Reactive Company Control in Company Knowledge Graphs

Davide Magnanimi
Banca d'Italia[1] &
Politecnico di Milano
davide.magnanimi@polimi.it

Luigi Bellomarini
Banca d'Italia[1]
luigi.bellomarini@bancaditalia.it

Stefano Ceri
Politecnico di Milano
stefano.ceri@polimi.it

Davide Martinenghi
Politecnico di Milano
davide.martinenghi@polimi.it

***Abstract*—The Company Control Problem consists in understanding who exerts decision power in companies. Central banks, financial intelligence units, and market regulators are all interested in this problem, which is crucial for their core goals. In the context where these actors operate, changes in company control call for immediate reactions.**

Yet, computing control relationships is a computationally expensive problem that involves traversing the entire shareholding structure and aggregating shares over multiple paths.

In the context of the joint European banking supervision, the Bank of Italy will soon handle the shareholding graph of all European companies, which comprises hundreds of millions of entities (firms and individuals) and billions of edges and properties. This graph is highly volatile as the Bank continuously receives updates about shareholding relationships with unpredictable high frequency. This makes the straightforward bulk solution, where all the company control relationships are computed and materialized whenever a change occurs, unaffordable in practice.

In this work, we present an incremental rule-based formalization of the problem, adopting the Vadalog fragment of the Datalog+/- families of languages. Our approach analyzes the specific change, singles out the portions of the graph that are affected by it, and selectively updates them. This allows one both to timely evaluate the impact of ownership variations on an extensive European-scale shareholding graph and to enable economists to perform the so-called "what-if analysis", i.e., simulation scenarios to proactively study the consequences of potential share acquisition operations, that currently are prohibitively time expensive. We provide an extensive experimental evaluation on very large company graphs, comparatively confirming the scalability of our technique in a real production setting.

***Index Terms*—company control, Datalog, knowledge graphs, materialization maintenance, reactive reasoning**

I. Introduction

Who exerts decision power in a company? Given a national or international network of millions of entities interconnected by billions of shareholding relationships, who controls—directly or indirectly—a given company?

This question, known as the *Company Control Problem* (CCP) [1], is of great interest to banks, national central banks, financial intelligence units, regulatory and supervisory authorities, and Fintech firms. Indeed, it impacts their core business: loan granting processes, *creditworthiness evaluation* and *know-your-customer* onboarding procedures are hinged on company control, to spot any possible conflict of interest between borrowers and lenders. In *banking supervision* and *anti-money laundering*, the authorities need to identify the actual centers of power so as to accurately detect the drivers and the ultimate beneficiaries of financial misconducts. In *monetary policy* in the Eurosystem, authorities oversee the credit market by assessing the so-called *collateral eligibility* [2], that is, the independence of the collateral issuer with respect to the entities involved in monetary policy operations, in order to reduce the credit risk. Company control is a relevant topic also in the broader macroeconomic community [3, 4], where *market concentration* sparks recurring debates.

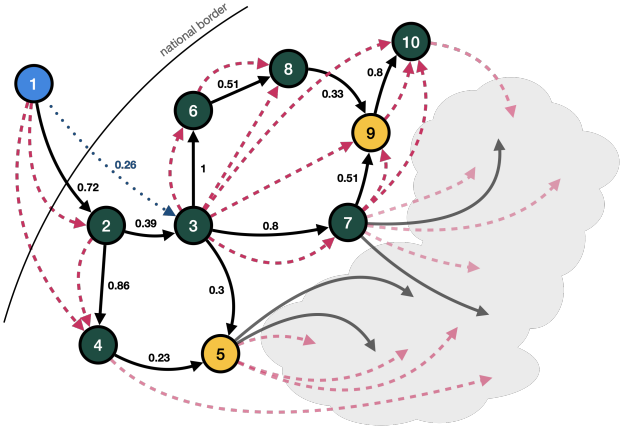


Fig. 1: An excerpt from the KG. Yellow nodes are companies with a national strategic relevance; the blue node is a non-EU shareholder. Solid edges are existing ownership relationships; dashed-magenta edges are control relationships. The dotted-blue edge represents a candidate share acquisition.

Industrial Scenario. The Bank of Italy holds a company network in the form of a *Company Knowledge Graph* (KG), stored as a *property graph* [5]. The nodes denote individuals and companies, and the edges represent shareholding relationships (or *ownerships*), with the respective percentage. Since the Bank of Italy operates as a regulator and supervisor in the European System of Central Banks, the graph, which was initially focussed on Italian companies, is being gradually expanded to span the entire EU-level space, with hundreds of millions of nodes and billions of edges and properties.

[1]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.

A core element of the graph are *control edges*, which are derived "x *controls* y" relationships computed from the shareholding edges according to the following definition [1]:

A company x controls a company y if x (i) directly owns the majority of y shares, or (ii) controls a set of companies that, possibly together with x itself, jointly hold the majority of y.

A static pre-computation of control edges is not suitable for uprising economic and financial applications, which need to capture dynamic phenomena involving frequent changes in ownership edges. A timely "reactive" computation of control edges is crucial for a host of institutional services of the Bank of Italy, which call for immediate action once a change of control is detected. Also in the analytical perspective, timeliness in updating control edges matters, e.g., when analysts need to perform interactive "*what-if analysis*" to seize the systemic effect on control, by simulating changes of ownership.

As a matter of fact, a large category of different industrial scenarios share a common trait: a very limited number of changes in the ownership edges in a very small time interval cause changes in control relationships, potentially in distant areas of the KG, to be accounted for as efficiently as possible.

Example 1. *Consider now the excerpt in Figure 1 of our KG: an analyst wants to swiftly assess the impact of the acquisition of 26% of the shares of Company* 3 *undertaken by a non-EU company* 1*. In fact, it would change the controllers of Companies* 5 *and* 9*, which are of strategic national relevance, and potentially also affect others. Company* 1 *already controls 39% of Company* 3*, and, with a further 26% acquisition would gain control of the majority. Consequently, Company* 1 *would exert control over Company* 7*, the strategic* 9*, as well as* 6*,* 8*, and* 10*, already controlled by* 3*. Finally, via the controlled Companies* 3 *and* 4*, our non-EU shareholder would take over the strategic Company* 5 *as well and, with a cascade effect, also other companies controlled by* 5*,* 7*,* 9*, and* 10*.* ■

As of now, the Bank of Italy computes the control relationships between companies in a batch fashion: daily ownership variations are collected from specific data sources, the ownership edges are updated and all the control relationships are finally recomputed. Considering the current rate of thousands of changes per day, rapidly soaring for the EU-level role of the Bank, not only is the batch approach incompatible with the mentioned on-line applications, but also causes longer and longer periods with outdated data, unacceptable in practice.

Goal. In this industrial work, we consider a declarative specification of the CCP problem in Datalog [6, 7] extended with features of practical utility such as aggregation, negation, and inequalities, and introduce a reactive formulation of it as an *inference task* that exploits changes of ownership so as to locally and efficiently update control relationships. We propose a production-ready system that provides our analysts with a *reactive Knowledge Graph*, having continuously and incrementally updated control edges. We implemented our solution in the Vadalog System [8], a state-of-the-art reasoner using the VADALOG language of the Datalog$^{\pm}$ family [9].

Contribution. This work contributes a *declarative*, *incremental*, *reasoning-based* approach to the computation of company control relationships over large and frequently updated Company Knowledge Graphs, which we name *reactive control computation*. In particular:

- We propose a **reactive logic-based formulation of the CCP**. We introduce specific predicates to reason on insertions and deletions of shareholding relationships in the graph and use them to incrementally update control edges. Our formulation leverages the topology of the graph to exploit forms of *update locality* and avoid massive cascade updates or full recomputation, like in the case of the batch approach.
- We **validate our approach in many experimental settings**. Specifically, we test reactive control computation on several time snapshots of the Italian Company KG of the Bank of Italy as well as on a large number of synthetic graphs, reproducing the real graph structure at an increasing scale. We evaluated performance along four dimensions: *size* (up to 16 times the nodes of the Italian graph); *density* (up to 3 times the ownership relationships of the Italian real graph); different *topology* of the shareholding network; and, finally, *percentage of updated ownership relationships* (from 1% to 100% of the entire graph). We show that, for the real graphs, the reactive control provides a speedup of up to 83% with respect to the batch approach, for daily changes. With synthetic graphs, our approach shows its full potential, providing the greatest advantage with a limited number of changes (e.g., in case of updates collected with daily frequency, or within "what-if analysis" settings) over a very large KG (e.g., European scale).

Overview. The remainder of the paper is organized as follows. In Section II we introduce the CCP and the needed background. Section III presents our reactive approach. The experimental evaluation is presented in Section IV. Related work is in Section V and Section VI concludes the paper.

II. THE COMPANY CONTROL PROBLEM

Before analyzing the CCP, let us first briefly present the Italian Company Knowledge Graph.

