A Query Engine for Probabilistic Preferences

Uzi Cohen
Technion, Israel
uzi.cohen@campus.technion.ac.il

Batya Kenig
Technion, Israel
batyak@cs.technion.ac.il

Haoyue Ping
Drexel University, USA
hp354@drexel.edu

Benny Kimelfeld*
Technion, Israel
bennyk@cs.technion.ac.il

Julia Stoyanovich†
Drexel University, USA
stoyanovich@drexel.edu

ABSTRACT

Models of uncertain preferences, such as Mallows, have been extensively studied due to their plethora of application domains. In a recent work, a conceptual and theoretical framework has been proposed for supporting uncertain preferences as first-class citizens in a relational database. The resulting database is probabilistic, and, consequently, query evaluation entails inference of marginal probabilities of query answers. In this paper, we embark on the challenge of a practical realization of this framework.

We first describe an implementation of a query engine that supports querying probabilistic preferences alongside relational data. Our system accommodates preference distributions in the general form of the Repeated Insertion Model (RIM), which generalizes Mallows and other models. We then devise a novel inference algorithm for conjunctive queries over RIM, and show that it significantly outperforms the state of the art in terms of both asymptotic and empirical execution cost. We also develop performance optimizations that are based on sharing computation among different inference tasks in the workload. Finally, we conduct an extensive experimental evaluation and demonstrate that clear performance benefits can be realized by a query engine with built-in probabilistic inference, as compared to a stand-alone implementation with a black-box inference solver.

ACM Reference Format:
Uzi Cohen, Batya Kenig, Haoyue Ping, Benny Kimelfeld, and Julia Stoyanovich. 2018. A Query Engine for Probabilistic Preferences. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196923

1 INTRODUCTION

Preferences state relative desirability among *items*, and they appear in various forms such as full item rankings, partial orders, and top-k choices. Large amounts of preference information are collected and analyzed in a variety of domains, including recommendation systems [3, 42, 46] (where items are products), polling and elections [7, 18, 19, 40] (where items are political candidates), bioinformatics [1, 31, 47] (where items are genes or proteins), and data cleaning [11, 13, 17, 29] (where items are tuples in relations). User preferences are often inferred from indirect input (such as purchases), or from the preferences of a population with similar demographics, making them uncertain in nature. This motivates a rich body of work on probabilistic preference models in the statistics literature [39].

The machine learning and computational statistics communities have developed effective models and analysis techniques for preferences [2, 4, 8, 18, 21, 24, 32, 34, 35]. An important model for uncertain preferences is the *Repeated Insertion Model* (RIM) [9]. RIM defines a probability distribution over the rankings of a set of items via a generative process that reorders a *reference ranking* $\boldsymbol{\sigma}$ by repeatedly inserting items into a random ranking in a left-to-right scan. A RIM model has two parameters. The first is a ranking $\boldsymbol{\sigma} = \langle \sigma_1, \ldots, \sigma_m \rangle$ called a *reference ranking*. The second is an *insertion function* Π that determines the insertion probabilities of the items during the left-to-right scan. Different realizations of RIM, defined by distinct specifications of the insertion function, give rise to different distributions over rankings [14, 16, 37].

Most notable is the *Mallows* model [37], and its extension, the Generalized Mallows Model, which have received much attention from the statistics [14, 16, 44], political science [44] and machine learning communities. The literature on these models ranges from the theoretical [43] to more practical applications such as genome rearrangements for the construction of phylogenetic trees [22], multilabel classification [5], and explaining the diversity in a given sample of voters [45]. From the machine learning perspective, the problem is to find the parameters of the Mallows model – the reference ranking $\boldsymbol{\sigma}$, and the dispersion parameter ϕ, used to define the insertion function in the Mallows realization of RIM. A large body of work has dealt with the problem of learning these parameters using independent samples from the distribution [8, 35, 38, 45].

In this paper we follow the pursuit of *database systems* for managing preferences, wherein preferences, uncertain preferences, and specialized operators on preferences are first-class citizens [23, 26, 28]. This pursuit is motivated by two standard projected benefits. One is *reduced software complexity*—preference analysis will be done by simple SQL queries that access both preferences and ordinary (contextual) data, rather than by complex programs that

*This work was supported in part by ISF Grant No. 5921551, and by BSF Grant No. 2014391.
†This work was supported in part by NSF Grants No. 1464327, 1539856 and 1741047, and by BSF Grant No. 2014391.

