MARITA: Mining Approximate Rules in Tables

Pierre-Henri Paris[1], François Amat[2], Paolo Papotti[3] and Fabian Suchanek[2]

[1]Paris-Saclay University, CNRS, LISN, France
[2]Telecom Paris and Institut Polytechnique de Paris, France
[3]EURECOM, France

Abstract

Rule mining has been extensively studied for RDF knowledge bases. However, it remains difficult on multi-table relational databases, because the search space grows rapidly with the number of relations and attributes. We present MARITA, an approach for mining approximate Horn rules directly from relational databases. MARITA combines a compact, non-redundant rule representation with foreign-key-guided joins and a novel *relation-disjoint semantics* that prevents repeated relation occurrences from being satisfied by the same tuple. Its search procedure exploits Horn-specific pruning and evaluates candidate rules directly in SQL. Experiments on 83 public relational benchmarks show that MARITA frequently runs in subsecond time and achieves speed-ups of up to $10^4\times$ over existing systems.

Keywords

Rule mining, relational databases, Horn rules

1. Introduction

Horn rules can capture patterns in structured data, such as, e.g., the observation that married partners usually live in the same place: $married(x, y) \wedge livesIn(x, z) \Rightarrow livesIn(y, z)$. Such rules serve a variety of purposes: They can be used to impute missing data, to flag inconsistencies, to find exceptions to general phenomena, to propose constraints, or to extract general insights about the data [1]. Finding such rules automatically is called *rule mining*.

Several successful rule mining approaches exist for RDF knowledge bases, with prominent systems being AMIE [2], Rudik [3], and AnyBURL [4]. These systems use different strategies to scale to millions of triples. However, rule mining is so far much less successful on relational databases: Systems such as Metanome [5], SPIDER [6], or FastFDs [7] mine only specific sub-types of rules such as functional dependencies or inclusion dependencies. Systems that mine more general rules (*inductive logic programming* approaches), such as Popper [8] or FOIL [9] are powerful, but require significant amounts of working memory. Even MATILDA [10], our own previous system for mining Tuple-generating dependencies (TGDs) on databases, struggles on larger databases. This is not surprising: Rule mining scales exponentially with the number of columns and is thus inherently expensive [11].

In this paper, we propose MARITA[1], an approach that modifies MATILDA to mine Horn rules instead of TGDs. In the MATILDA evaluation, 96% of the mined rules were Horn rules anyway [10], and we thus do not lose much by focusing on this class. We do not just optimize the MATILDA algorithm for the case of Horn rules, but we also introduce a relation-disjoint semantics, by which repeated occurrences of the same relation that use different primary-key variables must be witnessed by tuples with different primary-key values. Our experiments show that MARITA finds these rules orders of magnitude faster than competitors.

ISWC 2026 Companion Volume, October 25–29, 2026, Bari, Italy
pierre-henri.paris@universite-paris-saclay.fr (P. Paris); i.tiddi@vu.nl (F. Amat); papotti@eurecom.fr (P. Papotti); i.tiddi@vu.nl (F. Suchanek)
https://phparis.net/ (P. Paris); https://kmitd.github.io/ilaria/ (F. Amat); https://papotti.eurecom.io/ (P. Papotti); https://kmitd.github.io/ilaria/ (F. Suchanek)
0000-0002-9665-1187 (P. Paris); 0000-0001-7116-9338 (F. Amat); 0000-0003-0651-4128 (P. Papotti); 0000-0001-7116-9338 (F. Suchanek)

[1]Mining Approximate Rules In TAbles

MARRIED	**Partner1**	Partner2
	Elvis	Priscilla
	Priscilla	Elvis

RESIDENCE	**Person**	Zip	City
	Elvis	37501	Memphis
	Priscilla	37501	Memphis
	Lisa	37502	Memphis

