PClean: Bayesian Data Cleaning at Scale with Domain-Specific Probabilistic Programming

Alexander K. Lew **Monica Agrawal** **David Sontag** **Vikash K. Mansinghka**
Massachusetts Institute of Technology

Abstract

Data cleaning is naturally framed as probabilistic inference in a generative model of ground-truth data and likely errors, but the diversity of real-world error patterns and the hardness of inference make Bayesian approaches difficult to automate. We present PClean, a probabilistic programming language (PPL) for leveraging dataset-specific knowledge to automate Bayesian cleaning. Compared to general-purpose PPLs, PClean tackles a restricted problem domain, enabling three modeling and inference innovations: (1) a non-parametric model of relational database instances, which users' programs customize; (2) a novel sequential Monte Carlo inference algorithm that exploits the structure of PClean's model class; and (3) a compiler that generates near-optimal SMC proposals and blocked-Gibbs rejuvenation kernels based on the user's model and data. We show empirically that short (<50-line) PClean programs can: be faster and more accurate than generic PPL inference on data-cleaning benchmarks; match state-of-the-art data-cleaning systems in terms of accuracy and runtime (unlike generic PPL inference in the same runtime); and scale to real-world datasets with millions of records.

1 INTRODUCTION

Real-world data is often noisy and incomplete, littered with NULLs, typos, duplicates, and inconsistencies. Cleaning dirty data is important for many workflows, but can be difficult to automate, requiring judgment calls about objects in the world (e.g., to decide whether two records refer to the same hospital, or which of several cities called "Jefferson" someone lives in). Generative models provide a conceptually appealing approach to automating this sort of reasoning, but the diversity of real-world errors (Abedjan et al., 2016) and the difficulty of inference pose significant challenges.

This paper presents PClean, a domain-specific generative probabilistic programming language (PPL) for Bayesian data cleaning. As in some existing PPLs (e.g. BLOG (Milch and Russell, 2006)), PClean programs encode prior knowledge about relational domains, and quantify uncertainty about the latent networks of objects that underlie observed data. However, PClean's approach is inspired by *domain-specific* PPLs, such as Stan (Carpenter et al., 2017) and Picture (Kulkarni et al., 2015): it aims not to serve all conceivable relational modeling needs, but rather to deliver fast inference, concise model specification, and accurate cleaning on large-scale problems. It does this via three modeling and inference contributions:

1. PClean introduces a domain-general non-parametric prior on the number of latent objects and their link structure. PClean programs customize the prior via a relational schema and via generative models for objects' attributes.
2. PClean inference is based on a novel sequential Monte Carlo (SMC) algorithm, to initialize a latent object database with plausible guesses, and novel rejuvenation updates to fix mistakes.
3. PClean provides a compiler that generates near-optimal SMC proposals and Gibbs rejuvenation kernels given the user's data, PClean program, and inference hints. These proposals improve over generic top-down PPL inference by incorporating local Bayesian reasoning within user-specified subproblems, and heuristics from traditional cleaning systems.

Together, these innovations improve over generic PPL inference, enabling fast and accurate cleaning of challenging real-world datasets with millions of rows.

All code available at github.com/probcomp/PClean

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

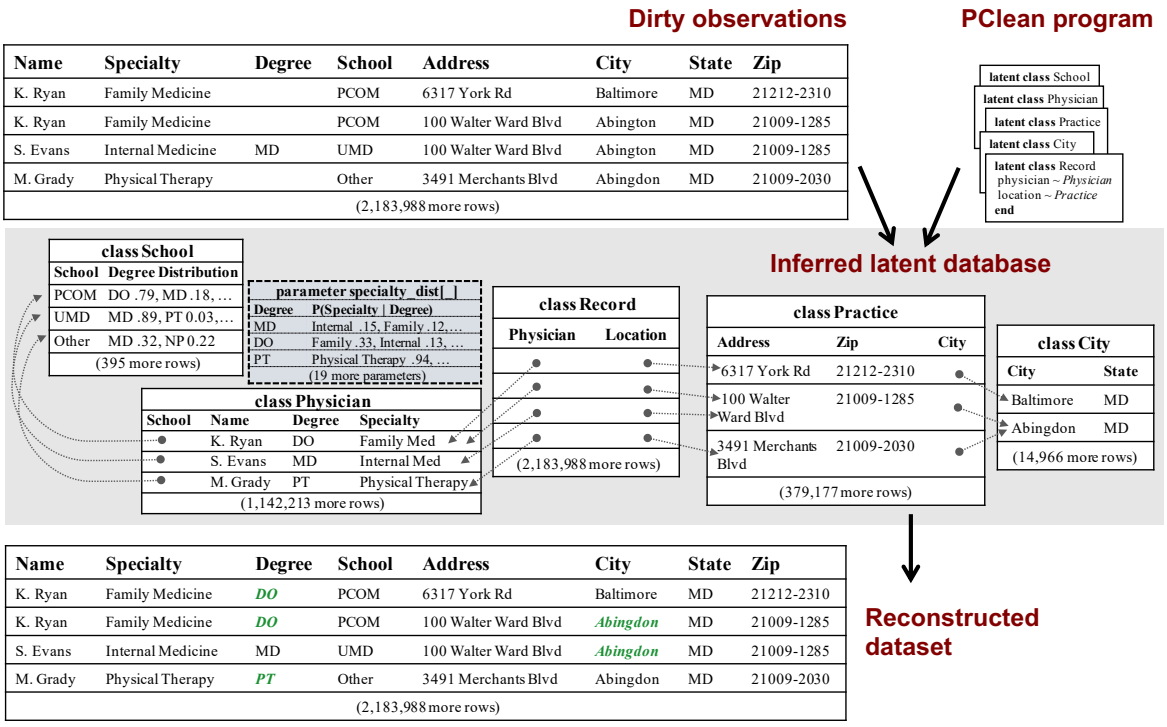

Figure 1: PClean applied to Medicare's 2.2-million-row Physician Compare National database. Based on a user-specified relational model, PClean infers a latent database of entities, which it uses to correct systematic errors (e.g. the misspelled *Abington, MD* appears 152 times in the dataset) and impute missing values.

1.1 Related Work

Many researchers have proposed generative models for cleaning specific datasets or fixing particular error patterns (Pasula et al., 2003; Kubica and Moore, 2003; Mayfield et al., 2009; Matsakis, 2010; Xiong et al., 2011; Hu et al., 2012; Zhao et al., 2012; Abedjan et al., 2016; De et al., 2016; Steorts et al., 2016; Winn et al., 2017; De Sa et al., 2019; Eduardo et al., 2020; Marchant et al., 2021). Such formulations specify priors over ground truth data, and likelihoods that model errors. In contrast, PClean's PPL makes it easy to write short (<50 line) programs to specify custom models of new datasets, and automates an inference algorithm that delivers fast, accurate cleaning results.

