Probabilistic Data Programming with ENFrame

Dan Olteanu and Sebastiaan J. van Schaik
University of Oxford

Abstract

This paper overviews ENFrame, a programming framework for probabilistic data. In addition to relational query processing supported via an existing probabilistic database management system, ENFrame allows programming with loops, assignments, conditionals, list comprehension, and aggregates to encode complex tasks such as clustering and classification of probabilistic data.

We explain the design choices behind ENFrame, some distilled from the wealth of work on probabilistic databases and some new. We also highlight a few challenges lying ahead.

1 Motivation and Scope

Probabilistic data management has gone a long, fruitful way in the last decade [20]: We have a good understanding of the space of possible relational and hierarchical data models and its implication on query tractability; the community already delivered several open-source systems that exploit the first-order structure of database queries for scalable inference, e.g., MystiQ [3], Trio [22], MayBMS/SPROUT [11], and PrDB [19] to name very few, and applications in the space of web data management [8, 6]. Significantly less effort has been spent on supporting complex data processing beyond mere querying, such as general-purpose programming.

There is a growing need for computing frameworks that allow users to build applications feeding on uncertain data without worrying about the underlying uncertain nature of such data or the computationally hard inference task that comes along with it. For tasks that only need to query probabilistic data, existing probabilistic database systems do offer a viable solution [20]. For more complex tasks, however, successful development requires a high level of expertise in probabilistic databases and this hinders the adoption of existing technology as well as communication between potential users and experts.

A similar observation has been recently made in the areas of machine learning [5] and programming languages [10]. Developing programming languages that allow probabilistic models to be expressed concisely has become a hot research topic [18]. Such programming languages can be imperative (C-style) or declarative (first-order logic), with the novelty that they allow to express probability distributions via generative stochastic models, to draw values at random from such distributions, and to condition values of program variables on observations. For inference, the programs are usually grounded to Bayesian networks and fed to MCMC methods [15]. In the area of databases, MCDB [13] and SimSQL [4] have been visionary in enabling stochastic analytics in the database by coupling Monte Carlo simulations with declarative SQL extensions and parallel database techniques.

The thesis of this work is that one can build powerful and useful probabilistic data programming frameworks that *leverage* existing work on probabilistic databases. ENFrame [21] is a framework that aims to fit this vision:

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- Its programming language is a fragment of Python with constructs such as bounded-range loops, if-then-else statements, list comprehension, aggregates, variable assignments, and query calls to external database engines. A user program can express complex tasks such as clustering and classification intermixed with structured querying.
- The users are oblivious to the probabilistic nature of the input data: They program as if the input data were plain relational, with no uncertainty or layout intricacy. It is the job of ENFrame to make sense of the underlying data formats, probabilities, and input correlations, thus allowing low-level entry for users without expert knowledge on inference and probabilistic models.

  ENFrame relies on the Πgora probabilistic data integration system [17] for exposing a uniform relational view of the underlying data, which may be Bayesian networks for some sources and probabilistic c-tables for others (or special cases such as tuple-independent or block-independent disjoint tables).
- ENFrame adheres to the possible worlds semantics for its whole processing pipeline. Under this semantics, the input is a probability distribution over a finite set of possible worlds, with each world defining a database. The result of a user program is equivalent to executing it within each world and is thus a probability distribution over possible outcomes of the program variables. This distribution can be inspected by the user and can serve as input to probabilistic database queries and subsequent ENFrame programs.
- ENFrame uses a rich language of probabilistic events to symbolically express input correlations, trace the correlations introduced during computation, and enable result explanation, sensitivity analysis [14], incremental maintenance, and knowledge compilation-based approaches for approximate inference [9].

  While propositional formulas over random variables are expressive enough to capture computation traces of ENFrame programs, they can be very large and expensive to manage. ENFrame's event language extends algebraic formalisms based on semimodules for probabilistic data [2, 7] that can succinctly[1] encode probabilistic events with constructs mirroring those in the user language.
- ENFrame exploits the structure of queries and programs for efficient inference; it relies on SPROUT for query processing on probabilistic data [16, 9].

  To avoid iterating over all possible worlds, ENFrame employs approximation schemes and exploits the fact that many of the possible worlds are alike. To further speed up inference on networks of highly interconnected probabilistic events, such as those for clustering and classification programs, ENFrame uses parallel algorithms that exploit multi-core architectures.