The Company Knowledge Graph. It contains detailed information about companies, individuals, corporate events, many other entities, and the relationships among them. In this work, we focus on entities and relationships regarding *ownership (of shares) relationships*, which connect subsidiaries to their shareholders (either other companies or individuals), and define an *ownership graph* $G = (V, E, L)$ as a directed, weighted graph in which V is the set of nodes representing such companies or individuals; E is the set of directed edges, with $E \subseteq V^2$, that represent shareholding relationships; and $L : E \rightarrow (0, 1]$ is a labeling function that assigns to each edge $e = (v, w) \in E$ a label $L(e)$ representing the percentage of total equity of company w held by the shareholder v. Clearly, for every node $w \in V$, we require that $\sum_{v \in pred_w} L((v, w)) \leq 1$, with $pred_w$ indicating the set of nodes $v \in V$ that are shareholders of w.

For our purposes, we model the ownership graph as a database $\mathcal{D}$ of facts $\mathsf{Own}(x, y, w)$, where x is a shareholder (company or individual), y is a company, and w is the owned share percentage. For example, the graph in Figure 1 would be encoded as $\mathcal{D} = \{\mathsf{own}(1, 2, 0.72), \mathsf{own}(2, 4, 0.86), \mathsf{own}(2, 3, 0.39), \mathsf{own}(4, 5, 0.23), \ldots\}$.

We will encode the CCP and our reactive solution to it as an *inference task*, expressed in a Datalog-based formalism, over the database of the ownership graph. Let us start with the needed preliminary concepts and background.

Relational Foundations and VADALOG. Let $\mathbf{C}$ and and $\mathbf{V}$ be disjoint countably infinite sets of *constants* and *variables*, respectively. A *(relational) schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. A *term* is either a constant or variable. An *atom* over $\mathbf{S}$ 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* over $\mathbf{S}$ associates with each relation symbol in $\mathbf{S}$ a relation of the respective arity over the domain of constants. A Datalog *rule* is a first-order sentence $\forall\bar{x}\forall\bar{y}(\varphi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}))$, where φ (the *body*) is a conjunction of atoms and ψ (the *head*) is an atom. We will omit universal quantifiers in the rest of the work. A rule having an empty body, understood as *true*, is named *fact*. A *relation* is a set of facts, and thus a database can also be regarded as a set of facts. A database $\mathcal{D}$ satisfies a Datalog rule $\varphi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x})$ if for each fact $\varphi(\bar{t}, \bar{t}')$ of $\mathcal{D}$, there exists a tuple $\bar{t}$ of constants such that the facts $\psi(\bar{t})$ are also in $\mathcal{D}$.

The semantics of a set of rules Σ over a database $\mathcal{D}$, denoted as $\Sigma(\mathcal{D})$, is a set of facts defined via the CHASE *procedure* [6]: the chase updates $\mathcal{D}$ by adding new facts, namely, "applying" rules, until the final result $\Sigma(\mathcal{D})$ satisfies all the rules of Σ. We call the process of deriving $\Sigma(\mathcal{D})$ an *inference task*. In the chase, we say that $\sigma = \varphi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x})$ is *applicable* to $\Sigma(\mathcal{D})$ if there is a unifier θ_σ such that $\varphi(\bar{x}\theta_\sigma, \bar{y}\theta_\sigma) \subseteq \Sigma(D)$ and θ_σ has not been used to generate new facts in $\Sigma(\mathcal{D})$.

The languages of the Datalog$^\pm$ family [9], technically *fragments*, extend Datalog with features of practical utility such as existential quantification to increase expressive power and, at the same time, introduce syntactic limitations for tractability reasons [10]. In this work, we will use an extended version of Datalog borrowing aggregations, negations, and inequalities from VADALOG [11], a Datalog$^\pm$ fragment.

Aggregations have been introduced in multiple logical contexts and the need for a careful definition of their semantics (procedural and model-based) arose in all of them. Among the types of aggregations VADALOG supports, in this work we adopt a simple stratified semantics [12], which is enough for our purposes. Given a rule of the form $\varphi(\bar{x}, \bar{y}), v = \mathit{aggr}(q) \rightarrow \psi(\bar{x}, v)$, where q is a variable of $\bar{x}$ and *aggr* is a generic aggregation operator, the value of variable v is computed in the chase by aggregating the values of q over the distinct groups defined by the values of $\bar{x}$.

VADALOG features *stratified negation*, according to the usual stratified semantics [13], the principle of which is operationally simple: given a negated atom "$\neg a$" appearing in a rule σ, for each logical unifier for which, not considering a, σ is applicable, the rule is applied only if such a unifier does not bind a to any fact that has already been generated.

Finally, we will adopt inequalities in the rule body as syntactic sugar for standard comparison predicates (e.g., $>$, $\geq$, $<$, $\leq$, ...) over body variables and constants. When aggregations appear in the body, we may use inequalities to directly constrain the aggregated variable v.

The Company Control Problem. We have already provided a definition of the CCP in the introduction, which is majority-based and broadly accepted in the corporate economics community [4]. Let us consolidate the definition by focusing on a portion of Figure 1. Shareholder 3 directly owns 100% and 80% of the total equity of 6 and 7, respectively. Therefore, shareholder 3 obtains the control over them, as well as over the shares they own of other companies. In particular, 3 is able to exert control on firm 8, as 6 directly controls it. Consequently, by controlling 7 and 8, the shareholder 3 also controls the sum of the shares of 9 they own. As the sum is above 50%, it allows 3 to control company 9 as well.

The CCP has been studied in the database literature and proved to be quadratic, in its pairwise decision formulation, that is, given an ownership graph and two companies x and y from this graph, decide whether a control relationship between them holds [1]. In this paper, we will focus on the more general inference task of deriving all the control relationships. Thanks to its recursive nature, an elegant VADALOG formulation of CCP can be provided as follows [1]:

$$\mathsf{company}(x) \rightarrow \mathsf{control}(x, x) \quad (1)$$

$$\mathsf{control}(x, z), \mathsf{own}(z, y, w) \wedge v = \mathit{sum}(w) \wedge v > 0.5 \rightarrow \mathsf{control}(x, y) \quad (2)$$

Given that every company x controls itself (Rule 1), by Rule 2 we have that x controls y if the sum of the shares w of y owned by companies z, over all companies z controlled by x, is above the 50% threshold.

III. REACTIVE COMPANY CONTROL

Given the company KG, a *materialization* and *batch* approach to the CCP would consist in periodically deleting all the control edges, considering the most recent version of the graph, and computing a new materialization of the relationships—unaffordable in practice, as we have discussed. Conversely, in this section we introduce an incremental solution, which "*reacts*" to changes and updates the control edges.

Towards an incremental approach, we enrich our vocabulary of VADALOG atoms with adorned versions. In detail, given a set of changes of the KG performed in the interval $[t_1, t_2]$, for an atom ϱ, we denote as *new*ϱ, an atom binding to all facts for ϱ holding after t_2; as ϱ^+, an atom binding to all the facts for ϱ that have been inserted during $[t_1, t_2]$; as ϱ^-, an atom binding to all the facts for ϱ that have been removed during $[t_1, t_2]$; finally, ϱ binds to all facts holding before t_1.

Then, in our context we will refer to the set of ownerships holding before (own) and after the update (newOwn); to those

inserted (own^+) and removed (own^-) by the update; to the old materialization of the control relationship ($\mathsf{control}$) and to the facts to be inserted ($\mathsf{control}^+$) or deleted ($\mathsf{control}^-$) to materialize the new one ($\mathsf{newControl}$). For the sake of simplicity, we will refer to a generic update (interval) and will not explicitly refer to $[t_1, t_2]$ in the predicate syntax. We model modifications to share amounts as a deletion of the respective ownership fact followed by a re-insertion. Inserted or deleted nodes are represented by adding (own^+) resp. removing (own^-) all the ownership facts they participate in.

Our framework models the scenario at hand as an instance of the *materialization maintenance* problem [14, 15, 16, 17], in particular, as the task of maintaining the materialization of $\Sigma(\mathcal{D})$, where Σ is the company control program and $\mathcal{D}$ is the database encoding the KG. We adopt a *Delete/Rederive* (DRed) [18] strategy to handle deletions: we first delete the affected control facts, identifying them with an *overstimation/refinement* process. Then, we apply the insertions and extend $\mathcal{D}$ with all the consequences of the inserted facts.

More precisely, given $\mathcal{D}$ and a set of updates occurred during $[t_1, t_2]$ that can be referred to via adorned predicates, the materialization maintenance task is solved by two sets of VADALOG rules Σ_D and Σ_I with two associated inference tasks that, respectively, infer the sets $\mathbf{C}^- = \Sigma_D(\mathcal{D})$ of control facts to be deleted from $\mathcal{D}$, and the set $\mathbf{C}^+ = \Sigma_I(\mathcal{D} \setminus \mathbf{C}^-)$ to be added to $\mathcal{D} \setminus \mathbf{C}^-$. In total, $(\mathcal{D} \setminus \mathbf{C}^-) \cup \mathbf{C}^+$ is the updated version of $\Sigma(\mathcal{D})$.

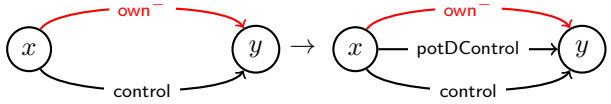
We describe rules of Σ_D in Section III-A and those of Σ_I in Section III-B. We conclude with a discussion, in Section III-C.

A. Deletion rules (Σ_D)

The rules of Σ_D are conceptually organized in three groups that (i) identify the facts that are potentially deleted as a consequence of ownership changes; (ii) for each of those facts, try to identify alternative derivation paths; (iii) single out the facts deleted by group (i) and not reinserted by group (ii).

