d-blink: Distributed End-to-End Bayesian Entity Resolution

Neil G. Marchant[a] Rebecca C. Steorts[b] Andee Kaplan[c]
Benjamin I. P. Rubinstein[a] Daniel N. Elazar[d]

[a]School of Computing and Information Systems, University of Melbourne
[b]Department of Statistical Science, Duke University
[c]Department of Statistics, Colorado State University
[d]Methodology Division, Australian Bureau of Statistics

September 16, 2019

Abstract

Entity resolution (ER) (record linkage or de-duplication) is the process of merging together noisy databases, often in the absence of a unique identifier. A major advancement in ER methodology has been the application of Bayesian generative models. Such models provide a natural framework for clustering records to unobserved (latent) entities, while providing exact uncertainty quantification and tight performance bounds. Despite these advancements, existing models do not scale to realistically-sized databases (larger than 1000 records) and they do not incorporate probabilistic blocking. In this paper, we propose "distributed Bayesian linkage" or `d-blink`—the first scalable and distributed end-to-end Bayesian model for ER, which propagates uncertainty in blocking, matching and merging. We make several novel contributions, including: (i) incorporating probabilistic blocking directly into the model through auxiliary partitions; (ii) support for missing values; (iii) a partially-collapsed Gibbs samper; and (iv) a novel perturbation sampling algorithm (leveraging the Vose-Alias method) that enables fast updates of the entity attributes. Finally, we conduct experiments on five data sets which show that `d-blink` can achieve significant efficiency gains—in excess of 300×—when compared to existing non-distributed methods.

Keywords: auxiliary variable, distributed computing, Markov chain Monte Carlo, partially-collapsed Gibbs sampling, record linkage

1 INTRODUCTION

Entity resolution (ER) (record linkage and de-duplication) is the process of identifying and merging records across data sources that refer to the same underlying entities. In the absence of unique identifiers (e.g. social security numbers), entities are resolved by searching for clusters of records that have sufficiently similar attributes. This can be a challenging task due to disparities between data sources, poor data quality and temporal variation.

Much of the ER literature advocates a pipeline approach (Christen, 2012; Dong and Srivastava, 2015) comprising four main stages. In the first stage, known as *attribute/schema alignment*, a set of common attributes are identified among the data sets. In the second stage, known as *blocking* or *indexing*, the set of all record pairs is pruned to yield a smaller set of candidate matches. This set is refined in the third stage, known as *matching* or *scoring*, typically by thresholding a weighted sum of pairwise attribute similarities. Finally, in the fourth stage, known as *merging* or *fusion*, the matching record pairs are mapped to consistent clusters, which are then merged to produce single representative records.

While a pipeline approach can be flexible and efficient at scale, it suffers from a major deficiency: uncertainties encountered at each stage are not propagated through the pipeline, nor are they included in the final output. As a result, an incorrect decision at an earlier stage (such as a poor blocking design) cannot be corrected at later stages. In short, information is lost and accuracy may suffer. Furthermore, analysts are unable to incorporate ER uncertainty in a post-ER analysis, e.g. an inferential or prediction task.

Bayesian modeling offers a promising solution to the problem of rigorous uncertainty propagation. Indeed, Bayesian modeling for ER has been an active area of research in recent years (Copas and Hilton, 1990; Belin and Rubin, 1995; Larsen and Rubin, 2001; Tancredi and Liseo, 2011; Zhao et al., 2012; Gutman et al., 2013; Sadinle, 2014; Steorts, 2015; Steorts et al., 2016; Zanella et al., 2016; Sadinle, 2017). In addition to providing a principled treatement of uncertainty, the Bayesian paradigm allows for flexible modeling assumptions, including the incorporation of prior information. The latter can provide a regularizing effect which is useful when little or no training data is available. Recent work has also shown that theoretical performance bounds can be obtained for Bayesian ER models (Steorts et al., 2017), whereas counterparts for rule-based and discriminative methods seem implausible.

Despite the successes of Bayesian models for ER, we are not aware of any models that scale to realistically-sized databases (larger than $n \sim 10^3$ records). In conventional ER pipelines, scalability is addressed at the blocking/indexing stage, where a large fraction of the $O(n^2)$ possible record comparisons are eliminated (Christen, 2012). "Traditional blocking" (Steorts et al., 2014) has been applied in an ad-hoc fashion to Bayesian ER models (Steorts et al., 2016), however it does not preserve the posterior distribution, nor does it propagate uncertainty between the blocking and matching stages.

In this paper, we propose a scalable and distributed extension to the `blink` model for end-to-end Bayesian ER (Steorts, 2015), which we call "distributed `blink`" or `d-blink`. Our key insight is an auxiliary variable representation that induces a partitioning over the entities and records, while preserving the marginal posterior. The partitions play a similar role as traditional blocks, however the assignment of entities and records to partitions is probabilistic and is inferred jointly with the other variables. This representation

enables distributed inference at the partition level, as variables in distinct partitions become conditionally independent. The fact that this is possible gives the `blink` model a clear advantage in terms of scalability. It is unclear whether a similar approach can be adapted to other Bayesian ER models with more complex (e.g. non-parametric) linkage priors.

To construct the partitions, we propose a method based on k-d trees which co-partitions similar entities, while achieving well-balanced partitions. We make several other novel contributions including support for record values missing completely at random; a partially-collapsed Gibbs sampler that balances constraints imposed by distributed computing; and insights into improving computational efficiency. The latter insights include: (i) a sub-quadratic algorithm for updating links based on indexing; (ii) a truncation approximation for the attribute similarities; and (iii) a novel perturbation sampling algorithm for updating the entity attributes, which leverages the Vose-Alias method (Vose, 1991).

We implement our proposed methodology as an open-source Apache Spark package and conduct empirical evaluations on two synthetic and three real data sets. One of the synthetic data sets—`RLdata10000`—mimics administrative data with distortion and duplication and has been commonly used as a benchmark in the record linkage literature. The second synthetic data set (`ABSEmployee`) is motivated by real linkages of employment survey data undertaken at the Australian Bureau of Statistics (ABS). It is not derived from real data due to strict confidentiality restrictions. We include this data set in the supplementary material so that others may use it for evaluation of data linkage methods.

Additionally, we apply our methodology to three real applications of entity resolution, each of which is an important and challenging problem. First we link two snapshots of the North Carolina Voter Registration (`NCVR`) data set. It is necessary to identify duplicates within and across snapshots before computing statistics or performing prediction or inferential tasks. The second application is from the medical field, where it is common for records to be duplicated across locally-managed databases. Specifically, we examine de-identified person data from a longitudinal study called the National Long Term Care Survey (`NLTCS`). There are duplicates across all three waves of the study, which must be removed before any analysis can be performed. The third data set is derived from the 2008 and 2010 waves of the Survey of Household Income and Wealth (`SHIW`). It is a de-identified categorical data set and duplicates must be removed to draw valid statistical conclusions. We choose to work with three very different real applications to understand the scalability and accuracy of our methods, in cases where ground truth is known.

Our computational and sampling improvements can yield efficiency gains in excess of $300\times$ compared to `blink`, and we observe near-linear gains with increasing numbers of partitions. Crucially, our results indicate that these efficiency dividends do not come at the cost of accuracy.

2 RELATED WORK

The related work spans three key areas: ER methodology, inference for Bayesian ER models, and distributed MCMC.

Entity resolution methodology. Matching pairs of tables was first addressed with a probabilistic approach by Newcombe et al. (1959). Fellegi and Sunter (1969) later formalized this approach within the framework of decision theory. Many extensions to the Fellegi-Sunter (FS) model have been proposed (for surveys, see Winkler, 2006, 2014), including a recent extension to multiple databases (Sadinle and Fienberg, 2013). However, the FS model has been criticized due to its lack of support for duplicates within databases; misspecified independence assumptions; and its dependence on subjective thresholds (Tancredi and Liseo, 2011). These limitations have prompted development of more sophisticated Bayesian generative models, including models for bipartite matching (Copas and Hilton, 1990; Belin and Rubin, 1995; Larsen and Rubin, 2001; Tancredi and Liseo, 2011; Gutman et al., 2013; Sadinle, 2017), de-duplication (Sadinle, 2014; Steorts, 2015; Steorts et al., 2016) and matching across multiple databases (Steorts, 2015; Steorts et al., 2016). All of these models operate on structured data, and most (Copas and Hilton, 1990; Belin and Rubin, 1995; Larsen and Rubin, 2001; Tancredi and Liseo, 2011; Gutman et al., 2013; Steorts et al., 2016) rely solely on binary attribute-level comparisons to inform matching. Recent models (Steorts, 2015; Sadinle, 2014, 2017) additionally utilize attribute-level similarity scores.

Apart from these advances in Bayesian approaches to matching (largely undertaken in statistics), there have been an abundance of contributions from the database and machine learning communities (see surveys by Getoor and Machanavajjhala, 2012; Christen, 2012). Their focus has typically been on rule-based approaches (Fan et al., 2009; Singh et al., 2017), supervised learning approaches (Mudgal et al., 2018), hybrid human-machine approaches (Wang et al., 2012; Gokhale et al., 2014), and scalability (Papadakis et al., 2016). Broadly speaking, all of these approaches rely on either humans in-the-loop or large amounts of labelled training data, which is not generally the case in the Bayesian setting.

Inference for Bayesian ER models. Most prior work on Bayesian generative models for ER (e.g. Tancredi and Liseo, 2011; Gutman et al., 2013; Steorts, 2015) has relied on Gibbs sampling for inference. Compared to other Markov chain Monte Carlo (MCMC) algorithms, Gibbs sampling is relatively easy to implement, however it may suffer from slow convergence and poor mixing owing to its highly local moves (Liu, 2004).

In the broader context of clustering models, the *split-merge algorithm* (Jain and Neal, 2004) has been proposed as an alternative to Gibbs sampling. It traverses the space of clusterings through proposals that split single clusters or merge pairs of clusters, and is thus less susceptible to becoming trapped in local modes. Steorts et al. (2016) applied this algorithm to an ER model similar to `blink`. However, it remains unclear whether the split-merge algorithm is efficient for ER problems, where the number of clusters is expected to grow almost linearly in the number of records (Zanella et al., 2016).

More recently, Zanella et al. (2016) proposed the *chaperones algorithm* for inference in microclustering models. It is similar in spirit to the split-merge algorithm, however it is based on restricted Gibbs sampling rather than the Metropolis-Hastings framework. It "focuses on cluster reassignments with higher probabilities" by maintaining a biased distribution over the space of record pairs. However, its scalability is likely to be limited as a result.

In this work, we apply partially-collapsed Gibbs sampling (van Dyk and Park, 2008), as we expect it will be more efficient for ER models than the aforementioned methods, while still yielding an improvement over regular Gibbs sampling.

Parallel/distributed MCMC. Recent literature has focused on using parallel and distributed computing to scale up MCMC algorithms, where applications have included Bayesian topic models (Newman et al., 2009; Smola and Narayanamurthy, 2010; Ahn et al., 2014) and mixture models (Williamson et al., 2013; Chang and Fisher, 2013; Lovell et al., 2013; Ge et al., 2015). We review the application to mixture models, as they are conceptually similar to ER models.

Existing work has concentrated on Dirichlet process (DP) mixture models and Hierarchical DP mixture models. The key to enabling distributed inference for these models is the realization that a DP mixture model can be reparameterized as a mixture of DPs. Put simply, the reparameterized model induces a *partitioning* of the clusters, such that clusters assigned to *distinct partitions* are conditionally independent. As a result, variables within partitions can be updated in parallel.

Williamson et al. (2013) exploited this idea at the thread level to parallelize inference for a DP mixture model. Chang and Fisher (2013) followed a similar approach, but included an additional level of parallelization within partitions using a parallelized version of the split-merge algorithm. Others (Lovell et al., 2013; Ge et al., 2015) have developed distributed implementations in the Map-Reduce framework.

We do not consider DP mixture models in our work, as their behaviour is ill-suited for ER applications.[1] However we do borrow the reparameterization idea, albeit with a more flexible partition specification which permits similar entities to be co-partitioned and facilitates load balancing. It would be interesting to see whether similar ideas can be applied to microclustering models (Zanella et al., 2016), however preserving the marginal posterior distribution seems challenging in this case.

3 A SCALABLE MODEL FOR BAYESIAN ER

We now present our extension to the `blink` model for Bayesian ER (Steorts, 2015), which incorporates auxiliary partitions, support for missing values, and generic attribute similarity functions. We describe notation and assumptions in Section 3.1, before outlining the generative process in Section 3.2. Attribute similarity measures are defined in Section 3.3, including a truncation approximation which improves scalability. In Section 3.4, we demonstrate that the marginal posterior of `d-blink` is equivalent to `blink` under certain conditions. Finally, in Section 3.5 we provide intuition for the auxiliary partition representation.

[1] With a DP prior, the number of clusters grows logarithmically in the number of records, but empirical observations call for near-linear growth (Zanella et al., 2016).

3.1 Notation and assumptions

Consider a collection of T tables[2] (databases) indexed by t, each with R_t records (rows) indexed by r and A aligned attributes (columns) indexed by a. We assume each record (t, r) links to a single entity, denoted by λ_{tr}, from a fixed population of entities of size E indexed by e. The value of the a-th attribute for record (t, r) is denoted by x_{tra}, and is assumed to be a noisy observation of the linked entity's true attribute value $y_{\lambda_{tr}a}$. We allow for the fact that some attributes x_{tra} may be missing completely at random through a corresponding indicator variable o_{tra} (Little and Rubin, 2002, p. 12).

Table 1 summarizes our notation, including model-specific variables which will be introduced shortly. We adopt the following rules to compactly refer to sets of variables:

- A boldface lower-case variable denotes the set of *all attributes*: e.g. $\boldsymbol{x}_{tr} = [x_{tra}]_{a=1\ldots A}$.
- A boldface capital variable denotes the set of *all index combinations*: e.g. $\boldsymbol{X} = [x_{tra}]_{t=1\ldots T;r=1\ldots R_t;a=1\ldots A}$.

We also define notation to separate the record attributes $\mathbf{X}$ into an observed part $\mathbf{X}^{(o)}$ (those x_{tra}'s for which $o_{tra} = 1$) and a missing part $\mathbf{X}^{(m)}$ (those x_{tra}'s for which $o_{tra} = 0$).

After specifying a model, our goal is to perform end-to-end ER, by inferring the *joint* posterior distribution over the links $\boldsymbol{\Lambda} = [\lambda_{tr}]_{t=1\ldots T;r=1\ldots R_t}$ and true entity attribute values $\boldsymbol{Y} = [y_{ea}]_{e=1\ldots E;a=1\ldots A}$. We condition only on the observed record attribute values $\mathbf{X}^{(o)}$. In other words, we operate in a fully unsupervised setting, assuming no ground truth data is available for the links or entities.

Note that inferring $\boldsymbol{\Lambda}$ is equivalent to the *matching* stage of the ER pipeline, while inferring $\boldsymbol{Y}$ is equivalent to the *merging* stage. By inferring $\boldsymbol{\Lambda}$ and $\boldsymbol{Y}$ *jointly*, we are able to propagate uncertainty between the two stages. We'll later show how to incorporate probabilistic blocking, while allowing full propagation of uncertainty.

3.2 Model specification

We now describe the generative process for our proposed model `d-blink`. A plate diagram is depicted in Figure 1, with key differences from `blink` highlighted in a dashed blue line style.

Entities. Each entity in the population (of fixed size E) is described by A "true" attributes. The value of attribute a for entity e is denoted by y_{ea} and is assumed to be drawn independently from the domain of the attribute $\mathcal{V}_a$ according to the empirical distribution (derived from the tables) $\phi_a(\cdot)$:

$$y_{ea} \overset{\text{ind.}}{\sim} \text{Discrete}_{v \in \mathcal{V}_a}[\phi_a(v)]. \tag{1}$$

[2]We define a *table* as an ordered (indexed) collection of records, which may contain duplicates (records for which all attributes are identical).

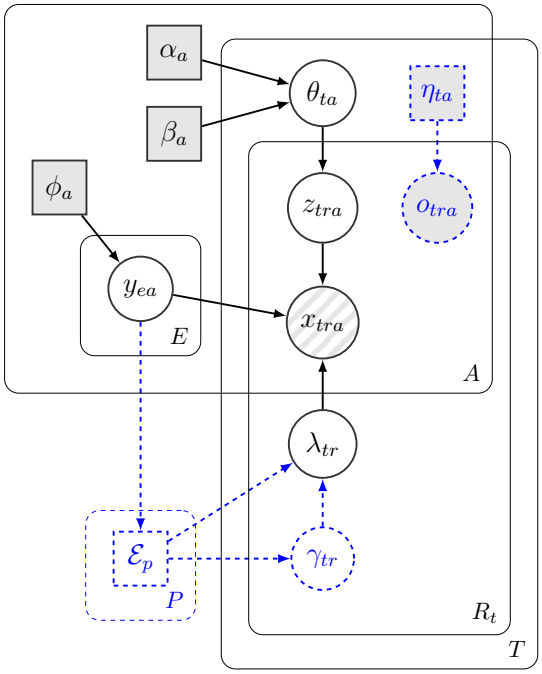

Figure 1: Plate diagram for `d-blink`. Extensions to `blink` are highlighted in a dashed blue line style. Circular nodes represent random variables; square nodes represent deterministic variables; (un)shaded nodes represent (un)observed variables; arrows represent conditional dependence; and plates represent replication over an index.

Table 1: Summary of notation.

Symbol	Description	Symbol	Description
$t \in 1 \dots T$	index over tables	γ_{tr}	assigned partition for record r in table t
$r \in 1 \dots R_t$	index over records in table t	λ_{tr}	assigned entity for record r in table t
$e \in 1 \dots E$	index over entities	θ_{ta}	prob. attribute a in table t is distorted
$p \in 1 \dots P$	index over partitions	α_a, β_a	distortion hyperparams. for attribute a
$a \in 1 \dots A$	index over attributes	η_{ta}	prob. attribute a in table t is observed
$v \in 1 \dots \lvert\mathcal{V}_a\rvert$	index over domain of attribute a	$\mathcal{V}_a$	domain of attribute a
$R = \sum_t R_t$	total number of records	$\phi_a(\cdot)$	distribution over domain of attribute a
x_{tra}	attribute a for record r in table t	$\texttt{sim}_a(\cdot,\cdot)$	similarity measure for attribute a
z_{tra}	distortion indicator for x_{tra}	$\mathcal{R}_e$	set of records assigned to entity e
o_{tra}	observed indicator for x_{tra}	$\mathcal{E}_p$	set of entities assigned to partition p
y_{ea}	attribute a for entity e	$\texttt{PartFn}(\cdot)$	partition assignment function

Partitions. `d-blink` deviates from `blink` by splitting the entities into P partitions[3] according to their attribute values. The assignment of entities to partitions is achieved through a deterministic *partition function*:

$$\texttt{PartFn} : \bigotimes_{a} \mathcal{V}_a \to \{1, \ldots, P\}, \tag{2}$$

where $\otimes$ is the Kronecker product. Further details are given in Section 4, along with an example based on k-d trees. We also introduce the notation $\mathcal{E}_p(\boldsymbol{Y}) = \{e : \texttt{PartFn}(\boldsymbol{y}_e) = p\}$, which allows us to concisely refer to the set of entities assigned to partition p.