There are key differences that set ENFrame apart from the myriad of recent probabilistic programming [18] and probabilistic data processing [1] approaches:

- ENFrame uses the semantics and a probabilistic data model compatible with probabilistic databases [20]. Its input can consist of arbitrarily correlated (and not only independent) probabilistic events. This enables processing pipelines mixing programming (via ENFrame) and querying (via SPROUT).
- The user need not be aware of the probabilistic nature of data or the possibly different probabilistic formalisms used by the input sources. One implication is that probability distributions can only be supplied as input data and not in the actual program. So far, input distributions are discrete and can only be given explicitly and not symbolically (e.g., Normal distribution with parameters mean and standard deviation).
- The probabilistic event language allows ENFrame to leverage existing work on incremental maintenance of query answers and extend it to incremental maintenance of the program output in the face of updates to input probabilities, insertions and deletions of uncertain objects (though this is subject to future work).

[1] Such events can be exponentially more succinct than equivalent propositional formulas over random variables or Bayesian networks.

```
1 (O, n) = loadData(db)      # list and number of objects
2 (k, iter) = loadParams()   # number of clusters/iterations
3 M = init()                 # initialise medoids
4
5 for it in range(0,iter):   # clustering iterations
6  InCl = [None] * k         # assignment phase
7  for i in range(0,k):
8   InCl[i] = [None] * n
9   for l in range(0,n):
10   InCl[i][l] = reduce_and(
11    [(dist(O[l],M[i]) <= dist(O[l],M[j]))
12      for j in range(0,k)])
13  InCl = breakTies1(InCl)  # each object in one cluster
14
15  DistSum = [None] * k     # update phase
16  for i in range(0,k):
17   DistSum[i] = [None] * n
18   for l in range(0,n):
19    DistSum[i][l] = reduce_sum(
20      [dist(O[l],O[p]) for p in range(0,n) if InCl[i][p]])
21
22  Centre = [None] * k
23  for i in range(0,k):
24   Centre[i] = [None] * n
25   for l in range(0,n):
26    Centre[i][l] = reduce_and(
27      [DistSum[i][l] <= DistSum[i][p] for p in range(0,n)])
28  Centre = breakTies2(Centre) # one medoid per cluster
29
30  M = [None] * k
31  for i in range(0,k):
32   M[i] = reduce_sum(
33      [O[l] for l in range(0,n) if Centre[i][l]])
```

$\forall i$ in $0..n-1 : O^i \equiv \Phi(o_i) \otimes \vec{o}_i$

$M^0_{-1} \equiv \Phi(o_{\pi(0)}) \otimes \vec{o}_{\pi(0)}; \ldots ; M^{k-1}_{-1} \equiv \Phi(o_{\pi(k-1)}) \otimes \vec{o}_{\pi(k-1)}$

$\forall$it in $0..$iter -1 :

  $\forall i$ in $0..k-1$:

    $\forall l$ in $0..n-1$:

      $\text{InCl}^{i,l}_{\text{it}} \equiv \bigwedge_{j=0}^{k-1} \left[\text{dist}(O^l, M^i_{\text{it}-1}) \leq \text{dist}(O^l, M^j_{\text{it}-1})\right]$

// Encoding of breakTies1 omitted

$\forall i$ in $0..k-1$:

  $\forall l$ in $0..n-1$:

    $\text{DistSum}^{i,l}_{\text{it}} \equiv \sum_{p=0}^{n-1} \text{InCl}^{i,p}_{\text{it}} \otimes \text{dist}(O^l, O^p)$

$\forall i$ in $0..k-1$:

  $\forall l$ in $0..n-1$:

    $\text{Centre}^{i,l}_{\text{it}} \equiv \bigwedge_{p=0}^{n-1} \left[\text{DistSum}^{i,l}_{\text{it}} \leq \text{DistSum}^{i,p}_{\text{it}}\right]$

// Encoding of breakTies2 omitted

$\forall i$ in $0..k-1$:

  $M^i_{\text{it}} = \sum_{l=0}^{n-1} \text{Centre}^{i,l}_{\text{it}} \wedge O^l$

Figure 1: k-medoids clustering specified as user program (left) and event program (right).

2 ENFrame by Example: k-medoids clustering

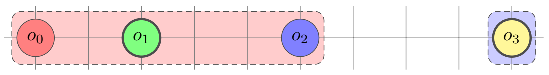
We illustrate ENFrame using the k-medoids algorithm, an iterative clustering algorithm with three phases. During the *initialisation phase*, k initial medoids (cluster centres) are selected at random or using a heuristic. Then, the algorithm iteratively assigns data points to the closest medoid during the *assignment phase*, followed by the *update phase* during which the new medoids are selected.

Figure 1 shows a user program in ENFrame encoding this algorithm and the corresponding event program.

