Formal Verification & Analysis of Graph Databases

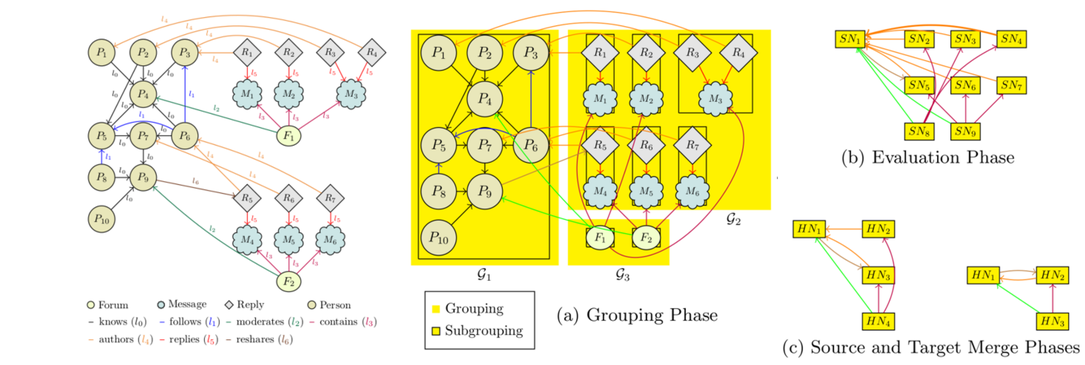
Stefania Dumbrava
ENSIIE & Methods Team, Samovar Laboratory (Institut Polytechnique de Paris)
stefania.dumbrava@ensiie.fr

Graph Models & Database Management Systems

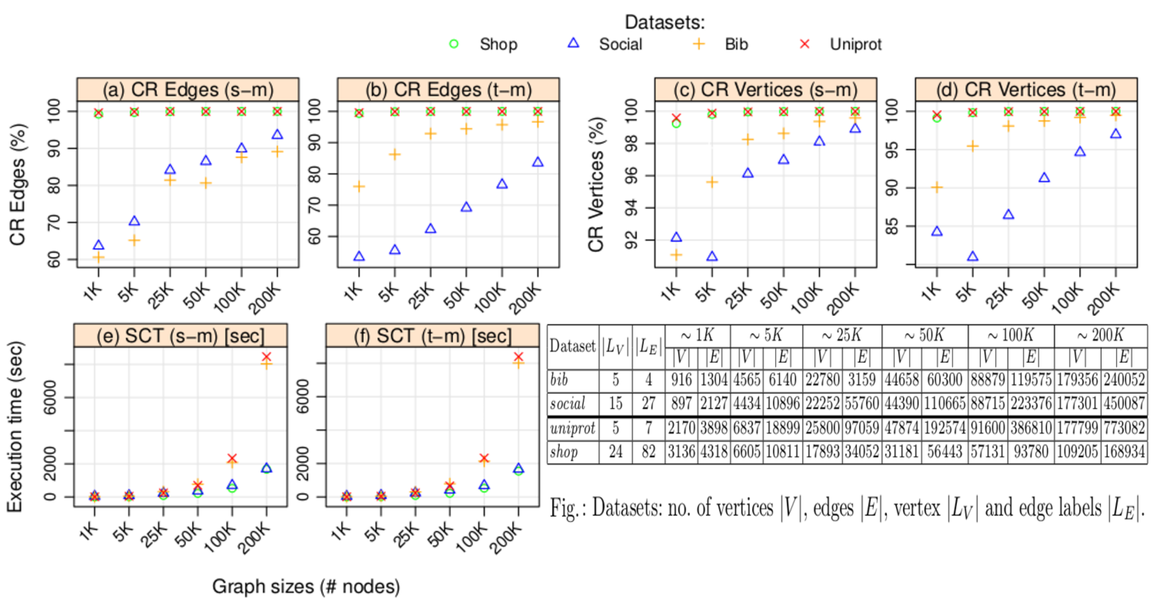
Graphs are semantically rich data models able to inherently capture the structure of complex objects and their interconnecting relationships. Due to their high expressiveness, graphs are used in numerous domains, including Knowledge Representation and the Semantic Web, Linked Open Data, geolocation data, as well as in life science repositories. Graph models cover the spectrum from simple, edge-labeled to property enriched, on both nodes and edges, and lie at the foundation of modern *graph database systems*. These systems leverage *native graph storage* and *index-free adjacency* to replace the costly table joins from the relational setting with efficient graph traversals. We overview recent work on computing compact abstractions of graph datasets and on using these to approximate the result of intractable path query fragments ([1-4]). We also highlight the implementation and formal verification of a prototype graph database engine capable of evaluating regular queries. To this end, we rely on a fragment of the Datalog logic programming language as a unifying formalism for the multitude of graph query dialects that currently underpin commercial implementations and that are yet to be standardized ([5-6]).