PClean draws on a rich literature of Bayesian approaches to modeling relational data (Friedman et al., 1999), including open-universe models with identity and existence uncertainty (Milch and Russell, 2006). Many PPLs could express PClean-like data cleaning models (Milch et al., 2005; Goodman et al., 2008; Goodman and Stuhlmüller, 2014; Mansinghka et al., 2014; Gordon et al., 2014; Tolpin et al., 2016; Ścibior et al., 2018; Bingham et al., 2019; Cusumano-Towner et al., 2019), but in practice, generic PPL inference is often too slow. This paper introduces new algorithms that scale better, and demonstrates external validity of the results by calibrating PClean's runtime and accuracy against SOTA data-cleaning baselines (Dallachiesat et al., 2013; Rekatsinas et al., 2017) that use machine learning and weighted logic (typical of discriminative approaches (Mccallum and Wellner, 2003; Wellner et al., 2004; Wick et al., 2013)).

Some of PClean's inference innovations have close analogues in traditional cleaning systems; for example, PClean's preferred values from Section 3.3 are related to HoloClean's notion of domain restriction. In fact, PClean can be viewed as a scalable, Bayesian, domain-specific PPL implementation of the PUD framework from De Sa et al. (2019) (which abstractly characterizes the HoloClean implementation from Rekatsinas et al. (2017), but does not itself include PClean's modeling or inference innovations). Some of PClean's inference contributions also have precursors in LibBi and Birch (Murray, 2015; Murray and Schön, 2018), which, like PClean, employ sequential Monte Carlo algorithms with data-driven proposals. However, Birch's delayed sampling technique (Murray et al., 2018) would *not* yield intelligent, data-driven proposals in PClean's non-parametric model class, and in Section 4, we show that PClean's novel inference contributions (including its static generation of model-specific proposal code, and its per-object rejuvenation schedule) are necessary for efficient, accurate cleaning.

2 MODELING

In this section, we present the PClean modeling language, which is designed for the concise encoding of domain-specific knowledge about data and likely errors into generative models. PClean programs specify (i) a prior $p(\mathbf{R})$ over a latent ground-truth relational database of entities, and (ii) an observation model $p(\mathbf{D} \mid \mathbf{R})$ describing how the attributes of entities from $\mathbf{R}$ are reflected in an observed flat data table $\mathbf{D}$. Unlike general-purpose PPLs, PClean does not afford complete freedom in specifying $p(\mathbf{R})$. Instead, we impose a novel domain-general structure prior $p(\mathbf{S})$ on the *skeleton* of the database $\mathbf{R}$: $\mathbf{S}$ determines how many entities are in each latent database table, and which entities are related. The user's program encodes only $p(\mathbf{R} \mid \mathbf{S})$, a probabilistic relational model over the attributes of the objects whose existence and relationships are given by $\mathbf{S}$. This decomposition limits the PClean model class, but enables the development of an efficient SMC inference algorithm (Section 3).

2.1 PClean Modeling Language

A PClean program defines a set of *classes* $\mathcal{C} = (C_1, \ldots, C_k)$, one for each type of object underlying the user's data, and a *query* $\mathbf{Q}$ that describes how latent objects inform the observed flat dataset $\mathbf{D}$.

Class Declarations. The declaration of a PClean class C includes three kinds of statement: *reference statements* ($Y \sim C'$), which define a reference slot $C.Y$ connecting objects of class C to objects of a target class $T(C.Y) = C'$; *attribute statements* ($X \sim \phi_{C.X}(\ldots)$), which define a new *attribute* $C.X$ that objects of the class possess, and declare the prior distribution $\phi_{C.X}$ that it follows; and *parameter statements* (**parameter** $\theta_C \sim p_{\theta_C}(\ldots)$), which introduce mutually independent hyperparameters shared by all objects of the class C, to be learned from the noisy data. The prior $\phi_{C.X}$ for an attribute may depend on the values of a *parent set* $Pa(C.X)$ of attributes, potentially accessed via reference slots. For example, in Figure 2, the *Physician* class has a *school* reference slot with target class **School**, and a *degree* attribute whose value depends on *school.degree_dist*. Together, the attribute statements specify a *probabilistic relational model* Π for the user's schema (possibly parameterized by hyperparameters $\{\theta_C\}_{C \in \mathcal{C}}$) (Friedman et al., 1999).

Query. A PClean program ends with a *query*, connecting the schema of the latent relational database to the fields of the observed dataset. The query has the form **observe** $(U_1$ **as** $x_1), \cdots, (U_k$ **as** $x_k)$ **from** C_{obs}, where C_{obs} is a class that models the records of the observed dataset (*Record*, in Figure 2), x_i are the names of the columns in the observed dataset, and U_i are dot-

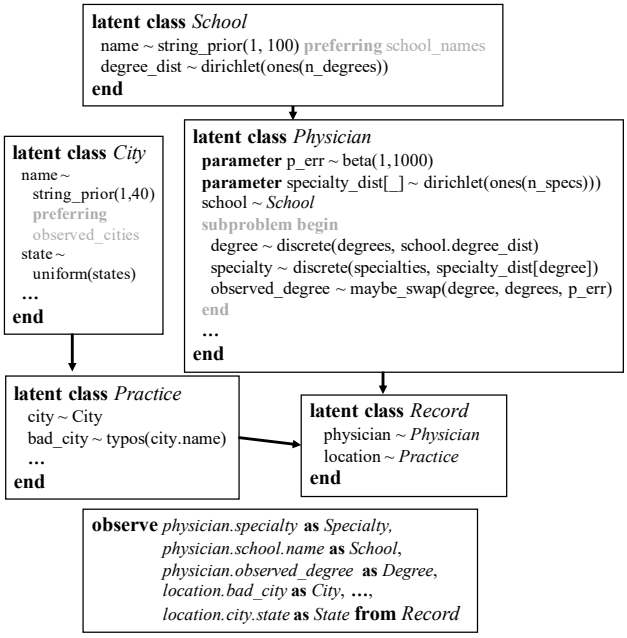

Figure 2: An example PClean program. PClean programs define: (i) an *acyclic* relational schema, comprising a set of classes $\mathcal{C}$, and for each class C, sets $\mathcal{A}(C)$ of attributes and $\mathcal{R}(C)$ of reference slots; (ii) a probabilistic relational model Π encoding priors for object attributes; and (iii) a query $\mathbf{Q}$ (last line), specifying how attributes are observed in the flat data table $\mathbf{D}$. *Inference hints* (in gray) do not change the model.

expressions (e.g., *physician.school.name*), picking out attributes accessible via zero or more reference slots from C_{obs}. The records of the dataset $\mathbf{D}$ are modeled as directly recording the values of the attributes named by U_i, each for a distinct object in C_{obs}. As such, errors must be modeled as *part* of the latent database $\mathbf{R}$, not as a separate stage of the generative process. For example, Figure 2 models systematic typos in the *City* field, by associating each *Practice* with a possibly misspelled version *bad_city* of the practice's city.

We emphasize that the relational schema and query are modeling choices: much of PClean's expressive power comes from the freedom to posit latent relational structure that is not directly reflected in the dataset. Figure 3 shows how this freedom can be used to capture several common data-cleaning motifs.