Distortion. Associated with each table t and attribute a is a distortion probability θ_{ta}, with assumed prior distribution:

$$\theta_{ta} | \alpha_a, \beta_a \overset{\text{ind.}}{\sim} \text{Beta}[\alpha_a, \beta_a], \tag{3}$$

where α_a and β_a are hyperparameters. We provide recommendations for setting α_a and β_a in Appendix F. The distortion probabilities feed into the record-generation process below.

Records. We assume the records are generated by selecting an entity uniformly at random and copying the entity's attributes subject to distortion. The process for generating record r in table t is outlined below. Steps (i), (ii), and (v) deviate from `blink`.

(i) Choose a partition assignment γ_{tr} at random in proportion to the partition sizes:

$$\gamma_{tr} | \boldsymbol{Y} \overset{\text{ind.}}{\sim} \text{Discrete}_{p \in \{1 \ldots P\}}[|\mathcal{E}_p|/E]. \tag{4}$$

(ii) Choose an entity assignment λ_{tr} uniformly at random from partition γ_{tr}:

$$\lambda_{tr} | \gamma_{tr}, \boldsymbol{Y} \overset{\text{ind.}}{\sim} \text{DiscreteUniform}[\mathcal{E}_{\gamma_{tr}}]. \tag{5}$$

(iii) For each attribute a, draw a distortion indicator z_{tra}:

$$z_{tra} | \theta_{ta} \overset{\text{ind.}}{\sim} \text{Bernoulli}[\theta_{ta}]. \tag{6}$$

(iv) For each attribute a, draw a record value x_{tra}:

$$x_{tra} | z_{tra}, y_{\lambda_{tr} a} \overset{\text{ind.}}{\sim} (1 - z_{tra}) \delta(y_{\lambda_{tr} a}) + z_{tra} \, \text{Discrete}_{v \in \mathcal{V}_a}[\psi_a(v | y_{\lambda_{tr} a})] \tag{7}$$

where $\delta(\cdot)$ represents a point mass. If $z_{tra} = 0$, x_{tra} is copied directly from the entity. Otherwise, x_{tra} is drawn from the domain $\mathcal{V}_a$ according to the distortion distribution ψ_a. In the literature, this is known as a hit-miss model (Copas and Hilton, 1990).

(v) For each attribute a, draw an observed indicator o_{tra}:

$$o_{tra} \overset{\text{ind.}}{\sim} \text{Bernoulli}[\eta_{ta}]. \tag{8}$$

If $o_{tra} = 1$, x_{tra} is observed, otherwise it is missing.

[3] In the database and distributed computing communities, *partition* may refer to a part of divided object. We use the term in this sense, which differs from the usage in set theory.

Detail on the distortion distribution. $\psi_a(\cdot|w)$ chooses a distorted value for attribute a conditional on the true value w. In our parameterization of the model, it is defined as

$$\psi_a(v|w) = h_a(w)\phi_a(v)\mathrm{e}^{\mathtt{sim}_a(v,w)}, \tag{9}$$

where $h_a(w) = 1/\sum_{v\in\mathcal{V}_a} \phi_a(v)\mathrm{e}^{\mathtt{sim}_a(v,w)}$ is a normalization constant and $\mathtt{sim}_a$ is the similarity measure for attribute a (see Section 3.3). Intuitively, this distribution chooses values in proportion to their empirical frequency, while placing more weight on those that are "similar" to w. This reflects the notion that distorted values are likely to be close to the truth, as is the case when modeling typographical errors.

3.3 Attribute similarity measures

We now discuss the attribute similarity measures that appear in the distortion distribution of Equation (9). The purpose of these measures is to quantify the propensity that some value v in the attribute domain is chosen as a distorted alternative to the true value w.

Definition (Attribute similarity measure)**.** *Let $\mathcal{V}$ be the domain of an attribute. An* attribute similarity measure *on $\mathcal{V}$ is a function $\mathtt{sim} : \mathcal{V}\times\mathcal{V} \to [0, s_{\max}]$ that satisfies $0 \le s_{\max} < \infty$ and $\mathtt{sim}(v,w) = \mathtt{sim}(w,v)$ for all $v, w \in \mathcal{V}$.*

Note that our parameterization in terms of attribute *similarity* measures differs from `blink`, which uses *distance* measures. This allows us to make use of a more efficient sampling method, as described in Section 6.3. The next proposition states that the two parameterizations are in fact equivalent, so long as the distance measure is bounded and symmetric (a proof is provided in Appendix B.1).

Proposition 1. *Let $\mathtt{dist}_a : \mathcal{V}\times\mathcal{V} \to [0, d_{\max;a}]$ be the attribute distance measure that appears in* `blink`*, and assume that $0 \le d_{\max;a} < \infty$ and $\mathtt{dist}_a(v,w) = \mathtt{dist}_a(w,v)$ for all $v, w \in \mathcal{V}$. Define the corresponding attribute similarity measure for* `d-blink` *as*

$$\mathtt{sim}_a(v,w) := d_{\max;a} - \mathtt{dist}_a(v,w). \tag{10}$$

Then the parameterization of ψ_a used in `d-blink` *is equivalent to* `blink`*.*

In this paper, we restrict our attention to the following similarity measures for simplicity:

- *Constant similarity measure.* This measure is appropriate for categorical attributes, where there is no reason to believe one value is more likely than any other as a distortion to the true value w. Without loss of generality, it may be defined as $\mathtt{sim}_{\mathrm{const}}(v,w) = s_{\max}$ for all $v, w \in \mathcal{V}$.

- *Normalized edit similarity measure.* This measure is based on the edit distance metric, and is suitable for modeling distortion in generic string attributes. Following Yujian and Bo (2007), we define a normalized edit distance metric,

$$\mathtt{dist}_{\mathrm{nEd}}(v,w) = \frac{2\ \mathtt{dist}_{\mathrm{Ed}}(v,w)}{|v| + |w| + \mathtt{dist}_{\mathrm{Ed}}(v,w)},$$

where $\texttt{dist}_{\text{Ed}}$ denotes the regular edit distance and $|v|$ denotes the length of string v. Note that alternative definitions of the normalized edit distance could be used (see references in Yujian and Bo, 2007), however the above definition is unique in that it yields a proper metric. Since the normalized edit distance is bounded on the interval $[0, 1]$ we can define a corresponding normalized edit similarity measure:

$$\texttt{sim}_{\text{nEd}}(v, w) = 1 - \texttt{dist}_{\text{nEd}}(v, w). \tag{11}$$

Ideally, one should select attribute similarity measures based on the data at hand. There are many possibilities to consider, such as Jaccard similarity, numeric similarity measures (Lesot et al., 2008) and other domain-specific measures (Bilenko and Mooney, 2003).

3.4 Posterior distribution and equivalence

Having described the `d-blink` model, we now consider the joint posterior distribution over the latent (unobserved) parameters. By reading the conditional dependence structure from the plate diagram (Figure 1) and marginalizing over the missing record attributes $\boldsymbol{X}^{(m)}$, one can show that the posterior distribution is of the following form:

$$\begin{aligned} p(\boldsymbol{\Gamma}, \boldsymbol{\Lambda}, \boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{\Theta} | \boldsymbol{X}^{(o)}, \boldsymbol{O}) \propto \prod_{e,a} p(y_{ea}|\phi_a) \times \prod_{t,a} p(\theta_{ta}|\alpha_a, \beta_a) \times \prod_{\substack{t,r,a \\ o_{tra}=1}} p(x_{tra}|z_{tra}, \lambda_{tr}, y_{\lambda_{tr}a}) \\ \times \prod_{t,r} \Big\{ p(\gamma_{tr}|\boldsymbol{Y}) p(\lambda_{tr}|\gamma_{tr}, \boldsymbol{Y}) \prod_{a} p(z_{tra}|\theta_{ta}) \Big\}. \end{aligned} \tag{12}$$

For further detail on the derivation and an expanded form of the posterior, we refer the reader to Appendix A.

The structure of the posterior for `d-blink` is very similar to the one for `blink`, apart from the factors for γ_{tr} and λ_{tr} and the introduction of the "observed" indicator variables o_{tra} (as highlighted in Figure 1). In fact, one can show that `d-blink` is equivalent to `blink` in the following sense:

Proposition 2. *Suppose the conditions of Proposition 1 hold and that $\alpha_a = \alpha$ and $\beta_a = \beta$ for all a. Assume furthermore that all record attributes are observed, i.e. $o_{t,r,a} = 1$ for all t, r, a. Then the marginal posterior of $\boldsymbol{\Lambda}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$ and $\boldsymbol{\Theta}$ for* `d-blink` *(i.e. marginalized over $\boldsymbol{\Gamma} = [\gamma_{tr}]_{t=1\ldots T; r=1\ldots R_t}$) is identical to the posterior for* `blink`.

A proof is provided in Appendix B.2.

3.5 Rationale for introducing partitions

We now briefly explain the role of the auxiliary partitions in `d-blink`. Firstly, we note that without the partitions ($P = 1$), the Markov blanket for λ_{tr} includes the attribute values for *all* of the entities $\boldsymbol{Y}$. This presents a major obstacle when it comes to distributing the inference on a compute cluster, as the data is not separable. By incorporating partitions,

we restrict the Markov blanket for λ_{tr} to include only a subset of the entity attribute values—those in the same partition as record (t, r). As a result, it becomes natural to distribute the inference so that each compute node is responsible for a single partition (see Section 5.2 for details). Secondly, we can interpret the partitions as performing a probabilistic blocking in the context of MCMC sampling (introduced in Section 5), which improves computational efficiency. In a given iteration, the possible links for a record are restricted to the entities residing in the same partition. However, unlike conventional blocking, the partition assignments are not fixed—between iterations the entities and linked records may move between partitions.

4 PARTITION FUNCTIONS

In Section 3.2 we defined a generic partition function (Equation 2) that is responsible for assigning entities to partitions. This function may be regarded as a free parameter, since it has no bearing on the model equivalence by Theorem 2. However, from a practical perspective it ought to be chosen carefully, as it can impact the efficiency of the inference. We suggest some guidelines for choosing a partition function in Section 4.1, before presenting an example based on k-d trees in Section 4.2.

4.1 Interpretation and guidelines

Recall that the partition function assigns an entity to a partition p according to its attributes $\boldsymbol{y}_e = [y_{ea}]_{a=1\ldots A}$. Since $\boldsymbol{y}_e$ is *unobserved*, it must be treated as a random variable over the space of possible attributes $\mathcal{V}_\otimes := \bigotimes_a \mathcal{V}_a$. This means the partition function should not be interpreted as partitioning the entities directly. Rather, it should be interpreted as partitioning the space $\mathcal{V}_\otimes$ in which the entities reside, while taking the distribution over $\mathcal{V}_\otimes$ into account. With this interpretation in mind, we argue that the partition function should ideally satisfy the following properties:

(i) *Balanced weight.* The partitions should have equal weight (probability mass) under the distribution over $\mathcal{V}_\otimes$. This ensures the expected number of entities assigned to each partition is equal, which results in proper load balancing.

(ii) *Entity separation.* A pair of entities drawn at random from the same partition should have a high degree of similarity, while entities drawn from different partitions should have a low degree of similarity. This improves the likelihood that similar records will end up in the same partition, and allows them to more readily form likely entities.

These properties need not be satisfied strictly: the extent to which they are satisfied is merely expected to improve the efficiency of the inference. For example, satisfying the first property requires knowledge of the marginal posterior distribution over $\boldsymbol{y}_e$, which is infeasible to calculate. We note that there is likely to be tension between the two properties, so that a balance must be struck between them.

4.2 Example: k-d tree partition function

We now describe a partition function based on k-d trees, which is used in our experiments in Section 7.

Background. A k-d tree is a binary tree that recursively partitions a k-dimensional affine space (Bentley, 1975; Friedman et al., 1977). In the standard set-up, each node of the tree is associated with a data point that implicitly splits the input space into two half-spaces along a particular dimension. Owing to its ability to hierarchically group nearby points, it is commonly used to speed up nearest-neighbor search. This makes a k-d tree a good candidate for a partition function, since it can be balanced while grouping similar points.

Setup. Our setup differs from a standard k-d tree in several aspects. First, we consider a discrete space $\mathcal{V}_\otimes$ (not an affine space), where the "k dimensions" are the A attributes. Second, we do not store data points in the tree. We only require that the tree implicitly stores the boundaries of the partitions, so that it can assign an arbitrary $\boldsymbol{y} \in \mathcal{V}_\otimes$ to the correct partition (a leaf node). Finally, since we are working in a discrete space, the input space to a node is a countable set. The node must split the input set into two parts based on the values of one of the attributes.

Fitting the tree. Since it is infeasible to calculate the marginal posterior distribution over $\boldsymbol{y}_e$ exactly, we use the empirical distribution from the tables as an approximation. As a result, we treat the records (tables) as a sample from the distribution over $\boldsymbol{y}_e$, and fit the tree so that it remains balanced with respect to this sample. The depth of the tree d determines the number of partitions (2^d).

Achieving balanced splits. When fitting the tree, each node receives an input set of samples and a rule must be found that splits the set into two roughly equal (balanced) parts based on an attribute. We consider two types of splitting rules: the *ordered median* and the *reference set* (see Appendix C). We allow the practioner to specify an ordered list of attributes to be used for splitting. To ensure balanced splits, we recommend selecting attributes with a large domain. If possible, we recommend preferencing attributes which are known a priori to be reliable (low distortion), as this will reduce the shuffling of entities/ records between partitions. In principle, it is possible to automate the process of fitting a tree: one could grow several trees with randomly-selected splits and use the one that is most balanced. We examine balance empirically in Appendix H.

5 INFERENCE

We now turn to approximating the full joint posterior distribution over the unobserved variables $\boldsymbol{Z}$, $\boldsymbol{Y}$, $\boldsymbol{\Theta}$, $\boldsymbol{\Gamma}$ and $\boldsymbol{\Lambda}$, as given in Equation (12). Since it is infeasible to sample from this distribution directly, we use an MCMC algorithm called partially-collapsed Gibbs (PCG) sampling (van Dyk and Park, 2008). In addition, we show how to exploit the

conditional independence induced by the partitions to distribute the PCG sampling across multiple threads or machines.

5.1 Partially-collapsed Gibbs sampling

Following the original `blink` paper (Steorts, 2015), we initially experimented with regular Gibbs sampling. However, the resulting Markov chains exhibited slow convergence and poor mixing. This is a known shortcoming of Gibbs sampling which may be remedied by collapsing variables and/or updating correlated variables in groups (Liu, 2004). These ideas form the basis for a framework called *partially-collapsed Gibbs (PCG) sampling*—a generalization of Gibbs sampling that has "much better convergence properties" (van Dyk and Park, 2008).

Under the PCG framework, variables are updated in groups by sampling from their conditional distributions. These conditional distributions may be taken with respect to the joint posterior (like regular Gibbs), or with respect to *marginal distributions* of the joint posterior (unique to PCG). The latter case is called *trimming* and must be handled with care so as not to alter the stationary distribution of the Markov chain.

In applying PCG sampling to `d-blink`, we must decide how to apply the three tools: *marginalization* (equivalent to grouping), *permutation* (changing the order of the updates) and *trimming* (removing marginalized variables). In theory, the convergence rate should improve with more marginalization and trimming, however this must be balanced with the following: (i) whether the resulting conditionals can be sampled from efficiently, and (ii) whether the resulting dependence structure is compatible with our distributed set-up (see Section 5.2). We consider two samplers, PCG-I and PCG-II, described below. Of the two, we recommend PCG-I as it is more efficient in our empirical evaluations (see Section 7.1). We include the PCG-II sampler, as one would expect the PCG-II sampler to perform better than the PCG-I sampler in terms of mixing, however when computational efficiency is taken into account the performance is worse (see Figure 6).

5.1.1 PCG-I sampler

The PCG-I sampler uses regular Gibbs updates for θ_{ta}, λ_{tr} and z_{tra} for all t, r and a. The conditional distributions for these updates are listed in Appendix D. When updating the entity attributes y_{ea} and the record partition assignments γ_{tr}, marginalization and trimming are used. Specifically, we apply marginalization by jointly updating $\boldsymbol{y}_e$ and $\{\gamma_{tr}, \boldsymbol{z}_{tr}\}_{\mathcal{R}_e}$ (the set of γ_{tr}'s and $\boldsymbol{z}_{tr}$'s for records (t, r) linked to entity e). We then trim (analytically integrate over) $\{\boldsymbol{z}_{tr}\}_{\mathcal{R}_e}$.