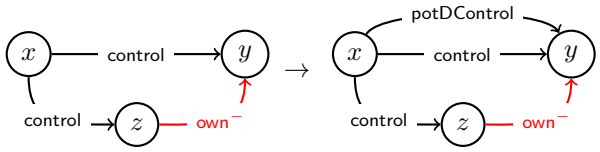
Group 1 - Potentially deleted controls. The rules of this group identify, by overestimation, the control facts $\mathsf{potDControl}$ that are potentially deleted as a consequence of ownership changes. This deletion event can take place in three scenarios, that we see next. For each scenario, we present the rationale, the VADALOG rules, and an explanatory snippet. Each snippet offers a visual representation of the rule application, where deleted facts are denoted by red edges.

- **Deletion of a direct ownership.** The deletion of a direct ownership $x \to y$ reduces the total shares of y controlled by x. Consequently, a control relationship from x to y may not hold anymore, as denoted by a new $\mathsf{potDControl}$ fact.



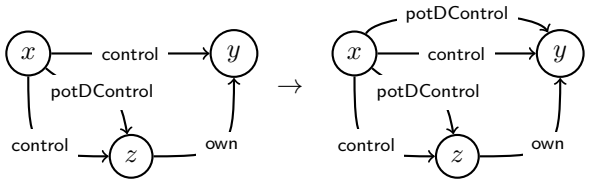
$$\mathsf{own}^-(x, y, q) \wedge \mathsf{control}(x, y) \to \mathsf{potDControl}(x, y) \quad (3)$$

- **Deletion of an indirect ownership.** Let x control y via direct and indirect undertakings, e.g., via z. If the indirect undertaking is removed, x may lose its control on y.



$$\mathsf{control}(x, z) \wedge \mathsf{own}^-(z, y, q) \wedge \mathsf{control}(x, y) \to \mathsf{potDControl}(x, y) \quad (4)$$

- **Recursive propagation.** If x controls y also via a controlled company z retaining shares of y, a potential deletion of the control from x to z could possibly cause x to lose its control on y, recursively.



$$\mathsf{potDControl}(x, z) \wedge \mathsf{own}(z, y) \wedge \mathsf{control}(x, y) \to \mathsf{potDControl}(x, y) \quad (5)$$

Group 2 - Recomputing affected controlled shares. The rules of this group seek for alternative derivations of the control relationships affected by ownership deletions.

In this group, we recompute direct and indirect control shares only for the pairs of nodes interested by a $\mathsf{potDControl}$ relationship, hence minimizing the portion of the traversed graph. To find alternative derivations, given a potentially deleted control, we collect all the shares q of y controlled by x via z into the predicate $\mathsf{potDControlShare(x, z, y, q)}$, so that they are eventually aggregated. Rules 6 and 7 together represent the base case of this aggregation; Rules 8 and 9, which are mutually recursive, represent the inductive case.

To start, we need to introduce an auxiliary predicate to consider only the ownerships that have not been deleted, i.e.: $\mathsf{own}(x, y, q) \wedge \neg \mathsf{own}^-(x, y, q) \to \mathsf{stillOwn}(x, y, q)$.

$$\mathsf{potDControl}(x, y) \wedge \mathsf{stillOwn}(x, y, q) \to \mathsf{potDControlShare}(x, y, y, q) \quad (6)$$

With Rule 7, we want to collect the shares of indirect controls. We consider the shares q for the pairs x and y connected by a potentially deleted control, in the case x certainly controls some other node z, i.e., $\mathsf{potDControl}(x, z)$ does not hold, and z still retains q shares of y.

$$\mathsf{potDControl}(x, y) \wedge \mathsf{stillOwn}(z, y, q) \wedge \mathsf{control}(x, z) \wedge \neg\ \mathsf{potDControl}(x, z) \to \mathsf{potDControlShare}(x, z, y, q) \quad (7)$$

The goal of Rules 8 and 9 is to collect the pairs for which while potentially the control may have been deleted, in fact, it is not, as alternative derivations do exist. Rule 8 creates potStillControl facts, which witness that a control of x on y still exists if the sum of the newly derived controlled shares (those computed by Rules 6, 7, and 9) is above 0.5.

$$\mathsf{potDControlShare}(x,z,y,q) \wedge j = \mathit{sum}(q) \wedge j > 0.5 \rightarrow \mathsf{potStillControl}(x,y) \quad (8)$$

In Rule 9 we collect the shares that x controls of y via z, in the case the potentially deleted control between x and z instead still holds (as witnessed by potStillControl facts produced by Rule 8) as well as the ownership between z and y.

$$\mathsf{potDControl}(x,y) \wedge \mathsf{potStillControl}(x,z) \wedge \mathsf{stillOwn}(z,y,q) \rightarrow \mathsf{potDControlShare}(x,z,y,q) \quad (9)$$

Group 3 - Computing $\mathbf{C}^-$. We are now ready to single out the control facts to be removed in the absence of alternative derivations (Rule 10).

$$\mathsf{potDControl}(x,y) \wedge \neg\, \mathsf{potStillControl}(x,y) \rightarrow \mathsf{control}^-(x,y) \quad (10)$$

Example 2. *Let us consider the example in Figure 1 again and suppose Company 6 sells all its shares of 8, as captured by the fact* $\mathsf{own}^-(6,8,0.51)$. *In Group 1, we obtain* $\mathsf{potDControl}(6,8)$ *and* $\mathsf{potDControl}(3,8)$ *by Rules 3 and 4 respectively;* $\mathsf{potDControl}(3,9)$ *and* $\mathsf{potDControl}(3,10)$ *by Rule 5, respectively. When recomputing the affected control shares, Group 2 produces only* $\mathsf{potDControlShare}(3,7,9,0.51)$ *(Rule 7) and, therefore,* $\mathsf{potStillControl}(3,9)$ *(Rule 8). Company 3 still control 9 as the total controlled shares is above 50%. Rule 9 derives* $\mathsf{potDControlShare}(3,9,10,0.8)$. *Again, another still holding control is derived by Rule 8, i.e.,* $\mathsf{potStillControl}(3,10)$. *Note that the control facts of Company 3 on 8 and 6 on 8 do not exist any longer. Finally, Rule 10 produces the deleted controls* $\mathsf{control}^-(6,8)$ *and* $\mathsf{control}^-(3,8)$. ■

Applying $\mathbf{C}^-$. Given the outcome of the inference task $\Sigma_D(\mathcal{D})$, Rule 11 removes the deleted control facts $\mathbf{C}^-$ from $\mathcal{D}$. Note that we are actually performing a "virtual" deletion by producing facts for stillControl by set difference.

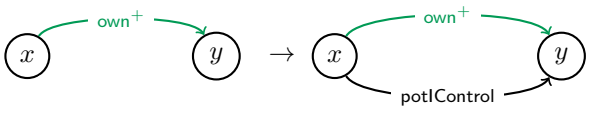
$$\mathsf{control}(x,y) \wedge \neg\, \mathsf{control}^-(x,y) \rightarrow \mathsf{stillControl}(x,y) \quad (11)$$

B. Insertion rules (Σ_I)

The rules of Σ_I are organized in three groups that (i) identify the control facts that are potentially added due to new ownerships; (ii) for each of those new control candidates, compute the respective control shares; (iii) based on these shares, single out the actual new control facts.

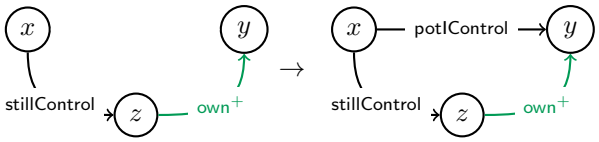
Group 1 - Potentially inserted controls. The rules of this group identify three scenarios in which it is possible to derive by overestimation new control facts potIControl whenever new ownership relationships (own^+) or, recursively, new controls ($\mathsf{control}^+$) are added. Let us introduce the scenarios; we will use green edges to denote the added facts.

- **Insertion of a direct ownership.** The addition of a new ownership from x to y contributes to the total shares of y controlled by x and a new control may arise.