Figure 1: A toy database, keys in bold

2. Problem Statement

We refer the reader to [10, 12, 13] for a definition of relational databases, relations, schemas, keys, attributes, and foreign keys. Figure 1 shows a simple database with two tables, 5 attributes, and 2 keys (in bold). Here, MARRIED.PARTNER1 and MARRIED.PARTNER2 are foreign keys for RESIDENCE.PERSON. In what follows, we consider a fixed database instance, and write $r(a_1, ..., a_n)$ to mean that the tuple $\langle a_1, ..., a_n \rangle$ is in the relation r in that database instance (we say the relation *holds* for that tuple). We work under the closed world semantics, meaning that $r(a_1, ..., a_n)$ is false iff the tuple is not in r. We do not consider NULL values. A *Horn rule* r is a universally quantified implication $B_1(\vec{x}_1) \wedge \cdots \wedge B_m(\vec{x}_m) \Rightarrow H(\vec{y})$, where $B_1, \ldots, B_m, H$ are relations, each component is called an *atom*, the part left of the arrow is the *body*, and the right part is the *head*. In our toy example, the rule "married people live in the same place" is

$$married(x, y) \wedge residence(x, z, c) \Rightarrow residence(y, z, c)$$

All variables in the head must occur in the body. The *length* of r is $|r| = m + 1$. We consider only *foreign-key joinable* rules, i.e., all different attributes that are joined by the same variable are linked by a foreign key [10]. We further focus on rules that are *safe*, i.e., all variables of the head atom appear also in at least one body atom, and that are *connected*, i.e., each atom is transitively connected to each other atom by shared variables [2]. Two rules are *equivalent* if one can be obtained from the other by renaming variables, and permuting body atoms without changing the head atom, the multiset of body atoms, or the induced join equalities. The *support* of a rule in a database instance is the number of assignments from variables in the head of the rule to constants for which there exists an assignment for the remaining variables so that all atoms hold in the instance. The support of our example rule is 2, because there are two assignments of y, z, c that can be extended to make all atoms of the rule hold.

The *proto-rule* of a rule is the conjunction of its atoms. A *recursive proto-rule* is a proto-rule where at least one relation appears more than once. We index each relation by its occurrence number, so that a recursive proto-rule takes the form $\bigwedge_r \bigwedge_{i=1}^{n_r} r_i(\vec{x_{r,i}})$. In our example, the protorule is $married_1(x, y) \wedge residence_1(x, z, c) \wedge residence_2(y, z, c)$.

Given a database instance D, a support threshold κ, and length bound L, *rule mining* is the task of returning safe and connected Horn rules r such that (i) all joins in r correspond to foreign keys, (ii) r is not equivalent to a previously returned rule, (iii) $|r| \leq L$, and (iv) the support of r in D is at least κ.

3. MARITA

To mine rules, MARITA takes inspiration from MATILDA: It constructs a *candidate graph*, where each node is a pair of indexed attribute occurrences of the form $\langle t_i.a, t_{i'}'.a' \rangle$. Each node thus corresponds to a join condition in the rule, and we write such a node as $t_i.a = t_{i'}'.a'$. In our example rule from Section 2, one join condition is $residence_1.zip = residence_2.zip$. An edge connects an attribute pair $t_i.a = t_{i'}'.a'$ to an attribute pair $t_{i''}''.a'' = t_{i'''}'''.a'''$ if a relation occurrence of the first pair appears in the second pair, or if an attribute from the first pair is joinable with an attribute from the second pair. Traversing an edge thus corresponds to adding a join condition to the rule. Traversing the entire graph in a systematic fashion enumerates all possible protorules [10].

MARITA departs in three ways from MATILDA: First, the transformation of protorules into rules is simpler. As MATILDA mines full-fledged TGDs, it has to consider all subsets of atoms of the protorule

for the head. MARITA, which mines only Horn rules, can limit itself to choosing each atom of the protorule as a head atom to create as many Horn rules per protorule as there are atoms in it. This allows for pruning, as follows: Any rule with a support lower than the given support threshold allows pruning all rules that have the same head atom and contain all body atoms – an optimization that was not possible in full generality in MATILDA.