2.2 Non-Parametric Structure Prior $p(\mathbf{S})$

A PClean program's class declarations specify a probabilistic relational model that can be used to generate the attributes of objects in the latent database, but does not encode a prior over how many objects exist in each class or over their relationships. (The one exception is C_{obs}, the designated observation class,

Noisy Categorical with Unknown Domain

```
latent class Category   # Model true categories as latent objects
  name ~ string_prior(1, 10) preferring observed_labels
end                  # ^ min and max string length

latent class Record    # Each record has:
  category ~ Category  # a latent reference to a true category
  observed_category ~ typos(category.name) # + an observed label
end
observe observed_category as label from Record
```

Data Integration with Learned Reliability Rates per Source

```
latent class DataSource          latent class Object
  name ~ string_prior(1, 10)       attr ~ discrete(values, probs)
  reliability ~ beta(10, 100)    end
end
latent class DataRecord  # Each record contains information:
  src ~ DataSource       #   from an unreliable data source,
  obj ~ Object           #   about a latent object.
  observed_attr ~ maybe_swap(obj.attr, values, 1-src.reliability)
end               # ^ source's reliability determines error probability
observe src.name as source, observed_attr as attr from DataRecord
```

Systematic Unit Errors

```
latent class Country               miles(x) = 0.62x
  name ~ string_prior(1, 30)       km(x) = x
  miles_prevalence ~ beta(1, 1) # how often will measurements from
end                             # this country be in miles, not km?

latent class Trip
  parameter avg_km ~ normal(100, 10) # avg. trip distance, learned
  country ~ Country
  is_miles ~ bernoulli(country.miles_prevalence) # choose units
  distance ~ scaled_normal(is_miles ? miles : km,  avg_km, 10)
end          # transform sampled value (in km) according to units
observe country.name as location, distance as dist from Trip
```

Figure 3: PClean programs can concisely model a variety of data-cleaning scenarios.

whose objects are assumed to be in one-to-one correspondence with the rows of the observed dataset $\mathbf{D}$.) In this section, we introduce a domain-general structure prior $p(\mathbf{S}; |\mathbf{D}|, \mathcal{C})$ that encodes a non-parametric generative process over the *object sets* $\mathbf{S}_C$ associated with each class C, and over the values of each object's reference slots. The parameter $|\mathbf{D}|$ is the number of observed data records; $p(\mathbf{S}; |\mathbf{D}|, \mathcal{C})$ places mass only on relational skeletons in which there are exactly $|\mathbf{D}|$ objects in C_{obs} and every other object is connected via some chain of reference slots to one of them.

PClean's generative process for relational skeletons is shown in Figure 4. First, with probability 1, we set $\mathbf{S}_{C_{obs}} = \{1, \ldots, |\mathbf{D}|\}$ (a set of $|\mathbf{D}|$ distinct object IDs). PClean requires that the directed graph with edges $(C, T(C.Y))$ for each reference slot $C.Y$ be acyclic, which allows us to generate the remaining object sets class-by-class, processing a class only *after* any classes with reference slots targeting it. To generate an object set for class C, we first consider the reference set $\mathbf{Ref}_\mathbf{S}^C$ of all objects with reference slots targeting C:

$$\mathbf{Ref}_\mathbf{S}^C = \{(r, Y) \mid Y \in \mathcal{R}(C') \wedge T(C'.Y) = C \wedge r \in \mathbf{S}_{C'}\}$$

GenerateSkeleton$(\mathcal{C}, |\mathbf{D}|)$:
- ▷ Create one C_{obs} object per observed record
- $\mathbf{S}_{C_{obs}} := \{1, \ldots, |\mathbf{D}|\}$
- ▷ Generate a class *after* all referring classes:
- **for** class $C \in$ TopoSort$(\mathcal{C} \setminus \{C_{obs}\})$ **do**
  - ▷ Collect references to class C
  - $\mathbf{Ref}_\mathbf{S}^C := \{(r, Y) \mid r \in \mathbf{S}_{C'}, T(C'.Y) = C\}$
  - ▷ Generate targets of those references
  - $\mathbf{S}_C \sim$ GenerateObjectSet$(C, \mathbf{Ref}_\mathbf{S}^C)$
  - ▷ Assign reference slots pointing to C
  - **for** object $r' \in \mathbf{S}_C$ **do**
    - **for** referring object $(r, Y) \in r'$ **do**
      - $r.Y := r'$
- ▷ Return the skeleton
- **return** $\{\mathbf{S}_C\}_{C \in \mathcal{C}}, (r, Y) \mapsto r.Y$

GenerateObjectSet$(C, \mathbf{Ref}_\mathbf{S}^C)$:
- $s_C \sim Gamma(1, 1)$; $d_C \sim Beta(1, 1)$
- ▷ Partition references into co-referring subsets
- $\mathbf{S}_C \sim CRP(\mathbf{Ref}_\mathbf{S}^C, s_C, d_C)$
- **return** $\mathbf{S}_C$

Figure 4: PClean's non-parametric structure prior $p(\mathbf{S})$ over the relational skeleton $\mathbf{S}$ for a schema $\mathcal{C}$.

The elements of $\mathbf{Ref}_\mathbf{S}^C$ are pairs (r, Y) of an object and a reference slot; if a single object has two reference slots targeting class C, then the object will appear twice in the reference set. The point is to capture all of the places in $\mathbf{S}$ that will refer to objects of class C.

We then generate a *co-reference partition* of $\mathbf{Ref}_\mathbf{S}^C$, i.e., we partition the references to class C into disjoint subsets, within each of which we take all references to target the same object. To do this, we use the two-parameter Chinese restaurant process $CRP(X, s, d)$, which defines a non-parametric distribution over partitions of its set-valued parameter X. The strength s and discount d control the sizes of the clusters. The CRP generates a partition of all references to class C, and *we treat the resulting partition as the object set* $\mathbf{S}_C$, i.e., each component defines one object of class C:

$$\mathbf{S}_C \mid \mathbf{Ref}_\mathbf{S}^C \sim CRP(\mathbf{Ref}_\mathbf{S}^C, s_C, d_C)$$

To set the reference slots $r.Y$ with target class $T(\mathbf{Class}(r).Y) = C$, we simply look up which partition component (r, Y) (viewed as an element of $\mathbf{Ref}_\mathbf{S}^C$) was assigned to. Since we have equated these partition components with objects of class C, we can directly set $r.Y$ to point to the component (object) that contains (r, Y) as an element:

$$r.Y := \text{the unique } r' \in \mathbf{S}_{T(\mathbf{Class}(r).Y)} \text{ s.t. } (r, Y) \in r'$$

This procedure can be applied iteratively to generate object sets for every class and fill all reference slots.