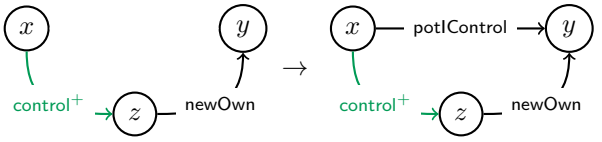
$$\mathsf{own}^+(x,y,q) \wedge \neg\, \mathsf{stillControl}(x,y) \rightarrow \mathsf{potIControl}(x,y) \quad (12)$$

- **Insertion of an indirect ownership.** Let z be a company controlled by x. The addition of a new ownership from z to y contributes to the total shares of y that x controls, directly or indirectly, possibly resulting in a new control.



$$\mathsf{stillControl}(x,z) \wedge \mathsf{own}^+(z,y,q) \wedge \neg\, \mathsf{stillControl}(x,y) \rightarrow \mathsf{potIControl}(x,y) \quad (13)$$

- **Recursive propagation.** Let z own shares of y. A new control ($\mathsf{control}^+$) from x to z may contribute to the total amount of shares of y, directly and indirectly, controlled by x. Recursively, this may lead to a new control from x to y.



$$\mathsf{control}^+(x,z) \wedge \mathsf{newOwn}(z,y,q), \wedge \neg\, \mathsf{stillControl}(x,y) \rightarrow \mathsf{potIControl}(x,y) \quad (14)$$

Group 2 - Computation of new controlled shares. In this group, we collect all the direct (Rule 15) and indirect (Rule 16, 17) controlled shares in $\mathcal{D} \setminus \mathbf{C}^-$ between the pairs singled out by potIControl facts.

$$\mathsf{potIControl}(x,y) \wedge \mathsf{newOwn}(x,y,q) \rightarrow \mathsf{potIControlShare}(x,y,y,q) \quad (15)$$

$$\mathsf{potIControl}(x,y) \wedge \mathsf{stillControl}(x,z) \wedge \mathsf{newOwn}(z,y,q) \rightarrow \mathsf{potIControlShare}(x,z,y,q) \quad (16)$$

$$\mathsf{potIControl}(x,y) \wedge \mathsf{control}^+(x,z) \wedge \mathsf{newOwn}(z,y,q) \rightarrow \mathsf{potIControlShare}(x,z,y,q) \quad (17)$$

Group 3 - Computing $\mathbf{C}^+$. Finally, the new controls ($\mathsf{control}^+$) are derived by summarizing all the newly computed controlled shares and selecting only those above 0.5.

$$\mathsf{potIControlShare}(x, z, y, q) \wedge j = \mathit{sum}(q) \wedge j > 0.5 \rightarrow \mathsf{control}^+(x, y) \quad (18)$$

Note that the new controls are in turn fed into Rule 14 and 17.

Example 3. *Let us suppose the candidate share acquisition operation in Figure 1 is settled and the reactive approach is employed for updating controls. The fact* $\mathsf{own}^+(1, 3, 0.26)$ *captures the update. We obtain* $\mathsf{potIControl}(1, 3)$ *by Rule 12, and* $\mathsf{potIControlShare}(1, 3, 3, 0.26)$ *and* $\mathsf{potIControlShare}(1, 2, 3, 0.39)$ *by Rules 15 and 16, respectively. As the sum of the two computed controlled shares is above 50%, Rule 18 yields* $\mathsf{control}^+(1, 3)$*. Then, by Rule 14 we generate* $\mathsf{potIControl}(1, 5)$*,* $\mathsf{potIControl}(1, 6)$*, and* $\mathsf{potIControl}(1, 7)$*. By evaluating all the direct and indirect contributions in the recursive activation of rules, we finally also obtain* $\mathsf{control}^+(1, 5)$*,* $\mathsf{control}^+(1, 6)$*,* $\mathsf{control}^+(1, 8)$*,* $\mathsf{control}^+(1, 7)$*,* $\mathsf{control}^+(1, 9)$*, and* $\mathsf{control}^+(1, 10)$*.* ■

Applying $\mathbf{C}^+$. Once the inference task $\Sigma_I(\mathcal{D} \setminus \mathbf{C}^-)$ has been completed, the obtained control edges $\mathbf{C}^+$ are added to $\mathcal{D} \setminus \mathbf{C}^-$ with Rules 19 and 20. Also in this case, we opt for a "virtual" insertion with a new control predicate (newControl).

$$\mathsf{stillControl}(x, y) \rightarrow \mathsf{newControl}(x, y) \quad (19)$$

$$\mathsf{control}^+(x, y) \rightarrow \mathsf{newControl}(x, y) \quad (20)$$

C. Discussion

We now address a few interesting points, whose full technical discussion would be too involved and beyond the scope of this work. Here, our goal is pointing out the specific choices that have been taken in this industrial application with respect to the possibilities offered by the extensive experience of the database literature on materialization maintenance and Datalog-based reasoning.

Runtime and optimization. We execute the rules of Σ_D and Σ_I in the VADALOG System. It offers *chase-based query-driven* reasoning with *forward propagation*. In particular, the system relies on *isomorphism chase* [19] enhanced with *semi-naïve* evaluation [6, Ch.13], particularly suitable to handle insertions [18]. To support deletions, DRed algorithms search for alternative derivations of deleted facts. This is also the case of more advanced *B/F* (backward-forward) approaches [15], that even avoid overdeletions, by interleaving backward and forward steps and continuously searching for other derivations. In this work, we purely submit our inference tasks to the VADALOG runtime, and do not develop a custom DRed or B/F algorithm for materialization maintenance, as our choice is towards a production-ready Datalog reasoner. Hence, we do not rely on smart combinations of DRed and B/F with derivation counting strategies [17], nor on a native search of alternative derivations, which would have been laborious to implement in a forward chaining system. However, to avoid potential inefficiencies, we resort to *magic sets* [6] and adorn our rules with high selectivity atoms (e.g., stillOwn, etc.) that restrict the derivations to the precomputed pairs of nodes affected by ownership changes.

Correctness and complexity. The correctness of our technique can be shown in a straightforward but boring way by proving the equivalence $\Sigma(\mathcal{D}) = (\mathcal{D} \setminus \mathbf{C}^-) \cup \mathbf{C}^+$, for a set of changes of ownership applied to $\mathcal{D}$. The data complexity of the incremental updates is polynomial; more precisely, we have a quadratic worst-case upper bound. Intuitively, when all ownership facts are updated, our incremental technique degenerates into a constant number of instances of CCP, which is in turn quadratic [1].

Stratification. Let Σ_D^A be a set containing Rule 11, which applies $\mathbf{C}^-$ to $\mathcal{D}$, and let Σ_I^A contain Rules 19 and 20, generating the new controls. The program $M = \Sigma_D \cup \Sigma_D^A \cup \Sigma_I \cup \Sigma_I^A$ is such that the extension of newControl in M is the updated materialization of the control relation in $\mathcal{D}$. We designed M in such a way it is decomposable into the sequence of strata $\langle \Sigma_D, \Sigma_D^A, \Sigma_I, \Sigma_I^A \rangle$, so that the i-th stratum only depends on the i-1-th. Moreover, Σ_D can be further decomposed into a chain of strata. This offers the chance to interleave multiple materialization points in the processing as well as supporting different storage back-ends that either feature explicit insertion/deletion primitives (e.g., INSERT/DELETE in relational databases) or require to replace physical structures with their updated versions, like in the case of filesystem-based materialization.

IV. EXPERIMENTAL EVALUATION

We have evaluated our reactive approach empirically in multiple settings. We tested the performance on real scenarios with data from the Italian Company KG as well as on synthetic graphs that push realistic topologies to extreme scale. We also evaluated challenging topologies and densities. In Section IV-A we describe the adopted datasets. Experiments on real-world data are in Section IV-B. The synthetic scenarios are presented in Section IV-C. A discussion of the relative impact of deletions and insertions is in Section IV-D. Finally, Section IV-E pursues a real case where our reactive approach is applied to "what-if analysis" in both real and synthetic graphs.

Configuration. The rules presented in Section III have been executed in the VADALOG System. We conducted all experiments on a cloud-hosted virtual machine with 20 cores (AMD EPYC™ architecture) and 160GB of RAM.

Data organization. The input data (i.e., own, newOwn, own^+, own^- and control) are stored in the file system. Each experimental evaluation concludes with the materialization of the output data (i.e., newControl, $\mathsf{control}^+$ and $\mathsf{control}^-$).

A. Input datasets

The Italian Company KG. The graph is characterized by a large weakly connected component. Its structure, often referred to as "the lung graph" [20] or "bow-tie" [21, 22, 23], is characterized by 8.589M nodes and 7.749M edges, revolving around

	# of updates		# of Recomputed Control Shares		
Interval	Own$^-$	Own$^+$	Baseline	Reactive	Improv. (%)
Reference	-	-	-	-	-
1-day	0.3K	0.9K	8.06M	1.9K	99.8%
7-days	13.5K	14K	8.07M	16K	99.8%
30-days	83.1K	84K	8.08M	99K	99.7%
90-days	193K	234K	8.1M	276K	96.5%
180-days	332K	418K	8.18M	495K	93.8%
365-days	542K	690K	8.25M	813K	90.14%

Fig. 8: Changes of ownership with respect to the reference snapshot (left side); number of recomputed control shares when updating the reference materialization for both the baseline and the reactive approach (right side).

	Values
Size	1M, 2M, 4M, 8M, 16M, 32M, 64M, **128M**
Topology	***scale-free***, *small-world*, *random*
Density	***sparse***, *normal*, *dense*
Updates	**1%**, 2%, 4%, 8%, 16%, 32%, 64%, 82%, 100%

Fig. 9: The values explored along the four dimensions of the synthetic scenarios. The default values used to visualize the results of the experiments are in bold.

a small number of nodes having a high out-degree value, while the rest of the graph is quite fragmented and exhibits a relevant number of nodes in small components with low out-degree. In fact, there are 1.657M Weakly Connected Components (WCC) with 5 nodes each, on average, while the largest WCC has more than 2.263M nodes. The graph contains 8.587M small Strongly Connected Components (SCC). The average in-degree, i.e., number of shareholders, is ≈ 2.41 (ignoring nodes with no incoming edges); the average out-degree, i.e., number of subsidiary companies, is ≈ 1.3 (ignoring nodes with no outgoing edges). Moreover, it is interesting to observe that the maximum in-degree and out-degree are approximately 4K and 1.1K, respectively. As confirmed in the corporate economics literature [24], in terms of topology, the Italian graph is a *scale-free network*, in which the degree distribution follows a *power-law* distribution, with few nodes acting as hubs.

