Distributed Company Control in Company Shareholding Graphs

Andrea Gulino*, Stefano Ceri*, Georg Gottlob†, Emanuel Sallinger† and Luigi Bellomarini‡ §
*Politecnico di Milano, name.surname@polimi.it
†TU Wien and Univ. of Oxford, surname@dbai.tuwien.ac.at
‡Banca d'Italia, luigi.bellomarini@bancaditalia.it

***Abstract*—The Company Control Problem is of central importance to banks, financial intermediaries, financial intelligence units, regulatory and supervisory authorities such as the Central Banks. It consists in understanding who takes decisions in a large company network, that is, who controls the majority of votes for each single company. This has an impact on a large number of business areas, with examples including evaluation of creditworthiness, economic analysis of the control dispersion, anti-money laundering, prevention of potentially hostile takeovers, evaluation of risks, and shock propagation.**

This paper is based on our experience with the Central Bank of Italy and presents an approach to the solution of the company control problem in distributed settings, especially relevant, as large and distributed ownership graphs reflect European-size applications where scalability is paramount.

In particular, we formalize the problem as query answering on a large distributed database. We study how independent subqueries can be executed in each partition and the partial results assembled at a master site to produce the answer. We study the formal properties of the problem, that is not easily parallelizable, and then present a method that supports parallelism at best.

We present a thorough experimental evaluation of our approach with the Italian company graph of the Bank of Italy and the European Register of Financial Intermediaries and Affiliates as well as many artificial graphs to fully assess scalability.

I. Introduction

Understanding who takes decisions in company networks, exerting the so-called *company control*, is a well-known problem and of primary interest to a very broad spectrum of financial authorities, banks, central banks, national statistical offices, Financial Intelligence Units (FIUs), financial intermediaries, including both the rising FinTechs as well as the incumbent firms. All such actors wish to understand who, for a given company, can control the vote majority, as it is a generally accepted assumption [11] that voting rights have a one-to-one correspondence to company shares. Control of shares can be direct, in the sense that x directly owns such shares of y (i.e., it is a shareholder of y), or indirect, in case x controls a set of companies that jointly own shares of y. This latter, inductive facet of control, makes its computation by no means trivial and indeed much harder than standard reachability queries, or variants thereof, over graphs.

The *company control problem* serves many different business goals of the financial authorities, including *financial and bank supervision*, *anti-money laundering*, *credit worthiness* evaluation and *collateral eligibility*.

§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 paper is based on our experience with the Central Bank of Italy. In its role of central bank within the European System of Central Banks (ESBC), the Bank of Italy operates as a EU-wide supervisory and regulatory body. In such a global setting, the company control problem has a supernational span and involves processing datasets of very large directed graphs represented and stored in either relational or graph databases, expressing the ownership relationship. In fact, the ESCB has consolidated an EU-level supervision mechanism and, more in general, is heading towards an integrated financial and statistical systems that increasingly calls for a cooperation of the national banks in data collection, processing and analysis tasks. In these settings, for scalability, responsibility, and privacy reasons, distributed data processing where multiple and sometimes all the ESBC banks are involved are becoming more and more common.

Yet, to the best of our knowledge, the algorithms that have been proposed so far to solve the company control problem, in both the technical [24] and the economics [11] communities, are sequential and, unfortunately, do not meet the needs of the real-world applications, which require highly parallelizable methods to support a distributed version of company control, with the following desiderata:

- **Volume**. The graph includes millions of companies which ere extensively interconnected.
- **Locality**. The graph may be partitioned, owned and maintained by different sites for organisational reasons (e.g., different central banks, countries).
- **Scalability** (*scaling-out*). The algorithm should support growing graphs.
- **Performance** (*scaling-up*). For specific applications (e.g., interactive analytics) the processing time must be limited.

In this work, we account for all these aspects and propose an approach that combines effective graph data management techniques with *parallel and distributed* query execution. We see the problem as a query answering task on a large distributed database and study how independent subqueries can be executed at each partition (*inter-site parallelism*) with a further level of *intra-site parallelism*, and the partial results be assembled by a master site, producing the answer.

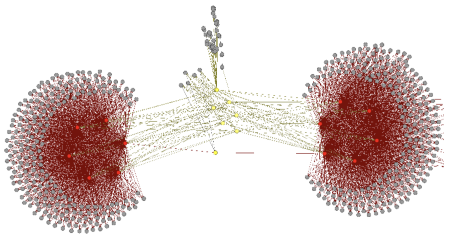


Figure 1: Subgraph of Italian ownership graph from [24], with the 12 shareholders having the highest out-degree.

Contribution. The main contributions of this work are:

- Relevant industrial scenarios motivated by real processes of the Bank of Italy within the ESCB, where the company control problem is of major interest.
- A formalization of the company control problem and the first characterization of its computational complexity.
- A parallel algorithm for solving the company control problem through recursive queries over large databases of ownership relationships.
- The definition of the distributed control problem as well as a distributed parallel algorithm for it.
- A detailed experimental evaluation of our approach in the ESCB scenario, using two real-world ownership graphs and testing overall scalability with exhaustive synthetic settings.

Overview. The remainder of the paper is organized as follows. In Section II we describe our industrial settings. In Section III we formally characterize the problem, whose complexity is studied in Section IV. In Section V, we propose a sequential algorithm as well as its properties, an essential building block of our parallel version, which is then presented in Section VI. Section VII describes how the problem can be effectively managed when data distribution reflects a national organization and a transnational control. Section VIII evaluates our approach in action. Section IX presents the related work and in Section X we draw our conclusions.

II. Industrial Scenarios

Let us start by describing the characteristics of the graphs interested by the business goals described in the introduction.

The Italian graph. The Italian ownership graph is characterized by a main large weakly connected component. As it shown in Figure 1 (from [24] and sometimes referred to as "the lung graph"), it contains the main 12 shareholding companies, with the highest out-degree. Six of them own more than 200 other companies (the right lung), whereas another group of six (the left lung) owns in turn a similar number. The main 12 shareholders are all owned, though not controlled, by 7 non-Italian companies, located at the center of the lung graph and displayed in yellow. More in detail, the graph counts 4.059M nodes and 3.960M edges; it exhibits a high number of small strongly connected components (SCC), 4.058M conglomerates with 15 nodes in the largest one; also, although the graph has one huge weakly connected component (WCC) with 1.598M nodes, most of the nodes are scattered around small WCCs: 635.324K in total, with 6.390 nodes on average and none with more than 175 nodes. On average, each node owns 1.431 companies and is owned by 2.716. There are 30 nodes owning more than 225 firms, with 2 owning more than $1K$.

In terms of topology, the graph is a *scale-free network*, i.e., the degree distribution asymptotically follows a particular degree law [22]. This topology is also confirmed by corporate economics literature [10], which justifies this with portfolio differentiation reasons.

The EU graph. The overall European graph is composed of a set of scale-free networks, the local graphs, which are interconnected via cross-country ownership and control relationships. Towards a production application of our approach to distributed company control, we used the Italian graph as a *proxy version* of the EU-level company graph, where the real Italian graph of the Bank of Italy is used to train graph generators and produce a realistic simulation version of the overall EU graph. Scale-free networks are characterized by a set of parameters, which control the distribution (e.g., in terms of density) and give shape to each specific network [3]. With this assumption, we generated graphs for all European countries as scale-free networks (whose parameters have been fitted from the Italian graph) and connected them via a set of "border companies" (between 2% and 5% of the total).

The Register of Intermediaries and Affiliates. We also successfully applied our technique to the *Register of Intermediaries and Affiliates* (RIAD) [23], the ownership network of European financial companies run by the Bank of Italy for the European Central Bank. While at the moment, this graph is entirely stored at a single site, being able to swiftly compute company control on RIAD data is extremely valuable business-wise. In fact, control can be used as a predictor of the *collateral eligibility* for *asset-backed securities*, a core application RIAD is used for in the ESCB [14]. Due to its structure, the RIAD graphs highlights the effectiveness of parallelism, in our technique: 91% of the nodes are within a one-node SCC, with one large SCC containing 88 nodes, and all the others with less than 10 nodes; there is one huge WCC, with 57% of the nodes, with the others scattered around small WCCs with 11.968 nodes on average and (apart from the largest one), none with more than 472 nodes.

