [12, 14]). Nevertheless, thanks to clustering, VADA-LINK can achieve good behaviour also in this case, clearly at the cost of loss in accuracy.

6.2 Evaluating Accuracy

As typically done in the validation of link prediction settings [26], we consider a graph with some edges arbitrarily removed; then, it is our goal to predict those missing edges and in so-doing we are interested in recall, i.e., how many of those removed edges we have recovered. We wish to infer the connections that need to be present and whose definitions are expressed by the polymorphic but deterministic implementations of the `Candidate` predicate, for example, Algorithm 7.

In some settings, `Candidate` implements certain models, like in the case of company control and asset eligibility, for which we would just need to discuss correctness from case to case; in others, `Candidate` encodes a classification model, for which the usual validation methodologies (confusion matrix, accuracy, precision, recall, ROC, AUC, etc.) would be valuable. This is indeed the case of the detection of family connections, our third setting. However, our goal here is not to concentrate on the statistical properties of specific link prediction models: in fact, besides our simple but effective Bayesian approach, more sophisticated models can be plugged in into VADA-LINK, each with specific fine tuning. Instead, here our interest is in evaluating the overall tradeoff between scalability and recall.

Varying the number of clusters. We seek to observe the impact of the number of clusters on the recall of VADA-LINK. We artificially built 10 random graphs S_i (with $1 \leq i \leq 10$) having the same number of nodes, topology and features as the real-world graph presented in Section 2. For each of them, we ran VADA-LINK in "no cluster mode", that is, we forced the system to concentrate all nodes inside one single cluster in order to produce all the theoretically possible links by means of the naive exhaustive comparison. Therefore, for each subgraph S_i we produced an augmented one $\hat{S}_i$. Then, from each $\hat{S}_i$, we randomly selected 10 edge sets Θ_{ij} (with $1 \leq j \leq 10$), each containing 20% of the predicted links and generated new subgraphs $S^{\Theta_{ij}}$ without those edges. For each $S^{\Theta_{ij}}$ we ran VADA-LINK by varying the number of clusters with 20 configurations from 1 to 500, with the technique described in Section 6.1. For each case and cluster, we obtained an augmented graph $\hat{S}^{\Theta_{ij}}$ and computed the recall R_{ijc} as the fraction of edges cardinality $|E(\hat{S}^{\Theta_{ij}})|/|E(\hat{S}_i)|$, that is, the percentage of removed edges that have been recovered. For each of the 20 clusters, we averaged the 100 computed recall R_{ij}.

Results. Figure 4(e) shows how recall decreases with the number of clusters: It is clearly maximum for the single cluster case; then, the recall obtained for 20 clusters, 99.4%, is certainly acceptable for our use cases and, in general reasonable for industrial settings in ownership graphs. We then observe 98.6% for 50 clusters and the approach becomes ineffective for more than 400 clusters, with a recall steadily under 50%. A comparison with Figure 4(c) is interesting and confirms that with more than 10 clusters, processing time is under 10 seconds. This implies that for our case, an effective balance between efficiency and recall is between 10 and 20 clusters. More generally, Figure 4(e) shows a slow decrease of recall, which proves the robustness of Vada-Link. We explain that with the recursive interaction between first- and second-level clustering we have in Algorithm 3. Whenever in a second-level cluster new links are predicted, they are used by the aggregate function #GraphEmbedClust to improve the embedding and provide better first-level grouping and, as a consequence, increase the likelihood that in the second-level clustering, #GenerateBlocks considers new candidates. In other words, the recursive interplay between the two clustering functions compensate for increases in the number of clusters and contributes to a favourable balance between scalability and recall.

7 CONCLUSION

Company ownership graphs confront us with several problems, relevant in the financial realm. Discovering company control, reasoning on asset eligibility or finding out family connections are three interesting examples we focus on in this paper, motivated by the construction of the Enterprise Knowledge Graph of Banca d'Italia. Many more such settings exist, all with the common goal of enriching the company graph with new links. The need for actionable solutions is felt by the the national central banks of the European system, the national statistical offices and many more financial authorities

In this paper, we proposed Vada-Link, a new approach for the creation of valuable links in company graphs that leverages the vast amount of domain knowledge typically present in financial realm. On the basis of recent developments in the KG research area, we model the input graph as the extensional component of a logic-defined KG, where domain knowledge is encoded in Vadalog and link prediction is operated as a reasoning task within the KG. We discussed several favourable properties of our approach, which is general and applies to different data models and schemas. We also discussed the architecture of Vada-Link and provided experimental evaluation, highlighting good performance. We believe that with Vada-Link we are shedding light on problems of the financial realm that have specific scientific relevance per se and a graph- and logic-based view on them can certainly contribute to their full characterization.