We shall now derive this update. Referring to Equation (12), the joint posterior of $\boldsymbol{y}_e$, $\{\gamma_{tr}, \boldsymbol{z}_{tr}\}_{\mathcal{R}_e}$ conditioned on the other parameters has the form

$$p\big(\boldsymbol{y}_e, \{\gamma_{tr}, \boldsymbol{z}_{tr}\}_{\mathcal{R}_e} \big| \boldsymbol{Z}^{\neg\mathcal{R}_e}, \boldsymbol{\Gamma}^{\neg\mathcal{R}_e}, \boldsymbol{\Theta}, \boldsymbol{\Lambda}, \boldsymbol{X}^{(o)}, \boldsymbol{O}\big) \propto$$
$$\prod_a \Big\{ p(y_{ea}|\phi_a) \times \prod_{(t,r)\in\mathcal{R}_e} p(\gamma_{tr}|\boldsymbol{Y}) p(\lambda_{tr}|\gamma_{tr}, \boldsymbol{Y}) p(z_{tra}|\theta_{ta}) \times \prod_{\substack{(t,r)\in\mathcal{R}_e \\ o_{tra}=1}} p(x_{tra}|z_{tra}, \lambda_{tr}, y_{ea}) \Big\},$$

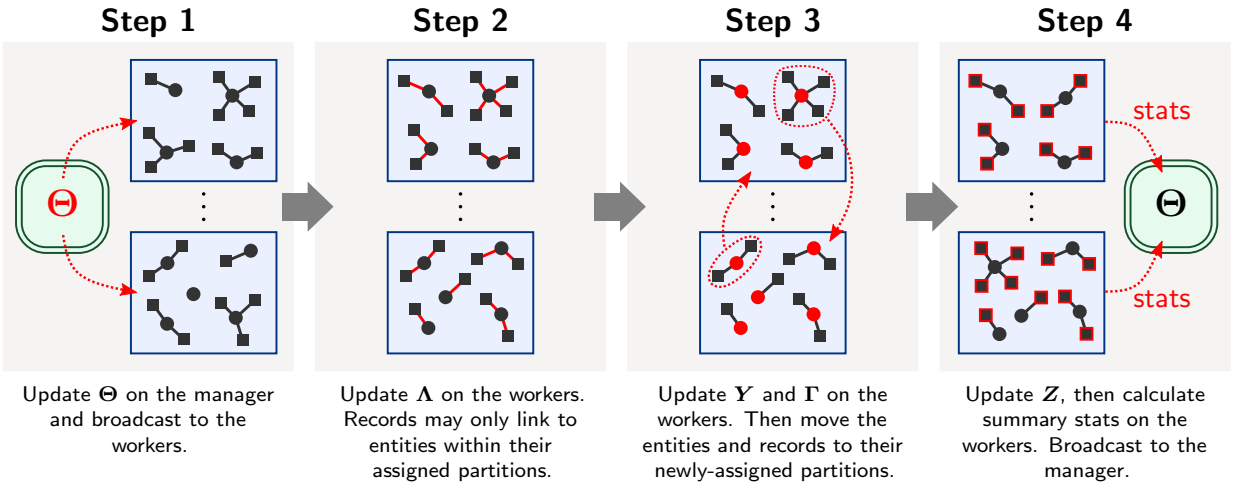

Figure 2: Schematic depicting a single iteration of distributed PCG sampling. The entity attributes ($\boldsymbol{Y}$—circular nodes), record attributes and their distortion indicators ($\boldsymbol{X}$, $\boldsymbol{Z}$—square nodes), and links from records to entities ($\boldsymbol{\Lambda}$—node connectors) are distributed across the workers (blue rectangular plates) according to their assigned partitions. The distortion probabilities ($\boldsymbol{\Theta}$) reside on the manager (green rounded-rectangular plate).

where superscript $\neg\mathcal{R}_e$ denotes exclusion of any records $(t, r) \in \mathcal{R}_e$ (those currently linked to entity e). Substituting the distributions and trimming $\{\boldsymbol{z}_{tr}\}_{\mathcal{R}_e}$ yields

$$p\big(\boldsymbol{y}_e, \{\gamma_{tr}\}_{\mathcal{R}_e} \big| \boldsymbol{Z}^{\neg\mathcal{R}_e}, \boldsymbol{\Gamma}^{\neg\mathcal{R}_e}, \boldsymbol{\Theta}, \boldsymbol{\Lambda}, \boldsymbol{X}^{(o)}, \boldsymbol{O}\big) = p(\{\gamma_{tr}\}_{\mathcal{R}_e} | \mathcal{R}_e, \boldsymbol{y}_e) \prod_a p(y_{ea} | \mathcal{R}_e, \boldsymbol{\Theta}, \boldsymbol{X}^{(o)}, \boldsymbol{O}) \tag{13}$$

where

$$p(y_{ea} | \mathcal{R}_e, \boldsymbol{\Theta}, \boldsymbol{X}^{(o)}, \boldsymbol{O}) \propto \phi_a(y_{ea}) \prod_{\substack{(t,r)\in\mathcal{R}_e \\ o_{tra}=1}} \{(1-\theta_{ta})\mathbb{I}[x_{tra} = y_{ea}] + \theta_{ta}\psi_a(x_{tra}|y_{ea})\}$$

and

$$p(\{\gamma_{tr}\}_{\mathcal{R}_e} | \mathcal{R}_e, \boldsymbol{y}_e) \propto \prod_{(t,r)\in\mathcal{R}_e} \mathbb{I}[\gamma_{tr} = \texttt{PartFn}(\boldsymbol{y}_e)]\,.$$

Note that the update for $\{\gamma_{tr}\}_{\mathcal{R}_e}$ is deterministic, conditional on $\boldsymbol{y}_e$ and $\mathcal{R}_e$.

Since we have applied trimming, we must permute the updates so that the trimmed variables $\boldsymbol{Z}$ are not conditioned on in later updates. This means the updates for $\boldsymbol{y}_e$ and $\{\gamma_{tr}, \boldsymbol{z}_{tr}\}_{\mathcal{R}_e}$ must come *after* the updates for θ_{ta} and λ_{tr}, but *before* the updates for z_{tra}.

5.1.2 PCG-II sampler

The PCG-II sampler is identical to PCG-I, except that it replaces the regular Gibbs update for λ_{tr} with an update that marginalizes and trims $\boldsymbol{z}_{tr}$. To derive the distribution for

Table 2: Dependencies for the conditional updates used in PCG-I.

Update variables	Dependencies
θ_{ta}	$z_{t\cdot a} = \sum_r z_{tra}$
λ_{tr}	$\boldsymbol{z}_{tr}$, $\boldsymbol{x}_{tr}$, γ_{tr}, $\mathcal{E}_{\gamma_{tr}}$, $\{\boldsymbol{y}_e\}_{e\in\mathcal{E}_{\gamma_{tr}}}$
y_{ea}, $\{\gamma_{tr}, z_{tra}\}_{(t,r)\in\mathcal{R}_e}$	$\mathcal{R}_e$, $\{x_{tra}\}_{(t,r)\in\mathcal{R}_e}$, $\{\theta_{ta}\}_{(t,r)\in\mathcal{R}_e}$
z_{tra}	x_{tra}, λ_{tr}, $y_{\lambda_{tr}a}$, θ_{ta}

this update, we first consider the joint posterior of λ_{tr} and $\boldsymbol{z}_{tr}$ conditioned on the other parameters:

$$p(\lambda_{tr}, \boldsymbol{z}_{tr}|\boldsymbol{\Gamma}, \boldsymbol{Y}, \boldsymbol{\Theta}, \boldsymbol{Z}^{\neg(t,r)}, \boldsymbol{X}^{(o)}, \boldsymbol{O}) \propto$$
$$p(\lambda_{tr}|\gamma_{tr}, \boldsymbol{Y}) \times \prod_a p(z_{tra}|\theta_{ta}) \times \prod_{\substack{a \\ o_{tra}=1}} p(x_{tra}|z_{tra}, \lambda_{tr}, y_{\lambda_{tr}a})$$

where superscript $\neg(t,r)$ denotes exclusion of record (t,r). Substituting the distributions and trimming $\boldsymbol{z}_{tr}$ yields

$$p(\lambda_{tr}|\boldsymbol{\Gamma}, \boldsymbol{Y}, \boldsymbol{\Theta}, \boldsymbol{Z}^{\neg(t,r)}, \boldsymbol{X}^{(o)}, \boldsymbol{O}) \propto$$
$$\mathbb{I}[\lambda_{tr} \in \mathcal{P}_{\gamma_{tr}}(\boldsymbol{Y})] \times \prod_{\substack{a \\ o_{tra}=1}} \Big\{(1-\theta_{ta})\mathbb{I}[x_{tra} = y_{\lambda_{tr}a}] + \theta_{ta}\psi_a(x_{tra}|y_{\lambda_{tr}a})\Big\}. \tag{14}$$

5.2 Distributing the sampling

By examining the conditional distributions derived in the previous section and those listed in Appendix D, one can show that the updates for the variables associated with entities and records (z_{tra}, λ_{tr}, γ_{tr} and y_{ea}) only depend on variables associated with entities and records assigned to the *same* partition (excluding $\boldsymbol{\Theta}$). These dependencies are summarized in Table 2 for the PCG-I sampler. The distortion probability θ_{ta} is an exception—it is not associated with any partition and may depend on z_{tra}'s across *all* partitions.

This dependence structure—in particular, the conditional independence of entities and records across partitions—makes the PCG sampling amenable to distributed computing. As such, we propose a manager-worker architecture where:

- the *manager* is responsible for storing and updating variables *not* associated with any partition (i.e. $\boldsymbol{\Theta}$); and
- each *worker* represents a partition, and is responsible for storing and updating variables associated with the entities and records assigned to it.

The manager/workers may be processes running on a single machine or on machines in a cluster. If using a cluster, we recommend that the nodes be tightly coupled, as frequent communication between them is required.

Figure 2 depicts a single iteration of PCG sampling using our proposed manager-worker architecture. Of the four steps depicted, steps 2 and 3 are the most computationally intensive, so we can potentially achieve a significant speed-up by distributing them across the workers. The communication cost is likely to be greatest in step 3, where the entities and linked records are shuffled to their newly-assigned partitions. A well-chosen partition function can minimize this cost. If similar records/entities are co-partitioned, there should be little movement between partitions. It is important to note that Figure 2 constitutes a *single* iteration of PCG sampling, or more precisely, a *single* application of the Markov transition operator. To generate a Markov chain of length T, we begin with some initial state and iteratively apply the transition operator T times.

6 COMPUTATIONAL EFFICIENCY CONSIDERATIONS

6.1 Efficient pruning of candidate links

In this section, we describe a trick that is aimed at improving the computational efficiency of the Gibbs update for λ_{tr} (used in the Gibbs and PCG-I samplers). This particular trick does *not* apply to the joint PCG update for λ_{tr} and $\boldsymbol{z}_{tr}$ (used in the PCG-II sampler).

Consider the conditional distribution for the λ_{tr} update in Equation (S5) of Appendix D:

$$p(\lambda_{tr} = e|\boldsymbol{\Gamma}, \boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{X}^{(o)}, \boldsymbol{O}) \propto \\ \mathbb{I}[e \in \mathcal{P}_{\gamma_{tr}}(\boldsymbol{Y})] \times \prod_{\substack{a \\ o_{tra}=1}} \Big\{(1 - z_{tra})\mathbb{I}[x_{tra} = y_{ea}] + z_{tra}\psi_a(x_{tra}|y_{ea})\Big\}. \tag{15}$$

The support of this distribution is the *set of candidate links* for record (t, r), which we denote by $\mathcal{L}_{tr}$. Looking at the first indicator function above, we see that $\mathcal{L}_{tr} \subseteq \mathcal{E}_{\gamma_{tr}}$, i.e. the candidate links are restricted to the entities in the *same partition* as record (t, r). Thus, a naïve sampling approach for this distribution takes $O(|\mathcal{E}_{\gamma_{tr}}|)$ time.

We can improve upon the naïve approach by exploiting the fact that $\mathcal{L}_{tr}$ is often considerably smaller than $\mathcal{E}_{\gamma_{tr}}$. To see why this is the case, note that the second indicator function in Equation (15) further restricts $\mathcal{L}_{tr}$ if any of the distortion indicators for the observed record attributes are zero. Specifically, if $z_{tra} = 0$ and $o_{tra} = 1$, $\mathcal{L}_{tr}$ cannot contain any entity whose a-th attribute y_{ea} does not match the record's a-th attribute x_{tra}. This implies $\mathcal{L}_{tr}$ is likely to be small in the case of low distortion.

Putting aside the computation of $\mathcal{L}_{tr}$ for the moment, this means we can reduce the time required to update λ_{tr} to $O(|\mathcal{L}_{tr}|)$. To compute $\mathcal{L}_{tr}$ efficiently, we propose maintaining an inverted index over the entity attributes within each partition. Specifically, the index for the a-th attribute in partition p should accept a query value $v \in \mathcal{V}_a$ and return the set of entities that match on v:

$$\mathcal{M}_{pa}(v) = \{n \in \mathcal{E}_p : y_{ea} = v\}. \tag{16}$$

Once the index is constructed, we can efficiently retrieve the set of candidate links for record

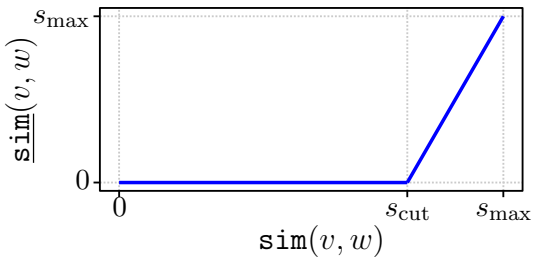

Figure 3: Transformation from a raw similarity function (`sim`) to a truncated similarity function (`sim`).

(t, r) by computing a multiple set intersection:

$$\mathcal{L}_{tr} = \bigcap_{\{a: z_{tra}=0 \wedge o_{tra}=1\}} \mathcal{M}_{\gamma_{tr}a}(x_{tra}). \tag{17}$$

This assumes at least one of the observed record attributes is not distorted. Otherwise $\mathcal{L}_{tr} = \mathcal{E}_{\gamma_{tr}}$.

Since the sizes of the sets $\mathcal{M}_{\gamma_{tr}a}(x_{tra})$ are likely to vary significantly, we advise computing the intersection iteratively in increasing order of size. That is, we begin with the smallest set and retain the elements that are also in the next largest set, and so on. With a hash-based set implementation, this scales linearly in the size of the first (smallest) set.

6.2 Caching and truncation of attribute similarities

We have not yet emphasized that the updates for $\boldsymbol{\Lambda}$, $\boldsymbol{Y}$ and $\boldsymbol{\Gamma}$ depend on the attribute similarities between pairs of values in the attribute domains. Specifically, for each attribute a, we need to access the indexed set $\mathcal{S}_a = \{\texttt{sim}_a(v, w) : v, w \in \mathcal{V}_a \times \mathcal{V}_a\}$. These similarities may be expensive to evaluate on-the-fly, so we cache the results in memory on the workers.

To manage the quadratic scaling of $\mathcal{S}_a$, and in anticipation of another trick introduced in Section 6.3, we transform the similarities so that those *below* a cut-off $s_{\text{cut};a}$ are regarded as completely disagreeing. We achieve this by applying the following truncation transformation to the raw attribute similarity $\texttt{sim}_a(v, w)$:

$$\underline{\texttt{sim}}_a(v, w) = \max\left(0,\ \frac{\texttt{sim}_a(v, w) - s_{\text{cut};a}}{1 - s_{\text{cut};a}/s_{\max;a}}\right). \tag{18}$$

as illustrated in Figure 3. Whenever a raw attribute similarity is called for, we replace it with this truncated version. Only pairs of values with positive truncated similarity are stored in the cache—those not stored in the cache have a truncated similarity of zero by default. Note that attributes with a constant similarity function $\texttt{sim}_{\text{const}}$ are treated specially—there is no need to cache the index set of similarities, since they are all identical.

It is important to acknowledge that the truncated similarities are an approximation to the original model. We claim that the approximation is reasonable on the following grounds:

- *Low loss of information.* Below a certain cut-off, the attribute similarity function is unlikely to encode much useful information for modeling the distortion process. For example, the fact that $\texttt{sim}_{\text{nEd}}(\text{“Smith”}, \text{“Chiu”}) = 0.385$ whereas $\texttt{sim}_{\text{nEd}}(\text{“Smith”}, \text{“Chen”}) = 0.286$, doesn't necessarily suggest that “Chiu” is more likely than “Chen” as a distorted alternative to “Smith”.

- *Precedent.* In the record linkage literature, value pairs with similarities below a cut-off are regarded as completely disagreeing (Winkler, 2002; Enamorado et al., 2017).

- *Efficiency gains.* As we shall soon see in Section 6.3, we can perform the combined $\boldsymbol{Y}$, $\boldsymbol{\Gamma}$, $\boldsymbol{Z}$ update more efficiently by eliminating pairs below the cut-off from consideration.

6.3 Fast updates of entity attributes using perturbation sampling

We now present a novel sampling algorithm that allows us to efficiently perform the PCG update for y_{ea} and $\{\gamma_{tr}, z_{tra}\}_{\mathcal{R}_e}$. The algorithm relies on the observation that the conditional distribution for y_{ea} can be expressed as a mixture over two components:

(i) a *base distribution* over $\mathcal{V}_a$ which is ideally constant for all entities; and

(ii) a *perturbation distribution* which varies for each entity, but has a much smaller support than $\mathcal{V}_a$.

With this representation, we can avoid computing and sampling from the full distribution over $\mathcal{V}_a$, which varies for each y_{ea} update. Rather, we only need to compute the perturbation distribution over a much smaller support, and then sample from the mixture, which can be done efficiently using the Vose-Alias method (Vose, 1991). We refer to this algorithm as *perturbation sampling*.

6.3.1 Perturbation sampling

Although we're interested in applying perturbation sampling to a specific conditional distribution, we describe the idea in generality below.

Consider a target probability mass function (pmf) $p(x|\omega)$ with finite support $\mathcal{X}$, which varies as a function of parameters $\omega \in \Omega$. In general, one must recompute the probability tables to draw a new variate whenever ω changes—a computation that takes $O(|\mathcal{X}|)$ time. However, if the dependence on ω is of a certain restricted form, we show that it is possible to achieve better scalability by expressing the target as a mixture. This is made precise in the following result.

Proposition 3. *Let $p(x|\omega)$ be a pmf with finite support $\mathcal{X}$, which depends on parameters $\omega \in \Omega$. Suppose there exists a “base” pmf $q(x)$ over $\mathcal{X}$ which is independent of ω and a non-negative bounded perturbation term $\epsilon(x|\omega)$, such that $p(x|\omega)$ can be factorized as $p(x|\omega) \propto q(x)(1 + \epsilon(x|\omega))$. Then $p(x|\omega)$ can be expressed as a mixture over the base pmf $q(x)$ and a “perturbation” pmf $v(x|\omega) := c\, q(x)\epsilon(x|\omega)$ over $\mathcal{X}^\star = \{x \in \mathcal{X} : \epsilon(x|\omega) > 0\}$ as follows:*

$$p(x|\omega) = \frac{c}{1+c} q(x) + \frac{1}{1+c} v(x|\omega) \tag{19}$$

where $c^{-1} := \sum_{x \in \mathcal{X}^\star} q(x)\epsilon(x|\omega)$.

Proof. The result is straightforward to verify by substitution. □

Algorithm S1 (in Appendix E) shows how to apply this result to draw random variates from a target pmf. Briefly, it consists of three steps: (i) the perturbation pmf v and its normalization constant c are computed; (ii) a biased coin is tossed to choose between the base pmf q and the perturbation pmf v; and (iii) a random variate is drawn from the selected pmf. If q is selected, a pre-initialized Alias sampler is used to draw the random variate (reused for all ω). Otherwise if v is selected, a new Alias sampler is instantiated. The result below states the time complexity of this algorithm (see Appendix E for a proof).

Proposition 4. *Algorithm S1 (in Appendix E) returns a random variate from the target pmf* $p(x|\omega)$ *for any* $\omega \in \Omega$ *in* $O(|\mathcal{X}^\star|)$ *time.*

This is a promising result, since the size of the perturbation support $|\mathcal{X}^\star|$ is typically of order 10 for our application, while the size of the full support $|\mathcal{X}|$ may be as large as 10^5. Hence, we expect a significant speed-up over the naïve approach.

