Expected Shapley-Like Scores of Boolean Functions: Complexity and Applications to Probabilistic Databases

Pratik Karmakar[1,3] Mikaël Monet[2,5]
Pierre Senellart[1,2,3,6,7] Stéphane Bressan[1,4,7]

[1]CNRS@CREATE LTD, Singapore

[2]Inria, Lille & Paris, France

[3]DI ENS, ENS, PSL University, CNRS, Paris, France

[4]National University of Singapore, Singapore

[5]CRIStAL, Université de Lille, CNRS, Lille, France

[6]Institut Universitaire de France, Paris, France

[7]IPAL, CNRS, Singapore, Singapore

Dagstuhl, 18 January 2024

Motivation

Practical Motivation

- Power indices (Shapley, Banzhaf, etc.) [Laruelle, 1999]: reasonable ways to quantify the responsibility of a data item for a complex task such as query evaluation

Motivation

Practical Motivation

- Power indices (Shapley, Banzhaf, etc.) [Laruelle, 1999]: reasonable ways to quantify the responsibility of a data item for a complex task such as query evaluation
- Real data: marred with uncertainty, which may be represented by probability distributions

Motivation

Practical Motivation

- Power indices (Shapley, Banzhaf, etc.) [Laruelle, 1999]: reasonable ways to quantify the responsibility of a data item for a complex task such as query evaluation
- Real data: marred with uncertainty, which may be represented by probability distributions
- But how to assess responsibility of data items when they are both uncertain and involved in a complex task?

Motivation

Practical Motivation

- Power indices (Shapley, Banzhaf, etc.) [Laruelle, 1999]: reasonable ways to quantify the responsibility of a data item for a complex task such as query evaluation
- Real data: marred with uncertainty, which may be represented by probability distributions
- But how to assess responsibility of data items when they are both uncertain and involved in a complex task?

Theoretical Motivation

- Tractability landscape of Shapley value computation and probabilistic query evaluation strikingly similar

Motivation

Practical Motivation

- Power indices (Shapley, Banzhaf, etc.) [Laruelle, 1999]: reasonable ways to quantify the responsibility of a data item for a complex task such as query evaluation
- Real data: marred with uncertainty, which may be represented by probability distributions
- But how to assess responsibility of data items when they are both uncertain and involved in a complex task?

Theoretical Motivation

- Tractability landscape of Shapley value computation and probabilistic query evaluation strikingly similar
- Can we do Shapley value computation on top of probabilistic databases tractably if we can do probabilistic query evaluation tractably?

Shapley-Like Scores

- V: finite set of Boolean variables
- $\varphi : 2^V \to \{0,1\}$ Boolean function over V
- $c : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$: coefficient function (assumed to have PTIME evaluation when input in unary)

Shapley-Like Scores

- V: finite set of Boolean variables
- $\varphi : 2^V \to \{0,1\}$ Boolean function over V
- $c : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$: coefficient function (assumed to have PTIME evaluation when input in unary)

$$\mathsf{Score}_c(\varphi, V, x) \stackrel{\text{def}}{=} \sum_{E \subseteq V \setminus \{x\}} c(|V|, |E|) \times [\varphi(E \cup \{x\}) - \varphi(E)].$$

Shapley-Like Scores

- V: finite set of Boolean variables
- $\varphi : 2^V \to \{0,1\}$ Boolean function over V
- $c : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$: coefficient function (assumed to have PTIME evaluation when input in unary)

$$\mathsf{Score}_c(\varphi, V, x) \stackrel{\text{def}}{=} \sum_{E \subseteq V \setminus \{x\}} c(|V|, |E|) \times [\varphi(E \cup \{x\}) - \varphi(E)].$$

Example

- $c_{\text{Shapley}}(k, \ell) \stackrel{\text{def}}{=} \frac{\ell!(k-l-1)!}{k!} = \binom{k-1}{l}^{-1} k^{-1}$: Shapley value [Shapley et al., 1953]
- $c_{\text{Banzhaf}}(k, \ell) \stackrel{\text{def}}{=} 1$: Banzhaf value [Banzhaf III, 1964]
- $c_{\text{PB}}(k, \ell) \stackrel{\text{def}}{=} 2^{-k+1}$: Penrose–Banzhaf power [Kirsch and Langner, 2010]

Probabilistic Setting

- Product distribution on Boolean variables, $\Pr(x) \in [0, 1]$ for $x \in V$ (i.e., every Boolean variable is assumed to be independent)