REFERENCES

[1] 2014. GUIDELINE (EU) 2011/14 OF THE ECB. https://www.ecb.europa.eu/ecb/legal/pdf/l_33120111214en000100951.pdf. [Online; accessed 15-Feb-2019].
[2] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley.
[3] Renzo Angles. 2018. The Property Graph Database Model. In *AMW.*
[4] Paul Asquith and David W. Mullins. 1986. Signalling with Dividends, Stock Repurchases, and Equity Issues. In *Financial Management.* 27–44.
[5] Paolo Atzeni, Luigi Bellomarini, Francesca Bugiotti, and Giorgio Gianforme. 2009. A runtime approach to model-independent schema and data translation. In *EDBT (ACM International Conference Proceeding Series)*, Vol. 360. ACM, 275–286.
[6] Dzmitry Bahdanau, Tom Bosc, Stanislaw Jastrzkebski, Edward Grefenstette, Pascal Vincent, and Yoshua Bengio. 2017. Learning to Compute Word Embeddings On the Fly. *CoRR* abs/1706.00286 (2017).
[7] Albert-László Barabási. 2009. Scale-Free Networks: A Decade and Beyond. *Science* 325, 5939 (2009), 412–413.
[8] Albert-Laszlo Barabasi and Reka Albert. 1999. Emergence of Scaling in Random Networks. *Science (New York, N.Y.)* 286 (11 1999), 509–12.
[9] Fabrizio Barca and Marco Becht. 2001. The Control of Corporate Europe. Oxford University Press, European Corporate Governance Network. (2001).
[10] Luigi Bellomarini, Daniele Fakhoury, Georg Gottlob, and Emanuel Sallinger. 2019. Knowledge Graphs and Enterprise AI: The Promise of an Enabling Technology. In *ICDE.* IEEE, 26–37.
[11] Luigi Bellomarini, Ruslan R. Fayzrakhmanov, Georg Gottlob, Andrey Kravchenko, Eleonora Laurenza, Yavor Nenov, Stéphane Reissfelder, Emanuel Sallinger, Evgeny Sherkhonov, and Lianlong Wu. 2018. Data Science with Vadalog: Bridging Machine Learning and Reasoning. In *MEDI (Lecture Notes in Computer Science)*, Vol. 11163. Springer, 3–21.
[12] Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. 2017. Swift Logic for Big Data and Knowledge Graphs. In *IJCAI.*
[13] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. [n.d.]. The Vadalog System (technical report). https://drive.google.com/drive/u/0/folders/0B2xKYNJdpJzQaEh5VzlhTURHU3M.
[14] Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. 2018. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *PVLDB* 11, 9 (2018), 975–987.
[15] Angela Bonifati, George H. L. Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. *Querying Graphs.* Morgan & Claypool Publishers.
[16] Luca Cabibbo. 1998. The Expressive Power of Stratified Logic Programs with Value Invention. *Inf. Comput.* 147, 1 (1998), 22–56.
[17] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. 2010. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. In *LICS.*
[18] Stefano Ceri, Georg Gottlob, and Letizia Tanca. 2012. *Logic programming and databases.* Springer.
[19] Ariane Chapelle and Ariane Szafarz. 2005. Controlling firms through the majority voting rule. *Physica A: Statistical Mechanics and its Applications* 355, 2-4 (2005), 509–529.
[20] Peter Christen. 2012. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer. I–XIX, 1–270 pages.
[21] Caroline Fohlin. 1998. Trends in Business Organization: Do Participation and Cooperation Increase Competitiveness? Edited by Horst Siebert. Tubingen: J. C. B. Mohr, 1995. Pp. viii, 292. DM 108. *The Journal of Economic History* 58, 1 (1998), 292–293.
[22] Diego Garlaschelli, Stefano Battiston, Maurizio Castri, Vito Servedio, and Guido Caldarelli. 2004. The scale-free topology of market investments. http://goo.gl/h7tVVK. http://goo.gl/h7tVVK
[23] James B Glattfelder. 2010. *Ownership networks and corporate control: mapping economic power in a globalized world.* Ph.D. Dissertation. ETH Zurich.
[24] Georg Gottlob and Andreas Pieris. 2015. Beyond SPARQL under OWL 2 QL Entailment Regime: Rules to the Rescue. In *IJCAI.* 2999–3007.
[25] Paul Graham. 2003. Better bayesian filtering. In *Proceedings of Spam Conference*, Vol. 10 (8). 707–710.
[26] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD.* ACM, 855–864.
[27] Shan Shan Huang, Todd Jeffrey Green, and Boon Thau Loo. 2011. Datalog and emerging applications: an interactive tutorial. In *SIGMOD.*
[28] Seyed Mehran Kazemi and David Poole. 2018. SimplE Embedding for Link Prediction in Knowledge Graphs. In *NeurIPS.* 4289–4300.
[29] David Liben-Nowell and Jon M. Kleinberg. 2003. The link prediction problem for social networks. In *CIKM.* ACM, 556–559.
[30] Maximilian Marx, Markus Krötzsch, and Veronika Thost. 2017. Logic on MARS: Ontologies for Generalised Property Graphs. In *IJCAI 2017.* 1188–1194.
[31] Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. 1990. The magic of duplicates and aggregates. In *VLDB.* 264–277.
[32] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, Elena Baralis, Michele Osella, and Enrico Ferro. 2018. Knowledge Graph Embeddings with node2vec for Item Recommendation. In *ESWC (Satellite Events) (Lecture Notes in Computer Science)*, Vol. 11155. Springer, 117–120.
[33] Tiago Pimentel, Adriano Veloso, and Nivio Ziviani. 2018. Fast Node Embeddings: Learning Ego-Centric Representations. In *ICLR (Workshop).* OpenReview.net.
[34] Cesar A. Hidalgo R. and Albert-László Barabási. 2008. Scale-free networks. *Scholarpedia* 3, 1 (2008), 1716.
[35] Nicoleta Rogovschi, Jun Kitazono, Nistor Grozavu, Toshiaki Omori, and Seiichi Ozawa. 2017. t-Distributed stochastic neighbor embedding spectral clustering. In *IJCNN.* IEEE, 1628–1632.
[36] Andrea Romei, Salvatore Ruggieri, and Franco Turini. 2015. The layered structure of company share networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA).* IEEE, 1–10.
[37] Alexander Shkapsky, Mohan Yang, and Carlo Zaniolo. 2015. Optimizing recursive queries with monotonic aggregates in DeALS. In *ICDE.* 867–878.
[38] Leslie G. Valiant. 1979. The complexity of enumeration and reliability problems. In *Journal on Computing*, Vol. 8. SIAM, 410–421.
[39] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. In *NeurIPS.* 5171–5181.