6.3.2 Application of perturbation sampling

We now return to our original objective: performing the joint PCG update for y_{ea} and $\{\gamma_{tr}, z_{tra}\}_{\mathcal{R}_e}$. Referring to Equation (13), we can express the conditional distribution for y_{ea} (i.e. the target distribution) as

$$p(y_{ea} = v|\mathcal{R}_e, \boldsymbol{\Theta}, \boldsymbol{X}^{(o)}, \boldsymbol{O}) \propto q_a(v|\mathcal{R}_e, \mathbf{O}) \left(1 + \epsilon_a(v|\mathcal{R}_e, \boldsymbol{\Theta}, \boldsymbol{X}^{(o)}, \boldsymbol{O})\right). \tag{20}$$

The base distribution is given by

$$q_a(v|\mathcal{R}_e, \boldsymbol{O}) \propto \phi_a(v) \left(h_a(v)\right)^{n_a(\mathcal{R}_e, \boldsymbol{O})} \tag{21}$$

where $n_a(\mathcal{R}_e, \boldsymbol{O}) = |\{(t,r) \in \mathcal{R}_e : o_{tra} = 1\}|$ is the number of records linked to entity e with observed values for attribute a; and the perturbation term is given by

$$\epsilon_a(v|\mathcal{R}_e, \boldsymbol{\Theta}, \{x_{tra}\}_{\mathcal{R}_e}) = \prod_{\substack{(t,r)\in\mathcal{R}_e \\ o_{tra}=1}} \left\{ \mathrm{e}^{\underline{\mathtt{sim}}_a(x_{tra}, v)} + \frac{(\theta_{ta}^{-1} - 1)\, \mathbb{I}[x_{tra} = v]}{\phi_a(x_{tra}) h_a(x_{tra})} \right\} - 1. \tag{22}$$

The full support of the target pmf is $\mathcal{V}_a$, while the perturbation support is given by

$$\{x_{tra} : (t,r) \in \mathcal{R}_e \wedge o_{tra} = 1\} \cup \{v \in \mathcal{V}_a : \underline{\mathtt{sim}}_a(v, x_{tra}) > 0 \wedge o_{tra} = 1 \text{ for any } (t,r) \in \mathcal{R}_e\}.$$

In words, this set consists of the observed values for attribute a in the records linked to entity e, plus any sufficiently similar values from the attribute domain (for which the truncated similarity is non-zero). The size of the perturbation set will vary depending on the cut-off used for the truncation transformation—the higher the cut-off, the smaller the set. This implies that there is a trade-off between efficiency (small perturbation set) and accuracy (lower cut-off).

Table 3: Summary of data sets. Those marked with a '⋆' are synthetic.

Data set	# records (M)	# files (K)	# entities	# attributes (A)	
				categorical	string
⋆ `ABSEmployee`	600,000	3	400,000	4	0
`NCVR`	448,134	2	296,433	3	3
`NLTCS`	57,077	3	34,945	6	0
`SHIW0810`	39,743	2	28,584	8	0
⋆ `RLdata10000`	10,000	1	9,000	2	3

Remark. *The astute reader may have noticed that the base distribution q_a given in Equation* (21) *is not completely independent of the conditioned parameters, as is required by Proposition 3. In particular, q_a depends on $n_a(\mathcal{R}_e, \boldsymbol{O})$—roughly the size of entity e. Fortunately, we expect the range of regularly encountered entity sizes to be small, so we sacrifice some memory by instantiating multiple Alias samplers for each $n_a(\mathcal{R}_e, \boldsymbol{O})$ in some expected range. In the worst case, when a value is encountered outside the expected range and the base distribution is required (unlikely since the weight on the base component is typically small), we instantiate the base distribution on-the-fly (same asymptotic cost as the naïve approach).*

7 EMPIRICAL EVALUATION

We evaluated `d-blink` using two synthetic and three real data sets, as summarized in Table 3. All results presented here were obtained using our local server in pseudocluster mode, however some were replicated on a cluster in the Amazon public cloud (see Appendix G) to test the effect of higher communication costs. Further details about the data sets, hardware, implementation and parameter settings are provided in Appendix F.

7.1 Computational and sampling efficiency

Following Turek et al. (2016), we measured the efficiency of `d-blink` using the rate of effective samples produced per unit time (ESS rate), which balances sampling efficiency (related to mixing/autocorrelation) and computational efficiency. We used the `mcmcse` R package (Flegal et al., 2017) to compute the effective sample size (ESS), which implements a multivariate method proposed by Vats et al. (2015).

Since the number of variables in the model is unwieldy (there are at least $(E+R+T)A+R$ unobserved variables) we computed the ESS for the following summary statistics:

- the number of observed entities (scalar);
- the aggregate distortion for each attribute (vector); and
- the cluster size distribution (vector containing frequency of 0-clusters, 1-clusters, 2-clusters, etc.).

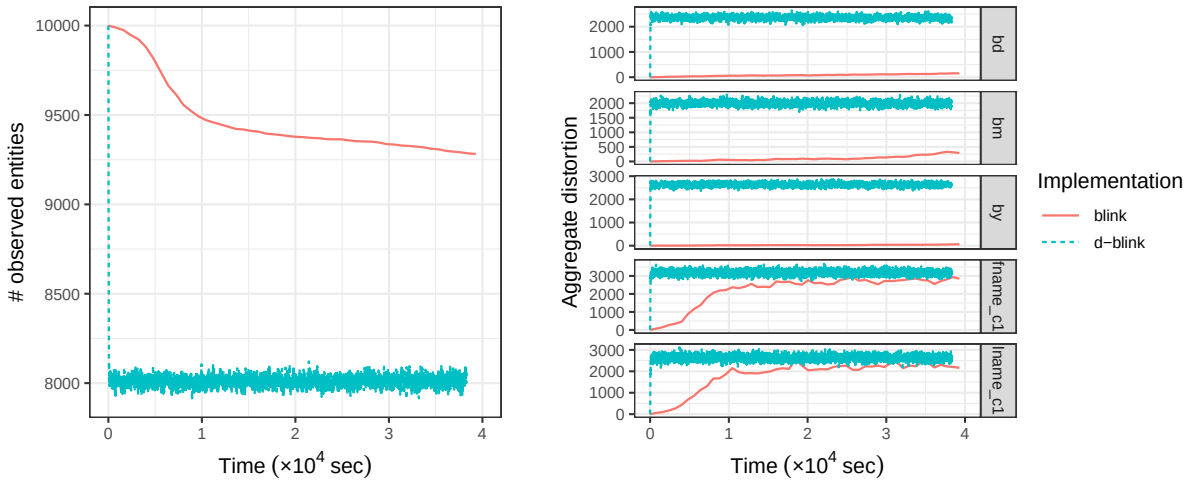

Figure 4: Comparison of convergence rates for `d-blink` and `blink`. The summary statistics for `d-blink` (number of observed entities on the left and attribute distortions on the right) rapidly converge to equilibrium, while those for `blink` fail to converge within 11 hours.

`d-blink` versus `blink`. We compared `d-blink` (using the PCG-I sampler) to our own implementation of `blink` (i.e. a Gibbs sampler without any of the tricks described in Section 6). For a fair comparison, we switched off partitioning in `d-blink`. We used the relatively small `RLdata10000` data set, as `blink` cannot cope with larger data sets. Figure 4 contains trace plots for two summary statistics as a function of running time. It is evident that `blink` has not converged to the equilibrium distribution within the allotted time of 11 hours, while `d-blink` converges to equilibrium in 100 seconds. Looking solely at the time per iteration, `d-blink` is at least 200× faster than `blink`.

Partitioning and efficiency. We tested the effect of varying the number of partitions P on the efficiency of `d-blink`. For each value of P, we computed the ESS rate averaged over 3000 iterations. We used the `NLTCS` data set and the PCG-I sampler. Figure 5 presents the results in terms of the speed-up relative to the ESS rate for $P = 1$. We observe a near-linear speed-up in P, with the exception of $P = 32$. The speed-up is expected to taper off with increasing numbers of partitions, as parallel gains in efficiency are overcome by losses due to communication costs and/or poorer mixing. This tipping point seems difficult to predict for a given set up, as it depends on complex factors such as the data distribution, the splitting rules used, and the hardware characteristics.

Sampling methods and efficiency. We evaluated the efficiency of the three samplers introduced in Section 5.1 (Gibbs, PCG-I and PCG-II). As above, we computed the ESS rate as an average over 3000 iterations. We set $P = 16$ and used the `NLTCS` data set. The results, shown in Figure 6, indicate that the PCG-I sampler is considerably more efficient (by a factor of 1.5–2×) than the baseline Gibbs sampler for this data set. We also observe that the PCG-II sampler performs quite poorly in comparison: between 20–30× slower than the Gibbs sampler. This is because the marginalization and trimming for the $\mathbf{\Lambda}$ update for

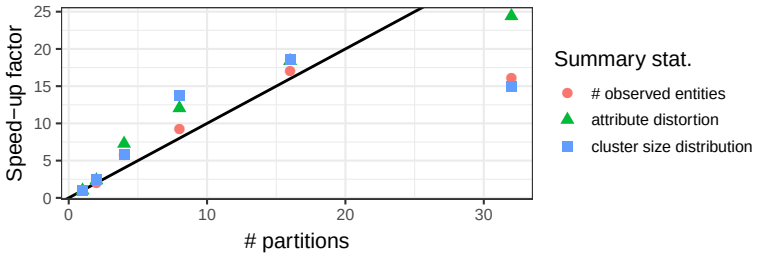

Figure 5: Efficiency of `d-blink` as a function of the number of partitions P and summary statistic of interest (larger is better). The speed-up measures the ESS rate relative to the ESS rate for $P = 1$ (no partitioning) for the `NLTCS` data set.

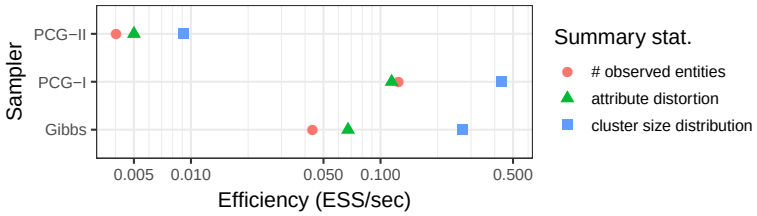

Figure 6: Efficiency of `d-blink` as a function of the sampler and summary statistic of interest (larger is better). All measurements are for the NLTCS data set with $P = 16$.

PCG-II prevents us from applying the trick described in Section 6.1. Thus although PCG-II is expected to be more efficient in terms of reducing autocorrelation, it is less efficient overall as each iteration is too computationally expensive.

7.2 Linkage quality

Though not our primary focus, we assessed the performance of `d-blink` in terms of its predictions for the linkage structure (the matching step) for the data sets in Table 3. This was not previously possible with `blink`, as it could only scale to small data sets of around 1000 records (Steorts, 2015).

Point estimate methodology. To evaluate the matching performance of `d-blink` with respect to the ground truth, we extracted a point estimate of the linkage structure from the posterior using the *shared most probable maximal matching sets (sMPMMS)* method (Steorts et al., 2016). This method circumvents the problem of label switching (Jasra et al., 2005)—where the identities of the entities do not remain constant along the Markov chain.

The sMPMMS method involves two main steps. In the first step, the most probable entity cluster is computed for each record based on the posterior samples. In general, these entity clusters will conflict with one another—e.g. the most probable entity cluster for r_1 might be (r_1, r_2) while for r_2 it is (r_1, r_2, r_3). The second step resolves these conflicts by assigning precedence to links between records and their most probable entity clusters. The result is a globally-consistent estimate of the linkage structure—i.e. it satisfies transitivity.

Table 4: Comparison of matching quality. "ARI" stands for adjusted Rand index and "Err. # clust." is the percentage error in the number of clusters.

Data set	Method	Pairwise measures			Cluster measures	
		Precision	Recall	F1-score	ARI	Err. # clust.
ABSEmployee	`d-blink`	0.9763	0.8530	**0.9105**	**0.9105**	**+1.667%**
	Fellegi-Sunter (10)	**0.9963**	0.8346	0.9083	—	—
	Fellegi-Sunter (100)	**0.9963**	0.8346	0.9083	—	—
	Near Matching	0.0378	**0.9930**	0.0728	—	—
	Exact Matching	0.9939	0.8346	0.9074	0.9074	+9.661%
NCVR	`d-blink`	0.9145	**0.9653**	**0.9392**	**0.9392**	**−3.562%**
	Fellegi-Sunter (10)	0.9868	0.7874	0.9083	—	—
	Fellegi-Sunter (100)	0.9868	0.7874	0.9083	—	—
	Near Matching	0.9899	0.7443	0.8497	—	—
	Exact Matching	**0.9925**	0.0017	0.0034	0.0034	+51.09%
NLTCS	`d-blink`	0.8319	0.9103	0.8693	0.8693	−22.09%
	Fellegi-Sunter (10)	**0.9094**	0.9087	**0.9090**	—	—
	Fellegi-Sunter (100)	**0.9094**	0.9087	**0.9090**	—	—
	Near Matching	0.0600	**0.9563**	0.1129	—	—
	Exact Matching	0.8995	0.9087	0.9040	**0.9040**	**+2.026%**
SHIW0810	`d-blink`	**0.2514**	0.5396	**0.3430**	**0.3429**	−37.65%
	Fellegi-Sunter (10)	0.0028	0.9050	0.0056	—	—
	Fellegi-Sunter (100)	0.0025	**0.9161**	0.0050	—	—
	Near Matching	0.0043	0.9111	0.0086	—	—
	Exact Matching	0.1263	0.7608	0.2166	0.2166	**−37.40%**
RLdata10000	`d-blink`	0.6334	**0.9970**	0.7747	**0.7747**	**−10.97%**
	Fellegi-Sunter (10)	0.9957	0.6174	0.7622	—	—
	Fellegi-Sunter (100)	0.9364	0.8734	0.9038	—	—
	Near Matching	0.9176	0.9690	**0.9426**	—	—
	Exact Matching	**1.0000**	0.0080	0.0159	0.0159	+11.02%

We distributed the computation of the sMPMMS method in the Spark framework. We used 9000 approximate posterior samples which were derived from a Markov chain of length 10^5 by discarding the first 10^4 iterations as burn-in[4] and applying a thinning interval of 10. These parameters were chosen by inspection of trace plots, some of which are reported in Appendix K. By contrast to the point estimates reported here, we also examined full posterior estimation in Appendix K.

Baseline methods. We compared the linkage quality of `d-blink` with three baseline methods as described below. We focused on scalable methods as we assumed very little to no training data was available.

- *Exact Matching.* Links records that match on all A attributes. It is unsupervised and ensures transitivity.

[4]We applied a burn-in of 60k iterations for `NCVR` as it was slow to converge.

- *Near Matching.* Links records that match on at least $L-1$ attributes. It is unsupervised, but does not guarantee transitivity.

- *Fellegi-Sunter.* Links records according to a pairwise match score that is a weighted sum of attribute-level dis/agreements. The weights are specified by the Fellegi-Sunter model (Fellegi and Sunter, 1969) and were estimated using the expectation-maximization algorithm, as implemented in the `RecordLinkage` R package (Sariyar and Borg, 2010). We chose the threshold on the match score to optimize the F1-score using a small amount of training data (size 10 and 100). This makes the method semi-supervised. Note that the training data was sampled in a biased manner to deal with the imbalance between the matches and non-matches (half with match scores above zero and half below). The method does not guarantee transitivity.

Results. Table 4 presents performance measures categorized by data set and method. The pairwise performance measures (precision, recall and F1-score) are provided for all methods, however the cluster performance measures (adjusted Rand Index, see Vinh et al., 2010, and percentage error in the number of clusters) are only valid for methods that guarantee transitivity of closure (`d-blink` and Exact Matching). Despite being fully unsupervised, `d-blink` achieves competitive performance when compared to the semi-supervised Fellegi-Sunter method. The two simple baselines, Near Matching and Exact Matching, are acceptable for data sets with low noise but perform poorly otherwise (e.g. `NCVR` and `RLdata10000`). We conducted an empirical sensitivity analysis for `d-blink` with respect to variations in the hyperparameters. The results for `RLdata10000` (included in Appendix J) show that `d-blink` is somewhat sensitive to all of the hyperparameters tested, however sensitivity is in general predictable, following clear and intuitive trends. One interesting observation is the fact that `d-blink` tends to overestimate the amount of distortion. This is perhaps not surprising given the absence of ground truth.

8 CONCLUDING REMARKS

We have proposed `d-blink`: a distributed, scalable extension of the `blink` model for unsupervised Bayesian ER. Our key insight was an auxiliary variable representation of the model that induces a partitioning over the entities and records. This integrates a kind of "blocking step" into the model, which improves scalability and enables distributed inference, while allowing blocking and ER errrors to propagate. We also contributed several ideas for improving the efficiency of the inference, including the use of partially-collapsed Gibbs sampling, inverted indices for updating links and a novel perturbation sampling algorithm for updating entity attributes. Our experiments show that `d-blink` can achieve significant efficiency gains compared to `blink`. With additional computing resources, we expect `d-blink` could be scaled to databases with 1–10 million records.

SUPPLEMENTAL MATERIALS

Code: An implementation of `d-blink` built on the Apache Spark distributed computing framework (Zip file). The source code is also hosted at `https://github.com/ngmarchant/dblink`.

Data: An archive containing data sets that we have permission to redistribute (Zip file).

ACKNOWLEDGEMENTS

N. Marchant acknowledges the support of an Australian Government Research Training Program Scholarship and the AMSIIntern program hosted by the Australian Bureau of Statistics. R. C. Steorts and A. Kaplan acknowledge the support of NSF SES-1534412 and CAREER-1652431. B. Rubinstein acknowledges the support of Australian Research Council grant DP150103710. N. Marchant and B. Rubinstein also acknowledge support of Australian Bureau of Statistics project ABS2018.363.

REFERENCES

Ahn, S., Shahbaba, B., and Welling, M. "Distributed Stochastic Gradient MCMC." In Xing, E. P. and Jebara, T. (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, 1044–1052. Bejing, China: PMLR (2014).

Belin, T. R. and Rubin, D. B. "A Method for Calibrating False-Match Rates in Record Linkage." *Journal of the American Statistical Association*, 90(430):694–707 (1995).

Bentley, J. L. "Multidimensional Binary Search Trees Used for Associative Searching." *Commun. ACM*, 18(9):509–517 (1975).

Bilenko, M. and Mooney, R. J. "Adaptive Duplicate Detection Using Learnable String Similarity Measures." In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, 39–48. New York, NY, USA: ACM (2003).

Chang, J. and Fisher, J. W., III. "Parallel Sampling of DP Mixture Models Using Sub-clusters Splits." In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, volume 1 of *NIPS'13*, 620–628. NY, USA: Curran Associates Inc. (2013).

Christen, P. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Berlin Heidelberg: Springer-Verlag (2012).

Copas, J. B. and Hilton, F. J. "Record Linkage: Statistical Models for Matching Computer Records." *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 153(3):287–320 (1990).

Dong, X. L. and Srivastava, D. "Big Data Integration." *Synthesis Lectures on Data Management*, 7(1):1–198 (2015).

Enamorado, T., Fifield, B., and Imai, K. "Using a probabilistic model to assist merging of large-scale administrative records." Technical report, Department of Politics, Princeton University (2017).

Fan, W., Jia, X., Li, J., and Ma, S. "Reasoning About Record Matching Rules." *Proc. VLDB Endow.*, 2(1):407–418 (2009).