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196923

integrate between ordinary databases, probabilistic preference models, and statistical solvers. The second is *enhanced performance* via development of smart execution plans, helping reduce the number of costly invocations of statistical solvers. As a starting point, we adopt the formalism of a *preference database* of Jacob's et al. [23], wherein a database consists of two types of relations: ordinary relations and *preference relations*. A preference relation represents a collection of orders, each referred to as a *session*. A tuple of a preference relation has the form $(\mathbf{s}; \sigma; \tau)$, and states that in the order $\succ_{\mathbf{s}}$ of session $\mathbf{s}$ it is the case that item σ is preferred to item τ, denoted $\sigma \succ_{\mathbf{s}} \tau$. For example, the tuple (Ann, Oct-5; Sanders; Clinton) denotes that in a poll conducted on October 5^{th}, Ann preferred Sanders to Clinton. Here, (Ann, Oct-5) identifies a session. Importantly, the internal representation of a preference need not store every pair-wise comparison, such as Sanders $\succeq$ Clinton, explicitly. Rather, the representation can be as compact as a vector of items in the case of a linear order.

Taking a step forward, Kenig et al. [26] extended the framework of Jacob's et al. [23] to allow for representations of probabilistic preferences, and illustrated their extension on the *RIM Probabilistic Preference Database*, abbreviated *RIM-PPD*. There, every session is internally associated with an independent RIM model that compactly represents a probability space over rankings. An example of a RIM-PPD is presented in Figure 1. Semantically, a RIM-PPD is a *probabilistic database* [48] where every random possible world (which is a deterministic database) is obtained by sampling from the stored RIM models. RIM-PPD adopts standard semantics of *query evaluation*, associating each answer with a *confidence value*— the probability of getting this answer in a random possible world [48]. Hence, query evaluation entails *probabilistic inference*, namely, computing the marginal probability of query answers.

In this paper, we embark on the challenge of a practical realization of a probabilistic preference database in scope of a general-purpose database system. We present our first step towards an implementation of a RIM-PPD, and describe a query-engine prototype that consists of several components: a *query parser*, an *execution engine*, and a *relational database* (PostgreSQL). The execution engine supports queries that have tractable complexity— *itemwise conjunctive queries* [26]. Intuitively, these are queries that state a preference among constants and variables (e.g., $x \succ y$ and $y \succ$ Trump) in addition to an independent condition on each item variable (e.g., x is a female candidate, and y is a male candidate who spoke at the 2016 Democratic National Convention).[1] Kenig et al. [26] show that, at least for the fragment of queries without self joins, itemwise queries are *precisely* the queries that can be evaluated in polynomial time.

The execution engine uses the relational query engine to translate query evaluation into an inference problem over the stored RIM models—*labeled RIM matching*. Intuitively, in this inference problem each item is associated with a set of *labels*, and the goal is to compute the marginal probability of the occurrence of a pattern among these labels. For instance, in the elections database in Figure 1 every candidate is associated with a set of labels concerning her party, age, education etc. In that context, we may wish to determine the probability that Ann ranks two female democrats, one junior (e.g., age$\leq$ 35), and one senior (e.g., age $>$ 60) before a republican candidate. The goal is to determine the probability that, in a random ranking, two candidates labeled with F,$\leq$ 35 and F,$>$ 60 precede a candidate labeled R. Our first attempt at solving this inference task is by implementing the algorithm *Top Matching* (or TM in notation) of Kenig et al. [26]. This algorithm is quite involved and the main challenge we encountered was very long execution times. For example, on a database of merely 1,000 voters and 20 candidates, a relatively small query Q, defining a partial order over 5 sets of candidates, took more than 6 hours to execute.

To attack the problem of bad performance, we develop two kinds of optimizations. The first optimization is a contribution of independent interest, which concerns the inference algorithm itself (regardless of the database). We devise a new inference algorithm for labeled RIM matching called *Lifted Top Matching* (or LTM). LTM applies the concept of *lifted inference* [20, 27, 30] over *Top Matching*, reducing the asymptotic (theoretical) execution cost and showing dramatic improvements in our empirical study. We describe LTM in Section 4, and present experimental results in Section 5. For example, evaluating query Q, previously described, takes 54 minutes with LTM, improving on the running time of TM by a factor of 7.