Second, the focus on Horn rules revealed a crucial shortcoming that impacts not just MATILDA, but also AMIE[2]: In the calculation of support, different occurrences of the same relation can have the variables bind to the same values. Take, e.g., the rule "People who live in the same city have the same ZIP code": $\mathit{residence}(p, z, c) \wedge \mathit{residence}(p', z', c) \Rightarrow \mathit{residence}(p', z, c)$. Intuitively, this rule should have a support of 2 in our toy example (because two people share city and ZIP code). But it has a support of 3. This is because p' can be bound to any of the three people in the table Residence, and p can simply be bound to that same value. To prevent repeated occurrences that use different primary-key variables from collapsing onto the same database tuple, we define the *relation-disjoint semantics*:

Definition 1. Let ϕ be a recursive proto-rule. For each relation r occurring in ϕ, let its occurrences be $r_1(\vec{x}_{r,1}), \ldots, r_{n_r}(\vec{x}_{r,n_r})$. Let $\mathrm{pk}(r)$ denote the set of primary-key attributes of r. For two occurrences $i < j$, define

$$K_{r,i,j} = \left\{ k \in \mathrm{pk}(r) \;\middle|\; \vec{x}_{r,i}[k] \text{ and } \vec{x}_{r,j}[k] \text{ are distinct variables} \right\}.$$

The relation-disjoint condition of ϕ is

$$DS(\phi) = \bigwedge_{r} \bigwedge_{\substack{1 \le i < j \le n_r \\ K_{r,i,j} \neq \emptyset}} \left(\bigvee_{k \in K_{r,i,j}} \vec{x}_{r,i}[k] \neq \vec{x}_{r,j}[k] \right).$$

Thus, if two occurrences of the same relation use different variables in at least one primary-key position, their primary-key values must differ in at least one such position. If they use identical variables in all primary-key positions, no additional constraint is introduced for that pair. The relation-disjoint support of a rule is the number of head assignments that can be extended to satisfy all rule atoms together with $DS(\phi)$, where ϕ is the proto-rule of the rule.

For our example rule, the proto-rule is

$$\phi = \mathit{residence}_1(p, z, c) \wedge \mathit{residence}_2(p', z', c) \wedge \mathit{residence}_3(p', z, c).$$

The only primary-key attribute of *residence* is the person. For the pair $(\mathit{residence}_1, \mathit{residence}_2)$, the primary-key variables are p and p', so the condition is $p \neq p'$. The same condition is obtained for $(\mathit{residence}_1, \mathit{residence}_3)$. For $(\mathit{residence}_2, \mathit{residence}_3)$, both primary-key positions contain the same variable p', so $K_{\mathit{residence},2,3} = \emptyset$ and no additional constraint is introduced. Hence

$$DS(\phi) = (p \neq p') \wedge (p \neq p') \equiv p \neq p'.$$

Adding this condition to the computation of support yields the desired value, 2.

Finally, MARITA departs from MATILDA by a completely revamped code bases. The system is implemented as a Python library and has no dependency on an external rule-mining system. Most notably, MARITA interfaces the input database through SQL, so that the system can work on any relational database without requiring any preprocessing.

Theoretical guarantees. MARITA inherits the completeness guarantee of MATILDA [10]. The algorithm enumerates all safe and connected protorules joined by foreign keys with a non-zero support. Since MARITA considers every atom of a protorule as head, the system enumerates all Horn rules in the hypothesis space, up to equivalence. Relation-disjointness affects the evaluation of candidate rules but not their enumeration: MARITA still considers every atom of each enumerated proto-rule as a possible head. In our experiments, support is computed using the relation-disjoint condition above.

[2]For AMIE the problem was not very important, because in knowledge bases it appears only with rule lengths greater than 3, which were previously shown to be less informative [14] and are thus generally not considered.