Probabilistic Setting

- Product distribution on Boolean variables, $\Pr(x) \in [0, 1]$ for $x \in V$ (i.e., every Boolean variable is assumed to be independent)
- For $Z \subseteq V$,
$\Pr(Z) \stackrel{\text{def}}{=} \left(\prod_{x \in Z} \Pr(x) \right) \times \left(\prod_{x \in V \setminus Z} (1 - \Pr(x)) \right)$

Probabilistic Setting

- Product distribution on Boolean variables, $\Pr(x) \in [0,1]$ for $x \in V$ (i.e., every Boolean variable is assumed to be independent)
- For $Z \subseteq V$,
  $\Pr(Z) \stackrel{\text{def}}{=} \left(\prod_{x \in Z} \Pr(x) \right) \times \left(\prod_{x \in V \setminus Z} (1 - \Pr(x)) \right)$
- $\Pr(\varphi) \stackrel{\text{def}}{=} \sum_{Z \subseteq V} \Pr(Z) \varphi(Z)$: the probability of the Boolean function φ to be true, aka, the expected value of the Boolean function

Probabilistic Setting

- Product distribution on Boolean variables, $\Pr(x) \in [0, 1]$ for $x \in V$ (i.e., every Boolean variable is assumed to be independent)
- For $Z \subseteq V$,
  $\Pr(Z) \stackrel{\text{def}}{=} \left(\prod_{x \in Z} \Pr(x) \right) \times \left(\prod_{x \in V \setminus Z} (1 - \Pr(x)) \right)$
- $\Pr(\varphi) \stackrel{\text{def}}{=} \sum_{Z \subseteq V} \Pr(Z) \varphi(Z)$: the probability of the Boolean function φ to be true, aka, the expected value of the Boolean function
- $\mathsf{EScore}_c(\varphi, x) \stackrel{\text{def}}{=} \sum_{\substack{Z \subseteq V \\ x \in Z}} (\Pr(Z) \times \mathsf{Score}_c(\varphi, Z, x))$ the expected score of x for φ

Problems studied

We consider classes of representations of Boolean functions, e.g., Boolean circuits, d-D circuits. We assume $\varphi(\emptyset)$ to be computable in PTIME.

- $\mathsf{EV}(\mathcal{F}) : \varphi \in \mathcal{F} \mapsto \Pr(\varphi)$
- $\mathsf{Score}_c(\mathcal{F}) : (\varphi \in \mathcal{F}, x \in V) \mapsto \mathsf{Score}_c(\varphi, V, x)$ for some coefficient function c
- $\mathsf{EScore}_c(\mathcal{F}) : (\varphi \in \mathcal{F}, x \in V) \mapsto \mathsf{EScore}_c(\varphi, x)$

We look for the complexity of these problem and for (Turing) polynomial-time reductions between problems, denoted $A \leqslant_{\mathsf{P}} B$, for class of Boolean functions (and $A \equiv_{\mathsf{P}} B$ for two-way reductions).

What is known?

- $\text{Score}_{c_{\texttt{Shapley}}}(\text{d-D})$ is PTIME [Deutch et al., 2022]

What is known?

- $\mathsf{Score}_{c_{\mathtt{Shapley}}}(\text{d-D})$ is PTIME [Deutch et al., 2022]
- $\mathsf{Score}_{c_{\mathtt{Banzhaf}}}(\text{d-D})$ is PTIME [Abramovich et al., 2023]

What is known?

- $\mathsf{Score}_{c_{\texttt{Shapley}}}(\text{d-D})$ is PTIME [Deutch et al., 2022]
- $\mathsf{Score}_{c_{\texttt{Banzhaf}}}(\text{d-D})$ is PTIME [Abramovich et al., 2023]
- $\mathsf{Score}_c(\mathcal{F}) \leqslant_{\mathsf{P}} \mathsf{EScore}_c(\mathcal{F})$ for any $\mathcal{F}$, c: just compute EScore_c with all probabilities set to 1

What is known?