In order to run our experimental scenarios with the real variations of the ownership graph, we considered a snapshot of the Italian Company KG at the beginning of 2021—our *reference* snapshot; then, we collected 6 more snapshots one day, one week, one month, one quarter, half-a-year, one year after the reference one. To build the adorned version of the atoms (i.e., own$^-$ and own$^+$), as described in Section III, we compared each snapshot to the reference one and identified the respective changes. They are reported in Figure 8.

Synthetic datasets. To stress our reactive approach, we built a number of synthetic graphs, by varying *number of updates*, *size*, *density*, and *topology*. The shareholding percentages have been generated by sampling from a Beta distribution with parameters fitted from the real-world graph.

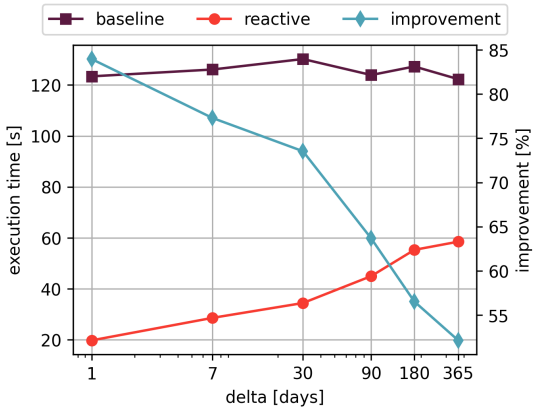


Fig. 10: Execution times of the baseline and the reactive approaches to update the reference materialization after days of ownership variations. The blue line represents the relative improvement on the execution time of the reactive approach over the baseline. The x-axis is on a logarithmic scale.

Size. We generated graphs by exponentially increasing their size from 1M up to 128M nodes, with a total of **8 different sizes**. As a benchmark, the largest generated graph is approximately 16 times larger than the Italian Company KG.

Topology. We adopted **3 graph topologies**: *scale-free* [25], with an adaptation of the Barabási–Albert model [25] for directed graphs; graphs based on the Watts-Strogatz model [26] for *small-world*; and the *random* directed graphs [27]. In particular, the scale-free topology mimics the structure of the Italian KG. The small-world topology is characterised by a high clustering coefficient that we use to simulate the presence of many corporate groups. Finally, the random graph enables a generic simulation, where no specific topology emerges.

Density. By adjusting the parameters of the generative models, we concentrated on graphs of 3 density levels: *sparse*, *normal*, and *dense*, which correspond to an average out-degree of approximately 1, 2, and 3, respectively. For reference, the average out-degree of the real Italian Company KG is ≈ 1.3.

Updates. For each synthetic graph, we randomly chose ownership edges and simulated updates as removals/reinsertions, by generating the corresponding facts for own$^-$ and own$^+$. To assess how our technique responds to more or less pervasive updates, we considered **9 levels of ownership update size** and accordingly generated the facts for own$^-$ and own$^+$. The adopted levels range from 1% to 100% of existing ownership edges, hence the last variation simulates full recomputation of the initial graph. Insertions and deletions of nodes are modeled just in terms of the insertions/deletions of the ownership relationships they are involved in.

In summary, we generated 72 (i.e., $8 \times 3 \times 3$) synthetic graphs and applied 9 levels of variations to each of them, for a total of 648 scenarios. A summary of the values used to generate the synthetic scenarios is shown in Figure 9.

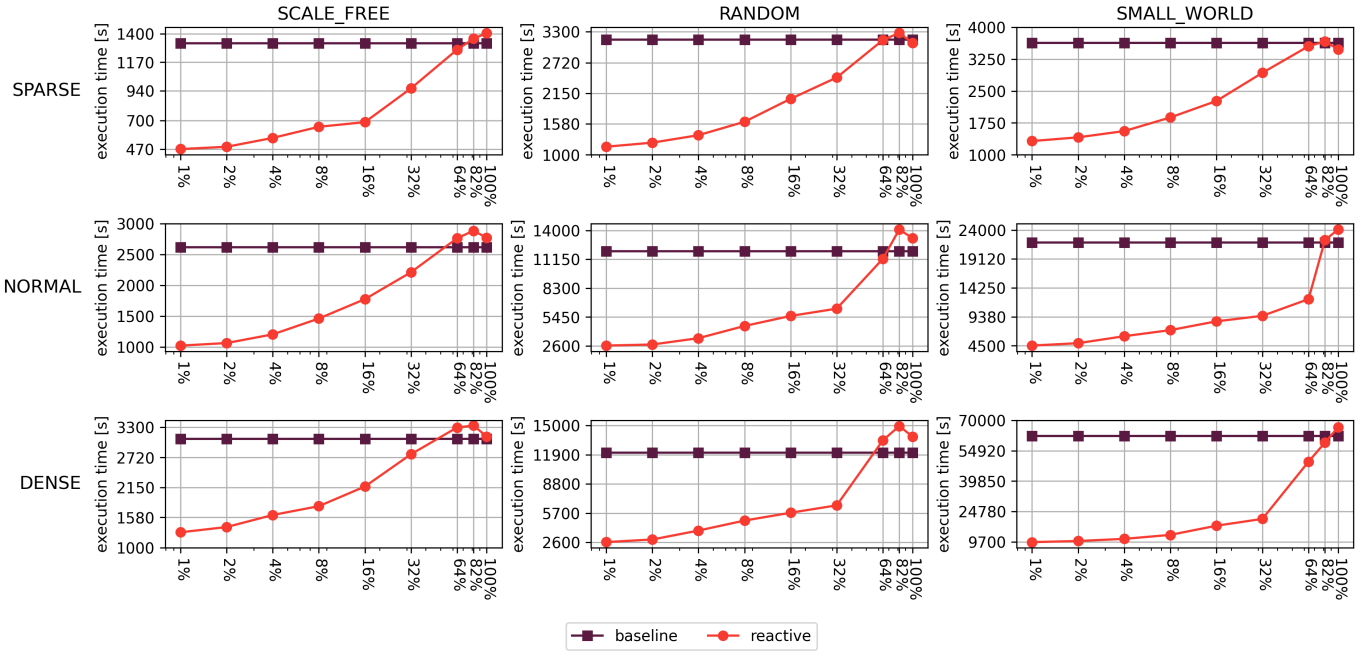


Fig. 11: Comparison of the execution times of the baseline and reactive approaches in deriving the new materialization by increasing the percentage of ownerships updates to process w.r.t. the total ones (x-axis in each plot) and considering different densities (rows of the plot matrix), and topologies (columns of the plot matrix). Results refer to graphs with a size of 128M nodes. The x-axes are on a logarithmic scale.

B. Experiments on the Italian Company KG

Execution times for updating the materialization. We measured the execution time to materialize the control relationships. We compared the baseline approach (full recomputation) with our reactive approach along our 6 time snapshots and reported the relative improvement. We repeated each run 5 times, averaging the elapsed times (in Figure 10).

The baseline approach takes nearly constant time to complete the task, independently of the snapshot span. We observe that, although the number of updates heuristically grows with the time span of the snapshot (Figure 8), with $\approx$ 12% of the ownership relationships updated after one year, the graph size remains almost stable. Conversely, the reactive approach shows a logarithmic trend that depends on the size of the time window in which changes are collected, thus exhibiting a significant reduction in the execution times. In particular, if the incremental update is executed daily, it is 83% faster than full recomputation; such an improvement logarithmically decreases to 53% for yearly updates.

Number of recomputed control shares. We also compared the two approaches in terms of how many single control shares they had to recompute to update the reference materialization. This indicator is crucial in the CCP, since it heuristically measures the impact of local changes on larger portions of the graph as a consequence of the interplay of aggregation and recursion. The results are reported in Figure 8.

We observe that while the baseline recomputed all the controlled shares ($\approx$ 8M), the reactive approach is faster as it considers way fewer control shares (almost four orders of magnitude in the best case). The overall improvement is considerable and ranges from $\approx$ 90% in the case of yearly shareholdings updates to $\approx$ 99.8% for the daily ones.

C. Experiments on synthetic scenarios

We compared the execution time of the full recomputation (baseline) with that of the reactive approach, in the described scenarios, i.e., varying the *number of updates*, *size*, *density*, and *topology*. We repeated the executions 5 times, for a total of 3240 runs, and averaged the results.

Impact of the number of ownership updates. The results are reported in Figure 11, where for visualization purposes, we set the graph size to 128M nodes.