To evaluate a query, our execution engine makes repeated calls to the LTM solver. The second type of optimizations we devise is that of *workload* optimizations, where we aim to reduce the execution cost of an entire sequence of calls to LTM. We consider two workload-based optimizations. The first applies to the case where RIM models exhibit the same *structure* (e.g., since they rely on similar partial training data), but different *parameters* (e.g., due to difference in variance). We show that, in the case of Mallows, we can simulate multiple executions of LTM in a single execution, obtaining substantial performance gains.

The second workload optimization applies to queries that look for the K most probable sessions satisfying a given itemwise query Q. There, we apply a two-step technique. We first call LTM with a *relaxed* pattern that has two properties: *(a)* its probability is no lower than that of the original pattern, and so it can serve as an upper bound on the true probability, *and (b)* the execution cost is far lower. We then use the upper bounds for pruning away executions of LTM that will (provably) not make it to the top-K. We again observe dramatic improvements in the running times. For example, retrieving the top-10 sessions for query Q executes within 7 minutes, improving on an LTM-based solution that does not use the upper-bounds optimization by a factor of 8. The success of these optimizations illustrates the importance of query evaluation within a database system — of having access to complete workloads, rather than being limited to individual inference tasks.

We conduct our experiments on Mallows models on real and synthetic datasets. We note that our algorithms do not utilize any difference between Mallows and general RIM, making our approach both general, and representative of the computational effort involved in reasoning with RIM distributions. Our experiments use a collection of itemwise queries, and study performance gains of various kinds. We compare between *Top Matching* [26] and our *Lifted Top Matching* algorithms on both vanilla labeled RIM matching and on general query evaluation (which entails a workload of labeled RIM matching tasks). In addition, we study the empirical impact of our workload-based performance optimizations.

[1] For a precise definition the reader is referred to Section 2.

In summary, the contributions of this paper are as follows. First, we describe an implementation of a query engine for RIM-PPD. Second, we devise a novel algorithm for labeled RIM matching, and prove its superiority over the state of the art both theoretically and empirically. Third, we devise workload optimizations: simulating multiple executions on a shared reference ranking, and top-K pruning based on pattern relaxation. Lastly, we conduct an experimental study of the empirical performance of our query engine.

The rest of this paper is organized as follows. In Section 2 we present the main concepts and terminology used throughout the paper. We describe the design and implementation of our query engine in Section 3, and present and analyze our novel algorithm for inference over labeled RIM in Section 4. We present our experimental evaluation in Section 5, and conclude in Section 6. Proofs and additional experiments are included in the Appendix.

2 PRELIMINARIES

In this section we present basic concepts, terminology and definitions that we use throughout the paper. We adopt much of the notation of Kenig et al. [26], who have established a theoretical framework that our system builds upon.

Orders and Rankings. An *order* is a binary relation $\succ$ over a set $I = \{\sigma_1, \ldots, \sigma_n\}$ of *items*. A *partial order* is an order that is irreflexive and transitive, that is, for all i, j and k we have $\sigma_i \not\succ \sigma_i$, and $\sigma_i \succ \sigma_j$ and $\sigma_j \succ \sigma_k$ imply $\sigma_i \succ \sigma_k$. A partial order where every two items are comparable is a *linear* order, or a *ranking*. Notationally, we identify a ranking $\sigma_1 \succ \cdots \succ \sigma_n$ with the sequence $\langle \sigma_1, \ldots, \sigma_n \rangle$. We denote by $rnk(I)$ the set of all rankings over I. For a ranking $\boldsymbol{\sigma} = \langle \sigma_1, \ldots, \sigma_n \rangle$ we denote by items($\boldsymbol{\sigma}$) the set $\{\sigma_1, \ldots, \sigma_n\}$, and by $\succ_{\boldsymbol{\sigma}}$ the order that $\boldsymbol{\sigma}$ stands for: $\sigma_i \succ_{\boldsymbol{\sigma}} \sigma_j$ whenever $i < j$.