3 INFERENCE

PClean's non-parametric structure prior ensures that PClean models admit a sequential representation, which can be used as the basis of a resample-move sequential Monte Carlo inference scheme (Section 3.1). However, if the SMC and rejuvenation proposals are made from the model prior, as is typical in PPLs, inference will still require prohibitively many particles to deliver accurate results. To address this issue, PClean uses a *proposal compiler* that exploits conditional independence in the model to generate fast enumeration-based proposal kernels for both SMC and MCMC rejuvenation (Section 3.2). Finally, to help users scale these proposals to large data, we introduce *inference hints*, lightweight annotations in the PClean program that can divide variables into subproblems to be separately handled by the proposal, or direct the enumerator to focus its efforts on a dynamically computed subset of a large discrete domain (Section 3.3).

3.1 Per-Observation Sequential Monte Carlo with Per-Object Rejuvenation

One version of the PClean model's generative process was given in Section 2: a skeleton can be generated from $p(\mathbf{S})$, then attributes can be filled in using the user's probabilistic relational model $p_\Pi(\mathbf{R} \mid \mathbf{S})$. Finally an observed dataset $\mathbf{D}$ can be generated according to the query $\mathbf{Q}$. But importantly, the model also admits a sequential representation, in which the latent database $\mathbf{R}$ is built in stages: at each stage, a single record is added to the observation class C_{obs}, along with any new objects in other classes that it refers to. Using this representation, we can run SMC on the model, building a particle approximation to the posterior that incorporates one observation at a time.

Database Increments. Let $\mathbf{R}$ be a database with designated observation class C_{obs}. Assume $\mathbf{R}_{C_{obs}}$, the object set for the class C_{obs}, is $\{1, \ldots, |\mathbf{D}|\}$. Then the database's i^{th} *increment* $\Delta^i_{\mathbf{R}}$ is the object set

$$\{r \in \mathbf{R} \mid \exists K,\, i.K = r \wedge \forall K', \forall j < i, j.K' \neq r\},$$

along with their attribute values and targets of their reference slots. Objects in $\Delta^i_{\mathbf{R}}$ may refer to other objects within the increment, or in earlier increments. Intuitively, the i^{th} increment of a database is the set of objects referenced by the i^{th} observation object, but *not* by any previous observation object $j < i$.

Sequential Generative Process. Figure 5 shows a generative process equivalent to the one in Section 2, but which generates the attributes and reference slots of each increment sequentially. Intuitively, the database is generated via a Chinese-restaurant 'social network': Consider a collection of restaurants, one

GenerateDataset(Π, $\mathbf{Q}$, $|\mathbf{D}|$):
  $\mathbf{R}^{(0)} \leftarrow \emptyset$ ▷ Initialize empty database
  for observation $i \in \{1, \ldots, |\mathbf{D}|\}$ **do**
    $\Delta^{\mathbf{R}}_i \leftarrow$ GenerateDbIncr($\mathbf{R}^{(i-1)}, C_{obs}$)
    $\mathbf{R}^{(i)} \leftarrow \mathbf{R}^{(i-1)} \cup \Delta^{\mathbf{R}}_i$
    $r \leftarrow$ the unique object of class C_{obs} in $\Delta^{\mathbf{R}}_i$
    $d_i \leftarrow \{X \mapsto r.\mathbf{Q}(X),\ \forall X \in \mathcal{A}(\mathbf{D})\}$
  return $\mathbf{R} = \mathbf{R}^{(|\mathbf{D}|)}, \mathbf{D} = (d_1, \ldots, d_{|\mathbf{D}|})$

GenerateDbIncr($\mathbf{R}^{(i-1)}$, root class C):
  $\Delta \leftarrow \emptyset$; $r_* \leftarrow$ a new object of class C
  for each reference slot $Y \in \mathcal{R}(C)$ **do**
    $C' \leftarrow T(C.Y)$
    for each object $r \in \mathbf{R}^{(i-1)}_{C'} \cup \Delta_{\mathbf{R}_{C'}}$ **do**
      $n_r \leftarrow |\{r' \mid r' \in \mathbf{R}^{(i-1)} \cup \Delta \wedge \exists \tau, r'.\tau = r\}|$
    $r_*.Y \leftarrow r$ w.p. $\propto n_r - d_{C'}$, or
      $\star$ w.p. $\propto s_{C'} + d_{C'} |\mathbf{R}^{(i-1)}_{C'} \cup \Delta_{\mathbf{R}_{C'}}|$
    if $r_*.Y = \star$ **then**
      $\Delta' \leftarrow$ GenerateDbIncr($\mathbf{R}^{(i-1)} \cup \Delta, C'$)
      $\Delta \leftarrow \Delta \cup \Delta'$
      $r_*.Y \leftarrow$ the unique r' of class C' in Δ'
  for each $X \in \mathcal{A}(C)$, in topological order **do**
    $r_*.X \sim \phi_{C.X}(\cdot \mid \{r_*.U\}_{U \in Pa(C.X)})$
  return $\Delta \cup \{r_*\}$

Figure 5: Sequential model representation.

for each class C, where each table serves a dish r representing an object of class C. Upon entering a restaurant, customers either sit at an existing table or start a new one, as in the usual generalized CRP construction. But these restaurants require that to start a new table, customers must first send $|\mathcal{R}(C)|$ friends to *other* restaurants (one to the target of each reference slot). Once the friends are seated at these *parent* restaurants, the original customer samples attributes $r.X$ of the new table's object, possibly informed by *their friends'* dishes (the objects $r.Y$ of class $T(C.Y)$). The process starts with $|\mathbf{D}|$ customers at the restaurant for C_{Obs}, who sit at separate tables; each customer who sits down triggers the sampling of one increment.

SMC Inference with Per-Object Rejuvenation. The sequential representation yields a sequence of intermediate unnormalized target densities $\tilde{\pi}_i$ for SMC:

$$\tilde{\pi}_i(\mathbf{R}) = \prod_{j=1}^{i} p(\Delta^{\mathbf{R}}_j \mid \Delta^{\mathbf{R}}_1, \ldots, \Delta^{\mathbf{R}}_{j-1}) p(d_j \mid \Delta^{\mathbf{R}}_1, \ldots, \Delta^{\mathbf{R}}_j).$$

Particles are initialized to hold an empty database, to which proposed increments $\Delta^{\mathbf{R}}_i$ are added each iteration. As is typical in SMC, at each step, the particles are reweighted according to how well they explain the new observed data, and resampled to cull low-weight particles while cloning and propagating promis-