Unlike the quasi-constant behaviour of the baseline, the reactive approach grows with a logarithmic or linear (depending on the topologies and densities) trend with the number of updates. The advantage of the approach decreases as the number of updates grows, to the point it becomes no longer convenient (we call it *crossover threshold*). In the cases of *small-world* and *random* topologies with *normal* and *dense* densities, the improvement of the reactive approach decreases more slowly than in the other scenarios, but plunges when the majority of ownerships have been updated. For such cases, the crossover threshold with the baseline is between $\approx$ 64% and $\approx$ 82%. The *small-world* settings allow us to investigate the main performance determinant: whenever the updates affect many and densely-connected corporate groups

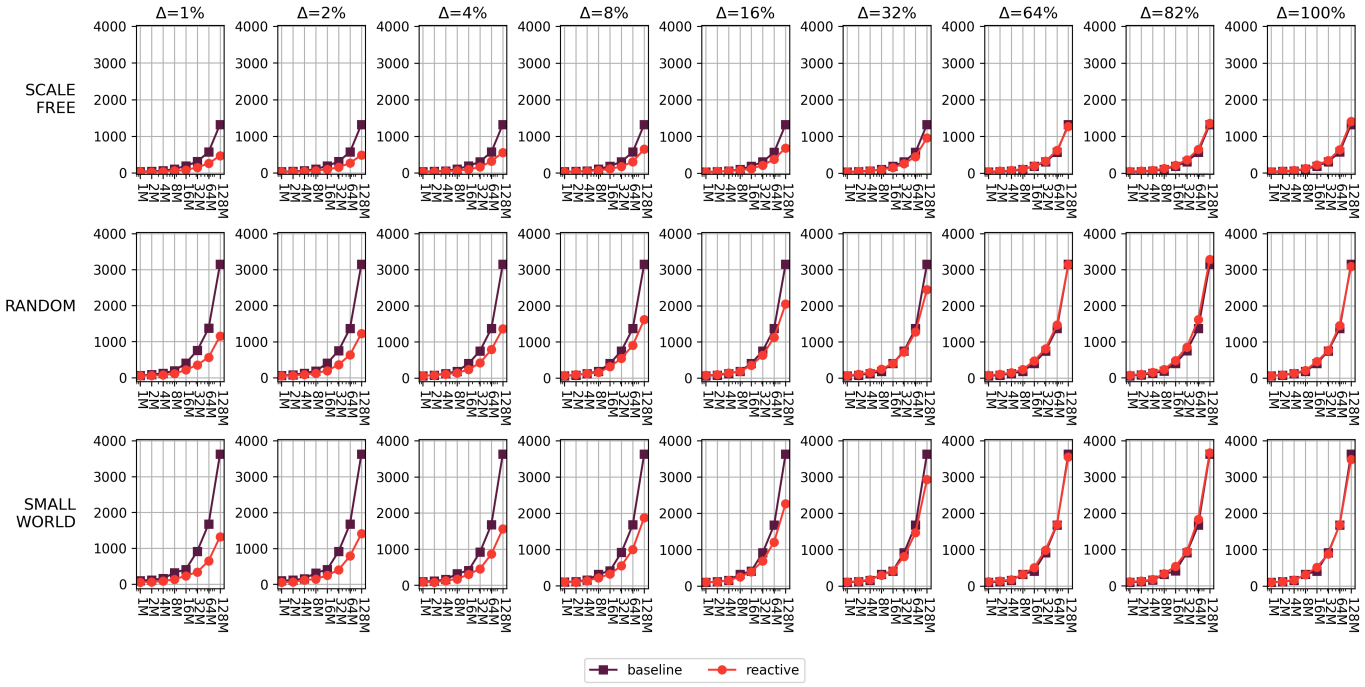


Fig. 12: Comparison of the execution times of the baseline and reactive approaches in deriving the new materialization as the size (x-axis of each plot), topology (rows of the plot matrix), and percentage of the updated ownerships w.r.t. the total ones (columns of the plot matrix) change. Results refer to graphs with a *sparse* density level. The x-axes are on a logarithmic scale.

the speed-up decreases, as the changes trigger many recursive steps. The *scale-free* topology is instead characterised by a smooth logarithmic growing trend for all the graph densities and the crossover point occurs between $\approx 32\%$ and $\approx 64\%$.

It is interesting to point out that in most cases, when updates impact more than 82% of the ownership edges, the linearly or logarithmically growing trend of execution times for the reactive approach is interrupted and times start to decrease. We motivate this behaviour as follows: the dominating cost in incremental recomputation depends on the recursive cases of deletions (i.e., Rule 5), which become less and less relevant with respect to insertions, when almost the entire graph is updated, as we shall see when discussing Figure 14.

Impact of the graph size. The results are reported in Figure 12, where we fix the *sparse* graph density. Both approaches show an increasing linear trend as the size of the graph grows. It is however evident that, if the percentage of updated ownerships is below $\approx 64\%$, the execution time of the reactive approach grows more slowly than the baseline. In that sweet spot, not only is the reactive approach always more convenient, but the improvement increases more and more as the size of the graph increases. However, if the updates regard more than $\approx 64\%$ of the ownerships, the reactive approach is not helpful any longer and, as the size of the graph increases, the disadvantage becomes more and more considerable. All the analyzed topologies show this behaviour.

Impact of the graph density. The results are in Figure 13, where we fixed 1% as the default percentage of updates.

We observe that for the *scale-free* and *small-world* topologies, higher density implies higher execution times; moreover, the increase in density improves the advantage of the reactive approach over the baseline. This behaviour is different in the case of *random* topology, where reactive performs better on graphs with *normal* density rather than on *dense* ones. In fact, in random graphs controls edges tend to be scattered throughout the graph as a result of the uniform distribution of ownership edges on which they depend. As a consequence of such a distribution, the lower execution times depend on the limited activations of the recursive steps.

Finally, in the case of larger synthetic graphs and for *random* and *small-world* topologies, the reactive approach completely outperforms the baseline, whose execution times soar dramatically. In fact, the high data volumes needed in such cases for the full recomputation require the reasoner to perform expensive swap-out/swap-in operations. It is worth mentioning that this limit is certainly dependent on the memory configuration, but poses a general scalability limitation to the baseline approach, entirely mitigated by reactive.

Impact of the graph topology. As shown in Figures 11, 12, and 13, the graph topology significantly affects the execution times, however, without substantial differences between the baseline and reactive approaches.

As expected, the *small-world* topology is the most challenging for the CCP, as the presence of dense and articulated clusters (i.e., business groups) implies the creation of complex control structures within the clusters. On the other hand, the

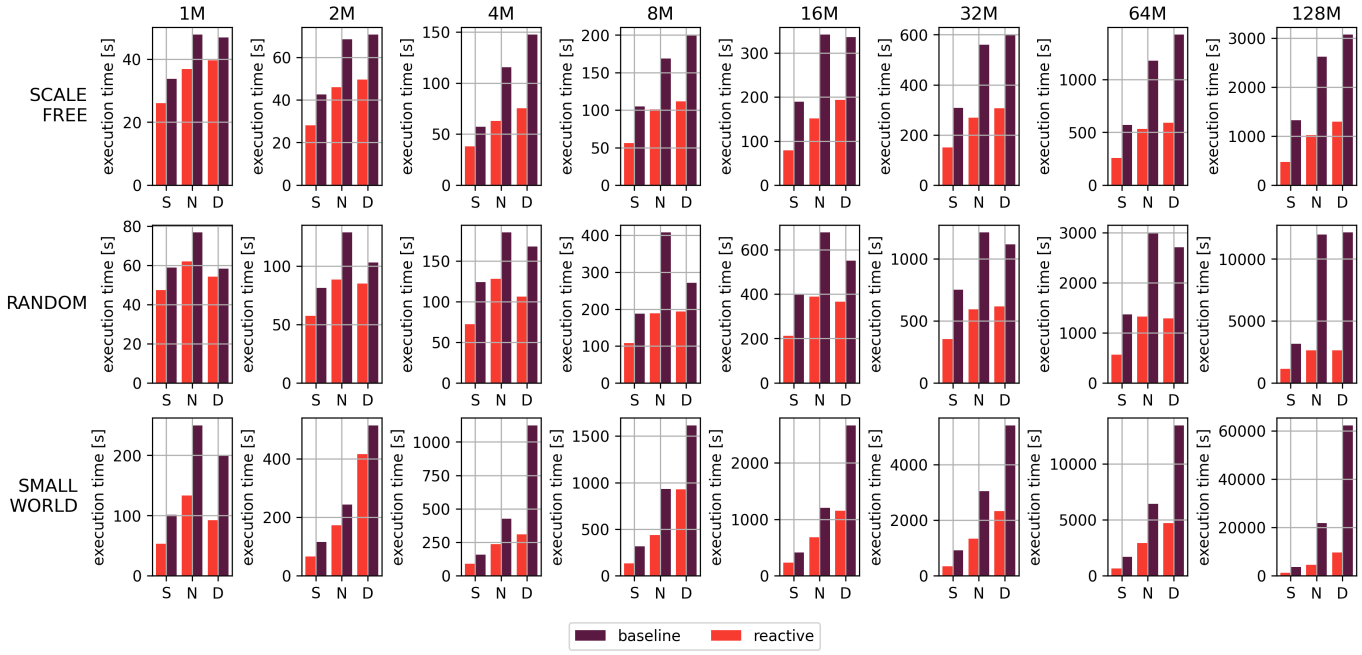


Fig. 13: Comparison of the execution times of the baseline and reactive approaches in deriving the new materialization for different combinations of densities (x-axis of each plot), topologies (rows of the plot matrix), and sizes (columns of the plot matrix) of the synthetic graphs. Results refer to the update of 1% of the total ownership relationships.

scale-free topology shows the lowest execution times, as the presence of many hubs leads to the concentration of control chains in a few single nodes, limiting the recursion depth.