The Repeated Insertion Model (RIM). A RIM model has two parameters. The first is a ranking $\boldsymbol{\sigma} = \langle \sigma_1, \ldots, \sigma_m \rangle$ called a *reference ranking*. RIM defines a probability distribution over the rankings of the items, that is, the probability space $rnk(\text{items}(\boldsymbol{\sigma}))$. The probability of each ranking is defined by a generative process, as described below, using the second parameter Π called an *insertion function*. The insertion function maps every pair (i, j) of integers with $1 \le j \le i \le m$ to a probability $\Pi(i, j) \in [0, 1]$, so that for all $i = 1, \ldots, m$ we have $\sum_{j=1}^{i} \Pi(i, j) = 1$. We denote the RIM model with the parameters $\boldsymbol{\sigma}$ and Π by $\mathsf{RIM}(\boldsymbol{\sigma}, \Pi)$.

The generative process of $\mathsf{RIM}(\boldsymbol{\sigma}, \Pi)$ produces a random ranking $\boldsymbol{\tau}$ as follows. Suppose that $\boldsymbol{\sigma} = \langle \sigma_1, \ldots, \sigma_m \rangle$. Begin with an empty ranking, and scan the items $\sigma_1, \ldots, \sigma_m$ left to right. Each σ_i is inserted to a random position $j \in \{1, \ldots, i\}$ inside the current $\langle \tau_1, \ldots, \tau_{i-1} \rangle$, pushing $\tau_j, \ldots, \tau_{i-1}$ forward, and resulting in the series $\langle \tau_1, \ldots, \tau_{j-1}, \sigma_i, \tau_j, \ldots, \tau_{i-1} \rangle$. The random position j is selected with the probability $\Pi(i, j)$, independently of previous choices.

Example 2.1. We use the example of Kenig et al. [26] from the domain of presidential elections in USA. Consider the model $\mathsf{RIM}(\boldsymbol{\sigma}, \Pi)$, where $\boldsymbol{\sigma} = \langle \texttt{Clinton}, \texttt{Sanders}, \texttt{Rubio}, \texttt{Trump} \rangle$. The following illustrates a random execution of the above generative process. We begin with an empty $\boldsymbol{\tau}$ and insert items as follows.

(1) With probability $\Pi(1, 1) = 1$ `Clinton` is placed first to get

$$\boldsymbol{\tau} = \langle \texttt{Clinton} \rangle ;$$

Candidates (o)

candidate	party	sex	age	edu	reg
Trump	R	M	70	BS	NE
Clinton	D	F	69	JD	NE
Sanders	D	M	75	BS	NE
Rubio	R	M	45	JD	S

Voters (o)

voter	sex	age	edu
Ann	F	20	BS
Bob	M	30	BS
Dave	M	50	MS

Polls (p)

voter	date	*Preference model* **MAL**($\boldsymbol{\sigma}$, ϕ)
Ann	Oct-5	⟨Clinton, Sanders, Rubio, Trump⟩, 0.3
Bob	Oct-5	⟨Trump, Rubio, Sanders, Clinton⟩, 0.3
Dave	Nov-5	⟨Clinton, Sanders, Rubio, Trump⟩, 0.5

Figure 1: A RIM-PPD inspired by [26].

(2) With probability $\Pi(2, 2)$ we insert `Sanders` to the second (last) position, resulting in

$$\boldsymbol{\tau} = \langle \texttt{Clinton}, \texttt{Sanders} \rangle ;$$

(3) With probability $\Pi(3, 2)$ we insert `Rubio` to the second position, pushing `Sanders` rightwards and resulting in

$$\boldsymbol{\tau} = \langle \texttt{Clinton}, \texttt{Rubio}, \texttt{Sanders} \rangle ;$$

(4) Finally, with probability $\Pi(4, 4)$ we put `Trump` fourth, to get

$$\boldsymbol{\tau} = \langle \texttt{Clinton}, \texttt{Rubio}, \texttt{Sanders}, \texttt{Trump} \rangle .$$

This produces the ranking $\langle \texttt{Clinton}, \texttt{Rubio}, \texttt{Sanders}, \texttt{Trump} \rangle$. It is a known fact that the resulting ranking is in a one-to-one correspondence with the insertion orders. In particular, the probability of $\langle \texttt{Clinton}, \texttt{Rubio}, \texttt{Sanders}, \texttt{Trump} \rangle$ is the product of the above insertion probabilities, that is, $\Pi(1, 1) \times \Pi(2, 2) \times \Pi(3, 2) \times \Pi(4, 4)$. □