III. The Company Control Problem

Let us now formally introduce the problem. A *business ownership graph* (or simply *ownership graph*) $G = (V, E, L)$ is a directed graph with nodes V representing companies, directed edges $E \subseteq (V^2 - \{(v,v) | v \in V\})$, and a labelling $L : E \rightarrow (0, 1]$ assigning each edge $e = (v, w) \in E$ a label $L(e)$ representing the fraction of the equity of w that v holds. For any node $v \in V$, it is required to hold that $\sum_{w \in pred_v} L((w, v)) \leq 1$.

We will use the following notation: for an edge $e = (h, t)$ we will sometimes call the node h (resp. t) the head (resp.

tail) of e. For $v \in V$ we denote by in_v (resp. out_v) the set of incoming (resp. outgoing) edges of v, and by $pred_v$ (resp. $succ_v$) the set of predecessor (resp. successor) nodes of v.

The company control problem. Intuitively, company x controls a company y when x directly owns more than 50% of y or x controls a set of companies that jointly own more than 50% of y. Formally, the control relation can be defined by the smallest set *Control* that satisfies the following logical recursive query[1] over a very large graph database - a classical but not trivial data management problem:

$$\mathit{Control}(x) \to \mathit{Control}(x,x) \tag{1}$$

$$\mathit{Control}(x,y), \mathit{Own}(y,z,w),$$
$$v = \mathit{msum}(w, \langle y \rangle), v > 0.5 \to \mathit{Control}(x,z) \tag{2}$$

Given an ownership graph $G = (V, E, L)$ together with two of its vertices, namely, a company s (*source*), and a company t (*target*), the *company control problem (CCP)* is the decision problem of determining whether $Controls(s,t)$ holds, i.e., understanding whether s can control t either directly or indirectly. We shall sometimes refer to this problem instance, or query, as $q_c(s,t)$ if the graph G is clear from the context. Observe that the structure of the inductive case (2) distances CCP from standard *pathfiding* or *weighted pathfinding* settings in graphs, and makes algorithms based on Dijkstra, A*, breadth-first or iterative depth-first traversals [17], not helpful due to the inherent difference between the problems. In fact, a company x may exert control on a company y as a result of multiple paths from x to y: a workable algorithm must be able to consider only those that potentially contribute to control and exclude the others. On the other hand, the standard algorithms either find specific paths from x to y (shortest, lowest/highest weight, etc.), or explore all of them. In particular, an approach to CCP based on path-enumeration would be inherently intractable and not parallelizable, as counting all s-t simple paths is a paradigmatic $\#\mathsf{P}$-hard problem of enumerative combinatorics [19].

IV. Computational Complexity

In this section, we explore the computational complexity of the company control problem (CCP) an see what are the margins of applicability for a parallel solution to it, suitable to our setting. We first argue that the CCP is tractable, and quadratic time in particular. Yet, unfortunately, we show that it is P-complete, from which we cannot immediately conclude it is parallelizable. In other words, an algorithm solving the problem by a polylogarithmic round of parallel steps on a polynomial number of processors (for example, via MapReduce or similar techniques) is by no means trivial or intrinsic of the problem itself and will be matter of Section V.

For proving tractability of the CCP, we have to provide a polynomial-time algorithm whose input (G, s, t), where G is an ownership graph and v and w are two of its nodes, determines whether v controls w. Actually, this algorithm already exists, as one just needs to execute the logic program given in Section III, for example using the Vadalog inference engine [2]. However, let us here state a strictly procedural program which does exactly the same:

Algorithm 1 Control by Expansion (CBE)

```
Input: (G,s,t), where $G = (V, E, L)$.
Output: yes if $s$ controls $t$ in $G$, otherwise no.
  $Controlled \leftarrow \{s\}$;
  while there is some $u \in V - Controlled$ such that
  $\Sigma_{u' \in Controlled \wedge (u',u) \in E} L((u',u)) > 0.5$ do
      $Controlled \leftarrow Controlled \cup \{u\}$;
  if $t \in Controlled$ then return yes else return no.
```

The above CBE algorithm thus starts with *Controlled* $= \{s\}$ (which would correspond to the first rule of the logic program of Section III applied to s, and during the while-loop exhaustively executes the second (recursive) rule. The CBE program, in its set variable *Controlled* thus correctly accumulates all companies that are controlled by s, and only those, and outputs *yes* if t is in this final set and otherwise *no*. If the input graph G has n company vertices, the while-loop is executed at most n times, and each time it sums over no more than n elements. Assuming appropriate data structures and a machine model where an arithmetic operation can be done in one unit step, the CBE algorithm thus runs in time $O(n^2)$. Therefore we can state:

Theorem 1. *The Company Control Problem (CCP) can be solved in quadratic time and is thus tractable.*

Now the question about the parallelizability of CCP arises. Could we use parallel frameworks, such as, for example Map-Reduce to solve CCP on a cluster of many computers with sub-polynomially many sequential rounds of parallel computations on all input instances? In more theoretical terms, is the CCP problem in the complexity class NC? We will show that this is actually not possible unless there were a dramatic, and generally disbelieved collapse of complexity classes. In fact, we are going to show that the CCP problem is P-hard, and thus P-complete (here P stands for the class of polynomial-time solvable decision problems). This means that CCP is among the hardest problems in P, being at least as hard as *any* other problem in P, and thus not in the class NC unless NC=P, which is considered to be similarly unlikely as P=NP. Problems that are P-hard are also referred-to as *inherently sequential* problems. For a very survey of P-hardness and the limits to parallel computations including a compendium of P-hard problems see [13].

Theorem 2. *The Company Control Problem (CCP) is P-hard and thus, as a member of P, also P-complete. CCP is therefore not parallelizable unless NC=P, that is, unless all polynomially decidable problems are parallelizable. CCP, moreover, remains P-hard even for acyclic company graphs that have less than three times more edges than nodes (and are thus sparse).*

[1] $v = \mathit{msum}(w, \langle y \rangle)$ is the monotonic sum of all w counting contributions only once for each y [2]

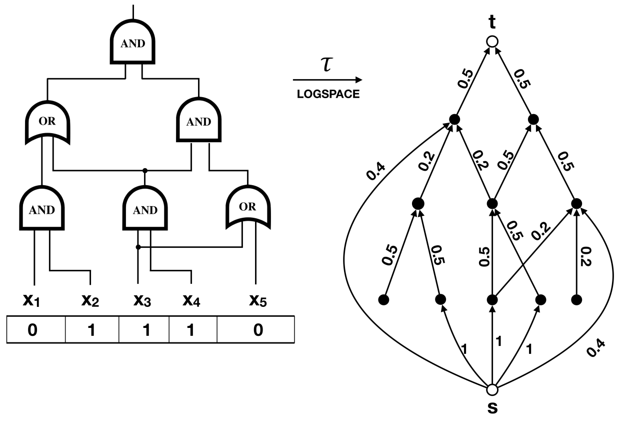


Figure 2: Illustration of the reduction of MVCP to CCP.

Proof. (Sketch) We reduce the well-known P-hard Monotone Circuit Value Problem (MCVP) via a logspace-reduction to the CCP problem. An instance C of the MCVP consists of a monotone Boolean circuit with input values, as depicted in the left part of Fig. 2. It is transformed into an equivalent instance of the CCP problem, as depicted in the right part of Fig. 2.