Graph Summarization & Approximate Query Processing

GRASP Summaries. The compact GRASP representation of a property graph $\mathcal{G}$ is computed as follows. A *grouping phase* partitions it into sub-groupings, based on the connectivity of its most frequent edge labels. An *evaluation phase* collapses the vertices and inner-edges of each maximally label-connected subgrouping. Finally, the source/target *merge phase* further collapses the common start/end nodes of same labeled edges.



Results. Experimental studies on datasets varying in density and label heterogeneity convey *high compression ratios* (CR) ($\approx$ 85% for graphs over 200K nodes) and *low summary computing times* (SCT), especially for sparser graphs, which also exhibit low sensitivity to size variations.



Dataset	$\lvert L_V\rvert$	$\lvert L_E\rvert$	$\sim 1K$		$\sim 5K$		$\sim 25K$		$\sim 50K$		$\sim 100K$		$\sim 200K$	
			$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert E\rvert$
bib	5	4	916	1304	4565	6140	22780	3159	44658	60300	88879	119575	179356	240052
social	15	27	897	2127	4434	10896	22252	55760	44390	110665	88715	223376	177301	450087
uniprot	5	7	2170	3898	6837	18899	25800	97059	47874	192574	91600	386810	177799	773082
shop	24	82	3136	4318	6605	10811	17893	34052	31181	56443	57131	93780	109205	168934

Fig.: Datasets: no. of vertices $\lvert V\rvert$, edges $\lvert E\rvert$, vertex $\lvert L_V\rvert$ and edge labels $\lvert L_E\rvert$.

Approximate Graph Querying (AQP). We approximate the evaluation of graph queries Q over $\mathcal{G}$ with that of their translation over the GRASP summarization of $\mathcal{G}$. We target queries that *count pair-wise label-constrained reachability*, e.g., `SELECT COUNT(*) MATCH` Q_i, as shown in the below table. These are frequent and intractable (their exact evaluation is #P-complete), hence a good case-study for designing efficient approximation heuristics.

ID	Query Body	Approx. Answer		Rel. Error (%)		Runtime (ms)	
		SumRDF	APP	SumRDF	APP	SumRDF	APP
Q_1	`(x0)-[:producer]->()<-[:paymentAccepted]-(x1)`	75	76	1.32	0.00	136.30	38.2
Q_2	`(x0)-[:totalVotes]->()<-[:price]-(x1)`	42.4	44	3.64	0.00	50.99	17
Q_3	`(x0)-[:jobTitle]->()<-[:keywords]-(x1)`	226.7	221	2.51	0.18	463.85	12.8
Q_4	`(x0)<-[:title]-()-[:performedIn]->(x1)`	19.5	20	2.50	0.00	831.72	8.8
Q_5	`(x0)-[:artist]->()<-[:employee]-(x1)`	143.3	133	7.19	0.37	196.77	10.6
Q_6	`(x0)-[:follows]->()<-[:editor]-(x1)`	524	528	0.38	0.48	1295.83	19

Results. Comparing our approach APP [4] with the state-of-the-art SumRDF AQP engine, we record an average relative estimation error of 0.15%. vs. 2.5% and an average query runtime of only 27.55 ms vs. 427.53 ms.

Formally Verified Graph Querying

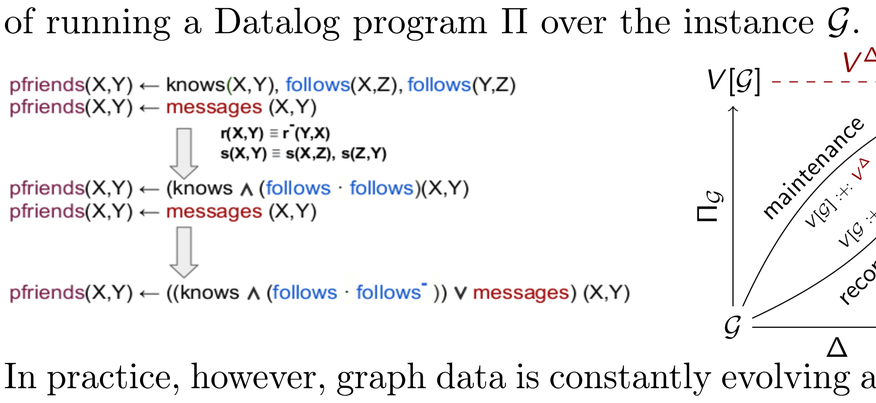
Graph queries used in commercial implementations are subsumed by a logic programming language called Datalog, which extends the expressivity of conjuctive queries with recursion. It consists of a set of facts and of if-then-else rules, allowing to *infer* new facts from existing ones.