The Mallows Model. As a special case of RIM, a *Mallows* model [37] is parameterized by a reference ranking $\boldsymbol{\sigma}$ and a *dispersion parameter* $\phi \in (0, 1]$. Intuitively, the probability of a ranking $\boldsymbol{\tau}$ becomes exponentially smaller as the distance of $\boldsymbol{\tau}$ from $\boldsymbol{\sigma}$ increases. Formally, this model, which we denoted by $\mathsf{MAL}(\boldsymbol{\sigma}, \phi)$, assigns to every ranking $\boldsymbol{\tau}$ over the items of $\boldsymbol{\sigma}$ the probability $\Pr(\boldsymbol{\tau} \mid \boldsymbol{\sigma}, \phi)$ defined as $\frac{1}{Z}\phi^{d(\boldsymbol{\tau}, \boldsymbol{\sigma})}$. Here, $d(\boldsymbol{\tau}, \boldsymbol{\sigma})$ is *Kendall's tau* [25] distance between $\boldsymbol{\tau}$ and $\boldsymbol{\sigma}$ that counts the pair-wise *disagreements* between $\boldsymbol{\tau}$ and $\boldsymbol{\sigma}$; that is, $d(\boldsymbol{\tau}, \boldsymbol{\sigma})$ is the number of item pairs (σ, σ') such that $\sigma \succ_{\boldsymbol{\sigma}} \sigma'$ and $\sigma' \succ_{\boldsymbol{\tau}} \sigma$. The number Z is the normalization constant, that is, the sum of $\phi^{d(\boldsymbol{\tau}, \boldsymbol{\sigma})}$ over all $\boldsymbol{\tau}$. Lower values of ϕ concentrate most of the probability mass around $\boldsymbol{\sigma}$, while $\phi = 1$ corresponds to the uniform probability distribution over the rankings. Doignon et al. [9] showed that $\mathsf{MAL}(\boldsymbol{\sigma}, \phi)$ can be represented as $\mathsf{RIM}(\boldsymbol{\sigma}, \Pi)$, where $\Pi(i, j)$ is equal to $\phi^{i-j}/(1 + \phi + \cdots + \phi^{i-1})$.

Labeled RIM Matching. We recall the definition of *labeled RIM matching* [26], an inference problem over RIM that we later use for query evaluation. A *labeled RIM* extends RIM by associating with each item a finite set of labels from an infinite set $\boldsymbol{\Lambda}$ of *item labels*. More formally, a labeled RIM is parameterized by $\boldsymbol{\sigma}$, Π and λ, where $\mathsf{RIM}(\boldsymbol{\sigma}, \Pi)$ is a RIM model and λ is a labeling function that maps every item σ_i in $\boldsymbol{\sigma}$ to a finite set $\lambda(\sigma_i)$ of labels in $\boldsymbol{\Lambda}$. We denote this model by $\mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}, \Pi, \lambda)$. A *label pattern* (or just *pattern*) is a directed graph g where every node is a label in $\boldsymbol{\Lambda}$. We denote by nodes(g) and edges(g) the sets of nodes and edges of g, respectively.

(a) A label pattern **(b) Label pattern of Ex. 4.4**

Figure 2: Label Patterns

Let $\boldsymbol{\tau} = \langle \tau_1, \ldots, \tau_m \rangle$ be a random ranking generated from the distribution $\mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}, \Pi, \lambda)$, and let g be a pattern. For a ranking $\boldsymbol{\tau}$, we denote by $\boldsymbol{\tau}(i)$ the item at position i, by $\boldsymbol{\tau}^{-1}(\tau)$ the position of item τ in $\boldsymbol{\tau}$, and by $\succ_{\boldsymbol{\tau}}$ the linear order that corresponds to $\boldsymbol{\tau}$ (i.e., $\tau_i \succ_{\boldsymbol{\tau}} \tau_j$ if $i < j$). An *embedding* of g in $\boldsymbol{\tau}$ (w.r.t. λ) is a function $\delta : \text{nodes}(g) \to \{1, \ldots, m\}$ that satisfies the following conditions.

(1) Labels match: $l \in \lambda(\boldsymbol{\tau}(\delta(l)))$ for all nodes l of g;
(2) Edges match: $\delta(l) < \delta(l')$ for all edges $l \to l'$ of g.