Each gate g is mapped to a company. In particular, the unique output gate of C is mapped into a vertex named t. In addition, there is an extra vertex s connected by an arc labeled 1 to each vertex that corresponds to an input gate with associated Boolean value 1. If g is an and-gate then an arc (v_a, v) and an arc (v_b, v), both labeled 0.5 are introduced. Intuitively this enforces that s must control both v_a and v_b in order to also control v. If g is an or-gate then an arc (s, v) labeled 0.4 and the two arcs (v_a, v), (v_b, v), both labeled 0.2 are introduced. Intuitively this enforces that s must control at least one of v_a and v_b in order to also control v. □

Although Theorem 2 is a negative result, it is a typical worst-case complexity result, and there may be many cases where a significant level of parallelization is still possible. In fact, in the rest of the paper we will present efficient parallel algorithms that perform very well in practice on the real-world and real-world inspired synthetic graphs in our experimental evaluation.

V. Algorithm

In this section, we first classify the nodes of an ownership graph into four *classes* that shall become central to the algorithms in this paper. After that, we describe *reduction rules* that act on these classes of nodes and apply simplifications to the graph. As the final ingredient for our algorithms, we then describe *termination conditions*.

Based on these parts, we are going to show how a *centralized* algorithm could be immediately constructed from these concepts. This is to help foster understanding for the more complex parallel and distributed algorithms that will follow.

A. Classes of Nodes

The reduction rules that we are going to present are *node-centric*, i.e., their application depends exclusively on the class of the individual node. We define four such classes:

- **Irrelevant nodes** (class C_1): a node is *irrelevant* if it misses incoming edges, outgoing edges or both. Formally we have that:
$$C_1 = \{v \in V^- \mid out_v = \emptyset \vee inc_v = \emptyset\}$$
where $V^- = V \setminus \{s, t\}$. Such a node cannot play any role in the control of s on t, and therefore can be removed. For obtaining non-overlapping classes, C_1 nodes will be excluded from C_2, C_3 and C_4.
- **Uncontrollable nodes:** (class C_2): a node is *uncontrollable* if the sum of the labels associated to its incoming edges does not exceed 0.5. Formally:
$$C_2 = \{v \in V^- \mid \sum_{e \in inc_v} L(e) \leq 0.5\} \setminus C_1$$
This kind of nodes can neither be controlled directly nor indirectly and, therefore, can be removed.
- **Directly controlled nodes** (class C_3): a node v is *directly controlled* if there exists a predecessor, namely w_{dc}, than owns more than 50% of v. Formally:
$$C_3 = \{v \in V^- \mid \exists w\, (w \in inc_v \wedge L(w) > 0.5)\} \setminus C_1$$
Observe that w_{dc} is the *only* predecessor that can directly control v, as the sum of ownership cannot be larger than 1. Hence, all other nodes in the graph can indirectly control v if and only if they control w_{dc}. Thus node v itself can be removed if we "transfer" its edges to w_{dc}.
- **Indirectly controllable nodes** (class C_4): a node is *indirectly controllable* if the sum of the labels associated to its incoming edges exceeds 0.5, but none of the labels individually exceeds 0.5 (i.e., C_3). Formally:
$$C_4 = \{v \in V^- \mid \sum_{e \in inc_v} L(e) > 0.5\} \setminus (C_1 \cup C_3)$$
Even if nodes in this class cannot be directly controlled by any of their predecessors, they may be indirectly controlled by other nodes in V (for example the source node). Therefore, those nodes may be relevant for answering $q_c(s, t)$ and cannot be immediately removed from the graph.

Note that the union of the defined classes contains all the possible nodes in a control graph except s and t and that, by construction, these classes are disjoint. To sum up, we have defined four classes: C_1 and C_2, describing nodes that can be removed from the graph, C_3, describing a relation between nodes that can be simplified, and C_4 containing nodes that cannot directly be removed from the graph.

B. Reduction Rules

In this subsection, we define reduction rules, the second component of our algorithm. A reduction rule is described by means of: (i) an activation **condition** defined on a node of the graph, that tells whether the rule can be applied to remove that node from the graph. For our rules, the condition checks whether the node belongs to one of the four classes

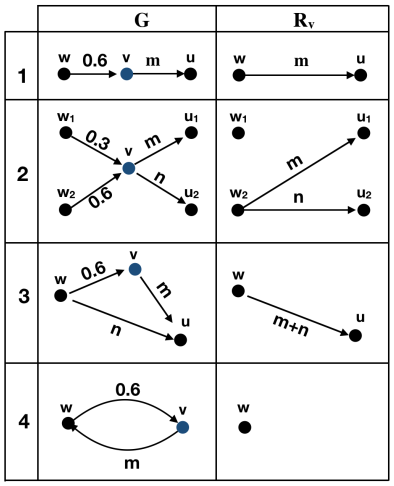


Figure 3: Four examples in which $R3$ is applied to a node $v \in G$. In each reduced graph R_v, resulting from the application of $R3$, v has been removed and the relations between its predecessors (denoted by letter w) and successors (denoted by letter u) have been simplified.

described in V-A and (ii) an **action** that tells how the graph should change as a consequence of the activation of a rule (e.g. remove the node and all its edges, add new outgoing edges to a given node).

In order to define reduction rules, consider an ownership graph $G = (V, E, L)$ and two nodes $s, t \in G$ that are respectively the source and target nodes for which we want to answer $q_c(s, t)$. In our notation, we generally use v to denote the node to which the rule is applied and that, eventually, will be removed from the graph; w and u to denote, respectively, a generic predecessor of v and a generic successor of v. We present the action associated to each rule as a function $a(R, v)$ that transforms the input graph $R = (V, E, L)$ into the reduced graph $R' = (V', E', L')$. The source and target nodes are excluded a-priori from the application of any rule (i.e. they should always remain in the graph).

1) R1, R2: Irrelevant or Uncontrollable Nodes: These two rules differ in their activation condition ($R1$ matches any node $v \in C_1$, while $R2$ matches nodes $v \in C_2$), but both have as their action to remove v and all its edges from the input graph $R = (V, E, L)$. The following is a formal description of the output graph $R' = (V', E', L')$:

$$a_{1/2}(R, v) = R' = \begin{cases} V' &= V \setminus \{v\} \\ E' &= E \setminus (inc_v \cup out_v) \\ L' &= L \setminus \{(e, k) \mid e \in (inc_v \cup out_v)\} \end{cases}$$

2) R3: Directly Controlled Nodes: This rule matches any node $v \in C_3$, i.e. any node that is directly controlled by one of its predecessors, namely w_{dc}. In this case, v and its incoming edges are removed from the graph, while the set of its outgoing edges, along with their labels, can be transferred to w_{dc}; i.e., what was previously owned by v is now owned by w_{dc}.

Let us consider a number of examples to understand the possible situations first. In Figure 3 (1) w directly controls v (i.e. $w = w_{dc}$), therefore v and its incoming edge are removed from the graph while its unique outgoing edge is transferred to w. In Figure 3 (2), w_2 directly controls v (i.e. $w_2 = w_{dc}$), therefore v and all its incoming edges are removed from the graph while all its outgoing edges are transferred to w_2.

In Figure 3 (3), we see that when the outgoing edges of v are assigned to w_{dc}, it may happen that there is already an edge connecting w_{dc} to a successor of s. In this situation, instead of introducing **parallel** edges in the graph, i.e. an edge with label m and another with label n, both connecting the same couple of nodes, we just generate a single edge whose label is the sum $m + n$.

In Figure 3 (4) we see a situation where w is both a predecessor and a successor of v. The application of $R3$ in this example would clearly introduce a self-loop on w_{dc}. Since for this instance of the company control problem self-loops are neglectable, we exclude the generation of self-loops from the application of $R3$.

We formalize the action of $R3$ as the following two steps:

Step 1. We remove v and *all* its edges from the graph in the same way as for R_1 and R_2, producing a *temporary* reduced graph $R^t = (V^t, E^t, L^t) = a_1(R, v)$.