Fellegi, I. P. and Sunter, A. B. "A Theory for Record Linkage." *Journal of the American Statistical Association*, 64(328):1183–1210 (1969).

Flegal, J. M., Hughes, J., Vats, D., and Dai, N. *mcmcse: Monte Carlo Standard Errors for MCMC*. Riverside, CA, Denver, CO, Coventry, UK, and Minneapolis, MN (2017). R package version 1.3-2.

Friedman, J. H., Bentley, J. L., and Finkel, R. A. "An Algorithm for Finding Best Matches in Logarithmic Expected Time." *ACM Trans. Math. Softw.*, 3(3):209–226 (1977).

Ge, H., Chen, Y., Wan, M., and Ghahramani, Z. "Distributed Inference for Dirichlet Process Mixture Models." In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 2276–2284. Lille, France: PMLR (2015).

Getoor, L. and Machanavajjhala, A. "Entity Resolution: Theory, Practice & Open Challenges." *Proc. VLDB Endow.*, 5(12):2018–2019 (2012).

Gokhale, C., Das, S., Doan, A., Naughton, J. F., Rampalli, N., Shavlik, J., and Zhu, X. "Corleone: Hands-off Crowdsourcing for Entity Matching." In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, 601–612. New York, NY, USA: ACM (2014).

Gutman, R., Afendulis, C. C., and Zaslavsky, A. M. "A Bayesian Procedure for File Linking to Analyze End-of-Life Medical Costs." *Journal of the American Statistical Association*, 108(501):34–47 (2013).

Jain, S. and Neal, R. M. "A Split-Merge Markov chain Monte Carlo Procedure for the Dirichlet Process Mixture Model." *Journal of Computational and Graphical Statistics*, 13(1):158–182 (2004).

Jasra, A., Holmes, C. C., and Stephens, D. A. "Markov Chain Monte Carlo Methods and the Label Switching Problem in Bayesian Mixture Modeling." *Statistical Science*, 20(1):50–67 (2005).

Larsen, M. D. and Rubin, D. B. "Iterative Automated Record Linkage Using Mixture Models." *Journal of the American Statistical Association*, 96(453):32–41 (2001).

Lesot, M.-J., Rifqi, M., and Benhadda, H. "Similarity measures for binary and numerical data: a survey." *International Journal of Knowledge Engineering and Soft Data Paradigms*, 1(1):63–84 (2008).

Little, R. J. A. and Rubin, D. B. *Statistical Analysis with Missing Data*. Wiley (2002).

Liu, J. S. *Monte Carlo Strategies in Scientific Computing*. Springer Series in Statistics. New York: Springer-Verlag (2004).

Lovell, D., Malmaud, J., Adams, R. P., and Mansinghka, V. K. "ClusterCluster: Parallel Markov Chain Monte Carlo for Dirichlet Process Mixtures." *arXiv:1304.2302 [cs, stat]* (2013). ArXiv: 1304.2302.

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. "Deep Learning for Entity Matching: A Design Space Exploration." In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, 19–34. New York, NY, USA: ACM (2018).

Newcombe, H. B., Kennedy, J. M., Axford, S. J., and James, A. P. "Automatic Linkage of Vital Records: Computers can be used to extract "follow-up" statistics of families from files of routine records." *Science*, 130(3381):954–959 (1959).

Newman, D., Asuncion, A., Smyth, P., and Welling, M. "Distributed algorithms for topic models." *Journal of Machine Learning Research*, 10(Aug):1801–1828 (2009).

Papadakis, G., Svirsky, J., Gal, A., and Palpanas, T. "Comparative Analysis of Approximate Blocking Techniques for Entity Resolution." *Proc. VLDB Endow.*, 9(9):684–695 (2016).

Sadinle, M. "Detecting duplicates in a homicide registry using a Bayesian partitioning approach." *The Annals of Applied Statistics*, 8(4):2404–2434 (2014).

—. "Bayesian Estimation of Bipartite Matchings for Record Linkage." *Journal of the American Statistical Association*, 112(518):600–612 (2017).

Sadinle, M. and Fienberg, S. E. "A Generalized Fellegi-Sunter Framework for Multiple Record Linkage With Application to Homicide Record Systems." *Journal of the American Statistical Association*, 108(502):385–397 (2013).

Sariyar, M. and Borg, A. "The RecordLinkage Package: Detecting Errors in Data." *The R Journal*, 2(2):61–67 (2010).

Singh, R., Meduri, V., Elmagarmid, A., Madden, S., Papotti, P., Quiané-Ruiz, J.-A., Solar-Lezama, A., and Tang, N. "Generating Concise Entity Matching Rules." In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, 1635–1638. New York, NY, USA: ACM (2017).

Smola, A. and Narayanamurthy, S. "An Architecture for Parallel Topic Models." *Proc. VLDB Endow.*, 3(1-2):703–710 (2010).

Steorts, R. C. "Entity Resolution with Empirically Motivated Priors." *Bayesian Analysis*, 10(4):849–875 (2015).

Steorts, R. C., Barnes, M., and Neiswanger, W. "Performance Bounds for Graphical Record Linkage." In Singh, A. and Zhu, J. (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, 298–306. Fort Lauderdale, FL, USA: PMLR (2017).

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

Steorts, R. C., Ventura, S. L., Sadinle, M., and Fienberg, S. E. "A Comparison of Blocking Methods for Record Linkage." In *Privacy in Statistical Databases*, Lecture Notes in Computer Science, 253–268. Springer, Cham (2014).

Tancredi, A. and Liseo, B. "A hierarchical Bayesian approach to record linkage and population size problems." *The Annals of Applied Statistics*, 5(2B):1553–1585 (2011).

Turek, D., Valpine, P. d., and Paciorek, C. J. "Efficient Markov chain Monte Carlo sampling for hierarchical hidden Markov models." *Environmental and Ecological Statistics*, 23(4):549–564 (2016).

van Dyk, D. A. and Park, T. "Partially Collapsed Gibbs Samplers." *Journal of the American Statistical Association*, 103(482):790–796 (2008).

Vats, D., Flegal, J. M., and Jones, G. L. "Multivariate Output Analysis for Markov chain Monte Carlo." *arXiv:1512.07713 [math, stat]* (2015). ArXiv: 1512.07713.
URL `http://arxiv.org/abs/1512.07713`

Vinh, N. X., Epps, J., and Bailey, J. "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance." *Journal of Machine Learning Research*, 11(Oct):2837–2854 (2010).

Vose, M. D. "A linear algorithm for generating random numbers with a given distribution." *IEEE Transactions on Software Engineering*, 17(9):972–975 (1991).

Wang, J., Kraska, T., Franklin, M. J., and Feng, J. "CrowdER: Crowdsourcing Entity Resolution." *Proc. VLDB Endow.*, 5(11):1483–1494 (2012).

Williamson, S., Dubey, A., and Xing, E. "Parallel Markov Chain Monte Carlo for Nonparametric Mixture Models." In Dasgupta, S. and McAllester, D. (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, 98–106. Atlanta, Georgia, USA: PMLR (2013).

Winkler, W. E. "Methods for Record Linkage and Bayesian Networks." Technical Report Statistics #2002-05, U.S. Bureau of the Census (2002).

—. "Overview of record linkage and current research directions." Technical Report Statistics #2006-2, U.S. Bureau of the Census (2006).

—. "Matching and record linkage." *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(5):313–325 (2014).

Yujian, L. and Bo, L. "A Normalized Levenshtein Distance Metric." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095 (2007).

Zanella, G., Betancourt, B., Wallach, H., Miller, J., Zaidi, A., and Steorts, R. C. "Flexible Models for Microclustering with Application to Entity Resolution." In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 1425–1433. NY, USA: Curran Associates Inc. (2016).

Zhao, B., Rubinstein, B. I. P., Gemmell, J., and Han, J. "A Bayesian Approach to Discovering Truth from Conflicting Sources for Data Integration." *Proc. VLDB Endow.*, 5(6):550–561 (2012).

Appendices for "d-blink: Distributed End-to-End Bayesian Entity Resolution"

Neil G. Marchant[a] Rebecca C. Steorts[b] Andee Kaplan[b]
Benjamin I. P. Rubinstein[a] Daniel N. Elazar[c]

[a]School of Computing and Information Systems, University of Melbourne
[b]Department of Statistical Science, Duke University
[c]Methodology Division, Australian Bureau of Statistics

September 16, 2019

A Derivation of the posterior distribution

Here we sketch the derivation of the joint posterior distribution over the unobserved variables conditioned on the observed record attributes $\boldsymbol{X}^{(o)}$, which is given in Equation (12) of the paper. First we read the factorization off the plate diagram in Figure 1, together with the conditional dependence assumptions detailed in Section 3.2 of the paper. We obtain the following expression, up to a normalisation constant:

$$\begin{aligned} p(\boldsymbol{\Gamma}, \boldsymbol{\Lambda}, \boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{\Theta}, \boldsymbol{X}^{(m)} | \boldsymbol{X}^{(o)}, \boldsymbol{O}) &\propto \prod_{e,a} p(y_{ea}|\phi_a) \times \prod_{t,a} p(\theta_{ta}|\alpha_a, \beta_a) \\ &\times \prod_{t,r} \Big\{ p(\gamma_{tr}|\boldsymbol{Y}) p(\lambda_{tr}|\gamma_{tr}, \boldsymbol{Y}) \prod_a p(z_{tra}|\theta_{ta}) \Big\} \times \prod_{\substack{t,r,a \\ o_{tra}=1}} p(x_{tra}|z_{tra}, \lambda_{tr}, y_{\lambda_{tr}a}) \\ &\times \prod_{\substack{t,r,a \\ o_{tra}=0}} p(x_{tra}|z_{tra}, \lambda_{tr}, y_{\lambda_{tr}a}). \end{aligned}$$

Ideally, we'd like to marginalize out all variables except $\boldsymbol{\Lambda}$ and $\boldsymbol{Y}$ (the variables of interest), however this is not tractable analytically. Fortunately, we can marginalize out the missing record attributes $\boldsymbol{X}^{(m)}$ which yields Equation (12) from the paper:

$$\begin{aligned} p(\boldsymbol{\Gamma}, \boldsymbol{\Lambda}, \boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{\Theta} | \boldsymbol{X}^{(o)}, \boldsymbol{O}) &\propto \prod_{e,a} p(y_{ea}|\phi_a) \times \prod_{t,a} p(\theta_{ta}|\alpha_a, \beta_a) \\ &\times \prod_{t,r} \Big\{ p(\gamma_{tr}|\boldsymbol{Y}) p(\lambda_{tr}|\gamma_{tr}, \boldsymbol{Y}) \prod_a p(z_{tra}|\theta_{ta}) \Big\} \times \prod_{\substack{t,r,a \\ o_{tra}=1}} p(x_{tra}|z_{tra}, \lambda_{tr}, y_{\lambda_{tr}a}). \end{aligned}$$

We can expand this further by substituting the conditional distributions given in Section 3.2 of the paper. This yields:

$$\begin{aligned} p(\boldsymbol{\Gamma}, \boldsymbol{\Lambda}, \boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{\Theta} | \boldsymbol{X}^{(o)}, \boldsymbol{O}) &\propto \prod_{e,a} \phi_a(y_{ea}) \times \prod_{t,a} \theta_{ta}^{\alpha_a - 1} (1 - \theta_{ta})^{\beta_a - 1} \\ &\times \prod_{t,r} \Big\{ \mathbb{I}[\lambda_{tr} \in \mathcal{E}_{\gamma_{tr}}(\boldsymbol{Y})] \prod_a \theta_{ta}^{z_{tra}} (1 - \theta_{ta})^{1 - z_{tra}} \Big\} \\ &\times \prod_{\substack{t,r,a \\ o_{tra}=1}} \Big\{ (1 - z_{tra}) \mathbb{I}[x_{tra} = y_{\lambda_{tr}a}] + z_{tra}\, \psi_a(x_{tra}|y_{\lambda_{tr}a}) \Big\}. \end{aligned} \tag{S1}$$

B Equivalence of `d-blink` and `blink`

In this section, we present proofs of Propositions 1 and 2, which show that the inferences we obtain from `d-blink` are equivalent to those we would obtain from `blink` under certain conditions.

B.1 Proof of Proposition 1: equivalence of distance/similarity representations

It is straightforward to show that `sim` as defined in Equation (10) of the paper satisfies the requirements of Definition 3.3. All that remains is to show that the two parameterizations of the distortion distribution ψ_a are equivalent. Beginning with ψ_a as parameterized in `blink`, we substitute Equation (10) and observe that

$$\psi_a(v|w) \propto \phi_a(v)\mathrm{e}^{-\mathtt{dist}_a(v,w)} = \phi_a(v)\mathrm{e}^{d_{\max;\mathrm{a}}+\mathtt{sim}_a(v,w)} \propto \phi_a(v)\mathrm{e}^{\mathtt{sim}_a(v,w)}.$$

This is identical to our parameterization in Equation (9). □

B.2 Proof of Proposition 2: equivalence of `d-blink` and `blink`

Given that

- Proposition 1 holds,
- the distortion hyperparameters are the same for all attributes, and
- all record attributes are observed,

the only factor in the posterior that differs from `blink` is:

$$\prod_{t,r} p(\lambda_{tr}|\gamma_{tr}, \boldsymbol{Y})p(\gamma_{tr}|\boldsymbol{Y}). \tag{S2}$$

Substituting the density for the conditional distributions for a single t, r factor yields:

$$p(\lambda_{tr}|\gamma_{tr}, \boldsymbol{Y})p(\gamma_{tr}|\boldsymbol{Y}) = \frac{\mathbb{I}[\lambda_{tr} \in \mathcal{E}_{\gamma_{tr}}(\boldsymbol{Y})]}{|\mathcal{E}_{\gamma_{tr}}(\boldsymbol{Y})|} \times \frac{|\mathcal{E}_{\gamma_{tr}}(\boldsymbol{Y})|}{E} = \frac{1}{E}\mathbb{I}[\lambda_{tr} \in \mathcal{E}_{\gamma_{tr}}(\boldsymbol{Y})]\,.$$

Putting this in Equation (S2) and marginalizing over $\boldsymbol{\Gamma}$ we obtain:

$$\prod_{t,r}\sum_{\gamma_{tr}=1}^{P} p(\lambda_{tr}|\gamma_{tr}, \boldsymbol{Y})p(\gamma_{tr}|\boldsymbol{Y}) = \prod_{t,r}\frac{1}{E}\sum_{\gamma_{tr}=1}^{P}\mathbb{I}[\lambda_{tr} \in \mathcal{E}_{\gamma_{tr}}(\boldsymbol{Y})] = \prod_{t,r}\frac{1}{E}\mathbb{I}[\lambda_{tr} \in \{1,\ldots,E\}]\,,$$

which is the factor that appears in the posterior for `blink`. □

C Splitting rules for the *k*-d tree partition function

In Section 4.2 of the paper we outline a partition function inspired by k-d trees. When inserting a node in the tree, we require a splitting rule that partitions the input set of values. In ordinary k-d trees, the median is often used for this purpose, however it is not appropriate for the discrete input sets that we encounter. As a result, we propose the following alternative splitting rules:

1. *Ordered median.* This rule is appropriate if the set of input attribute values is large and/or has a natural ordering. If there is no natural ordering, an artificial ordering must be applied (e.g. lexicographic ordering). The splitting rule is determined by sorting the input values and finding the median, accounting for the frequency of each value. Attribute values ordered before (after) the median are passed to the left (right) child node.

2. *Reference set.* This rule is appropriate if the set of input attribute values is small with no natural ordering. The splitting rule is determined by using a first-fit bin-packing algorithm to split the values into two roughly equal-sized bins, accounting for the frequency of each value. One of these bins is then labeled the "reference set". Attribute values (not) in the reference set are passed to the left (right) child node.

D Gibbs update distributions

Here we list the conditional distributions for the Gibbs updates. These are derived by referring to the posterior distribution in Equation (S1).

D.1 Update for θ_{ta}

$$\theta_{ta}|\boldsymbol{Z}, \boldsymbol{\Lambda}, \boldsymbol{\Gamma}, \boldsymbol{Y}, \boldsymbol{X}^{(o)}, \boldsymbol{O} \sim \text{Beta}[z_{t\cdot a} + \alpha_a, R_t - z_{t\cdot a} + \beta_a] \tag{S3}$$

where $z_{t\cdot a} := \sum_{r=1}^{R_t} z_{tra}$.

D.2 Update for z_{tra}

$$z_{tra}|\boldsymbol{\Lambda}, \boldsymbol{\Gamma}, \boldsymbol{Y}, \boldsymbol{\Theta}, \boldsymbol{X}^{(o)}, \boldsymbol{O} \sim (1 - o_{tra})\,\text{Bernoulli}[\theta_{ta}] + o_{tra}\,\text{Bernoulli}[\zeta_a(\theta_{ta}, x_{tra}, y_{\lambda_{tr}a})] \tag{S4}$$

where $\zeta_a(\theta, x, y) = \begin{cases} 1, & \text{if } x \neq y, \\ \frac{\theta\psi_a(x|y)}{\theta\psi_a(x|y) - \theta + 1}, & \text{otherwise.} \end{cases}$

D.3 Update for λ_{tr}

$$p(\lambda_{tr}|\boldsymbol{\Gamma}, \boldsymbol{Y}, \boldsymbol{\Theta}, \boldsymbol{Z}, \boldsymbol{X}^{(o)}, \boldsymbol{O}) \propto \mathbb{I}[\lambda_{tr} \in \mathcal{E}_{\gamma_{tr}}(\boldsymbol{Y})] \prod_{\substack{a \\ o_{tra}=1}} \Big\{(1 - z_{tra})\mathbb{I}[x_{tra} = y_{\lambda_{tr}a}] + z_{tra}\psi_a(x_{tra}|y_{\lambda_{tr}a})\Big\}. \tag{S5}$$

E Perturbation sampling algorithm

In Proposition 3 of the paper, we show how to express a target pmf p (from which we'd like to draw random variates) as a mixture over a base pmf q and a perturbation pmf v. Algorithm S1 demonstrates how to efficiently draw random variates from the target pmf using this mixture representation.

Algorithm S1 Perturbation sampling for $p(x|\omega)$

Input: map from $x, \omega \in \mathcal{X}^\star \times \Omega \to \epsilon(x|\omega)$; map from $x \in \mathcal{X} \to q(x)$; pre-initialized Alias sampler for q.

1: $v \leftarrow \emptyset$ ▷ empty map
2: **for** $x \in \mathcal{X}^\star$ **do**
3: $\quad v(x) \leftarrow q(x)\epsilon(x|\omega)$
4: **end for**
5: $c \leftarrow 1/\sum_{x \in \mathcal{X}^\star} v(x)$ ▷ normalization
6: $s \sim \text{Bernoulli}\left[\frac{c}{1+c}\right]$
7: **if** $s = 1$ **then**
8: $\quad$ **Return:** $x \sim q(\cdot)$ ▷ using input Alias sampler
9: **else**
10: $\quad v \leftarrow c * v$
11: $\quad$ **Return:** $x \sim v(\cdot)$ ▷ using new Alias sampler
12: **end if**

E.1 Proof of Proposition 4: complexity of perturbation sampling

Let us analyze the time complexity of Algorithm S1. Lines 2–6 are $O(|\mathcal{X}^\star|)$. By properties of the Alias sampler (Vose, 1991), line 8 is $O(1)$ and line 11 is $O(|\mathcal{X}^\star|)$. Thus the overall complexity is $O(|\mathcal{X}^\star|)$.