The pattern g is *embedded* in $\boldsymbol{\tau}$ (w.r.t. λ), denoted $(\boldsymbol{\tau}, \lambda) \models g$, if there is at least one embedding of g in $\boldsymbol{\tau}$. Equivalently, we can say that $\boldsymbol{\tau}$ *embeds* g. We denote by $\Delta(g, \boldsymbol{\tau})$ the set of all embeddings of g in the ranking $\boldsymbol{\tau}$. We take note of the difference between an embedding defined here, and the notion of *matching* defined in [26]. While an embedding maps the nodes of g to indexes in $\boldsymbol{\tau}$, a matching maps the nodes of g to specific items in $\boldsymbol{\sigma}$. This change in definition is crucial, and, as we will see, precisely what allows us to develop an algorithm with superior complexity guarantees.

We say that a function δ is *consistent with* g if, for every pair of labels $l, l' \in \text{nodes}(g)$, $\delta(l) < \delta(l')$ for all edges $l \to l'$ of g.

Example 2.2. Consider the model $\mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}, \Pi, \lambda)$ with reference ranking $\boldsymbol{\sigma} \stackrel{\text{def}}{=} \langle \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5 \rangle$. The labeling λ is given alongside the pattern g of Figure 2a: next to each label v the items with the corresponding label are mentioned (e.g., $\lambda(\sigma_3) = \{X, Y\}$). Consider the ranking $\boldsymbol{\tau}$ given by $\boldsymbol{\tau} \stackrel{\text{def}}{=} \langle \sigma_1, \sigma_5, \sigma_4, \sigma_3, \sigma_2 \rangle$. Then, $\Delta(g, \boldsymbol{\tau})$ contains the following embeddings:

- $\delta_1 \stackrel{\text{def}}{=} \{X \mapsto 1\ (\sigma_1), Y \mapsto 3\ (\sigma_4), Z \mapsto 5\ (\sigma_2)\}$
- $\delta_2 \stackrel{\text{def}}{=} \{X \mapsto 4\ (\sigma_3), Y \mapsto 4\ (\sigma_3), Z \mapsto 5\ (\sigma_2)\}$
- $\delta_3 \stackrel{\text{def}}{=} \{X \mapsto 1\ (\sigma_1), Y \mapsto 4\ (\sigma_3), Z \mapsto 5\ (\sigma_2)\}$

Note that in δ_2 labels X and Y are mapped to the same index (i.e., 4) corresponding to the item σ_3. □

The problem of *pattern embedding* on labeled RIM models is the following. Given a model $\mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}, \Pi, \lambda)$, and a pattern g, compute the probability that a ranking has an embedding of g; more formally, the goal is to compute $\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda)$ that we define as follows.

$$\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda) \stackrel{\text{def}}{=} \sum_{\substack{\boldsymbol{\tau} \in rnk(\text{items}(\boldsymbol{\sigma})) \\ (\boldsymbol{\tau}, \lambda) \models g}} \Pi_{\boldsymbol{\sigma}}(\boldsymbol{\tau}) \tag{1}$$

The challenge in computing (1) is that the number of possible rankings $\boldsymbol{\tau}$ to aggregate can be factorial in m. In the following section we present the improved algorithm for computing this probability.

Probabilistic Preference Databases. Jacob et al. [23] have proposed the concept of a *preference database*—a formalism for preference modeling and analysis within a general-purpose database framework. A preference database consists of two types of relations: ordinary relations (i.e., ones of ordinary relational databases) and *preference relations*. The two are abbreviated *o-relations* and *p-relations*, respectively. A p-relation represents a collection of orders (preferences), where each order is referred to as a *session*. A tuple of a p-relation has the form $(\mathbf{s}; \sigma; \tau)$, denoting that in the order $\succ_{\mathbf{s}}$ of the session $\mathbf{s}$ it is the case that $\sigma \succ_{\mathbf{s}} \tau$. Here, each of σ and τ is a one-dimensional value (representing an item), and $\mathbf{s}$ may have any arity. For example, the tuple (`Ann`, `Oct-5`; `Sanders`; `Clinton`) denotes that in the session of the voter `Ann` on `October 5th`, the candidate `Sanders` is preferred to the candidate `Clinton`.