- $\mathsf{Score}_{c_{\mathtt{Shapley}}}(\text{d-D})$ is PTIME [Deutch et al., 2022]
- $\mathsf{Score}_{c_{\mathtt{Banzhaf}}}(\text{d-D})$ is PTIME [Abramovich et al., 2023]
- $\mathsf{Score}_c(\mathcal{F}) \leqslant_{\mathsf{P}} \mathsf{EScore}_c(\mathcal{F})$ for any $\mathcal{F}$, c: just compute EScore_c with all probabilities set to 1
- $\mathsf{Score}_{c_{\mathtt{Shapley}}}(\mathcal{F}) \equiv_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ for any class $\mathcal{F}$ closed under $\vee$-substitutions [Kara et al., 2023] and when probabilities are uniform (unweighted model counting)

Outline

What have we shown?

Theorem

- $\mathsf{EScore}_c(\mathcal{F}) \leqslant_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$, c

What have we shown?

Theorem

- $\mathsf{EScore}_c(\mathcal{F}) \leqslant_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$, c
- $\mathsf{EScore}_{c_{\mathtt{Shapley}}}(\mathcal{F}) \equiv_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$

What have we shown?

Theorem

- $\mathsf{EScore}_c(\mathcal{F}) \leqslant_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$, c
- $\mathsf{EScore}_{c_{\mathtt{Shapley}}}(\mathcal{F}) \equiv_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$
- $\mathsf{EScore}_{c_{\mathtt{Banzhaf}}}(\mathcal{F}) \equiv_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$ *closed under conditioning and also closed under either conjunctions or disjunctions with fresh variables (e.g., d-Ds)*

What have we shown?

Theorem

- $\mathsf{EScore}_c(\mathcal{F}) \leqslant_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$, c
- $\mathsf{EScore}_{c_{\mathtt{Shapley}}}(\mathcal{F}) \equiv_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$
- $\mathsf{EScore}_{c_{\mathtt{Banzhaf}}}(\mathcal{F}) \equiv_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$ *closed under conditioning and also closed under either conjunctions or disjunctions with fresh variables (e.g., d-Ds)*

Proof techniques: inverting expected values and sums, decomposing sums by size of sets, polynomial interpolation

What have we shown?

Theorem

- $\mathsf{EScore}_c(\mathcal{F}) \leqslant_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$, c
- $\mathsf{EScore}_{c_{\mathtt{Shapley}}}(\mathcal{F}) \equiv_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$
- $\mathsf{EScore}_{c_{\mathtt{Banzhaf}}}(\mathcal{F}) \equiv_{\mathsf{P}} \mathsf{EV}(\mathcal{F})$ *for any* $\mathcal{F}$ *closed under conditioning and also closed under either conjunctions or disjunctions with fresh variables (e.g., d-Ds)*

Proof techniques: inverting expected values and sums, decomposing sums by size of sets, polynomial interpolation

$\Rightarrow$ the tractability landscape of $\mathsf{EScore}_{c_{\mathtt{Shapley}}}$ (and $\mathsf{EScore}_{c_{\mathtt{Banzhaf}}}$ under a mild condition) is exactly the same as that of EV

Concrete algorithms

In the case where we have a d-D C, possible to design specific algorithms (extending those of [Deutch et al., 2022, Abramovich et al., 2023]) for EScore_c with complexity (ignoring arithmetic costs):

- $O\left(|C| \times |V|^5 + \mathrm{T}_c(|V|) \times |V|^2\right)$ where $\mathrm{T}_c(\alpha)$ is the cost of computing the coefficient function on inputs $\leqslant \alpha$

Concrete algorithms

In the case where we have a d-D C, possible to design specific algorithms (extending those of [Deutch et al., 2022, Abramovich et al., 2023]) for EScore_c with complexity (ignoring arithmetic costs):

- $O\left(|C| \times |V|^5 + \mathrm{T}_c(|V|) \times |V|^2\right)$ where $\mathrm{T}_c(\alpha)$ is the cost of computing the coefficient function on inputs $\leqslant \alpha$
- $O\left(|V|^2 \times (|C||V| + |V|^2 + \mathrm{T}_c(|V|))\right)$ when all probabilities are identical

Concrete algorithms

In the case where we have a d-D C, possible to design specific algorithms (extending those of [Deutch et al., 2022, Abramovich et al., 2023]) for EScore_c with complexity (ignoring arithmetic costs):

- $O\left(|C| \times |V|^5 + \mathrm{T}_c(|V|) \times |V|^2\right)$ where $\mathrm{T}_c(\alpha)$ is the cost of computing the coefficient function on inputs $\leqslant \alpha$
- $O\left(|V|^2 \times (|C||V| + |V|^2 + \mathrm{T}_c(|V|))\right)$ when all probabilities are identical
- $O(|C| \times |V|)$ for c_{Banzhaf}