The **user program** is written in a subset of Python and is oblivious to the uncertainty of the input data; consequently, the users need not be conversant with probabilistic data models and inference.

The data is loaded from a database (if a query and database are provided), or an external data source (line 1). The initial medoids are then selected (line 3). In the assignment phase (lines 6 – 13), for each cluster C_i and object o_l, the Boolean vector `InCl` stores whether o_l is assigned to C_i. The Boolean value `Incl[i][l]` is true if the distance from o_l to the medoid M_i of cluster C_i is the smallest of the distances to all medoids M_j (line 10). After the assignment phase, the update phase is performed to select the new medoids `M[i]` (lines 15 – 33). An object becomes a medoid if its distance to all other objects in the same cluster is the smallest of all cluster members. For each cluster and object, the sum of distances to other cluster members is computed and stored in `DistSum` (lines 15 – 20). For cluster C_i and object o_l, the Boolean vector `Centre` stores whether o_l is the new medoid of C_i. The value `Centre[i][l]` is true if the distance sum of o_i to members of C_i is the smallest of all objects in C_i.

Example 1: For $k = 2$ clusters and Euclidean distance, the user program can be run on the following deterministic data consisting of four data points on a line to yield the following result with o_1 and o_3 as cluster medoids:

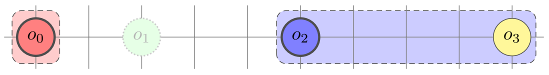


The **event program** gives a probabilistic interpretation of the user program: the deterministic variables in the user program become random variables. This gives rise to a probabilistic interpretation of k-medoids in which the object existence, cluster assignment, and medoid selection become uncertain.

ENFrame uses *conditional values* (c-values) to construct random variables whose sample space consists of real numbers and vectors in the feature space; the c-value construct is inspired by semimodule expressions used to capture provenance of queries with aggregates [2]. A c-value is denoted by $\Phi \otimes v$, where Φ is a propositional formula over Boolean random variables (or an arbitrary event) and v can be any real number or feature space vector. It is a probabilistic event whose interpretation is value v in case Φ is true and the neutral value for the domain of v otherwise (e.g., zero for reals); neutral values have a special interpretation in our formalism: An object whose value defaults to a neutral value in a possible world is interpreted as not existing in that world. Conditional values form the basic components of more complex events such as probabilistic sums, e.g., DistSum in the event program. Any discrete probability distribution $[(\vec{o}_1, p_1), \ldots, (\vec{o}_m, p_m)]$ over the positions $\vec{o}_1, \ldots, \vec{o}_m$ of a data point can be modelled using a sum of c-values $\sum_{j=1}^{m} \Phi_j \otimes \vec{o}_j$, where the Boolean probabilistic events $\Phi_1, \ldots, \Phi_m$ are mutually exclusive and have probabilities $p_1, \ldots, p_m$.

The random variables O_i are examples of c-values that define the input data points conditioned on probabilistic events. The Boolean random variable $\text{InCl}_{\text{it}}^{i,l}$ mirrors the Boolean variable `InCl[i][l]` in the user program; it represents the probabilistic event that object o_l is assigned to cluster C_i in iteration *it*. Similarly, $\text{DistSum}_{\text{it}}^{i,l}$ is a real-valued random variable representing all possible sums of distances from candidate medoid o_l to other objects in cluster C_i. Its probabilistic value is used to give rise to the Boolean random variable $\text{Centre}_{\text{it}}^{i,l}$, which eventually determines the new medoid M_{it}^{i}.

Example 2: Reconsider the clustering example from Example 1 and assume that the existence of each data point o_i is conditioned by a probabilistic event ϕ_i. Assume a total valuation ν of random variables, i.e., a possible world, that maps ϕ_1 to false and all other events ϕ_i to true. Then, the following result of 2-medoids clustering holds with a probability given by the product of the probabilities of the variable assignments in ν:



The program variables have a probabilistic interpretation and thus define a probability distribution that can be inspected by the user. For instance, the user can define a variable stating that two objects belong to the same cluster or co-occur together in clusters. ENFrame will compute the probability distributions for such variables. Such distributions can serve as input probabilistic data to queries and subsequent ENFrame programs.

3 Challenges Faced by ENFrame

ENFrame's goal is to enable probabilistic data programming beyond query processing. To achieve this goal, ENFrame faces several challenges, some of which are highlighted below.