Algorithm 1 Compiling SMC proposal to Bayesian network

procedure GENERATEINCREMENTBAYESNET(partial instance $\mathbf{R}^{(i-1)}$, data d_i)
▷ Set the vertices to all attributes and reference slots accessible from C_{obs}
$U \leftarrow \mathcal{A}(C_{obs}) \cup \{K \mid C_{obs}.K \text{ is a valid slot chain}\} \cup \{K.X \mid X \in \mathcal{A}(T(C_{obs}.K))\}$
▷ Determine parent sets and CPDs for each variable
for each variable $u \in U$ **do**
if $u \in \mathcal{A}(C_{obs})$ **then**
Set $Pa(u) = Pa^{\Pi}(C.u)$
Set $\phi_u(v_u \mid \{v_{u'}\}_{u' \in Pa(u)}) = \phi^{\Pi}_{C.u}(v_u \mid \{v_{u'}\}_{u' \in Pa(u)})$
else if $u = K.X$ for $X \in \mathcal{A}(T(C_{obs}.K))$ **then**
Set $Pa(u) = Pa^{\Pi}(T(C_{obs}.K).X) \cup \{K\} \cup \{u'.X \mid u' \text{ already processed} \wedge T(C_{obs}.u') = T(C_{obs}.K)\}$
Set

$$\phi_u(v_u \mid \{v_{u'}\}_{u' \in Pa(u)}) = \begin{cases} \mathbf{1}[v_u = v_K.X] & v_K \in \mathbf{R}^{(i-1)} \\ \phi^{\Pi}_{T(C_{obs}.K).X}(v_u \mid \{v_{u'}\}_{u' \in Pa^{\Pi}(T(C_{obs}.K).X)}) & v_K = \mathbf{new}_K \\ \mathbf{1}[v_u = v_{u'.X}] & v_K = \mathbf{new}_{u'}, u' \neq K \end{cases}$$

else
Set $Pa(u)$ to already-processed slot chains u' s.t. $T(C.u') = T(C.u)$, and K if $u = K.Y$
Set domain $V(u) = \mathbf{R}^{(i-1)}_{T(C.u)} \cup \{\mathbf{new}_{u'} \mid u' \in Pa(u) \cup \{u\}\}$
Set $\phi_u(v_u \mid \{v_{u'}\}_{u' \in Pa(u)})$ according to CRP, or to $\mathbf{1}[v_u = v_K.Y]$ if $u = K.Y$ and $v_K \in \mathbf{R}^{(i-1)}$
for attribute $X \in \mathcal{A}(\mathbf{D})$ **do**
Change node $\mathbf{Q}(u)$ to be observed with value $d_i.x$, **unless** $d_i.x$ is missing

ing ones. This process allows the algorithm to hypothesize new latent objects as needed to explain each new observation, but not to revise earlier inferences about latent objects (or delete previously hypothesized objects) in light of new observations; we address this problem with MCMC rejuvenation moves. These moves select an object r, and update all r's attributes and reference slots in light of all relevant data incorporated so far. In doing so, these moves may also lead to the "garbage collection" of objects that are no longer connected to the observed dataset, or to the insertion of new objects as targets of r's reference slots.

3.2 Compiling Data-Driven SMC Proposals

Proposal quality is the determining factor for the quality of SMC inference: at each step of the algorithm, a proposal $Q_i(\Delta_i^{\mathbf{R}}; \mathbf{R}^{(i-1)}, d_i)$ generates proposed additions $\Delta_i^{\mathbf{R}}$ to the existing latent database $\mathbf{R}^{(i-1)}$ to explain the i^{th} observed data point, d_i. A key limitation of the sequential Monte Carlo implementations in most general-purpose PPLs today is that the proposals Q_i are not *data-driven*, but rather based only on the prior: they make blind guesses as to the latent variable values and thus tend to make proposals that explain the data poorly. By contrast, PClean compiles proposals that use exact enumerative inference to propose discrete variables in a data-driven way. This approach extends ideas from Arora et al. (2012) to the block Gibbs rejuvenation and block SMC setting, with user-specified blocking hints. These proposals are *locally optimal* for models that contain only discrete finite-domain variables, meaning that of all possible proposals Q_i they minimize the divergence

$$KL(\pi_{i-1}(\mathbf{R}^{(i-1)})Q_i(\Delta_i^{\mathbf{R}}; \mathbf{R}^{(i-1)}, d_i) || \pi_i(\mathbf{R}^{(i-1)} \cup \Delta_i^{\mathbf{R}})).$$

The distribution on the left represents a perfect sample $\mathbf{R}^{(i-1)}$ from the target given the first $i-1$ observations, extended with the proposal Q_i. The distribution on the right is the target given the first i data points. In our setting the locally optimal proposal is given by

$$Q_i(\Delta_i^{\mathbf{R}}; \mathbf{R}^{(i-1)}, d_i) \propto p(\Delta_i^{\mathbf{R}} \mid \Delta_1^{\mathbf{R}}, \ldots, \Delta_{i-1}^{\mathbf{R}}) p(d_i \mid \Delta_1^{\mathbf{R}}, \ldots, \Delta_i^{\mathbf{R}}).$$

Algorithm 1 shows how to compile this distribution to a Bayesian network; when the latent attributes have finite domains, the normalizing constant can be computed and the locally optimal proposal can be simulated (and evaluated) exactly. This is possible because there are only a finite number of instantiations of the random increment $\Delta_i^{\mathbf{R}}$ to consider. The compiler generates efficient enumeration code separately for each pattern of missing values it encounters in the dataset, exploiting conditional independence relationships in each Bayes net to yield potentially exponential savings over naive enumeration. A similar strategy can be used to compile data-driven object-wise rejuvenation proposals, and to handle some continuous variables with conjugate priors; see supplement for details.

3.3 Scaling to Large Models and Data with Inference Hints

Scaling to models with large-domain variables and to datasets with many rows is a key challenge. In PClean, users can specify lightweight *inference hints* to the proposal compiler, shown in gray in Figure 2, to speed up inference without changing model's meaning.

Programmable Subproblems. First, users may group attribute and reference statements into blocks by wrapping them in the syntax **subproblem begin** ... **end**. This partitions the attributes and reference slots of a class into an ordered list of *subproblems*, which SMC uses as intermediate target distributions. This makes enumerative proposals faster to compute, at the cost of considering less information at each step; rejuvenation moves can often compensate for short-sighted proposals.

Adaptive Mixture Proposals with Dynamic Preferred Values. A random variable within a model may be intractable to enumerate. For example, `string_prior(1, 100)` is a distribution over all strings between 1 and 100 letters long. To handle these, PClean programs may declare *preferred values hints*. Instead of $X \sim d(E, \ldots, E)$, the user can write $X \sim d(E, \ldots, E)$ **preferring** E, where the final expression gives a list of values ξ_X on which the posterior mass is expected to concentrate. When enumerating, PClean replaces the CPD ϕ_X with a surrogate $\hat{\phi}_X$, which is equal to ϕ_X for preferred value inputs in ξ_X, but 0 for all other values. The mass not captured by the preferred values, $1 - \sum_{x \in \xi_X} \phi_X(x)$, is assigned to a special **other** token. Enumeration yields a partial proposal $\hat{Q}$ over a modified domain; the full proposal Q first draws from $\hat{Q}$ then replaces **other** tokens with samples from the appropriate CPDs $\phi_X(\cdot \mid Pa(X))$. This yields a mixture proposal between the enumerative posterior on preferred values and the prior: when none of the preferred values explain the data well, **other** will dominate, causing the attribute to be sampled from its prior. But if any of the preferred values are promising, they will almost certainly be proposed.