Step 2. We assign to w_{dc} the outgoing edges of v together with their labels, unless there is already an edge connecting w_{dc} to a successor of v; in that case we just add a new label to the already existing edge, as previously discussed. Formally, this is performed through constructing the following sets:

- E^+ and L^+: respectively, the set of new edges that should be assigned to w_{dc} and were previously successors of v, and the set of their labels. E^+ is simply the domain of L^+, that is defined as follows:

$$L^+ = \{ \big((w_{dc}, u), k\big) \mid (w_{dc}, u) \notin E \wedge \\ w_{dc} \neq u \ \wedge (v, u) \in out_v \ \wedge L((v, u)) = k\}$$

  where condition $w_{dc} \neq u$ prevents the creation of self-loops, and condition $(w_{dc}, u) \notin E$ excludes those edges that would create a parallel if they were assigned to w_{dc}.
- L^-: are the labels of the already existing edges connecting w_{dc} to a successor of v. We are going to remove those labels from the graph and replace them with new labels defined in the next point. The definition of this set is the following:

$$L^- = \{\big((w_{dc}, u), k\big) \mid \big((w_{dc}, u), k\big) \in L \wedge \\ (v, u) \in out_v\}$$

- $L^{\oplus}$: the new labels that should replace the labels defined in the previous point. The definition of this set is the following:

$$L^{\oplus} = \{\big((w_{dc}, u), k_1 + k_2\big) \mid \big((w_{dc}, u), k_1\big) \in L \wedge \\ (v, u) \in out_v \wedge \big((v, u), k_2\big) \in L\}$$

The overall action performed by $R3$, that produces the reduced graph $R' = (V', E', L')$ from $R^t = (V^t, E^t, L^t)$, is:

$$a_3(R, v) = R' = \begin{cases} V' & = V^t \\ E' & = E^t \cup E^+ \\ L' & = \left(L^t \setminus L^-\right) \cup L^+ \cup L^{\oplus} \end{cases}$$

Analysis. We are now going to show that the reduction rules preserve the behaviour of the graph for answering company control queries. This shall be essential for the completeness and correctness of our algorithms. We first define *control-equivalence* between graphs.

Definition 1. *Two ownership graphs G and G' are called* control-equivalent *w.r.t. a set of companies X, denoted*

$$G \equiv^c_X G'$$

if for any $s, t \in X$ it holds that s controls t in G if and only if s controls t in G'.

Note that the graph resulting from applying a reduction rule in general contains a subset of the nodes of the original graph. Therefore, control-equivalence is known to hold only for that subset:

Proposition 1. *Let R_i be a reduction rule ($i \in 1, 2, 3$) and let G be an ownership graph. Let $G' = a_i(G)$ and let V' denote the set of nodes of G'. It holds that $G \equiv^c_{V'} G'$.*

C. Termination Conditions

In principle, as our rules preserve control equivalence as shown in Prop. 1, any algorithm based on these rules could be executed exhaustively until no further rule is applicable. Given that each of our reduction rules removes at least one node, this would be guaranteed to terminate. Still, for obtaining optimized algorithms, we define the following conditions that allow us to terminate earlier:

- T_1: The source node s does not directly control any node in the graph. Formally:

  $$out_s = \emptyset \ \vee \neg\exists e \, (e \in out_s \wedge L(e) > 0.5) \qquad (T_1)$$

  In this case, there is no way for s to control t neither directly nor indirectly. Query $q_c(s, t)$ is therefore *false*.
- T_2: The target node t cannot be controlled by any other node in the graph. Formally:

  $$inc_t = \emptyset \vee \sum_{e \in inc_t} L(e) \leq 0.5 \qquad (T_2)$$

  Also in this case there is no way for s to control t. Query $q_c(s, t)$ is therefore *false*.
- T_3: s directly controls t. Formally:

  $$(s, t) \in E \wedge L((s, t)) > 0.5 \qquad (T_3)$$

  Query $q_c(s, t)$ is therefore *true*.

Procedure $checkTermination$ simply returns a boolean value indicating whether at least one of the termination conditions is satisfied on R, i.e.:

$$checkTermination(q_c, R) := T_1 \vee T_2 \vee T_3$$

We shall later also use T_3 to give the answer of our algorithm, i.e., we define $answer(q_c, R) := T_3$.

VI. Parallelism

In this section, we propose a parallel solution for graph reduction, based on the parallel application of rules $R1$ and $R2$ and enabling a certain degree of parallelism for the application of rule $R3$. We conclude the section with an example.
The proposed solution:

- organizes the execution in *steps* (somehow similar to the concept of super-steps used in Pregel [20]). Specifically, the algorithm iteratively applies two steps: (1) parallel testing of each node for the identification of their class; (2) parallel removal of nodes, depending on their class;
- separates the application of $R1$ and $R2$ from the application of $R3$, by splitting the overall execution in two *phases*: (1) parallel elimination of useless nodes ($v \in C_1, C_2$); (2) simplification of the remaining graph ($v \in C_3$). As shortly explained, while in the first phase $R1$ and $R2$ can be fully applied in parallel, in the second phase, we can also apply $R3$ partially in parallel, by identifying some non-overlapping fragments of the graph that can be processed in parallel.

```
procedure $parallelReduction(G, q_c, X)$
Input: a graph $G$, a query $q_c(s, t)$, an exclusion set $X$.
Output: a tuple $[globalAns, R]$.
 1: $G' = mark(G, q_c, X)$;
 2: while ($\nexists v \in G' \mid L^V(v) \in \{C_1, C_2\}$) do        ▷ Phase 1
 3:     if $checkTermination(G', q_c) == true$ then
 4:         return [ $answer(G, q_c)$, $\emptyset$];
 5:     $G' = clean(G')$;
 6:     $G' = mark(G', q_c, X)$;
 7: while ($\nexists v \in G' \mid L^V(v) = C_3\}$) do        ▷ Phase 2
 8:     if $checkTermination(G', q_c) == true$ then
 9:         return [ $answer(q_c, G')$, $\emptyset$];
10:     $G' = simplify(G')$;
11:     $G' = mark(G', q_c, X)$;
12: if $checkTermination(G', q_c) == true$ then
13:     return [ $answer(G, q_c)$, $\emptyset$];
14: else
15:     return [ $\emptyset$, $G'$];
```

The parallel algorithm is presented in the form of a procedure named *parallelReduction* to be next invoked in a distributed setting; it takes as input: (i) the ownership graph G; (ii) the *query* $q_c(s, t)$; (iii) an *exclusion set* X, containing nodes that are never removed from the graph. For this section, $X = \{s, t\}$, i.e., the algorithm will never remove nodes s and t, but the exclusion set will play an additional role in the distributed version of the algorithm.

The algorithm produces as output the tuple $[globalAns, G']$, where $globalAns$ is the answer to $q(s,t)$, while G' is the reduced graph that remains after applying reduction rules. The reduced graph G' is of course not required for giving the answer to the query, but will be used in the distributed setting. Moreover, we introduce an accessory labeling function L^V assigning each node $v \in V$ the label $L^V(v)$. The algorithm consists of two phases:

Phase 1: in this phase rules $R1$ and $R2$ are applied in parallel to remove irrelevant or uncontrollable nodes from the graph, by means of the following steps:

- *mark*: in parallel, each node v is labelled with a symbol representing its class (i.e. C_1, C_2, C_3 or C_4). If v is in the exclusion set X, it is marked with the symbol $\perp$.
- *clean*: in parallel, each node that was marked either with C_1 or with C_2 in the previous step is removed from the graph, along with all its edges.

After applying the *cleaning* step, some nodes that were previously labeled with C_3 or C_4 may have lost some of their incoming edges, thus changing their class into C_1 or C_2. If this is the case, the $mark$ and *clean* steps have to be repeated. Phase 1 terminates when all C_1 and C_2 nodes have been removed from the graph.

Phase 2: in this phase rule $R3$ is applied to simplify the graph produced in the previous phase. Note that now the graph can contain only nodes labeled with C_3, C_4 or $\perp$. In this phase, the following steps are iteratively applied:

- *mark*: the same procedure described for the first phase.
- *simplify*: in parallel for each node v that is marked either with C_4 or with $\perp$ (i.e. for each node that cannot be removed from the graph), perform a sequential reduction: $R3$ is sequentially applied to the C_3-successors of v until no successor of v is marked with C_3 (i.e. every successor is another non-removable node).