D. Impact of deletions vs. insertions

The cost of updating control materialization accounts for applying both deletions (Σ_D) and insertions (Σ_I), as we have discussed at length. What is the relative impact of these operations as the number of updates grows? In this section, we provide some elements to analyze this breakdown with respect to the times shown in Figure 11.

Specifically, we evaluated the elapsed time for $\Sigma_D(\mathcal{D})$ and $\Sigma_I(\mathcal{D} \setminus \mathbf{C}^-)$ varying the percentage of updated ownerships in the usual topologies and densities. To guarantee a fair comparison, we built the case in such a way that the volume of deletions (the number of facts for own^-) is the same as the one of the insertions (own^+). Results are reported in in Figure 14, where the graph size is fixed to 128M nodes.

We observe that with a low number of updates (less than $\approx 16\%$, on average), the deletion time dominates the insertion time. In fact, the recursive rules in Groups 1 and 2 of Σ_D that invalidate control facts motivate this behaviour. For the case of many updates (more than $\approx 82\%$), insertions significantly dominate deletions to the point that deleting all ownerships leads to the same recomputation time as deleting 1% of them, or even less for *small-world* topologies. Instead, the growing logarithmic trend of insertion does not reverse.

The progressively lower contribution of the deletions on the total elapsed times in the case of many updates is technically motivated by two factors. First, once the recursive invalidation of control edges (witnessed by potDControl facts) propagates to the most dense connected components, any further deletion will impact more and more marginal portions of the graph, thus with a limited subsequent propagation. Second, if many ownership facts are deleted, then the base cases of recursion (Rules 3 and 4) are sufficient to generate the potDControl facts, in such a way as to reduce the activation of the more expensive inductive case (Rule 5).

E. Simulating "what-if analysis"

To put our framework into action, we simulated a form of "what-if analysis", in which the user applies very few changes and is interested in swiftly evaluating the full range of impacts they have on the graph. To this aim, we randomly updated 20 ownership edges, captured as deletions followed by insertions, as usual, and measured the time needed by the baseline and the reactive approaches to materialize control. We considered the Italian Company KG as well as 3 synthetic graphs, with the discussed topologies, and composed of 128M nodes and *sparse* density. The results are shown in Figure 15.

This final example points out how in the presence of a scenario characterized by few changes, as realistic in most of the on-line and interactive use cases, the reactive approach outperforms batch execution, here by an order of magnitude, with an overall $\approx 90\%$ speed-up.

V. Related Work

Related literature about the CCP is interesting and multi-disciplinary. We briefly provide some references from inter-

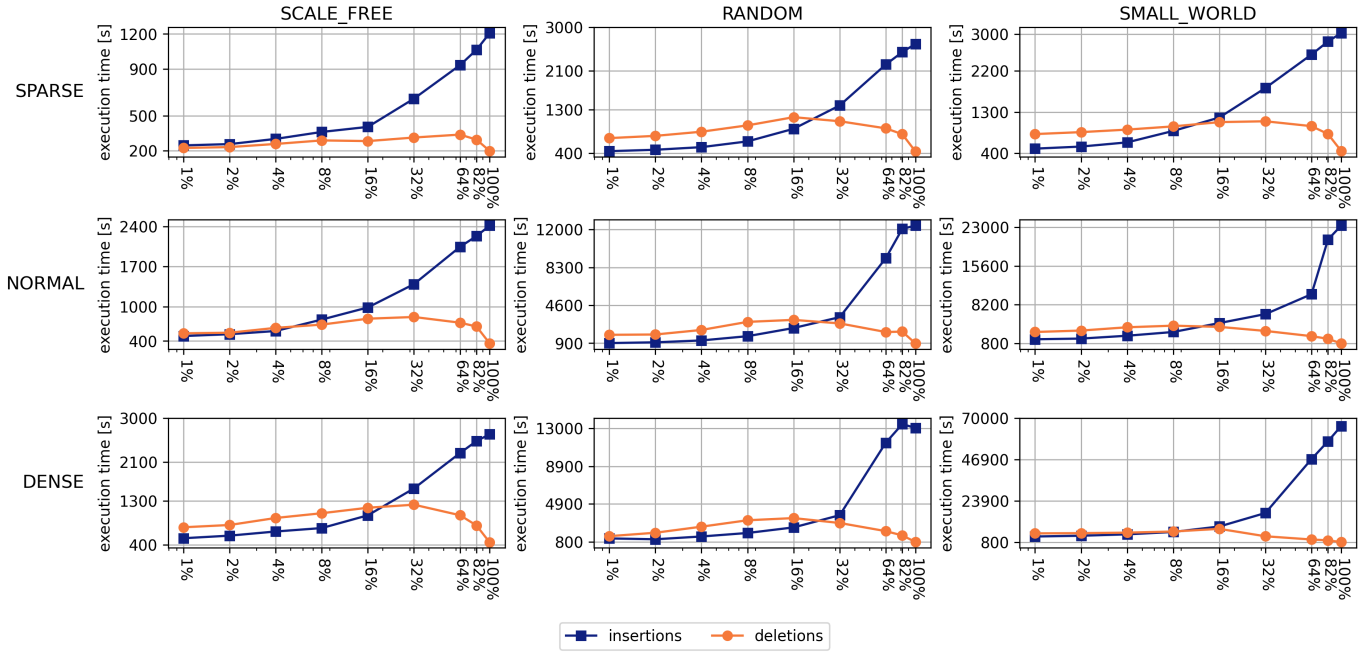


Fig. 14: Comparison of time required to derive control deletions, i.e., $\mathbf{C}^- = \Sigma_D(\mathcal{D})$, and control insertions, i.e., $\mathbf{C}^+ = \Sigma_I(\mathcal{D} \setminus \mathbf{C}^-)$. The analysis is performed on all the introduced graph densities (rows of the plot matrix) and topologies (columns of the plot matrix), while increasing the percentage of updated ownerships (x-axis of each plot) w.r.t to the total ones. The results refer to graphs with a size of 128M nodes. The x-axes are on a logarithmic scale.

secting research areas and then dedicate the rest of the section to related literature from the database community.

The company control problem. The CCP has been of primary importance in the corporate economics literature since the early times [28], to study macro-economic implications [3] related to control dispersion and concentration, and to understand and predict [29] complex control structures to try to prevent unwanted acquisitions [30].

Several studies in the data science [20], economics [4], and data engineering/AI [1, 31] communities have focused on achieving efficient computation of control relationships within an ownership network. To the best of our knowledge, however, all of the existing methods have been designed with a batch approach, do not exploit the information about ownership variation, and do not provide an efficient update of control edges, being therefore inapplicable for our purposes. Multiple quantitative models to exactly determine or approximate company control have been proposed [22], with a clear preference for linear threshold models, like the one discussed in this paper, formalized in logic programming contexts [32].

Materialization maintenance. The materialization maintenance problem is an instance of the database *view maintenance* problem [14, 15, 16, 17]. Given a database $\mathcal{D}$ and a knowledge base (KB) in the form of a set of Datalog rules Σ, let the KB be defined as $\Sigma(\mathcal{D})$, i.e., the set of facts entailed by the application of Σ on $\mathcal{D}$. When querying the KB, a typical optimization technique consists of pre-computing and materializing $\Sigma(\mathcal{D})$, with the consequent need to update it when new facts are added to or removed from $\mathcal{D}$. In this paper, we formulate the problem of incrementally updating control relationships as an instance of view maintenance. Several algorithms have been proposed in the literature, whose aim consists in identifying the minimal subset of the materialization that needs to be updated when some facts are modified. We have seen some of them in Section III. In our work, we do not provide a new algorithm to update materializations, but follow a declarative approach and formulate it as an inference task for the CCP case. In a more structured way, the existing algorithms for materialization maintenance can be organized into three groups.

Approaches based on *counting* [17, 33, 34] form the first category. For every derived fact, a counter enumerates all possible derivations. Then, because of explicit fact deletions, the number of derivations for an entailed fact may decrease. If the associated counter reaches zero, then as no viable derivations exist, the materialization is updated accordingly, by deleting the fact at hand. Typically, such counting algorithms do not support recursive rules and therefore cannot be employed in Datalog reasoners. However, a variant of such counting algorithms capable of handling recursive rules has also been proposed [35]: for each derived fact, it stores an array of counters to count deletions and derivations at each iteration. Still, they adopt a naïve evaluation approach [6, Ch.13], inefficient to handle pervasive recursion like in the case of traversals of large-scale KGs. Moreover, updating large graphs would also suffer from a significant memory footprint,

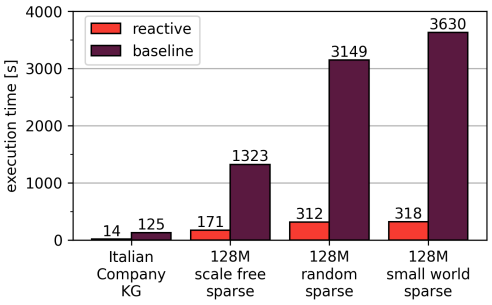


Fig. 15: Execution times for performing the "what-if analysis" on the Italian Company KG and three synthetic graphs.

needed to store the required counters.