4 EXPERIMENTS

In this section, we demonstrate empirically that (1) PClean's inference works when standard PPL inference strategies fail, (2) short PClean programs suffice to compete with existing data cleaning systems in both runtime and accuracy, (3) PClean can scale to large real-world datasets, and (4) PClean's inference can deliver calibrated and useful estimates of uncertainty. In Experiments (1)-(3), we evaluate PClean's accuracy using a single posterior sample (the last iterate of PClean's final MCMC rejuvenation sweep); in Experiment (4), we consider an uncertainty-aware, multi-sample estimator of the clean dataset, which exploits our Bayesian framework for higher-precision repairs. Experiments were run on a laptop with a 2.6 GHz CPU and 32 GB of RAM.

(1) Comparison to Generic PPL Inference. We evaluate PClean's inference against standard PPL inference algorithms reimplemented to work on PClean models, on a popular benchmark from the data cleaning literature (Figure 6). We do not compare directly to other PPLs' implementations, because many (e.g. BLOG) cannot represent PClean's non-parametric prior. Some languages (e.g. Turing) have explicit support for non-parametric distributions, but could not express PClean's recursive use of CRPs. Others could in principle express PClean's model, but would complicate an algorithm comparison in other ways: Venture's dynamic dependency tracking is thousands of times slower than SOTA; Pyro's focus is on variational inference, hard to apply in PClean models; and Gen supports non-parametrics only via the use of mutation in its slower dynamic modeling language (making SMC $O(N^2)$) or via low-level extensions that would amount to reimplementing PClean using Gen's abstractions. Nonetheless, the algorithms in Figure 6 are inspired by the generic automated inference provided in many PPLs, which use top-down proposals from the prior for SMC, MH (Goodman and Stuhlmüller, 2014; Ritchie et al., 2016), and PGibbs (Wood et al., 2014; Murray, 2015; Mansinghka et al., 2014). Our results show that PClean suffices for fast, accurate inference where generic techniques fail, and also demonstrate why inference hints are necessary for scalability: without subproblem hints, PClean takes much longer to converge, even though it eventually arrives at a similar F_1 value.

(2) Applicability to Data Cleaning. To check PClean's modeling and inference capabilities are good for data cleaning *in absolute terms* (rather than relative to generic PPL inference), we contextualize PClean's accuracy and runtime against two SOTA data-cleaning systems on three benchmarks with known ground truth (Table 1), described in detail in the supplement. Briefly, the datasets are *Hospital*, a standard benchmark with artificial typos in 5% of cells that can be corrected by leveraging duplication of entities across rows; *Flights*, a standard benchmark integrating flight information (e.g. arrival, departure times) from conflicting real-world data sources; and *Rents*, a synthetic dataset based on census statistics, featuring continuous and discrete values, misspelled county names, missing apartment sizes, and unit errors. The baseline systems are *HoloClean* (Rekatsinas et al., 2017), based on probabilistic machine learning,

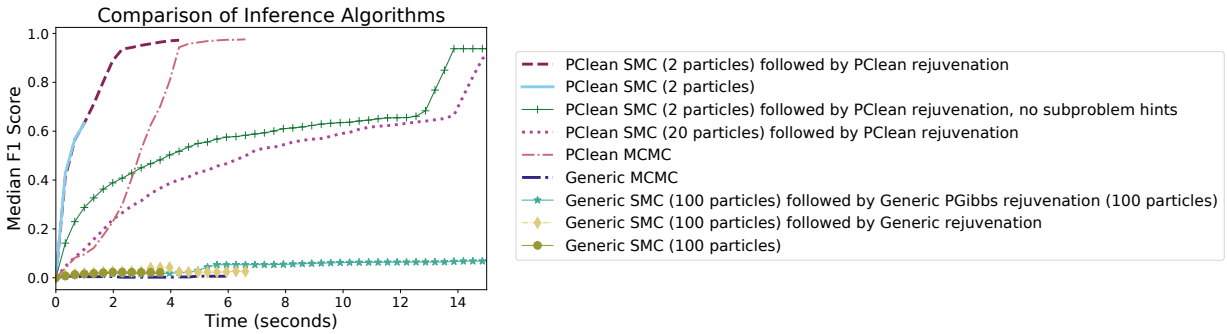

Figure 6: Median accuracy vs. runtime for five runs of alternative inference algorithms on the *Hospital* dataset (Chu et al., 2013), with an additional 20% of cells artificially deleted so as to test both repair and imputation.

Task	Metric	PClean	HoloClean (Unpublished)	HoloClean	NADEEF	NADEEF + Manual Java Heuristics
Flights	F_1	**0.90**	0.64	0.41	0.07	**0.90**
	Time	**3.1s**	45.4s	32.6s	9.1s	14.5s
Hospital	F_1	**0.91**	0.90	0.83	0.84	0.84
	Time	**4.5s**	1m 10s	1m 32s	27.6s	22.8s
Rents	F_1	**0.69**	0.48	0.48	0	0.51
	Time	1m 20s	20m 16s	13m 43s	13s	**7.2s**

Table 1: Results of PClean and various baseline systems on three diverse cleaning tasks.

and *NADEEF*, which uses a MAX-SAT solver to minimize violations of user-defined cleaning rules (Dallachiesat et al., 2013). For HoloClean, we consider both the original code and the authors' latest (unpublished) version on GitHub; for NADEEF, we include results both using NADEEF's streamlined rule-definition interface and with custom, handwritten Java rules (more expressive but also more cumbersome).

Table 1 reports F_1 scores and cleaning speed (see supplement for precision/recall). We do not aim to anoint a single 'best cleaning system,' since optimality depends on the available domain knowledge and the user's desired level of customization. Further, while we followed system authors' per-dataset recommendations where possible, a pure system comparison is difficult, since each system relies on its own rule configuration. Rather, we note that short (<50-line) PClean programs can encode knowledge useful in practice for cleaning diverse data, and inference is good enough to achieve F_1 scores as good or better than SOTA data-cleaning systems on all three datasets, often in less wall-clock time. As an illustration of the value and convenience of encoding relevant knowledge, on *Flights*, a baseline, 16-line PClean program earns an F_1 score of 0.60, but the F_1 can be boosted to 0.69 by encoding that sources have varying reliability (+1 line), and to 0.90 by encoding that for a given flight, an airline's own website is most likely to be reliable (+1 line). By contrast, adding a similar reliability heuristic to NADEEF required 50 lines of Java; see supplement.

(3) Scalability to Large, Real-World Data. We ran PClean on the Medicare Physician Compare National dataset, shown earlier in Figure 1. It contains 2.2 million records, each listing a clinician and a practice location; the same clinician may work at multiple practices, and many clinicians may work at the same practice. NULLs and systematic errors are common (e.g. consistently misspelled city names at a practice).