Virtually, each node v defines an *application context*, i.e. a sub-graph induced by all the nodes of type C_3 in any path from v to another non-removable node.

After applying the *simplify* step, some nodes that were previously labeled with C_4 may now be directly controlled, thus changing their class to C_3. If this is the case, the *mark* and *simplify* steps have to be repeated. Phase 2 terminates when all C_3 nodes have been removed from the graph.

Procedure *checkTermination*, identical to the one described in Section V, checks, at different steps, whether a termination condition is satisfied, possibly causing an early stop of the algorithm. Procedure *answer* produces the boolean answer to $q_c(s,t)$, as described in Section V.

At the end of phase 2, only s and t remain in the graph (no C_4 can remain under those conditions). Therefore, the check at line 14 will always yield $true$ and the algorithm will always terminate with the answer to $q_c(s,t)$ (given at line 15). Line 17 will be useful in the distributed context. All parallel procedures were implemented with GraphX, which is widely used for implementing parallel iterative graph computation algorithms like ours, as advised in [8].

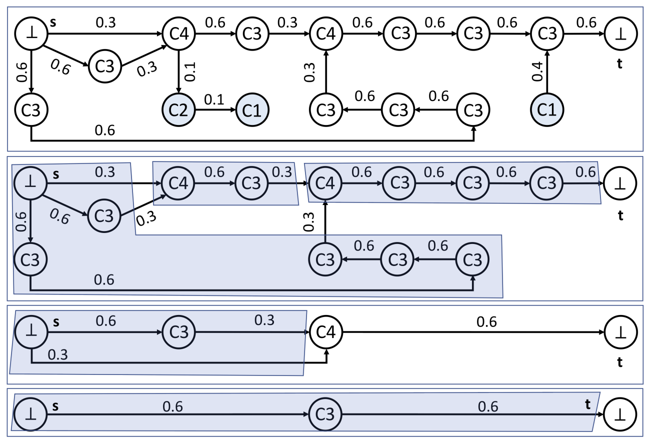


Figure 4: Example of parallel reduction.

Example. Figure 4 shows an example of application of *parallelReduction* to the graph represented in the top-most rectangle. Each rectangle represents an iteration of the algorithm, showing the initial marking of nodes and highlighting, in light blue, either nodes that will removed or application contexts that will be simplified in that iteration.

VII. Distributed Control

Our approach is based on **partial evaluation**, similar to what has been done in [7]. The query $q_c(s,t)$ is sent to all sites storing a partition of the graph. At each site, in parallel, a partial answer to $q_c(s,t)$, based on the local partition of the graph available at that site, is computed. Partial answers are then collected by a coordinator site that solves $q_c(s,t)$ on the graph built by merging all the partial answers. This method provides the following properties:

1) Each site in visited *at most once*.
2) Partial evaluation is performed *in parallel* at each site, thus sites do not have to synchronize their execution with any other site.
3) Partial answers are *reduced graphs*, usually much smaller than initial partitions, as many internal nodes (i.e. without direct connection to any external partition) are missing.
4) If the company control information is static (or at least it varies periodically, with rather long periods such as days or weeks), then at query time *at most two sites* need to perform local evaluation (i.e., compute a partial answer). Specifically, such local evaluation needs to be performed on those sites storing s or t; the partial answers produced by the remaining sites can be computed off-line and stored at the coordinator site.

No bounds on the size of the returned partial answers can be guaranteed: in the worst case, the whole partition must be returned to the coordinator. Nevertheless, in the experimental section we show that in real applications the size of

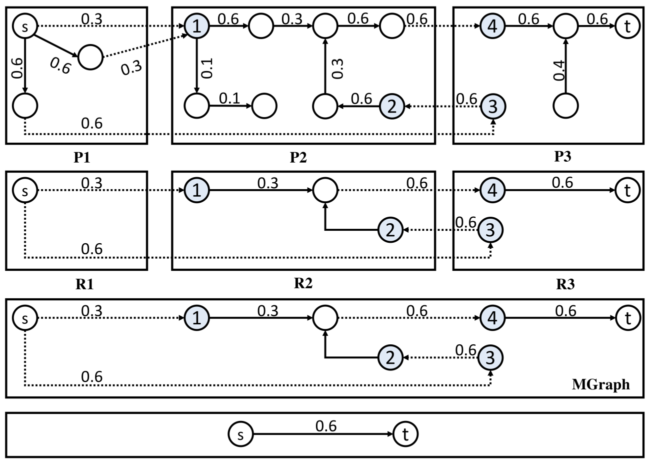


Figure 5: Distributed graph with 3 partitions $P1, P2$ and $P3$; partitions are first reduced to $R1$, $R2$ and $R3$ at each site and then merged into *MGraph* at the coordinator site, that finally can answer the query $q_c(s,t)$.

the returned partial answer is remarkably smaller than the initial partition size. In what follows, We start by introducing the notion of distributed graph, then we provide a detailed description on how partial evaluation is performed.

A. Distributed Graph

A partitioning Π of a graph G is defined as a pair (P, G_p), where P is a set of sub-graphs of $G = (V, E, L)$, as defined in Section III, and G_p is called the *partition graph* of Π, specifying edges across distinct sites with their head and tail nodes. Specifically, P and G_p are defined as:

1) $P = (P_1, ..., P_k)$, where each *partition* P_i is defined by $(V_i \cup V_i^{virt}, E_i \cup E_i^{cross}, L_i)$, with:
   - $V_i \subseteq V$,
   - (V_i, E_i, L_i) is the sub-graph of G induced by V_i,
   - if a node $u \in V_i$ is such that there is an edge $(u, v) \in E$ with $v \notin V_i$, then v is a *virtual node* in V_i^{virt},
   - E_i^{cross} is the set of *cross edges*, i.e. those edges (u, v) such that $u \in V_i$ and $v \notin V_i$.

   Moreover we define the set of *in-nodes* V_i^{in} as the set of those nodes in V_i having at least one predecessor in another partition. Formally, it is the set of nodes $u \in V_i$ such that there exists a cross edge $(v, u) \in E_j^{cross}$ with $j \neq i$. We generally refer to in-nodes and virtual nodes as the *boundary nodes* of a partition.
   Intuitively, for each partition P_i, $V_i \cup V_i^{virt}$ is the set of all nodes in V_i and, for each node in V_i having an edge to another partition, a virtual node indicating the connection. The set $E_i \cup E_i^{cross}$, instead, contains all the edges in E_i and those edges departing from a node in P_i that reach a node located in a different partition.
2) The partition graph G_p is defined as (V_p, E_p), in which
   - $V_p = \bigcup_{i\in[1,|P|]}(V_i^{virt} \cup V_i^{in})$, where the set $V_i^{virt} \cup V_i^{in}$ is the set of nodes in P_i having incoming cross-edges from different partitions or cross-edges reaching different partitions.
   - $E_p = \bigcup_{i\in[1,|P|]} E_i^{cross}$

In the first row of Figure 5, we show a distributed graph with three partitions: P_1, P_2 and P_3. Partition P_1 has no in-nodes, while $V_1^{virt} = \{1, 3\}$. For partition P_2, $V_2^{in} = \{1, 2\}$ and $V_2^{virt} = \{4\}$. Finally, $V_3^{in} = \{4, 3\}$ and $V_3^{virt} = \{2\}$. Note that the graph in Figure 5 is the same graph of Figure 4.