4. Evaluation

We evaluate MARITA on 83 multi-table datasets from the CTU Prague Relational Learning Repository [15], migrated to `SQLite`. All experiments run on a server with an Intel Xeon E5-2650 v2 @ 2.60 GHz processor, with a one-hour time limit, a 10 GB memory cap per job, a maximum rule length of 3, a maximum of 6 variables, and a minimal support of one. We compare against Popper [8], AMIE 3 [16], SPIDER [6], and MATILDA [10]. Table 1 reports representative results for a sample of ten databases (the complete benchmark results and all experimental settings are available on our GitHub repository). Our sample illustrates the range of outcomes across systems, including cases where runtimes vary substantially or a system times out or runs out of memory, showing that there is no uniform success and no uniformly fastest runtime for any one system.

Table 1
Representative benchmark results for ten datasets with their number of columns (#C). Time is in seconds; #R is the number of rules; OOM is Out of memory; OOT is Timeout

		Popper		Spider		AMIE 3		MATILDA		MARITA	
Database	#C	#R	Time	#R	Time	#R	Time	#R	Time	#R	Time
Biodegradability	14	0	20.55	7	1.58	2	3.74	OOT	3600.10	39	48.00
Bupa	16	0	17.09	58	1.07	0	1.04	2327	94.89	83	3.62
Carcinogenesis	23	OOT	3601.03	25	2.08	8	3.61	OOT	3600.45	198	493.20
CDESchools	90	OOM	302.30	55	12.43	834	94.05	OOT	3600.00	7	9.93
classicmodels	59	OOM	275.81	132	2.11	31	2.66	OOT	3600.52	59	5.76
Countries	67	OOM	298.75	11	11.86	67579	313.90	OOM	1709.95	15	45.94
Dunur	34	0	28.56	164	1.60	0	0.52	25164	3523.45	994	11.69
PTE	76	OOT	3601.03	300	2.11	0	3.61	OOT	3600.17	1813	207.26
tpcd	61	OOM	734.58	OOM	696.35	OOM	742.23	OOM	805.83	0	64.99
tpcds	425	OOM	1387.03	OOM	1383.50	OOM	1652.11	OOM	1539.00	OOM	104.23

Runtime results are not directly comparable, because the systems are designed to return different kinds of rules: SPIDER targets unary inclusion dependencies (and also returns dependencies without support in the data), AMIE 3 targets graph-level predicate associations after relational-to-RDF conversion, Popper searches a broader ILP space, and MATILDA mines approximate TGDs, which are a superset of Horn rules. To speed up MATILDA, we configured it to return only rules with a single head atom (it still returns rules where the head atom has an existential variable). MARITA targets non-vacuous, foreign-key–joinable, safe, connected, and non-duplicate Horn rules with a non-zero support. Its smaller rule sets are therefore expected. When we restrict the outputs of the other systems to MARITA's type of rules, only 825 rules remain in total across all competitors. We conducted an audit of these rules and found that MARITA indeed finds all of them (up to variable renaming). The rules not recovered by MARITA were predominantly either outside our foreign-key-joinable hypothesis class (165,771 rules) or vacuous self-witnessing patterns (13,115). We also disabled the disjoint semantics on the Biodegradability database for ablation. At $N = 2$, the disjoint version of MARITA finds 7 rules and terminates, while the non-disjoint version finds 12 rules and then times out. This shows that relation-disjointness does not just suppress redundant self-joins, but also reduces the search space measurably.

5. Conclusion

MARITA shows that Horn rules can be mined not just on RDF knowledge bases, but also on database tables. This becomes possible by focusing on foreign-key joinability and by relying on our newly introduced disjoint semantics. Future work will investigate how MARITA could work directly with RDF input, and how the rules can be used in R2RML [17] or SHACL [18]. All code and data, as well as example rules, all mined rules, and all experimental settings are available at https://github.com/PHParis/MARITA.