PClean took 7h36m, performing 8,245 repairs and 1,535,415 imputations. Out of 100 randomly chosen imputations, 90% agreed with manually obtained ground truth. We also verified that 7,954 repairs (96.5%) were correct (some were correct normalization, e.g. choosing a single spelling for cities whose names could be spelled multiple ways). By contrast, NADEEF changed 88 cells across the whole dataset, and HoloClean did not initialize in 24 hours, using the configuration provided by HoloClean's authors.

Figure 1 shows PClean's real behavior on four rows. Consider the misspelling *Abington, MD*, which appears in 152 entries. The correct spelling *Abingdon, MD* occurs in only 42. PClean still recognizes *Abington* as an error, because all 152 instances share a single practice address, and errors are modeled as systematic at the practice level. Next, consider PClean's correct inference that Ryan's degree is *DO*: more *Family Medicine* doctors are MDs than DOs, but the school *PCOM* awards many more DOs than MDs. All parameters enabling this reasoning are learned from the dirty data.

(4) MAP Estimation and Bayesian Uncertainty Quantification. Our previous experiments used a single posterior sample to estimate the clean dataset. This experiment investigates strategies for exploiting richer information about the posterior, namely: (1) using the most common predictions across multiple independent posterior samples (approximating the MAP clean dataset), and (2) setting a confidence threshold for repairs, to trade recall for higher precision. In particular, we ran PClean's inference with 10 parallel chains on the *Rents* dataset, and collected 1 posterior sample from each (the last iterate of MCMC rejuvenation). Figure 7's left panel shows, in blue, the precision and recall achieved by considering each sample individually, and in red, the various precision/recall tradeoffs achievable by using *most common* prediction (across all 10 samples) for each cell, or leaving a cell unmodified if the confidence (i.e., the proportion of samples in which the modal value was predicted) does not surpass a threshold. The optimal F_1 of 0.73 ($R = 0.70, P = 0.77$), a 4-point improvement over the results from Table 1, is achieved by thresholding at 0.5. The right panel of Figure 7 shows a calibration plot, obtained by binning the cells of the *Rents* dataset into confidence levels and measuring the proportion of cells in each bin for which the most common prediction was correct. A caveat of our approach to uncertainty-aware cleaning is that performing independent repairs *per cell* will not necessarily approximate the overall MAP clean dataset, unless the posterior is actually independent. Moreover, *Rents* is a challenging but synthetic dataset; the degree to which these calibration results replicate in real-world problems will depend on the fidelity of the user's PClean program. However, our results suggest that PClean's inference engine is capable of delivering not only accurate cleaning results but also useful estimates of uncertainty.

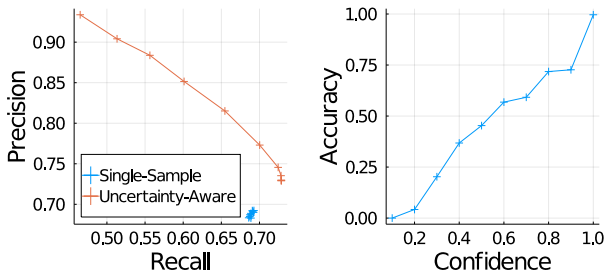

Figure 7: Exploiting Bayesian uncertainty with PClean on *Rents*. Left: Ten independent posterior samples were generated. Blue marks show precision and recall achieved by considering each sample separately, whereas red curve shows precision vs. recall tradeoff when using modal predictions across samples, making repairs only when predictions surpass a confidence threshold. Right: PClean's uncertainty quantification appears to be well-calibrated on *Rents*. 'Confidence' is the proportion of independent posterior samples that agree on the modal predicted value for a cell; 'accuracy' is the proportion of cells at a certain confidence level for which the modal prediction is correct.

5 DISCUSSION

PClean, like other domain-specific PPLs, aims to be more automated and scalable than general purpose PPLs, by leveraging structure in its restricted model class to deliver fast inference. At the same time, it aims to be expressive enough to concisely solve a broad class of real-world data cleaning problems.

Future development of PClean could build a more extensive standard library of primitives for modeling diverse data types (perhaps including neural models for text or image data), and a more robust proposal compiler for free-text and continuous latent variables (perhaps based on learning neural proposals for selected attributes). Integrating PClean with the abstractions of a mature probabilistic programming system, such as Gen, could facilitate implementation. PClean's scalability could also be improved, by exploring distributed variants of PClean based on Bayesian formulations of blocking (Marchant et al., 2021). A more speculative research direction is to (partially) automate PClean program authoring, by applying techniques such as automated error modeling (Heidari et al., 2019) or probabilistic program synthesis (Saad et al., 2019; Choi et al., 2020). It could also be fruitful to develop hierarchical variants of PClean that enable learned parameters and latent objects to transfer across datasets.

PClean could be described as a *data-driven* probabilistic expert system (Horvitz et al., 1988; Pearl, 1988; Heckerman et al., 1992; Shafer, 1996), incorporating ideas from probabilistic programming to scale to messy, real-world domain knowledge and data. Crucially, since PClean can infer the objects and parameters of a domain from data, users need only encode higher-level domain knowledge, not brittle details. It remains to be seen whether systems like PClean can be made to give meaningful explanations of individual judgments (like those offered by human experts).

Our results show that probabilistic programs can clean dirty, denormalized data with state-of-the-art accuracy and performance. More broadly, PClean joins existing domain-specific PPLs in demonstrating that it is feasible and useful to integrate sophisticated styles of modeling and inference, developed over years of research, into simple languages and specialized inference engines. We hope PClean proves useful to practitioners, and that it encourages researchers to develop new domain-specific PPLs for other important problems.

Acknowledgements

The authors are grateful to Zia Abedjan, Marco Cusumano-Towner, Raul Castro Fernandez, Cameron Freer, Divya Gopinath, Christina Ji, Tim Kraska, George Matheos, Feras Saad, Michael Stonebraker, Josh Tenenbaum, and Veronica Weiner for useful conversations and feedback, as well as to our anonymous referees for their constructive suggestions. This work is supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1745302; DARPA, under the Machine Common Sense (MCS) and Synergistic Discovery and Design (SD2) programs; gifts from the Aphorism Foundation and the Siegel Family Foundation; a research contract with Takeda Pharmaceuticals; and financial support from Facebook, Google, and the Intel Probabilistic Computing Center.

References

Abedjan, Z., Chu, X., Deng, D., Fernandez, R. C., Ilyas, I. F., Ouzzani, M., Papotti, P., Stonebraker, M., and Tang, N. (2016). Detecting data errors: Where are we and what needs to be done? In *Proceedings of the VLDB Endowment*.

Arora, N. S., Braz, R. d. S., Sudderth, E. B., and Russell, S. (2012). Gibbs sampling in open-universe stochastic languages. *arXiv preprint arXiv:1203.3464*.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M. A., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*.

Choi, Y., Dang, M., and Broeck, G. V. d. (2020). Group fairness by probabilistic modeling with latent fair decisions. *arXiv preprint arXiv:2009.09031*.

Chu, X., Ilyas, I. F., and Papotti, P. (2013). Holistic data cleaning: Putting violations into context. In *Proceedings - International Conference on Data Engineering*.