Expressiveness of user language. Its user language can express popular data mining tasks mixed with queries. So far, we experimented with clustering (Markov, k-means, k-medoids), classification (k-nearest neighbour), and relational queries with aggregates. The user language supports bounded loops, conditional (if-then-else) statements, assignments, list comprehension and aggregation, primitive and array-valued variables. Using conditional statements and independent input random variables, any Bayesian network can be constructed in the user program. As exemplified for clustering, aggregations are key ingredients supported by ENFrame's language but

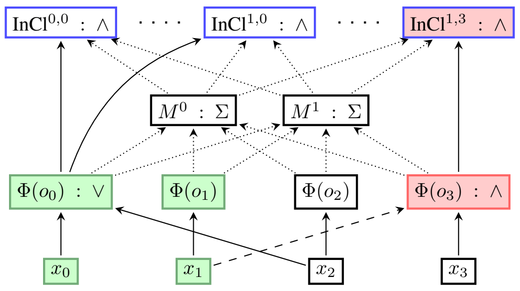


Figure 2: Excerpt of an event network for 2-medoids clustering with four data points $o_0, \ldots, o_3$.

missing in most existing probabilistic programming languages. The trade-off between the expressive power of the user language and the efficiency of probabilistic inference remains nevertheless an open research question.

For probabilistic inference, state-of-the-art approaches use Monte Carlo simulations that sample the input probability space, run the user program on each sample (possible world), and aggregate the results over the samples. With ENFrame, we seek to understand a complementary approach that is more in the spirit of earlier work on lineage-based query processing in probabilistic databases [20] and thus integrates well with the latter:

- (Grounding) We trace the lineage of the user program computation using probabilistic events, and then
- (Inference) We compute their exact probabilities or approximate them with error guarantees.

Each of these two tasks come with their own challenges beyond query processing[2].

Representation of event programs. Similarly to provenance management systems such as Orchestra [12] that represent provenance as graphs, ENFrame represents event programs as *event networks*, which are graphs whose nodes are Boolean connectives, comparisons, aggregates, and c-values. Expressions common to several events are only represented once in such graphs. Event networks for data mining tasks such as clustering are very repetitive and highly interconnected due to the combinatorial nature of the algorithms: The events at each iteration are expressions over the events at the previous iteration and have the same structure at each iteration. Moreover, the event networks can be cyclic, so as to account for program loops. While it is possible to unfold bounded-range loops, this can lead to prohibitively large event networks.

Example 3: Consider the event network in Figure 2. The lowest layer of the network consists of nodes for each of the random variables $x_0, \ldots, x_3$. On top of them are the events $\Phi(o_0), \ldots, \Phi(o_3)$ of the data points $o_0, \ldots, o_3$, which are $x_0 \vee x_2$, x_1, x_2, and $\neg x_1 \wedge x_3$ respectively. Furthermore, there are nodes for the medoids M^i and in-cluster Boolean values `InCl[i][l]`. Nodes in the lower layers in the network contribute to many (if not all) nodes in the next upper layer and sometimes even in higher layers. The dotted edges mean that nodes along them are not shown for readability reasons.

Inference via parallel approximate knowledge compilation. The inference techniques employed by ENFrame work for entire networks of interconnected events that are defined over several iterations, exhibit deep nesting structures, and use expressive constructs such as conditional values [21]. The common ground of these inference techniques is the compilation of the event networks into decision trees using Shannon expansion on the random variables occurring in the events. To scale to large networks and number of objects, we designed parallel approximate inference techniques. To approximate, we use an error budget to prune large, not-yet-explored fragments of the decision tree that only account for a small probability mass. To parallelise the computation, we divide on-the-fly the decision tree into fragments for concurrent exploration.

[2]Probabilistic query processing is a special instance of ENFrame processing: The lineage of a query on a probabilistic database is a probabilistic event that can be expressed in our event language and the probability of a query result is the probability of its lineage.

Example 4: Consider again the network in Figure 2 and a partial assignment of the random variables that maps both x_0 and x_1 to true. The nodes turned green, i.e., x_0, x_1, $\Phi(o_0)$, and $\Phi(o_1)$, are then also mapped to true, while the red ones are mapped to false. Subsequent mappings of the remaining variables to true or false eventually leads to a truth value for the events of interest, which are the events sitting on top of the network.

Experiments with k-medoids clustering and k-nearest neighbour classification show that our exact inference technique performs orders of magnitude better than the naïve approach of iterating over all possible worlds. Furthermore, the approximate techniques (with absolute error guarantees) perform orders of magnitude better than exact inference. Parallel inference techniques scale almost linearly in the number of CPU cores (benchmarked on machines with 4-core and 16-core CPU). The experiments were conducted for settings of up to 15,000 objects and 150 input random variables, where the objects exhibit either positive, mutually exclusive, or conditional correlations. When using both parallelisation (16 cores) and approximation (absolute error of 0.1), ENFrame's performance is up to several seconds for the above settings. We plan to investigate the effect of employing several machines on performance of our inference techniques.