Application to Probabilistic Databases

- TID database, Boolean query q in some query language
- Define Score_c, EScore_c of a tuple for a query as Score_c, EScore_c of the Boolean provenance of the query over the database
- We compare to PQE (Probabilistic Query Evaluation, i.e., computing the probability of a Boolean query)

Theorem

- $\mathrm{EScore}_c(q) \leqslant_{\mathrm{P}} \mathrm{PQE}(q)$ *for any c, query q (whatever the query language!)*
- $\mathrm{EScore}_{c_{\mathtt{Shapley}}} \equiv_{\mathrm{P}} \mathrm{PQE}(q)$ *for any query q (whatever the query language!)*

Application to Probabilistic Databases

- TID database, Boolean query q in some query language
- Define Score_c, EScore_c of a tuple for a query as Score_c, EScore_c of the Boolean provenance of the query over the database
- We compare to PQE (Probabilistic Query Evaluation, i.e., computing the probability of a Boolean query)

Theorem

- $\mathsf{EScore}_c(q) \leqslant_{\mathsf{P}} \mathsf{PQE}(q)$ *for any* c, *query* q *(whatever the query language!)*
- $\mathsf{EScore}_{c_{\mathtt{Shapley}}} \equiv_{\mathsf{P}} \mathsf{PQE}(q)$ *for any query* q *(whatever the query language!)*

$\Rightarrow$ We inherit all tractability and intractability results for PQE, e.g., dichotomy for UCQs [Dalvi and Suciu, 2012] or queries closed under homomorphisms [Amarilli, 2023]

Outline

Desiderata for a provenance-aware DBMS

- Extends a widely used database management system
- Easy to deploy
- Easy to use, transparent for the user
- Provenance automatically maintained as the user interacts with the database management system
- Provenance computation benefits from query optimization within the DBMS
- Allow probability computation based on provenance
- Any form of provenance can be computed: Boolean provenance, semiring provenance in any semiring (possibly, with monus), aggregate provenance, where-provenance, on demand

ProvSQL: Provenance within PostgreSQL (1/2)

[Senellart et al., 2018]

- Lightweight extension/plugin for PostgreSQL $\geqslant 9.5$ (tested against all versions – upgrade to a new version typically takes a couple of hours)
- Provenance annotations stored as Universally Unique Identifiers (UUIDs), in an extra attribute of each provenance-aware relation
- UUIDs of base tuples randomly generated; UUIDs of query results generated in a deterministic manner
- A provenance circuit relating UUIDs of elementary provenance annotations and arithmetic gates stored in shared memory of the DBMS (or on disk)
- All computations done in the universal semiring (more precisely, with monus, in the free semiring with monus; for where-provenance, in a free term algebra)

ProvSQL: Provenance within PostgreSQL (2/2)

[Senellart et al., 2018]

- Query rewriting (after parsing, before planning) to automatically compute output provenance attributes in terms of the query and input provenance attributes:
  - Duplicate elimination (DISTINCT, set union) results in aggregation of provenance values with $\oplus$
  - Cross products, joins results in combination of provenance values with $\otimes$
  - Difference rewritten in a join, with combination of provenance values with $\ominus$
- Additional circuit gates on projection, join for support of where-provenance
- Probability computation from the provenance circuits, via various methods (naive, sampling, compilation to d-Ds, tree decomposition)
- Expected Shapley value computation implemented directly within ProvSQL

ProvSQL: Current status

- Supported SQL language features:
  - Regular SELECT-FROM-WHERE queries (aka conjunctive queries with multiset semantics)
  - JOIN queries (regular joins and outer joins; semijoins and antijoins are not currently supported)
  - SELECT queries with nested SELECT subqueries in the FROM clause
  - GROUP BY queries
  - SELECT DISTINCT queries (i.e., set semantics)
  - UNION's or UNION ALL's of SELECT queries
  - EXCEPT queries
  - Aggregate queries (terminal, for simple aggregates)
- Try it (and see a demo) from https://github.com/PierreSenellart/provsql