F Further details on the experimental set-up

F.1 Data sets

We provide a brief description of each data set below, and specify the attributes used for matching.

- `ABSEmployee`. A synthetic data set used internally for linkage experiments at the Australian Bureau of Statistics. It simulates an employment census and two supplementary surveys (it is not derived from any real data sources). We used four attributes: `MB`, `BDAY`, `BYEAR` and `SEX`.

- `NCVR`. Two snapshots from the North Carolina Voter Registration database taken two months apart (Christen, 2014). The snapshots are filtered to include only those voters whose details changed over the two-month period. We use the full name, age, gender and zip code attributes.

- `NLTCS`. A subset of the National Long-Term Care Survey (Manton, 2010) comprising the 1982, 1989 and 1994 waves. A subset must be used as race was subsampled in the other three years, making it unsuitable for ER. We used the `SEX`, `DOB`, `STATE` and `REGOFF` attributes.

- `SHIW0810`. A subset from the Bank of Italy's Survey on Household Income and Wealth (Banca d'Italia, n.d.) comprising the 2008 and 2010 waves. We used 8 attributes: `IREG`, `SESSO`, `ANASC`, `STUDIO`, `PAR`, `STACIV`, `PERC` and `CFDIC`. Further details are available at `http://github.com/ngmarchant/shiw`.

- `RLdata10000`. A synthetic data set provided with the `RecordLinkage` R package (Sariyar and Borg, 2010). We used all attributes except for `fname_c2` and `lname_c2`, which consist mainly of missing values.

F.2 Implementation and hardware

Our implementation of `d-blink` is written in Scala and depends on Apache Spark 2.3.1 (a distributed computing framework). Since `d-blink` requires control over the partitioning (entities and linked records *must* reside on their assigned partitions), we used the RDD API with a custom partitioner. Our custom-built server ran in local (pseudo-cluster) mode, with 2× 28-core Intel Xeon Platinum 8180M CPUs for a total of 112 threads (with HyperThreading); and 128GB of allocated RAM on the driver.

F.3 Parameter settings and initialization

We used the following parameter settings for all experiments.

- The distortion hyperparameters α_a, β_a were set to encode a prior mean distortion probability of approximately 1%, with the strength varying in proportion to the total number of records R:

$$\alpha_a = R \times 10\% \times 1\% \text{ and } \beta_a = R \times 10\% \text{ for all } a.$$

- The size of the latent entity population E was set to R. This corresponds to a prior mean number of observed entities of $(1 - \mathrm{e}^{-1})R \approx 0.63R$, as shown by Steorts et al. (2016). It is important not to set E too low, as it places an upper bound on the number of entities.

- For simplicity, we treated all attributes as either "categorical-type" with similarity function $\mathtt{sim}_{\text{const}}$ or "string-type" with similarity function $10.0 \times \mathtt{sim}_{\text{nEd}}$ (these are defined in Section 3.3).

- The similarity cut-off for string-type attributes was set to 7.0, following advice in the `RecordLinkage` R package (Sariyar and Borg, 2010).

- We used the k-d tree partition function as defined in Section 4.2. The *reference set* splitting rule was used for input sets with 30 or fewer elements—the *ordered median* splitting rule was used otherwise.

To initialize the Markov chain, we linked each record to a unique entity and copied the record attributes into the entity attributes, assuming no distortion. Any entity attributes

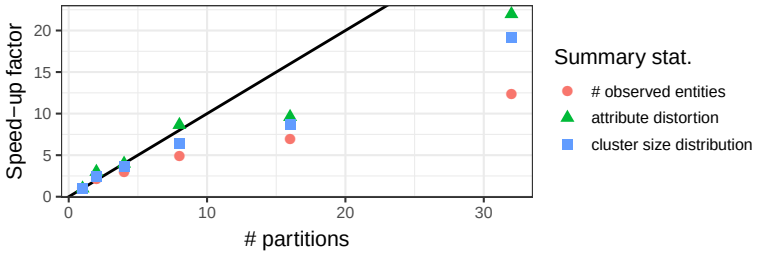

Figure S1: Efficiency of `d-blink` as a function of the number of partitions P and summary statistic of interest (larger is better). The speed-up measures the ESS rate relative to the ESS rate for $P = 1$ (no partitioning) for the `NLTCS` data set.

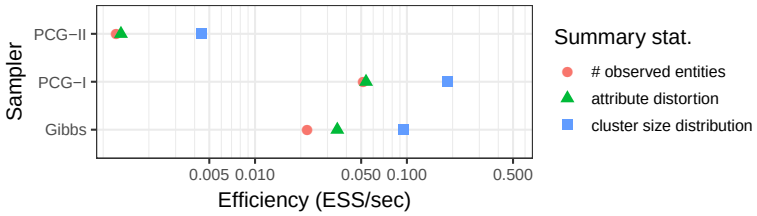

Figure S2: Efficiency of `d-blink` as a function of the sampler and summary statistic of interest (larger is better). All measurements are for the NLTCS data set with $P = 16$.

that were missing after this process (due to missing record attributes) were filled by drawing an attribute value from the empirical distribution. We set the thinning interval to 10—i.e. we only saved every tenth step along the chain. This increases the effective sample size for a given storage budget.

G Results on Amazon EC2

We repeated two of the experiments described in Section 7.1 of the main paper on a cluster running in the Amazon Elastic Compute Cloud (EC2). For the worker (executor) nodes, we used varying numbers of `m5.xlarge` instances with 4 vCores, 16 GiB memory and 32 GiB of Elastic Block Store (EBS) storage. Due to the increased latency and decreased bandwidth between the compute nodes, we expected the efficiency to decrease. This is indeed what we observed.

Figure S1 plots the speed-up as a function of the number of partitions P relative to a baseline with no partitioning. We observe poorer scaling with P compared to the results we obtained on our local server (c.f. Figure 5 in the main paper). Figure S2 plots the efficiency as a function of the sampling method with $P = 16$. The results are qualitatively similar to the ones we obtained using our local server (c.f. Figure 6 in the main paper). However, the ESS rate was reduced for all samplers as expected due to increased communication costs.

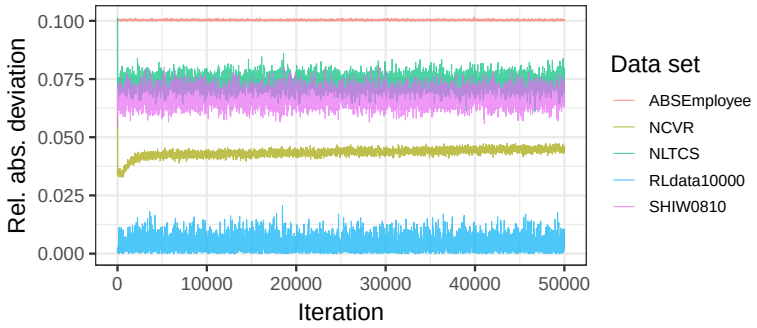

Figure S3: Balance of the partitions for a single run on each data set. The balance is measured in terms of the relative absolute deviation from the perfectly balanced configuration. The number of partitions $P = 64, 64, 16, 2, 8$ for each data set (in the order listed in the legend).

H Balance of the partitions

In Section 4.2, we proposed a partition function based on k-d trees, and argued that it could yield balanced partitions with good entity separation. While running `d-blink` with the k-d tree partition function, we recorded the size of the partitions ($|\mathcal{E}_p|$ for all p) to assess whether they were well-balanced. Figure S3 illustrates the results in terms of the relative absolute deviation from the perfectly balanced configuration (where the entities are divided equally among the partitions). We can see that the k-d tree partitioner is functioning quite well—the deviation from the perfectly balanced configuration is no more than 10% for all data sets.

I Uncertainty measures

`d-blink` allows for measures of uncertainty to be reported, unlike the baseline methods, since we have the full posterior distribution. For example, in Figure S4 we compute posterior estimates for the number of entities present in each data set, with 95% Bayesian credible intervals. Note that the posterior estimates are typically quite sharp. This seems to confirm arguments by Steorts et al. (2016) regarding the informativeness of the prior for the linkage structure in `blink`. Research on less informative priors is ongoing (Zanella et al., 2016).

J Sensitivity analysis

We conducted an empirical sensitivity analysis for `d-blink` using the `RLdata10000` data set. We selected this data set as it is relatively small, which made it quick to run the inference for various hyperparameter combinations. The parameters tested were:

- α_ℓ, β_ℓ: the shape parameters for the Beta prior on the distortion probabilities. We used the same values for all attributes (ℓ).

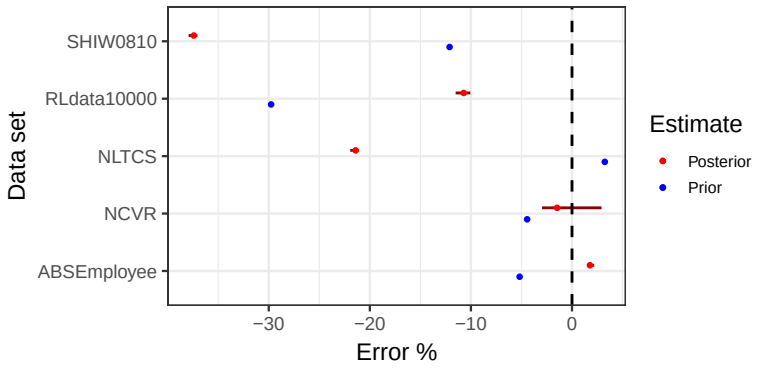

Figure S4: Percentage error in the posterior/prior estimates for the number of observed entities for `d-blink`. The posterior estimates are generally sharp and underestimate the true number of observed entities.

- E: the size of the latent population.
- $s_{\max}$: the scaling factor for the similarity function. This controls the inverse temperature of the softmax distribution for the distorted attribute values.

We varied each of these parameters in turn, while holding all other parameters fixed. For the Beta prior on the distortion probabilities, we first varied the strength while fixing the prior mean to $\sim 1\%$, then we varied the mean (1%, 5% and 10%) while fixing $\alpha + \beta$ (related to the strength). Table S1 presents the evaluation measures for each combination of parameters. The results indicate that the inferred linkage structure is relatively sensitive to all of the parameters, however sensitivity is in general predictable, following clear and intuitive trends. Of particular interest is the fact that the model performs best when the Beta prior on the distortion probabilities is sharply peaked near zero. It seems that the model has a tendency to overestimate the amount of distortion, particularly in the absence of ground truth.

Table S1: Sensitivity analysis for various parameters combinations using `RLdata10000`. The first group of rows tests the effect of varying the *strength* of the Beta prior, the second group tests the effect of varying the *mean* of the Beta prior, the third group tests the effect of varying the population size, and the fourth group tests the effect of varying the scaling factor for the similarity function.

Distortion		Pop. size	Max. sim.	Pairwise measures			Cluster measures	
α	β	E	$s_{\max}$	Precision	Recall	F1-score	ARI	Err. # clust.
0.1	**10.0**	10000	10.0	0.5342	0.9990	0.6962	0.6962	−17.47%
1.0	**100.0**	10000	10.0	0.5435	0.9990	0.7040	0.7040	−16.58%
10.0	**1000.0**	10000	10.0	0.6334	0.9970	0.7747	0.7747	−10.97%
100.0	**10000.0**	10000	10.0	0.9180	0.9850	0.9503	0.9503	−1.595%
10.0	**1000.0**	10000	10.0	0.6334	0.9970	0.7747	0.7747	−10.97%
50.5	**959.5**	10000	10.0	0.6132	0.9970	0.7593	0.7593	−11.90%
101.0	**909.0**	10000	10.0	0.5992	0.9970	0.7485	0.7485	−12.90%
10.0	1000.0	**9000**	10.0	0.5306	0.9970	0.6926	0.6926	−15.65%
10.0	1000.0	**10000**	10.0	0.6334	0.9970	0.7747	0.7747	−10.97%
10.0	1000.0	**11000**	10.0	0.6999	0.9960	0.8221	0.8221	−7.365%
10.0	1000.0	10000	**5.0**	0.6927	0.9940	0.8164	0.8164	−22.12%
10.0	1000.0	10000	**10.0**	0.6334	0.9970	0.7747	0.7747	−10.97%
10.0	1000.0	10000	**50.0**	0.2112	0.3920	0.2745	0.2745	−12.50%

K Trace plots

K.1 Attribute-level distortion

The following figures relate to the aggregate distortion per attribute for each data set. On the left are the trace plots, which show the aggregate distortion for each attribute (stacked vertically) along the Markov chain. On the right are the corresponding autocorrelation plots.