Algorithm 2 Distributed Control

```
/* executed at the coordinator site */

Input: Partitioning $(\Pi, G_p)$, query $q_c(s,t)$.
Output: The boolean answer to $q_c$ in G.
1:  post a query $q_c(s,t)$ for each partition in $\Pi$;
2:  $MGraph := \emptyset$;
3:  post query $q_c(s,t)$ to all the partitions in $\Pi$ ;
4:  for each partition $P_i \in \Pi$ do
5:      $X = \{s, t, V_i^{in}, V_i^{virt}\}$
6:      $[ans, R_i] = parallelReduction(P_i, q_c(s,t), X)$;
7:      if $ans \neq \emptyset$ then
8:          return ans;
9:      $MGraph := MGraph \cup R_i$;
10: $parallelReduction(MGraph, q_c(s,t))$;
11: return ans;
```

B. Distributed Algorithm

Assume that $Q = q_c(s,t)$ is posted on a site S_c, referred to as the **coordinator site**, in which a mapping m from the partitions in Π to different sites is stored.

1) *Distributing at site* S_c. Upon receiving query Q, the coordinator S_c posts Q to each site according to mapping m.
2) *Local evaluation* at each site S_i. Each site S_i evaluates (sub-queries) of Q in parallel, by processing the partition P_i stored in S_i as the known input to Q; the other partitions P_j are taken as the yet unavailable input, denoted by the untransformable nodes in V_i^{in} and V_i^{virt}. The partial answers are represented as a reduction of the graph P_i, and are sent back to S_c.
3) *Assembling at* S_c. Site S_c assembles and then reduces the received partial answers to get the final answer to Q.

The algorithm that performs the distributed evaluation of $q_c(s,t)$ is described by Alg. 2. The coordinator site invokes procedure *parallelReduction* (described in Section 4) on each site to compute partial answers. The reduced graph R_i must always contain the boundary nodes and, if contained in P_i, the source node and the target node. This is done by adding those nodes to the exclusion set passed to the procedure *parallelReduction*; i.e. $X = \{s, t, V_i^{virt}, V_i^{in}\}$. Unless any of the sites was able to independently produce the global answer to $q_c(s,t)$, all partial answers are collected and merged by the coordinator site that eventually invokes again *parallelReduction* on the merged partial answers to produce the final answer to $q_c(s,t)$.

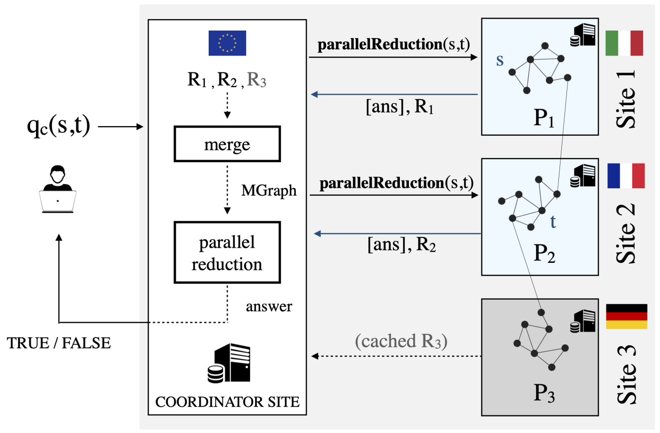


Figure 6: Upon receiving $q_c(s,t)$, the coordinator site forwards it to those sites storing either s or t. Sites return either the global answer ans or a reduced partition $R_{1/2}$. If none of the two sites was able to produce a global answer, R_1, R_2, together with any other pre-computed reduced partition (R_3), are merged and further reduced so as to produce the answer $q_c(s,t)$.

An important property of Algorithm 4 is highlighted at line 5: when a site does not contain s or t, its local evaluation depends only on the set of nodes in V_i^{in} and V_i^{virt}. These represent the control relationships between companies placed *at the borders* of partitions. If the control network is static (or has a slowly evolving dynamics) such control relationship can then be processed offline. Figure 6 highlights this property, as it shows that local evaluation takes place at the two sites 1 and 2 and that for all other sites (represented by site 3) the relevant information is precomputed offline.

Example. Figure 5 shows an example of how partial evaluation is applied to a distributed graph of three partitions. Procedure *parallelReduction* is first applied to P_1, P_2 and P_3. The resulting reduced partitions, R_1, R_2 and R_3 are collected by the coordinator. However, since P_2 does not contain s or t, R_2 could be pre-computed and already available at the coordinator site. The coordinator site merges the three reduced graphs into a single graph (*MGraph*) and applies *parallelReduction* to further reduce the graph and solve $q_c(s,t)$.

VIII. Implementation and Application

Towards an imminent production application of the parallel approach, we performed extensive experimentation with a realistic simulated version of the *EU graph* (Section VIII-A), which we also massively enriched with synthetic data to simulate extreme-scale settings, likely to arise in the near future. We also evaluated the approach against the real data from the *Italian ownership graph* and the *European Register of Intermediaries and Affiliates* (Section VIII-B). We analyzed the speedup led by the distributed approach and by pre-caching techniques (Section VIII-C). Finally, we compared our results with those obtained with a baseline serial algorithm and with a state-of-the-art graph processing engine (Section VIII-D).

The characteristics of all the graphs are in Section II. The algorithm and the architecture in Figure 6 have been implemented in Scala and are based on a Pregel-based computational model (GraphX [12]).

Configuration. All runs have been performed on a server equipped with two Intel® Xeon® E5-2650 processors at 2.00 GHz (32 hyper-threads in total) and 384 GB DDR3 RAM. The software stack includes Apache Spark 2.2.0.

A. Evaluating Distribution — the EU Graph

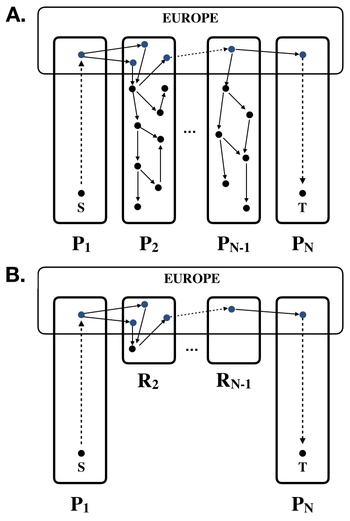
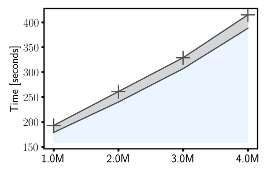


Figure 7: Overview of the distributed setting.

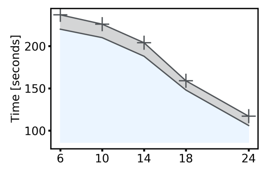
We generated a graph representing a real-world like Europe, assumed to contain 30 countries and, to allow more meaningful comparison between experimental results presented here, with the same number of companies in each country, assumed to be 5 million companies. In Figure 7 we show the distribution setting we assume. In Figure 7.A, we show partitions $P_1 \dots P_n$ representing the company graphs at each of the EU countries. Note that some nodes participate in cross-country links, e.g., a company in Italy may control a company in France or the other way around. When a query $q_c(s,t)$ is received at EU level, it is forwarded to all the n sites. Each of them computes *parallelReduction*. This reduced graph is submitted by each site to the EU level coordinator which on the union of the received reduced graphs, once again computes *parallelReduction* to obtain the final query result.

Note that at most two partitions contain s and t, whereas the remainder of partitions are query independent and therefore can be cached. Figure 7.B illustrates this latter setting, where we assume that P_1 resp. P_n contain s resp. t (where computation still needs to be performed before submitting to the European coordinator) whereas $P_2, \dots, P_{n-1}$ can be cached once computed (illustrated in Figure 7.B by the smaller areas R_i representing the reduced graph corresponding to P_i).