Acknowledgements. This work was partially supported by the Hi! PARIS, the ANR France 2030 program (ANR-23-IACL-0005), and the IA-cluster project (ANR-23-IACL-0001).

Declaration of use of generative AI

During the preparation of this work, the authors used ChatGPT for grammar and spelling checks and OpenAI Codex for the code. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] P. Betz, L. Galarraga, S. Ott, C. Meilicke, F. M. Suchanek, H. Stuckenschmidt, PyClause - Simple and Efficient Rule Handling for Knowledge Graphs, in: IJCAI demo track, 2024.

[2] L. Galárraga, C. Teflioudi, K. Hose, F. M. Suchanek, AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases , in: WWW, 2013.

[3] S. Ortona, V. V. Meduri, P. Papotti, Robust discovery of positive and negative rules in knowledge bases, in: ICDE, IEEE Computer Society, 2018, pp. 1168–1179. doi:10.1109/ICDE.2018.00108.

[4] C. Meilicke, M. W. Chekol, D. Ruffinelli, H. Stuckenschmidt, An introduction to anyburl, in: KI 2019: Advances in Artificial Intelligence: 42nd German Conference on AI, Kassel, Germany, September 23–26, 2019, Proceedings 42, Springer, 2019, pp. 244–248.

[5] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, F. Naumann, Data profiling with metanome, Proc. VLDB Endow. 8 (2015) 1860–1863. URL: http://dx.doi.org/10.14778/2824032.2824086. doi:10.14778/2824032.2824086.

[6] J. Bauckmann, U. Leser, F. Naumann, V. Tietz, Efficiently detecting inclusion dependencies, in: Proceedings of the IEEE International Conference on Data Engineering Workshops, 2007, pp. 1448–1455.

[7] C. Wyss, C. Giannella, E. Robertson, Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract, in: International Conference on Data Warehousing and Knowledge Discovery, Springer, 2001, pp. 101–110.

[8] A. Cropper, R. Morel, Learning programs by learning from failures, 2020. URL: https://arxiv.org/abs/2005.02259. arXiv:2005.02259.

[9] J. R. Quinlan, Learning logical definitions from relations, in: Machine Learning, volume 5, 1990, pp. 239–266.

[10] F. Amat, P.-H. Paris, P. Papotti, F. M. Suchanek, MATILDA: Mining Approximate Tuple-Generating Dependencies in Large Databases, in: TIST, 2026.

[11] X. Chu, I. F. Ilyas, P. Papotti, Discovering denial constraints, Proc. VLDB Endow. 6 (2013) 1498–1509. URL: http://www.vldb.org/pvldb/vol6/p1498-papotti.pdf. doi:10.14778/2536258.2536262.

[12] S. Abiteboul, R. Hull, V. Vianu, Foundations of databases, volume 8, Addison-Wesley Reading, France, Rocquencourt, 1995.

[13] E. F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (1970) 377–387. URL: https://doi.org/10.1145/362384.362685.

[14] L. Galárraga, C. Teflioudi, K. Hose, F. M. Suchanek, Fast Rule Mining in Ontological Knowledge Bases with AMIE+ , in: VLDBJ, 2015.

[15] J. Motl, O. Schulte, The ctu prague relational learning repository, 2024. URL: https://arxiv.org/abs/1511.03086. arXiv:1511.03086.

[16] J. Lajus, L. Galárraga, F. M. Suchanek, Fast and Exact Rule Mining with AMIE 3, in: ESWC, 2020.

[17] S. Das, S. Sundara, R. Cyganiak, R2RML: Rdb to rdf mapping language, W3C Recommendation, 2012. URL: https://www.w3.org/TR/r2rml/.

[18] H. Knublauch, D. Kontokostas, Shapes constraint language (SHACL), W3C Recommendation, 2017. URL: https://www.w3.org/TR/shacl/.