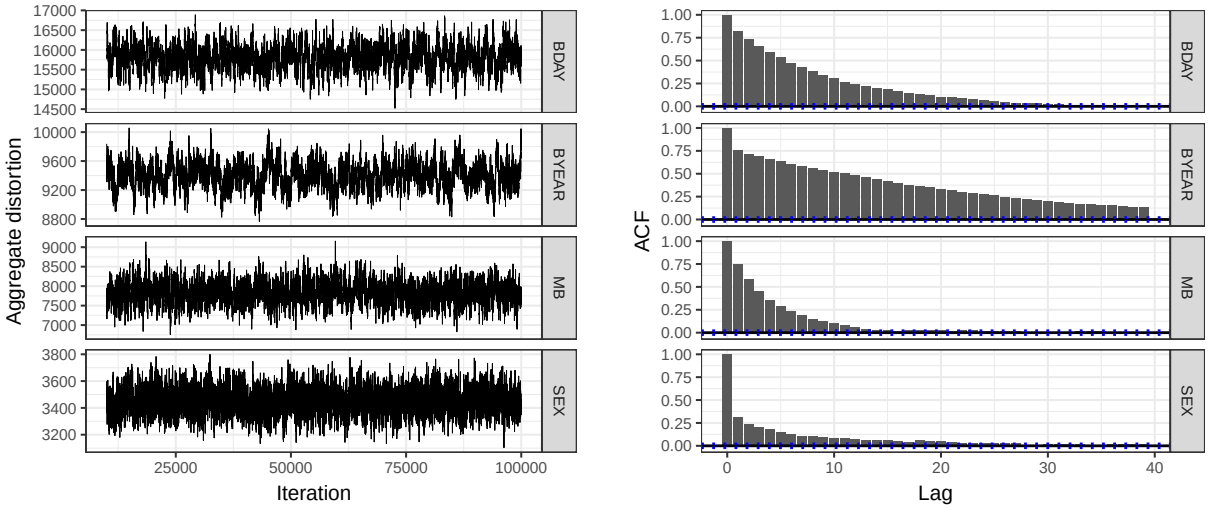

Figure S5: Attribute-level distortion for `ABSEmployee`

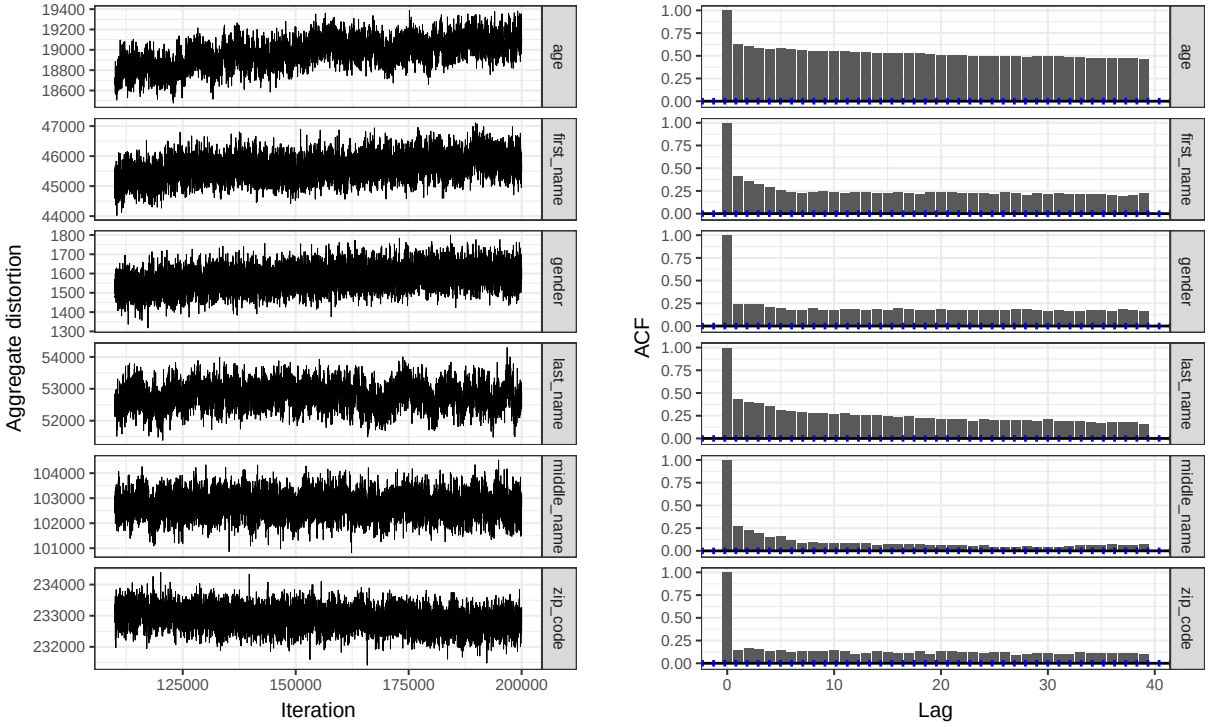

Figure S6: Attribute-level distortion for `NCVR`

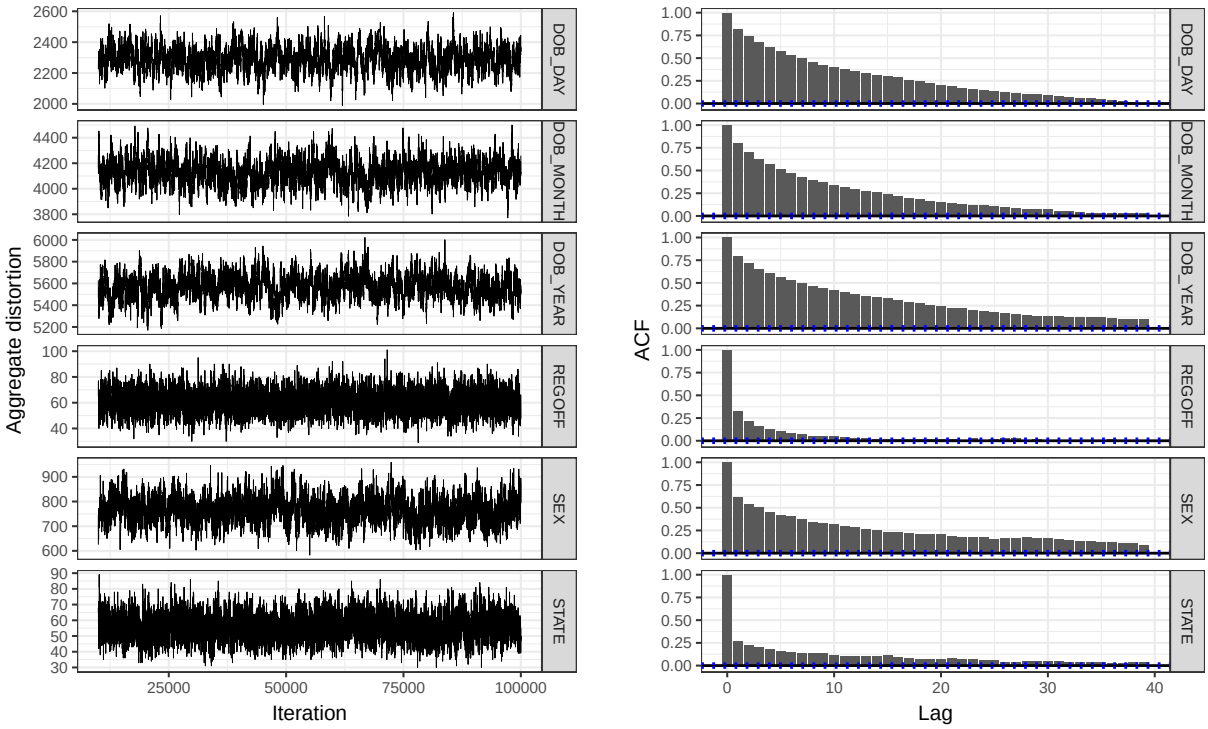

Figure S7: Attribute-level distortion for `NLTCS`

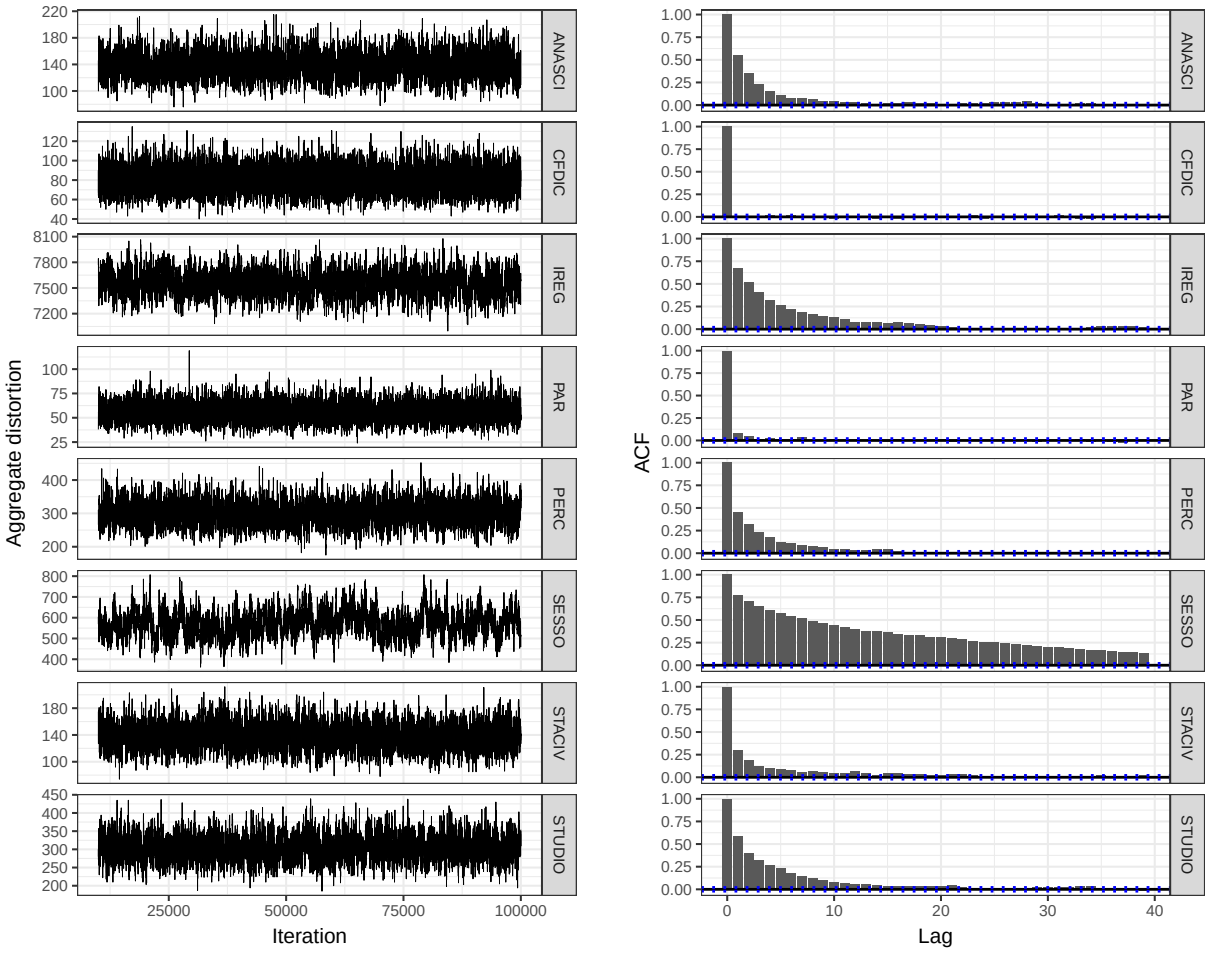

Figure S8: Attribute-level distortion for `SHIW0810`

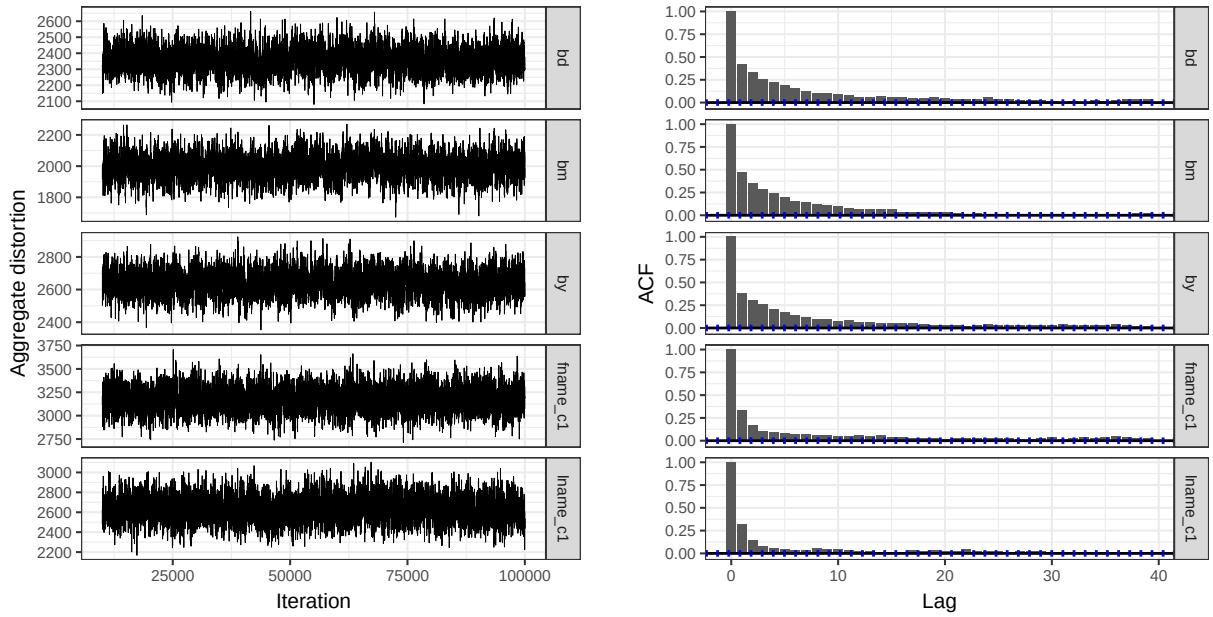

Figure S9: Attribute-level distortion for `RLdata10000`

K.2 Distribution of record distortion

The following figures relate to the distribution of record distortion for each data set. Specifically, we count the number of records with 0 distorted attributes, 1 distorted attribute, 2 distorted attributes, etc. On the left are the trace plots, which show the record counts for each distortion level (stacked vertically) along the Markov chain. On the right are the corresponding autocorrelation plots.

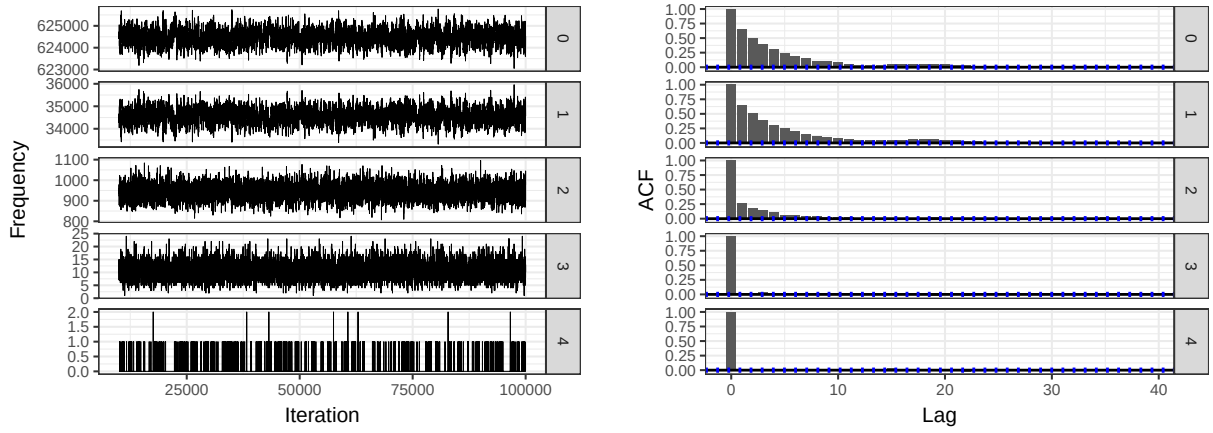

Figure S10: Distribution of record distortion for `ABSEmployee`

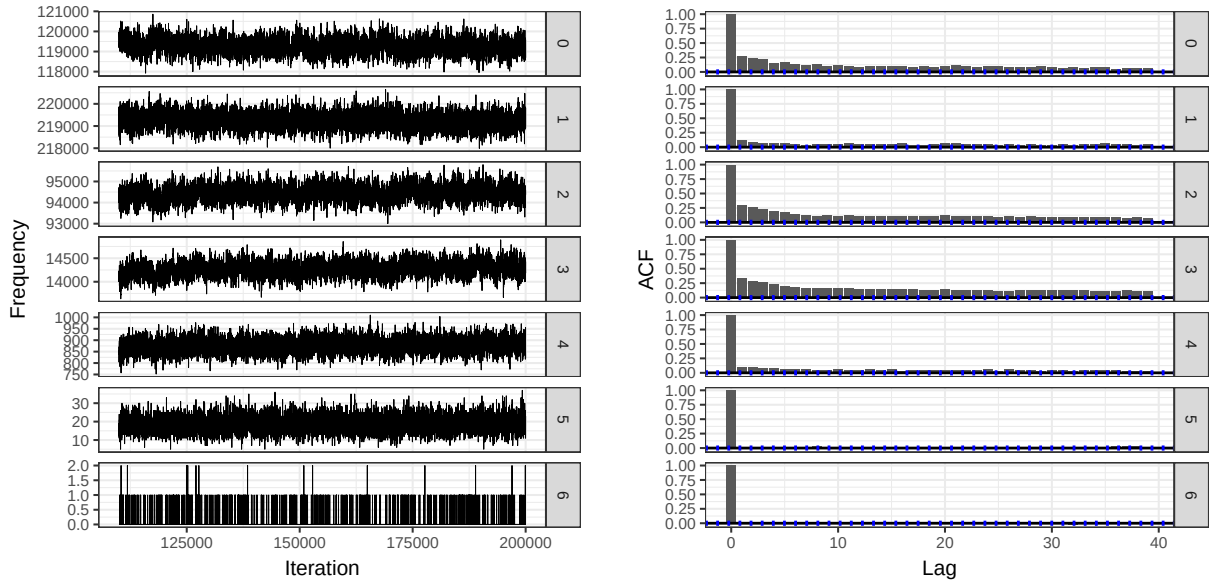

Figure S11: Distribution of record distortion for `NCVR`

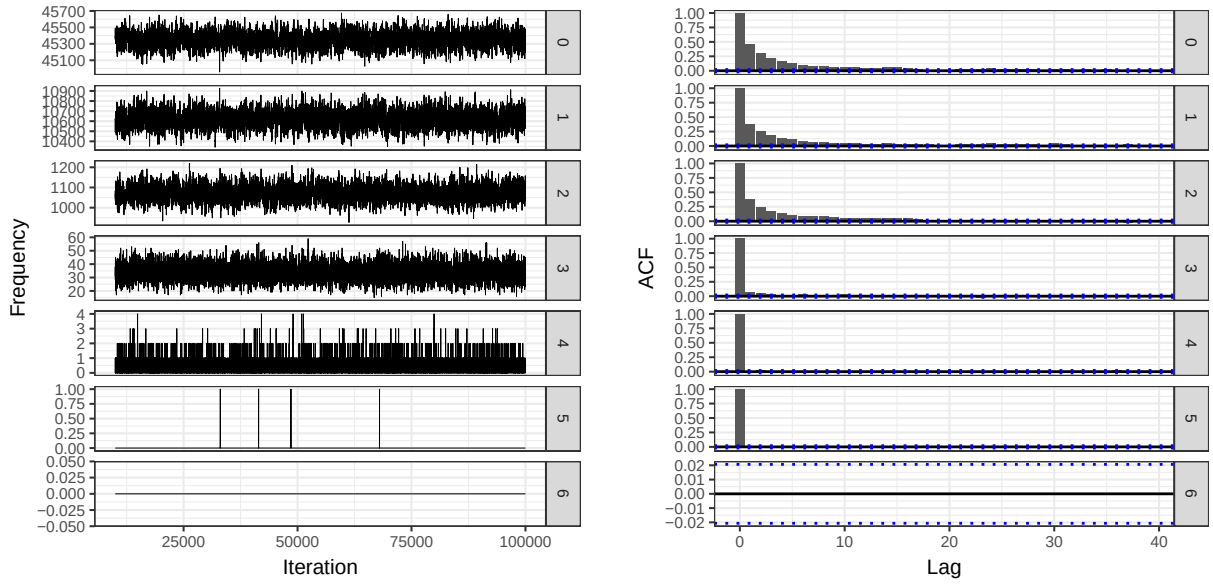

Figure S12: Distribution of record distortion for `NLTCS`

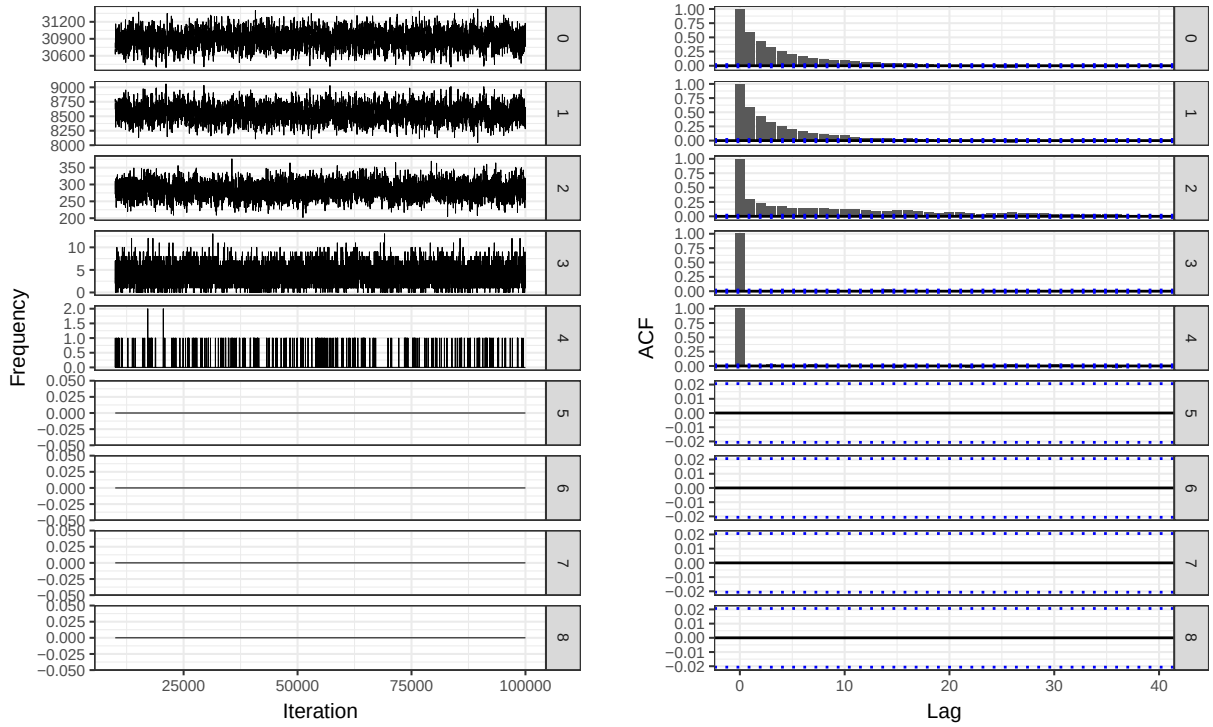

Figure S13: Distribution of record distortion for `SHIW0810`

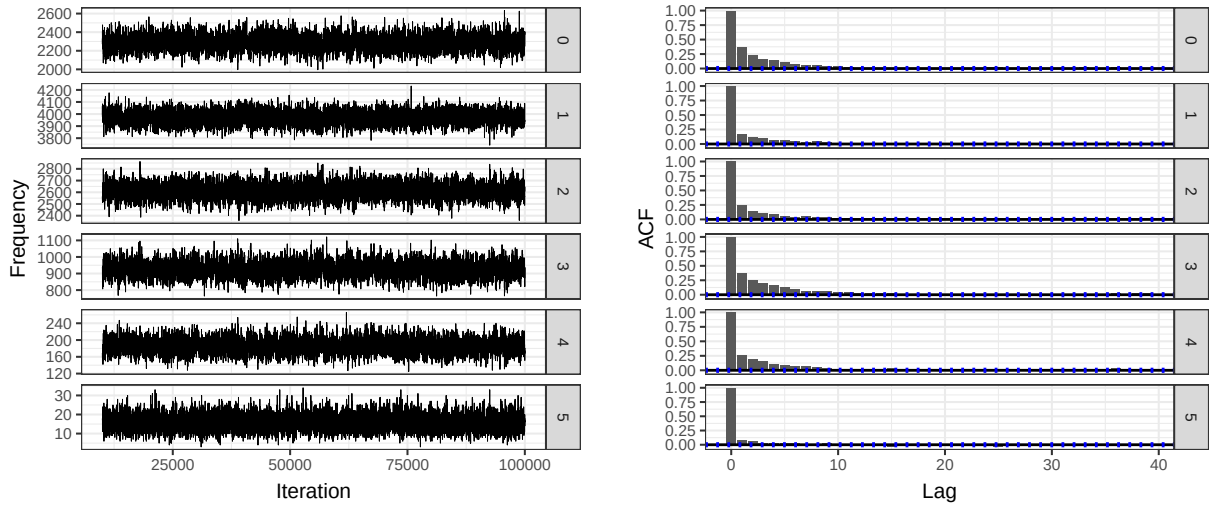

Figure S14: Distribution of record distortion for `RLdata10000`

K.3 Cluster size distribution

The following figures relate to the distribution of cluster (entity) sizes for each data set. Specifically, we count the number of entities with 0 linked records, 1 linked record, 2 linked records, etc. On the left are the trace plots, which show the counts for each cluster

size (stacked vertically) along the Markov chain. On the right are the corresponding autocorrelation plots.