The second group of approaches are known as *Delete/Rederive* (DRed) [14, 16, 17, 18] and we have already discussed them, as well as their B/F (backward forward) [15] and *Forward/Backward/Forward* (FBF) [36] variants, at length in Section III. DRed algorithms have been presented in the literature with both procedural and declarative forms. The procedural ones require a specific implementation of the maintenance features in the reasoner core, which was unsuitable for our need to adopt a fully engineered and production-ready reasoner. The declarative implementations feature rule-by-rule rewriting algorithms that generate a set of maintenance Datalog rules for incrementally updating the materialization of an intensional predicate. Unfortunately, to the best of our knowledge, none of these techniques fully satisfies our requirements for KGs: the seminal work [14] does not support aggregation and recursion; other proposals [16] do not disclose any practical rewriting algorithm for aggregations or examples of how it would be handled.

The third group comprises hybrid algorithms [15] that exploit a combination of DRed and B/F with counting: DRed is used to deal with recursive and non-recursive cases separately; B/F is used to evaluate non-recursive derivations avoiding the need for backward chaining.

Our framework provides a declarative implementation of a DRed approach to the CCP. Unlike existing techniques, we see the maintenance process itself as an inference task and formalize it with the program $\Sigma_D \cup \Sigma_I$. Therefore, we do not need to rely upon an evaluation strategy specifically crafted for incremental maintenance. Instead, we leverage the chase implemented in VADALOG. This approach fulfils the industrial requirement of a production-ready reasoner and at the same time benefits from the advantages of DRed strategies. On the other hand, we use the reasoner as a black box and fully rely on its pull-based forward chaining approach, offering semi-naïve processing for recursion, but calling for ad-hoc management of overdeletions in the Datalog code.

Stream reasoning. Finally, stream reasoning approaches [37, 38, 39] are also related to our work. Given a database $\mathcal{D}$, a KB and a query Q, the goal of streaming systems is providing an answer to Q, with facts holding within a sliding window that is continuously updated as $\mathcal{D}$ is modified. Most common approaches enrich facts with timestamps [37, 38], in such a way that handling deletions only involves dropping the expired facts. Yet, also in this case, details about how aggregation and negation would be handled are not available. Finally, further interesting yet still prototypical work introduces the so-called *data stream management system* languages, able to support non-monotonic negations as well as aggregations [39].

Besides the functional limitations, streaming approaches also have non-trivial impact on memory footprint due to the need to store temporal metadata.

VI. CONCLUSION

We contributed a declarative, incremental, and reasoning-based formulation of the CCP, aimed at solving the problem of efficiently updating an existing materialization of control edges in the shareholding network held by the Bank of Italy. We put our logical formulation into action by implementing and running it within the VADALOG System for logical reasoning.

The framework, currently in a pre-production stage, has production ambitions: since we demonstrated its effectiveness through the extensive experimental settings presented in this work, we want to deploy and use it for multiple applications of the Bank of Italy. We have touched on many of them, which range f. Moreover, with the increasingly global scope of the analyses a central bank is called to perform, our system will soon host hundreds of millions of entities and billions of relationships and properties.

This work has been the chance to connect a strategic problem for the Bank of Italy and an industrially felt challenge in the economic and financial context, to the large amount of experience developed by the database community in incremental materialization maintenance. Furthermore, from a theoretical point of view, we took the chance to push the boundaries of reasoning-based approaches to materialization, and, for the first time, adopted a full-fledged language like VADALOG to implement incremental updates. THAs future work, our approach could be further generalized: more in line with the rewriting-based DRed algorithms, our next goal is to design, architect and implement a generic "Datalog to *reactive* VADALOG" translator, able to support the CCP as a specific case. The goal of this industrial work has been however pragmatic and applied: once again, modern reasoning systems and languages have shown to be versatile and useful in practice to embark on another adventure by "walking the thin line between theory and practice" with Datalog [40].

ACKNOWLEDGMENT

This work is part of Davide Magnanimi's Executive PhD program at Politecnico di Milano.

References

[1] A. Gulino, S. Ceri, G. Gottlob, E. Sallinger, and L. Bellomarini, "Distributed company control in company shareholding graphs," in *ICDE*, 2021.

[2] "Guideline of the european central bank of 20 september 2011 on monetary policy instruments and procedures of the eurosystem (ecb/2011/14)," https://bit.ly/3VqqWtT.

[3] F. Barca and M. Becht, *The control of corporate Europe*, 2001.

[4] J. B. Glattfelder, "Ownership networks and corporate control: mapping economic power in a globalized world," Ph.D. dissertation, 2010.

[5] R. Angles, "The property graph database model," in *AMW*, ser. CEUR Workshop Proceedings, 2018.

[6] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, 1995.

[7] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about datalog (and never dared to ask)," *TKDE*, 1989.

[8] L. Bellomarini, D. Benedetto, G. Gottlob, and E. Sallinger, "Vadalog: A modern architecture for automated reasoning with large knowledge graphs," *Information Systems*, 2020.

[9] A. Calì, G. Gottlob, and A. Pieris, "New expressive languages for ontological query answering," in *AAAI*, 2011.

[10] G. Gottlob and A. Pieris, "Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue," in *IJCAI*, 2015.

[11] L. Bellomarini, E. Sallinger, and G. Gottlob, "The vadalog system: Datalog-based reasoning for knowledge graphs," *VLDB*, 2018.

[12] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan, "The magic of duplicates and aggregates," in *VLDB*, 1990.

[13] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," *ACM Comput. Surv.*, 2001.

[14] S. Ceri and J. Widom, "Deriving production rules for incremental view maintenance," in *VLDB*, 1991.

[15] B. Motik, Y. Nenov, R. E. F. Piro, and I. Horrocks, "Incremental update of datalog materialisation: the backward/forward algorithm," in *AAAI*, 2015.

[16] M. Staudt and M. Jarke, "Incremental maintenance of externally materialized views," in *VLDB*, 1996.

[17] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining views incrementally," in *SIGMOD*, 1993.

[18] P. Hu, B. Motik, and I. Horrocks, "Optimised maintenance of datalog materialisations," in *AAAI*, 2018.

[19] T. Baldazzi, L. Bellomarini, M. Favorito, and E. Sallinger, "On the relationship between shy and warded datalog+/-," in *KR*, 2022.

[20] A. Romei, S. Ruggieri, and F. Turini, "The layered structure of company share networks," in *DSAA*, 2015.

[21] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," *Computer networks*, 2000.

[22] J. B. Glattfelder and S. Battiston, "Backbone of complex networks of corporations: The flow of control," 2009.

[23] C. Piccardi, L. Calatroni, and F. Bertoni, "Communities in italian corporate networks," *Physica A: Statistical Mechanics and its Applications*, 2010.

[24] D. Garlaschelli, S. Battiston, M. Castri, V. D. Servedio, and G. Caldarelli, "The scale-free topology of market investments," *Physica A: Statistical Mechanics and its Applications*, 2005.

[25] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, 1999.

[26] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *Nature*, 1998.

[27] V. Batagelj and U. Brandes, "Efficient generation of large random networks," *Physical Review E*, 2005.

[28] R. La Porta, F. Lopez-de Silanes, and A. Shleifer, "Corporate ownership around the world," *The journal of finance*, 1999.

[29] L. Bellomarini, L. Bencivelli, C. Biancotti, L. Blasi, F. P. Conteduca, A. Gentili, R. Laurendi, D. Magnanimi, M. S. Zangrandi, F. Tonelli, S. Ceri, D. Benedetto, M. Nissl, and E. Sallinger, "Reasoning on company takeovers: From tactic to strategy," *DKE*, 2022.

[30] L. Bellomarini, M. Benedetti, S. Ceri, A. Gentili, R. Laurendi, D. Magnanimi, M. Nissl, and E. Sallinger, "Reasoning on company takeovers during the covid-19 crisis with knowledge graphs," in *RuleML+ RR*, 2020.

[31] P. Atzeni, L. Bellomarini, M. Iezzi, E. Sallinger, and A. Vlad, "Weaving enterprise knowledge graphs: The case of company ownership graphs." in *EDBT*, 2020.

[32] S. Ceri, G. Gottlob, and L. Tanca, *Logic Programming and Databases*, ser. Surveys in computer science, 1990.

[33] J. A. Blakeley, P.-A. Larson, and F. W. Tompa, "Efficiently updating materialized views," *SIGMOD*, 1986.

[34] A. Gupta, D. Katiyar, and I. S. Mumick, "Counting solutions to the view maintenance problem." in *JICSLP*, 1992.

[35] H. M. Dewan, D. Ohsie, S. J. Stolfo, O. Wolfson, and S. Da Silva, "Incremental database rule processing in paradiser," *Journal of Intelligent Information Systems*, 1992.

[36] B. Motik, Y. Nenov, R. Piro, and I. Horrocks, "Maintenance of datalog materialisations revisited," *Artificial Intelligence*, 2019.

[37] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, "Incremental reasoning on streams and rich background knowledge," in *Extended semantic web conference*, 2010.

[38] A. Ronca, M. Kaminski, B. C. Grau, and I. Horrocks, "The window validity problem in rule-based stream reasoning," in *KR*, 2018.

[39] C. Zaniolo, "Logical foundations of continuous query languages for data streams," in *Datalog 2.0*, 2012.

[40] G. Gottlob, "Adventures with Datalog: Walking the Thin Line Between Theory and Practice (Ext. Abs.)," 2014.