A Datalog program Π is evaluated by instantiating the heads of each rule with substitutions ν that uniformly match its body atoms against a set of facts I. The minimal model $\mathrm{MM}(\Pi)$ corresponds to the fixed-point iteration of $T_\Pi(I) = I \cup \{\nu(\mathsf{head}) \mid \mathsf{head} \leftarrow \mathsf{body} \in P \wedge \nu(body) \subseteq I\}$.

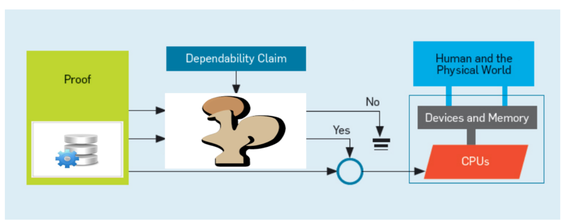
$$\Pi = \begin{cases} R_1(a,b). \\ R_2(b). \\ R_2(X) \leftarrow R_1(X,Y), R_2(Y). \end{cases}$$

$$T_\Pi \uparrow^1 = T_\Pi(\emptyset) = \{R_1(a,b), R_2(b)\}$$
$$T_\Pi \uparrow^2 = T_\Pi(T_\Pi \uparrow^1) = \{R_1(a,b), R_2(b), R_2(a)\}$$
$$\vdots$$
$$T_\Pi \uparrow^\omega = T_\Pi(T_\Pi \uparrow^{\omega-1}) = \mathrm{lfp}(T_\Pi) = \mathrm{MM}(\Pi)$$

Graph Queries in Datalog. A graph $\mathcal{G}$ can be seen as database instance by considering edge labels to be binary predicates. Datalog rules can then capture reachability through complex paths, as computed by `pfriends` below. These can represent unions, conjunctions, compositions, and inverses of paths with labels belonging to a regular language. Querying for pair-wise label constrained reachability thus amounts to *evaluating* the result $V[\mathcal{G}]$ of running a Datalog program Π over the instance $\mathcal{G}$.



In practice, however, graph data is constantly evolving and prone to updates Δ, i.e., to edge additions and removals. To avoid the costly recomputation of query results when a graph instance $\mathcal{G}$ is thus modified into $\mathcal{G}$:+: Δ, one can instead *maintain* previously computed results. This amounts to incrementally updating the existing result, in order to produce the same answer as a full recomputation (see above diagram). The correctness of the algorithm amounts to proving that: *If* $V[\mathcal{G}] \models \Pi_\mathcal{G}$, *then the engine outputs an incremental update* V^Δ, *such that* $V[\mathcal{G}]$:+: $V^\Delta \models \Pi_{\mathcal{G} :+: \Delta}$. We have used the Coq proof assistant to model the specification of a graph database that builds on the edge-labeled model and have implemented and formally proved an inference engine capable of both *incremental evaluation & maintenance* of graph queries belonging to the regular Datalog fragment.



Results. We have fully specified & verified our engine in less than 2K loc of Coq code [6]. We also extracted a *correct-by-construction* OCaml engine, which we ran on realistic graph datasets that we synthetically generated.

References

[1] A. Bonifati, S. Dumbrava. 2018. *Graph Queries: From Theory to Practice*. In Sigmod Record 47(4), p. 5–12.

[2] S. Dumbrava, A. Bonifati, A. Nazabal, R. Vuillemont. 2019. *Approximate Evaluation of Complex Queries on Property Graphs*. In Proc. of the 13th International Conference on Scalable Uncertainty Management (SUM), p. 250-265.

[3] A.Bonifati, S.Dumbrava, H. Kondylakis. 2021. *Graph Summarization*. Encyclopedia of Big Data Technologies.

[4] GRASP System. `https://github.com/grasp-algorithm`.

[5] A. Bonifati, S. Dumbrava, E. Gallego. 2018. *Certified Graph View Maintenance with Regular Datalog*. Theory and Practice of Logic Programming Journal, 18(3-4):372–389.

[6] VerdiLog System. `https://github.com/VerDILog`.