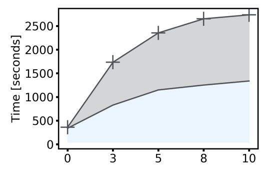
This setting allows for multiple scenarios, depending on the number of partitions. The real-world configuration is expected to be highly variable, with new companies being incorporated,



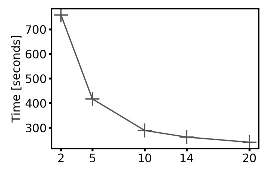
(a) varying the partition size



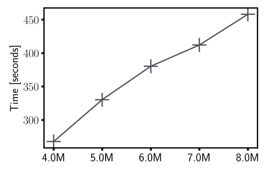
(b) varying the number of partitions



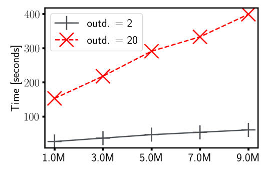
(c) varying the interconnection rate



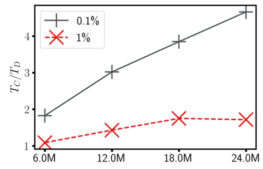
(d) varying the number of cores



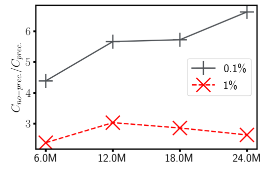
(e) varying the number of nodes



(f) varying the number of edges and density



(g) time speedup distributed vs centralized



(h) cost speedup with and without pre-computation

Figure 8: Experimental evaluation of parallelism (a-c) and distribution (d-h).

countries joining the distribution setting and others leaving, and the interconnection rate fluctuating. Let us consider the different cases.

Elapsed time varying the size of partitions. Figure 8.a shows how the execution time scales varying the size of each partition in terms of contained nodes (x-axis). The grey area under the curve represents the time spent at the coordinator site, while the light blue area represents the maximum execution time spent for partial result computation among all the involved sites. We can clearly observe a linear behaviour relative to the size of the partitions. This is consistent with what one would expect from the algorithm employed.

Elapsed time varying the number of partitions. We measured the influence of the number of partitions over the elapsed time in Figure 8.b. We still see a roughly linear behaviour here, with some deviations for the smaller number of partitions.

Elapsed time varying the interconnection rate. We measured the influence of the interconnection rate (the percentage of border nodes) on the elapsed times. In Figure 8.c, we observe that the higher the interconnection rate, the longer it takes to process the graph. Moreover, as the interconnection rate increases, most of the computation moves to the coordinator site; if all nodes in a partition were boundary nodes, all the computation would be performed at the coordinator.
In Europe, border nodes are approximately 1%, hence computation occurs mostly at the member states. Note that, assuming N partitions, each with M nodes, a 1% interconnection rate increase implies adding $0.01 \cdot M \cdot N$ edges to the overall graph, motivating the significant increase in execution time.

B. *Evaluating Parallelism – the Italian Graph and RIAD*

Elapsed time varying the number of cores. We measured the elapsed time of execution for the Italian graph, varying the number of available cores in the processing node, from 2 to 20 (Figure 8.d). We observe a roughly linear speed-up in the number of cores, with diminishing returns getting particularly stronger starting from 10 cores.

Elapsed time varying the number of nodes. We measured the elapsed time of execution for the Italian graph, varying the number of nodes in the graph from 4M to 8M. This reflects potential use cases, where different sets of companies are involved in the computation due to different properites (e.g., their legal form or size). As shown in Figure 8.e, doubling the number of nodes (from 4M to 8M), takes less than double the time (70% more time).

Elapsed time varying the number of edges and density of the graph. To assess extreme-scale settings, we measured the elapsed time of execution building synthetic scale-free graphs with increasing number of edges (x axis) and different densities, with out-degree of nodes from 2 to 20. As shown in Figure 8.f, in addition to the linear scalability with the number of edges, we can see that, dividing the out-degree by 10 (i.e. with a less denser graph) the execution time is 6 times faster.

Elapsed time with RIAD. The RIAD graph is smaller and less dense than the Italian and EU ones, yet, the computation of company control is a felt problem. The approach proposed in this work allows a runtime of 6.71 seconds. This is considered to be a good value by subject-matter experts.

C. *Evaluating speed-up wrt distribution and pre-caching*

Besides regulatory restrictions, we show that distribution has substantial performance advantage over centralization. We therefore conclude the experimental evaluation of our approach with further discussion of distribution vs. centralization and the pros of adopting data caching.

Speed-up of Distributed (w/o pre-caching) vs Centralized. We measured the speed-up in terms of elapsed time by partition size (x-axis) and for multiple interconnection rates of the distributed algorithm over the centralized one. The speedup is shown by plotting the ratio between the time spent to process the entire graph on a single site (T_C) and the distributed execution time T_D. Figure 8.g shows that with low interconnection rates (e.g., 0.1%) the distribution speed-up is relevant for small partitions and improves with larger ones (e.g., with $\sim 4.6x$ speed-up for $24M$-sized partitions). More realistic scenarios of company graphs exhibit higher interconnection rates (e.g., 1%), as we have seen: in these cases, although the speed-up is moderate for small partitions, it substantially improves with larger partitions and keeps stable, denoting a scalable behaviour of the approach.

Speedup of Distributed with pre-caching and w/o pre-caching. We measured the speed-up in terms of total computation cost by partition size (x-axis) and for multiple interconnection rates of the distributed algorithm with caching of query independent computations and without.
Again, as shown in Figure 8.h, an higher interconnection rate reduces the advantage of caching partial results, since a longer time is spent at coordinator.

Network traffic.

As argued in Section VII, sites storing partitions do not need to synchronize their execution and do not exchange data, The only data transfer consists in returning partial answers, mainly containing boundary nodes, to the master node. The following table shows the execution of the algorithm on a distributed graph partitioned across four different sites with 0.1% interconnection rate and variable partition size (from 4M to 8M nodes). Columns represent, in order: the average partition size (P); the average partial result size (R); the merged graph size (MGraph); the total network traffic in megabytes.

P (nodes | edges)	R (nodes | edges)	MGraph (nodes | edges)	Network Traffic
4M | 20M	8.2K | 7.3K	32.7K | 29K	1.2 MB
5M | 25M	10.3K | 9.3K	40.9K | 37K	1.5 MB
6M | 30M	12.3K | 11K	49.2K | 44.3K	1.8 MB
7M | 35M	14.7K | 15K	58.9K | 60K	2.4 MB
8M | 40M	18.1K | 21.8	72.5K | 87K	3.5 MB

D. Baseline Solution and Graph-native Systems

Speedup with respect to serial algorithm. We compared the parallel and distributed approach to CCP to a baseline version of the algorithm in production at the Bank of Italy for the RIAD graph, and which we consider our performance yardstick. On the RIAD graph, the parallel algorithm takes 6.71 seconds, i.e., $\sim 1/100$ of the time needed by the serial algorithm. We experimentally confirmed similar gain factors (from $1/60$ to $1/100$) with the synthetic settings. Artificially increasing the density beyond the one expected in real-world scenarios shows a reduced yet still significant (more than 60%) gain with respect to the baseline.

Comparison with Neo4J. Graph processing systems represent another yardstick to validate our approach. We encoded the CCP problem in *Cypher*, the query language of Neo4J, the most widely adopted graph management system. As Cypher is a navigational language, it adopts a limited form of recursion [1] which we experienced as insufficient to fully express CCP. Thus, we encoded in Cypher only the logic to detect all the paths needed for answering $q_c(s, t)$ and implemented a custom post-processing procedure taking them as input. We tested $q_c(s, t)$ by varying the number of nodes, edges and density as we previously did for our approach, and we measured only the Neo4J processing time, as a lower bound.

Figure 9.a-b, when compared with Figure 8.e-f, shows that under these assumptions our approach outperforms Neo4J and is more scalable for growing number of nodes, edges and graph density. In multiple cases we had to limit the exploration depth of Neo4J: runs with 7M nodes, with 9M edges and out-degree 2, and with 5M edges and out-degree 20 could not complete, while these cases are covered in Figure 8.e-f. Performance decay is due to the fact that Neo4J enumerates all the possible s-t paths. This experiment confirms that general purpose systems offer insufficient support to recursion for effectively computing CCP.