The central aspect of a preference database is that the internal representation of a preference need not store every pair comparison $\sigma \succ_{\mathbf{s}} \tau$ explicitly; rather, the representation of the session can be compact, such as a vector of items in the case of a linear order. Taking a step forward, Kenig et al. [26] extended the framework to allow for representations of probabilistic preferences, and conducted a complexity analysis for query evaluation over RIM models. Formally, in a *RIM Probabilistic Preference Database*, abbreviated *RIM-PPD*, every p-relation is represented as a pair (r, μ), where r is a relation of sessions, and μ maps every tuple (session) $\mathbf{s}$ of r into a RIM model $\mu(\mathbf{s}) = \mathsf{RIM}(\boldsymbol{\sigma}, \Pi)$. We call (r, μ) a *RIM relation*.

Semantically, a RIM-PPD $\mathcal{D}$ represents a *probabilistic database* [48] where every random possible world (database) is produced by the generative process that replaces each RIM relation (r, μ) with a unique preference relation P, constructed as follows.

1: Initialize P as an empty relation.
2: **for all** sessions $\mathbf{s}$ in r **do**
3: produce a random ranking $\boldsymbol{\tau}$ from the RIM model $\mu(\mathbf{s})$
4: **for all** items σ and σ' where $\sigma \succ_{\boldsymbol{\tau}} \sigma'$ **do**
5: insert into P the tuple $(\mathbf{s}; \sigma; \sigma')$

In particular, observe that different sessions (in the same or different RIM relations) are treated as *probabilistically independent* events.

Query Evaluation. We adopt the standard semantics of probabilistic databases for query evaluation, where the goal is to compute the probability of each output tuple [6, 48]. More precisely, for a query Q over a RIM-PPD $\mathcal{D}$ and a possible output tuple $\mathbf{a}$, the probability of $\mathbf{a}$ is defined as $\Pr(\mathbf{a} \in Q(\mathcal{D}))$; that is, the probability that we get $\mathbf{a}$ as an answer when evaluating Q over a random possible world. Our goal is to compute the probability of all output tuples.

In this paper we focus on the evaluation of Conjunctive Queries (CQs), and specifically on the class of *itemwise CQs* [26], over RIM-PPDs with a single p-relation. Next, we recall the definition of this class. A CQ over a preference relation has two types of atomic predicates—the ones of o-relations, called *o-atoms*, and the ones over p-relations, called *p-atoms*. Consider, for example, the CQ

$$Q(z) \leftarrow R(x, z), P(s; x; y), P(s; y; z), S(s, x), S(s, y)\,. \tag{2}$$

Here, the first, fourth and fifth atomic predicates are o-atoms, and the second and third are p-atoms. Intuitively, an itemwise CQ applies to each session separately, and the o-atoms constrain each item individually, but the o-atoms do not constrain *relationships* among items. For example, the Boolean CQ $Q() \leftarrow R(x, z), S(z, y), P(s; x; y)$ is *not* itemwise, since the o-atoms $R(x, z)$ and $S(z, y)$ describe a connection between x and y. The query $Q() \leftarrow R(x, z), S(z', y), P(s; x; y)$

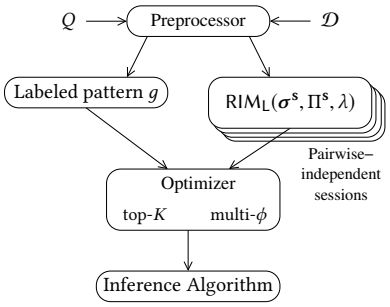


Figure 3: Architecture of the query engine

is itemwise. Nevertheless, we do allow connections through variables that occur in sessions or in the output. For example, in the CQ of (2) all connections go through the session variable s, or the output variable z. Therefore, the CQ is itemwise.

More formally, for a CQ Q over a preference database, a variable in an item attribute of a p-atom is referred to as an *item variable*, and a variable in a session attribute is referred to as a *session variable*. We say that Q is *itemwise* if both of the following hold.

(1) The session part of all p-atoms is the same (hence, there is no join across sessions).
(2) When removing from Q all of the p-atoms, session variables and output variables, no two item variables remain in the same connected component.

As an example, the CQ of (2) is itemwise. The session part of both p-atoms is s (hence, the first condition holds), and moreover, when removing from the query all of the p-atoms, session variables and output variables, we are left with the CQ $R(x), S(x), S(y)$, where no two item variables (or any two variables) are connected.

Kenig et al. [26] show that itemwise CQs are tractable (under data complexity). Moreover, in the absence of self joins, itemwise CQs are *precisely* the CQs that are computationally tractable over RIM-PPDs. That is, if Q is not itemwise and has no self joins, then its evaluation over RIM-PPDs is #P-hard.