Cusumano-Towner, M. F., Lew, A. K., Saad, F. A., and Mansinghka, V. K. (2019). Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

Dallachiesat, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I. F., Ouzzani, M., and Tang, N. (2013). NADEEF: A commodity data cleaning system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

De, S., Hu, Y., Meduri, V. V., Chen, Y., and Kambhampati, S. (2016). BayesWipe: A scalable probabilistic framework for improving data quality. *Journal of Data and Information Quality*, 8(1).

De Sa, C., Ilyas, I. F., Kimelfeld, B., Ré, C., and Rekatsinas, T. (2019). A formal framework for probabilistic unclean databases. In *Leibniz International Proceedings in Informatics, LIPIcs*.

Eduardo, S., Nazábal, A., Williams, C. K., and Sutton, C. (2020). Robust variational autoencoders for outlier detection and repair of mixed-type data. In *International Conference on Artificial Intelligence and Statistics*, pages 4056–4066. PMLR.

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999). Learning probabilistic relational models. In *IJCAI International Joint Conference on Artificial Intelligence*.

Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. (2008). Church: A Language for Generative Models. In *Proceedings of the 24th Annual Conference on Uncertainty in Artificial Intelligence (UAI 2008)*, pages 220–229. AUAI Press.

Goodman, N. D. and Stuhlmüller, A. (2014). The Design and Implementation of Probabilistic Programming Languages. http://dippl.org. Accessed: 2020-10-15.

Gordon, A. D., Graepel, T., Rolland, N., Russo, C., Borgström, J., and Guiver, J. (2014). Tabular: A schema-driven probabilistic programming language. *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, (1):321–334.

Heckerman, D. E., Horvitz, E. J., and Nathwani, B. N. (1992). Toward normative expert systems: Part I, the PATHFINDER project. *Methods of information in medicine*, 31(02):90–105.

Heidari, A., McGrath, J., Ilyas, I. F., and Rekatsinas, T. (2019). HoloDetect: Few-shot learning for error detection. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

Horvitz, E. J., Breese, J. S., and Henrion, M. (1988). Decision theory in expert systems and artificial intelligence. *International journal of approximate reasoning*, 2(3):247–302.

Hu, Y., De, S., Chen, Y., and Kambhampati, S. (2012). Bayesian Data Cleaning for Web Data.

Kubica, J. and Moore, A. (2003). Probabilistic noise identification and data cleaning. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 131–138.

Kulkarni, T. D., Kohli, P., Tenenbaum, J. B., and Mansinghka, V. (2015). Picture: A probabilistic programming language for scene perception. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Mansinghka, V., Selsam, D., and Perov, Y. N. (2014). Venture: a higher-order probabilistic programming platform with programmable inference. pages 1–78.

Marchant, N. G., Kaplan, A., Elazar, D. N., Rubinstein, B. I., and Steorts, R. C. (2021). d-blink: Distributed end-to-end bayesian entity resolution. *Journal of Computational and Graphical Statistics*, pages 1–16.

Matsakis, N. E. (2010). Active Duplicate Detection with Bayesian Nonparametric Models.

Mayfield, C., Neville, J., and Prabhakar, S. (2009). A statistical method for integrated data cleaning and imputation. Technical report.

Mccallum, A. and Wellner, B. (2003). Object Consolodation by Graph Partitioning with a Conditionally-Trained Distance Metric. *In Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 19–24.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L., and Kolobov, A. (2005). BLOG: Probabilistic Models With Unknown Objects. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 1352–1359. Morgan Kaufmann Publishers Inc.

Milch, B. and Russell, S. (2006). General-purpose MCMC inference over relational structures. *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence, UAI 2006*, pages 349–358.

Murray, L., Lundén, D., Kudlicka, J., Broman, D., and Schön, T. (2018). Delayed sampling and automatic rao-blackwellization of probabilistic programs. In *International Conference on Artificial Intelligence and Statistics*, pages 1037–1046.

Murray, L. M. (2015). Bayesian state-space modelling on high-performance hardware using LibBi. *Journal of Statistical Software*, 67(10):1–28.

Murray, L. M. and Schön, T. B. (2018). Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46:29–43.

Pasula, H., Marthi, B., Milch, B., Russell, S., and Shpitser, I. (2003). Identity uncertainty and citation matching. *Advances in Neural Information Processing Systems*.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

Rekatsinas, T., Chuy, X., Ilyasy, I. F., and Ré, C. (2017). HoloClean: Holistic data repairs with probabilistic inference. In *Proceedings of the VLDB Endowment*.

Ritchie, D., Stuhlmüller, A., and Goodman, N. (2016). C3: Lightweight incrementalized mcmc for probabilistic programs using continuations and callsite caching. In *Artificial Intelligence and Statistics*, pages 28–37.

Saad, F. A., Cusumano-Towner, M., Schaechtle, U., Rinard, M. C., and Mansinghka, V. K. (2019). Bayesian Synthesis of Probabilistic Programs for Automatic Data Modeling. *Proc. ACM Program. Lang.*, 3(POPL):37:1—-37:29.

Ścibior, A., Kammar, O., and Ghahramani, Z. (2018). Functional programming for modular bayesian inference. *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–29.

Shafer, G. (1996). *Probabilistic expert systems*. SIAM.

Steorts, R. C., Hall, R., and Fienberg, S. E. (2016). A Bayesian Approach to Graphical Record Linkage and Deduplication. *Journal of the American Statistical Association*, 111(516):1660–1672.

Tolpin, D., Van De Meent, J. W., Yang, H., and Wood, F. (2016). Design and implementation of probabilistic programming language anglican. In *ACM International Conference Proceeding Series*.

Wellner, B., McCallum, A., Peng, F., and Hay, M. (2004). An integrated, conditional model of information extraction and coreference with application to citation matching. *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 593–601.

Wick, M., Singh, S., Pandya, H., and McCallum, A. (2013). A joint model for discovering and linking entities. *AKBC 2013 - Proceedings of the 2013 Workshop on Automated Knowledge Base Construction, Co-located with CIKM 2013*, pages 67–71.

Winn, J., Guiver, J., Webster, S., Zaykov, Y., Kukla, M., and Fabian, D. (2017). Alexandria : Unsupervised High-Precision Knowledge Base Construction using a Probabilistic Program. pages 1–20.

Wood, F., Meent, J. W., and Mansinghka, V. (2014). A New Approach to Probabilistic Programming Inference. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, volume 33 of *Proceedings of Machine Learning Research*, pages 1024–1032. PMLR.

Xiong, L., Póczos, B., Schneider, J., Connolly, A., and VanderPlas, J. (2011). Hierarchical probabilis-

tic models for group anomaly detection. *Journal of Machine Learning Research*, 15:789–797.

Zhao, B., Rubinstein, B. I. P., Gemmell, J., and Han, J. (2012). A Bayesian approach to discovering truth from conflicting sources for data integration. *Proceedings of the VLDB Endowment*, 5(6):550–561.