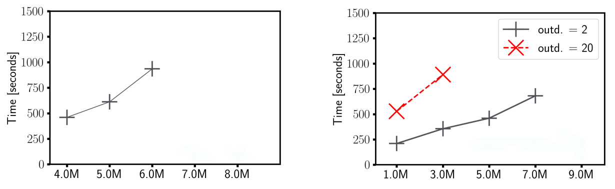
E. Use in Production and Lessons Learned

The parallel and distributed approach to CCP we have proposed is already used at the Bank of Italy for the production of internal research data products. It is part of a larger framework of solutions which will be imminently put into production for the delivery of external data products, some of which have been discussed in this paper. In particular, it is going to be the forefront to cope with the increasing volume and density of EU-level applications, unmanageable with the baseline algorithm.

Regulatory constraints make CCP a central problem. The experimental evaluation confirmed that a distributed solution though non-trivial is feasible in practice. Moreover, with volumes and densities resembling those of EU-level graphs, distribution leads to relevant performance gains, to be increasingly exploited. The graph density and the interconnection rate between partitions, reflecting the natural propensity of companies to establish national and cross-country links, respectively, are such as to undermine the approach effectiveness, although we clearly identified them as a key performance drivers.

IX. Related Work

The partial evaluation technique, on which the distribution of our algorithm relies on, has been used in compiler optimization [16], and querying XML trees [5]. Within the context of graph processing, the technique has been used to evaluate reachability queries [7], and graph simulation [18] [8] over graphs. Partial evaluation is used in Wenfei Fan et al [7] to solve a distributed reachability query q(s,t) over a master-slave architecture. In that case, partial results, which are represented by sets of boolean formulas, are computed sequentially within each node. Moreover, reachability is a much easier problem, NLOGSPACE-complete and with linear time complexity; it is



(a) varying the number of nodes (b) varying the number of edges and density

Figure 9: Experimental evaluation of CCP in Neo4J.

therefore highly parallelizable. Instead, the company control problem addressed in our work is PTIME-complete and the best known algorithms are quadratic, which is clearly harder than reachability and intrinsically non-parallelizable.

The work of Wenfei Fan [7] has been followed-up by [8], which basically extends the work to graph pattern matching queries, and by [9], where the authors present GRAPE a graph engine that, under certain conditions, allows to parallelize sequential algorithms such as graph traversal (e.g. reachability) and pattern matching algorithms. Others use a similar partial-evaluation based approach as in [7] ([21] [4] [15]). Again, none of the proposed solutions is suited for the complexity of company control queries.

A popular computational model for supporting efficient processing on large graphs is Pregel [20]. Programs in Pregel are expressed as a sequence of iterations, in each of which a node can receive messages sent by other nodes in the previous iteration, send messages to other nodes, modify its state or mutate the graph topology. The underlying architecture of Pregel consists of a master node and several worker node, each storing and processing a partition of the original graph. Messages sent to nodes belonging to different partitions imply a communication between different worker nodes. As pointed out in [7], the message passing model used in Pregel may serialize operations that can be conducted in parallel, and have no bound on the number of visits to each site. However Pregel can be combined with partial evaluation to support local processing of queries at each site, as we did, implementing the local evaluation procedure of our algorithm on top of Graphx [12], which is based on the computational model of Pregel.

X. Conclusion

In this paper, we provided a formal study of the company control problem and contributed a distributed and parallel algorithm, successful in handling the demand for high performance on graphs with millions of companies, possibly partitioned and maintained by different sites, even in the presence of access restrictions. We faithfully implemented an intrinsically parallel algorithm on top of a parallel framework. Thanks to our innovative methods, which exploit ad-hoc parallelism in a principled way, the problem can be solved with good performance in the real industrial setting, where thousands of control queries per minute can be asked.

This settings has one-to-one correspondence with multiple scenarios at EU level, featuring a set of national authorities or entities (e.g., national central banks, national statistical offices, etc.), each holding a specific partition of the company graph, and all facing the need to decide control relationships at super-national level. Many isomorphic scenarios to the company-control problem, with distributed graphs and control-like measures to be computed, exist and are of central interest. To the best of our knowledge, none of the existing solutions and systems adopts a distributed and parallel approach; existing solutions resort to ad-hoc combinations of the available local graphs and centralized processing, when feasible and allowed.

We believe that our algorithm could be at the basis of novel, more efficient systems for real settings at the EU level, as it could be key in unlocking the possibility to compute control-like measures that have been neglected so far, due to technical or organizational unfeasibility of centralized computation. Furthermore, our approach could drive a broader process change, towards relieving central institutions (e.g., the European Central Bank and Eurostat Register [6]) of centralized processing, while exploiting the computing power of local entities.

References

[1] Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J. L., and Vrgoc, D. Foundations of modern query languages for graph databases. *ACM Comput. Surv. 50*, 5 (2017), 68:1–68:40.

[2] Bellomarini, L., Sallinger, E., and Gottlob, G. The Vadalog system: Datalog-based reasoning for knowledge graphs. *VLDB 11*, 9 (2018).

[3] Bollobás, B., Borgs, C., Chayes, J., and Riordan, O. Directed scale-free graphs. In *SODA* (2003).

[4] Cheng, Y., Yuan, Y., Chen, L., Wang, G., Giraud-Carrier, C., and Sun, Y. Distr: A distributed method for the reachability query over large uncertain graphs. *IEEE Transactions on Parallel and Distributed Systems 27*, 11 (2016).

[5] Cong, G., Fan, W., Fan, W., and Kementsietsidis, A. Distributed query evaluation with performance guarantees. In *SIGMOD* (2007), ACM.

[6] Eurostat EuroGroup Register. https://cutt.ly/njJQoit.

[7] Fan, W., Wang, X., and Wu, Y. Performance guarantees for distributed reachability queries. *VLDB 5*, 11 (2012).

[8] Fan, W., Wang, X., Wu, Y., and Deng, D. Distributed graph simulation: Impossibility and possibility. *VLDB 7*, 12 (2014).

[9] Fan, W., Yu, W., Xu, J., Zhou, J., Luo, X., Yin, Q., Lu, P., Cao, Y., and Xu, R. Parallelizing sequential graph computations. *TODS 43*, 4 (2018).

[10] Garlaschelli, D., Battiston, S., Castri, M., Servedio, V., and Caldarelli, G. The scale-free topology of market investments. *Technical Report (https://arxiv.org/abs/cond-mat/0310503)* (2005).

[11] Glattfelder, J. B. *Ownership networks and corporate control: mapping economic power in a globalized world.* PhD thesis, ETH Zurich, 2010.

[12] Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., and Stoica, I. Graphx: Graph processing in a distributed dataflow framework. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)* (2014).

[13] Greenlaw, R., Hoover, H. J., Ruzzo, W. L., et al. *Limits to parallel computation: P-completeness theory*. Oxford University Press on Demand, 1995.

[14] Guideline (EU) 2011/14 of the ECB. https://cutt.ly/DjJQcup.

[15] Gurajada, S., and Theobald, M. Distributed set reachability. In *SIGMOD* (2016), ACM.

[16] Jones, N. D. An introduction to partial evaluation. *ACM Computing Surveys (CSUR) 28*, 3 (1996).

[17] Knuth, D. E. *The Art of Computer Programming, Volume 4*. Addison-Wesley Professional, 2006.

[18] Ma, S., Cao, Y., Huai, J., and Wo, T. Distributed graph pattern matching. In *WWW* (2012), ACM.

[19] Madras, N., and Slade, G. *The Self-Avoiding Walk*. Probability and Its Applications. Birkhäuser Boston, 1996.

[20] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. Pregel: a system for large-scale graph processing. In *SIGMOD* (2010), ACM.

[21] Peng, P., Zou, L., Özsu, M. T., Chen, L., and Zhao, D. Processing sparql queries over distributed rdf graphs. *VLDB 25*, 2 (2016).

[22] R., C. A. H., and Barabási, A. Scale-free networks. *Scholarpedia 3*, 1 (2008).

[23] Guideline (EU) 2018/876 of the ECB. https://cutt.ly/8jJQYys.

[24] Romei, A., Ruggieri, S., and Turini, F. The layered structure of company share networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* (2015), IEEE.