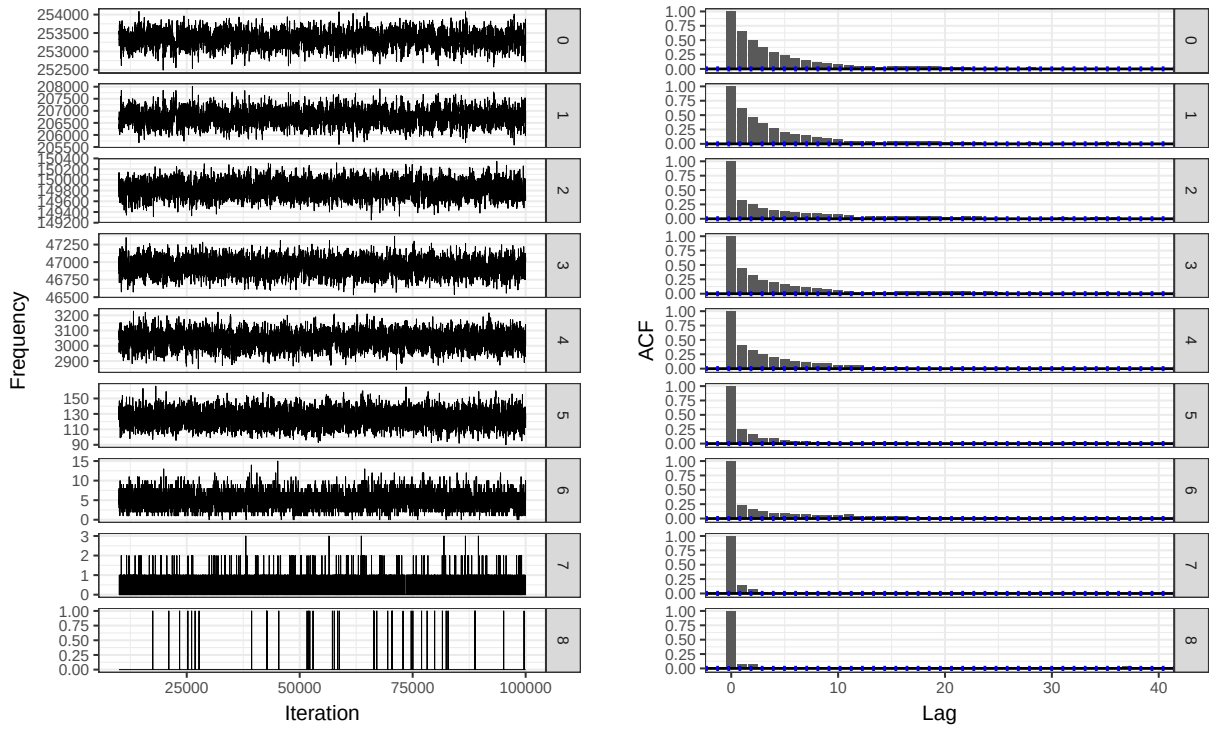

Figure S15: Cluster size distribution for `ABSEmployee`

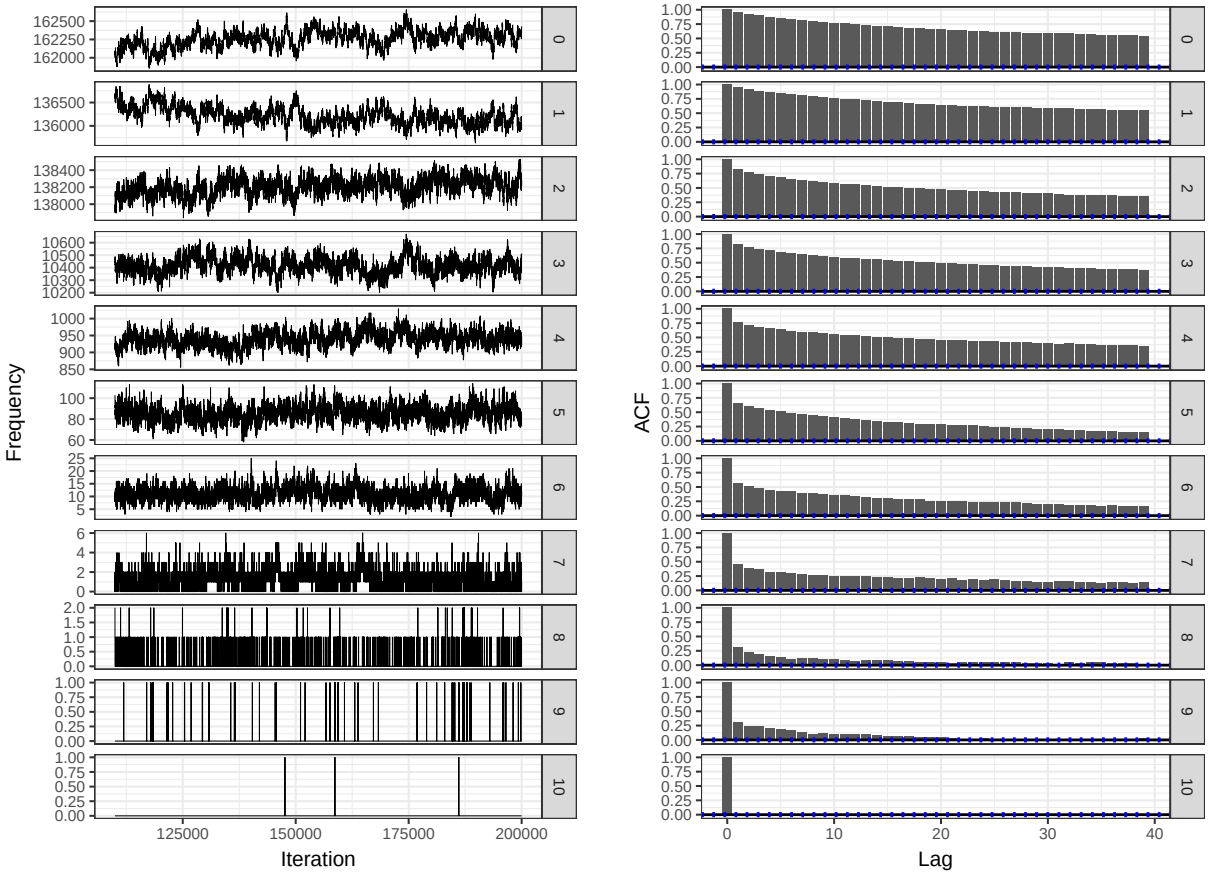

Figure S16: Cluster size distribution for `NCVR`

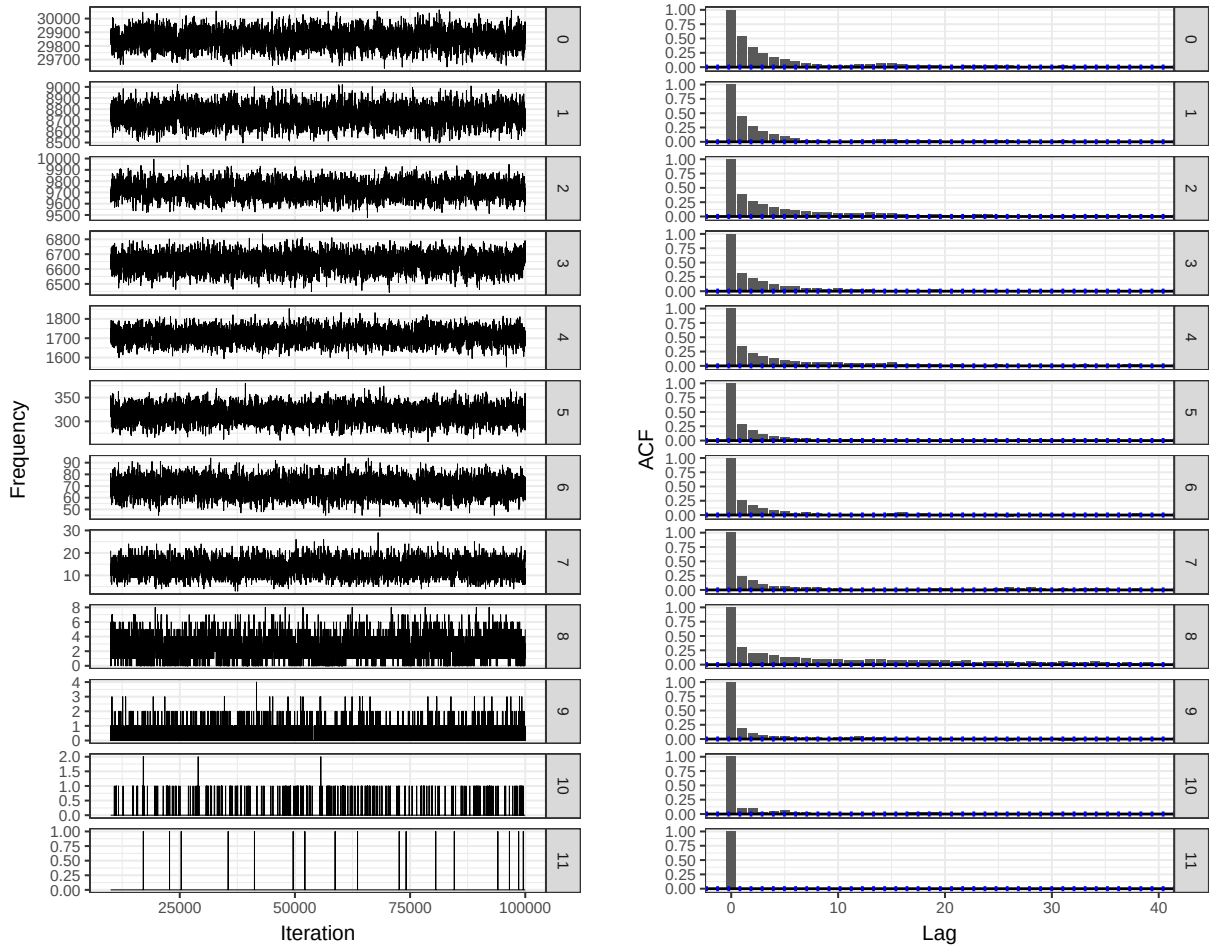

Figure S17: Cluster size distribution for `NLTCS`

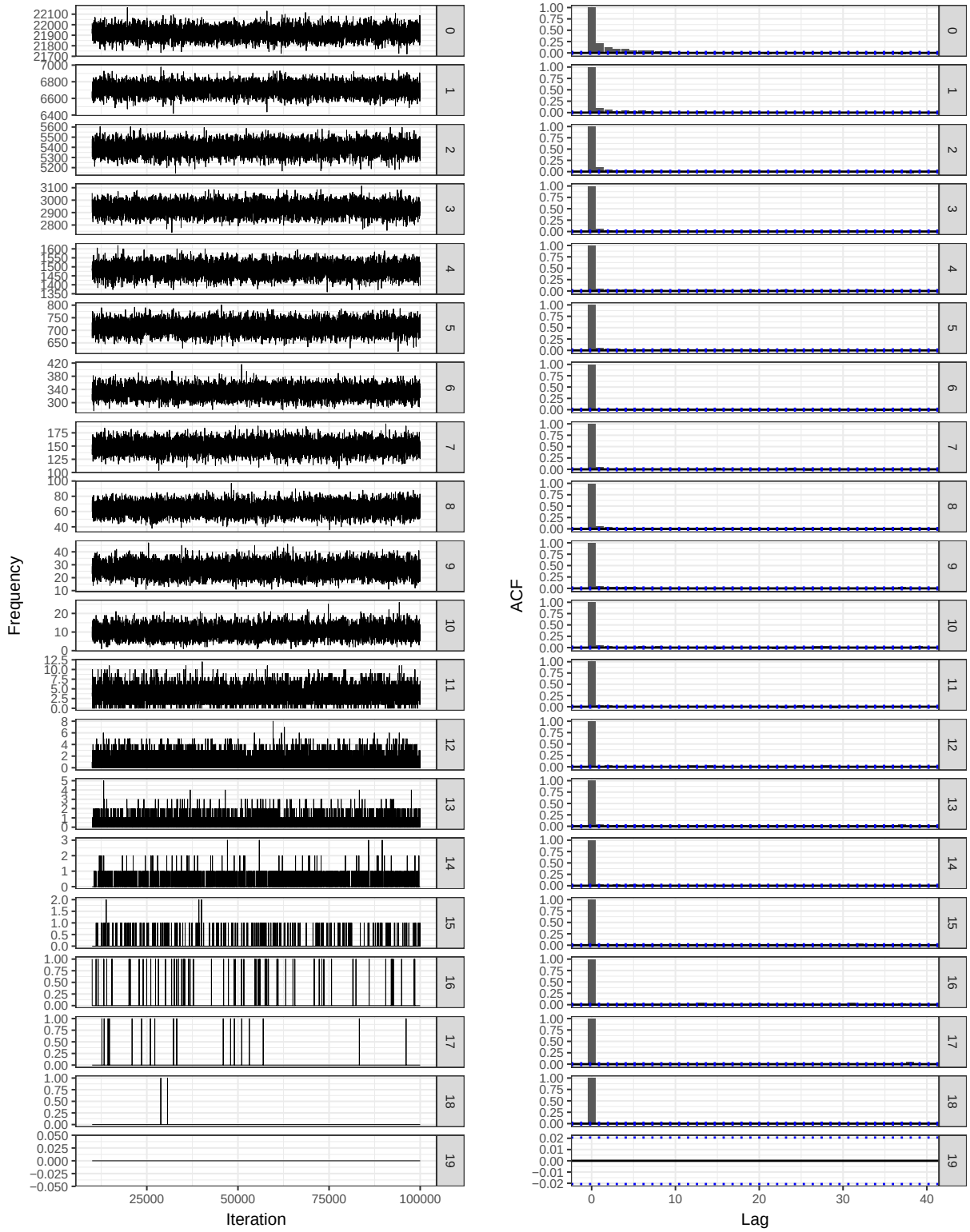

Figure S18: Cluster size distribution for `SHIW0810`

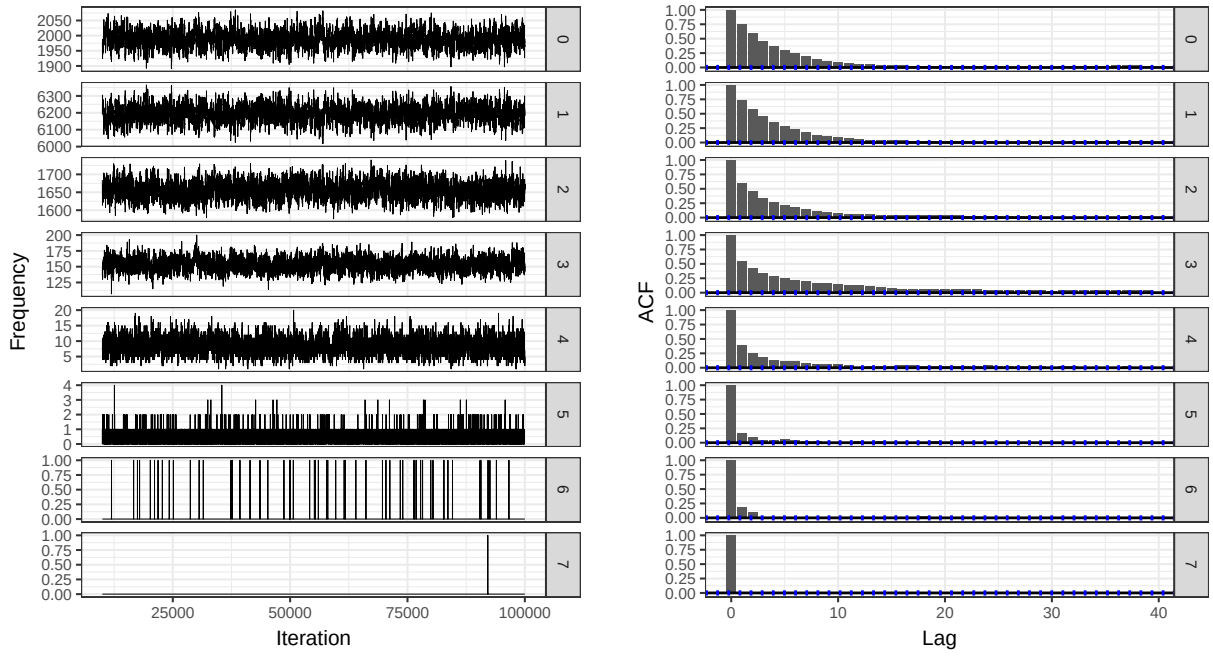

Figure S19: Cluster size distribution for `RLdata10000`

References

Banca d'Italia. "Bank of Italy – Survey on Household Income and Wealth." `http://www.bancaditalia.it/pubblicazioni/indagine-famiglie/index.html` (n.d.). Accessed: 9 March 2018.

Christen, P. "Preparation of a real temporal voter data set for record linkage and duplicate detection research." Technical report, Australian National University (2014).

Manton, K. G. "National Long-Term Care Survey: 1982, 1984, 1989, 1994, 1999 and 2004." (2010).

Sariyar, M. and Borg, A. "The RecordLinkage Package: Detecting Errors in Data." *The R Journal*, 2(2):61–67 (2010).

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

Vose, M. D. "A linear algorithm for generating random numbers with a given distribution." *IEEE Transactions on Software Engineering*, 17(9):972–975 (1991).

Zanella, G., Betancourt, B., Wallach, H., Miller, J., Zaidi, A., and Steorts, R. C. "Flexible Models for Microclustering with Application to Entity Resolution." In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 1425–1433. NY, USA: Curran Associates Inc. (2016).