Outline

Introduction

Theoretical Results

ProvSQL

Experimental Results

Conclusion

Set-Up

- Complexity of $O(n^6)$: is it really feasible?
- Same experiment set-up as in [Deutch et al., 2022]: 1 GB TPC-H database, 8 TPC-H queries with some adaptations (e.g., removing aggregates), computation of Shapley/Banzhaf scores for all input tuples
- Non-Boolean queries: computation for every output tuple
- Proof-of-feasibility rather than in-depth experiments
- Compilation to d-D:
  - Check whether Boolean circuit is already an independent circuit
  - Otherwise, try to find a low-treewidth decomposition of the circuit, and use it to build a d-D
  - Otherwise, use an external knowledge compiler (but never required)

Results

# Output tuples	Provenance time (s)	Compilation time (s)	Shapley time (s) Determ.	Shapley time (s) Expect.	Banzhaf time (s)
11620	2.125	1.226	0.762	1.758	0.467
5	1.117	0.044	0.766	40.910	0.191
4	1.215	0.017	0.269	9.381	0.085
1783	1.229	0.018	0.023	0.037	0.015
61	0.174	0.001	0.001	0.002	0.001
466	0.247	0.084	0.159	0.455	0.094
91159	2.711	0.749	0.655	1.008	0.489
56	1.223	0.000	0.000	0.000	0.000

Results

# Output tuples	Provenance time (s)	Compilation time (s)	Shapley time (s) Determ.	Shapley time (s) Expect.	Banzhaf time (s)
11620	2.125	1.226	0.762	1.758	0.467
5	1.117	0.044	0.766	40.910	0.191
4	1.215	0.017	0.269	9.381	0.085
1783	1.229	0.018	0.023	0.037	0.015
61	0.174	0.001	0.001	0.002	0.001
466	0.247	0.084	0.159	0.455	0.094
91159	2.711	0.749	0.655	1.008	0.489
56	1.223	0.000	0.000	0.000	0.000

Very encouraging! Shapley value computation does not have such a huge overhead!

Outline

Main message

- Expected Shapley value computation is not (much) more costly than probabilistic query evaluation
- Landscape seems clearer than for deterministic Shapley value computation
- PQE (and Expected Shapley value computation) is quite feasible in practice, even on large datasets
- Connection to SHAP-score [Van den Broeck et al., 2022] not quite clear (there is also a probability distribution, but not used in the same way)
- Approximations?

On the ProvSQL side: Perspectives

Usability: Support for larger subset of SQL, utility functions, better interface, documentation, ability to restrict to specific semirings

Efficiency: Benchmarks, optimizations of provenance and probability computation, scalability, manipulate circuit both on disk and in main memory

Knowledge compilation: closer integration with knowledge compilers

More complete probabilistic query evaluation: implementation of safe query plans, continuous probability distributions

Use cases: Work with users, provide semirings that implement useful behavior (e.g., the semiring of unions of real intervals for temporal databases)

Collaborators welcome!

ProvSQL tutorial:

`https://github.com/PierreSenellart/provsql/tree/master/doc/tutorial`

Bibliography I

Omer Abramovich, Daniel Deutch, Nave Frost, Ahmet Kara, and Dan Olteanu. Banzhaf Values for Facts in Query Answering. *arXiv preprint arXiv:2308.05588*, 2023.

Antoine Amarilli. Uniform reliability for unbounded homomorphism-closed graph queries. In *ICDT*, volume 255 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL `https://arxiv.org/abs/2209.11177`.

John F Banzhaf III. Weighted voting doesn't work: A mathematical analysis. *Rutgers L. Rev.*, 19:317, 1964.

Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2012.

Bibliography II

Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. Computing the Shapley value of facts in query answering. In *SIGMOD Conference*, pages 1570–1583. ACM, 2022.

Ahmet Kara, Dan Olteanu, and Dan Suciu. From Shapley Value to Model Counting and Back. *arXiv preprint arXiv:2306.14211*, 2023.

Werner Kirsch and Jessica Langner. Power indices and minimal winning coalitions. *Social Choice and Welfare*, 34(1):33–46, 2010. ISSN 01761714, 1432217X. URL http://www.jstor.org/stable/41108037.

Annick Laruelle. On the choice of a power index. Technical report, Instituto Valenciano de Investigaciones Económicas, 1999.

Bibliography III

Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. ProvSQL: provenance and probability management in postgresql. In *VLDB*, 2018. Demonstration.

Lloyd S Shapley et al. A value for n-person games. 1953.

Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of SHAP explanations. *Journal of Artificial Intelligence Research*, 74:851–886, 2022.