The output quality of our inference techniques has been evaluated against both the naïve method of iterating over all possible worlds (the golden standard), and inference in top-k most probable worlds. These experiments confirm that our error guarantees hold and show that top-k worlds inference only achieves the same output quality when k approaches the total number of worlds.

Managing large event networks. Full materialisation of event networks as obtained by exhaustive grounding of the user program can be prohibitive. We experimented with partial network compilation, where large network fragments are compiled down to C++ code. Initial experiments with k-medoids showed that this results in event networks that are two to three orders of magnitude smaller, while inference on the compressed event networks became up to an order of magnitude faster. All cluster membership events for a given cluster C_i can be lifted to one higher-order event, which expresses the membership of all data points to cluster C_i and can be compiled to C++ code for efficiency reasons. However, whereas the fine-grained events lead to large networks and less efficient inference, they support more accurately result explanation, sensitivity analysis, and incremental maintenance. We plan to investigate a functionality-performance trade-off brought by lifting the fine-grained events to their higher order realisation.

4 Conclusion

ENFrame is a programming framework for probabilistic data designed to integrate well with existing probabilistic database management systems such as MystiQ and MayBMS/SPROUT. So far, we investigated within this framework a rich language for random events that can trace complex computation such as clustering and classification, and distributed approximate inference techniques for networks of interconnected events with deep structure. Exciting directions for future work include:

- The trade-offs between the expressiveness of the user language and efficiency of inference and between the functionality of fine-grained events and performance brought by C++ compilation of event networks;
- Investigation of additional constructs in the event language needed to support a library of common algorithms for data analysis;
- Exploitation of the user program structure, in addition to query structure, for efficient inference;
- A better integration of queries and program constructs with relations and program data structures (e.g., multi-dimensional arrays) to be used interchangeably in both programs and queries;
- Improving the performance of inference by employing large-scale distributed systems.

References

[1] C. Aggarwal. *Managing and Mining Uncertain Data*. Kluwer, 2009.

[2] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011.

[3] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD*, pages 891–893, 2005.

[4] Z. Cai, Z. Vagena, L. L. Perez, S. Arumugam, P. J. Haas, and C. M. Jermaine. Simulation of database-valued Markov chains using SimSQL. In *SIGMOD*, pages 637–648, 2013.

[5] Defense Advanced Research Projects Agency. Probabilistic programming for advancing machine learning, April 2013. DARPA-BAA-13-31.

[6] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge Vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, 2014. To appear.

[7] R. Fink, L. Han, and D. Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5):490–501, 2012.

[8] R. Fink, A. Hogue, D. Olteanu, and S. Rath. SPROUT^2: A squared query engine for uncertain web data. In *SIGMOD*, pages 1299–1302, 2011.

[9] R. Fink, J. Huang, and D. Olteanu. Anytime approximation in probabilistic databases. *VLDB J.*, pages 823–848, 2013.

[10] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *ICSE*, 2014. Future of Software Engineering track (to appear).

[11] J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, pages 1071–1074, 2009.

[12] Z. Ives, T. Green, G. Karvounarakis, N. Taylor, V. Tannen, P. P. Talukdar, M. Jacob, and F. Pereira. The Orchestra collaborative data sharing system. *SIGMOD Rec.*, 37(2):26–32, 2008.

[13] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.

[14] B. Kanagal, J. Li, and A. Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *SIGMOD*, pages 841–852, 2011.

[15] B. Milch and et al. BLOG: Probabilistic models with unknown objects. In *IJCAI*, pages 1352–1359, 2005.

[16] D. Olteanu, J. Huang, and C. Koch. SPROUT: lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.

[17] D. Olteanu, L. Papageorgiou, and S. J. van Schaik. Πgora: An integration system for probabilistic data. In *ICDE*, pages 1324–1327, 2013.

[18] probabilistic-programming.org. Repository on probabilistic programming languages, 2014.

[19] P. Sen, A. Deshpande, and L. Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 18(5):1065–1090, 2009.

[20] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

[21] S. J. van Schaik, D. Olteanu, and R. Fink. ENFrame: A platform for processing probabilistic data. In *EDBT*, pages 355–366, 2014.

[22] J. Widom. Trio: a system for data, uncertainty, and lineage. In C. Aggarwal, editor, *Managing and Mining Uncertain Data*, chapter 5. Springer-Verlag, 2008.