3 QUERY ENGINE

In this section we describe a prototype implementation of a query engine that integrates inference over probabilistic preferences into a relational query processing framework. The prototype consists of a *query parser*, an *execution engine*, and a *relational database* (specifically PostgreSQL) to store and manage relational and preference data. Note that, although all data is stored in conventional relations, special processing of preference data is carried out by the query parser and execution engine, which we now describe in turn.

The query evaluation process is summarized in Figure 3, and each of its components is described below. Preprocessor receives an itemwise query Q with a single p-relation, represented by the pair (r, μ). Recall that r is a relation of sessions, and μ maps every session of r to the appropriate RIM model. Preprocessor extracts a labeled pattern g from Q, and a set of RIM models associated with the sessions of r. These are passed on to the Optimizer module, which applies multi-ϕ and top-K optimization techniques, and finally invokes the Inference Algorithm described in Section 4.

3.1 Preprocessor

We begin by describing how the evaluation of an itemwise CQ translates to instances of the labeled RIM matching. We follow the translation of Kenig et al. [26], where the reader can find the complete details. We consider the evaluation of an itemwise CQ Q over a RIM-PPD $\mathcal{D}$. Recall that Q uses one p-relation (possibly in multiple atoms). Let (r, μ) denote the instance associated to the preference symbol. This instance maps every session $\mathbf{s} \in r$ to the RIM model $\mathsf{RIM}(\boldsymbol{\sigma}^{\mathbf{s}}, \Pi^{\mathbf{s}})$. The assumption that different sessions in the same RIM relation are probabilistically independent allows us to focus on evaluating the query over a single RIM model $\mathsf{RIM}(\boldsymbol{\sigma}^{\mathbf{s}}, \Pi^{\mathbf{s}})$. For evaluating Q, we iterate over all possible output tuples, and compute their probability by translating Q into a Boolean CQ. Hence, below we will assume that Q is Boolean.

We map every term in an item attribute of the p-atoms in Q to a distinct label. A label l is associated with a database value σ if the constraints of the CQ can be met following the substitution of the terms labeled l with σ. Recall that the main feature of an itemwise query is that the set of ordinary atoms of the CQ define independent conditions on the individual item variables, and the connections between these variables are established only through the p-atoms. Therefore, the set of items that can substitute the label l, while meeting the constraints of the query, can be inferred individually for every label, solely by the ordinary relations connected to it.

We denote the set of labels associated with a database value σ by $\lambda(\sigma)$. The resulting labeled RIM model is $\mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}^{\mathbf{s}}, \Pi^{\mathbf{s}}, \lambda)$. Next, we construct a pattern g where nodes are the terms in the item positions of Q. The pattern g contains the edge $l_1 \rightarrow l_2$ for every p-atom of Q with l_1 and l_2 on the left and right attributes, respectively.

It follows from this construction that the probability of generating a possible world D, or specifically, an instance of a preference relation, that meets the constraints of the query, is equivalent to computing $\Pr(g \mid \boldsymbol{\sigma}^{\mathbf{s}}, \Pi^{\mathbf{s}}, \lambda)$, the probability of generating a ranking from $\mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}^{\mathbf{s}}, \Pi^{\mathbf{s}}, \lambda)$ that has an embedding of g.

For illustration, we will use the RIM-PPD instance in Figure 1, which is inspired by the running example of [26]. Here, **Candidates** and **Voters** are o-relations, while **Polls** is a p-relation. The instance **Polls** associates a session, identified by a (`voter`, `date`) pair, with a labeled Mallows model $\mathsf{MAL}_{\mathsf{L}}(\boldsymbol{\sigma}, \phi, \lambda)$. Consider the query Q:

$$Q(edu) \leftarrow P(v, \_; c_1; c_2), V(v, sex, \_, edu), \\ C(c_1, \_, sex, \_, \_, \_), C(c_2, \_, \_, \_, edu, \_)$$

The preprocessor takes as input a CQ Q and a RIM-PPD $\mathcal{D}$, and generates a sequence of *inference requests* $(g_1, \mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}_1, \Pi_1, \lambda_1)), \ldots (g_n, \mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}_n, \Pi_n, \lambda_n))$, where g_i is a label pattern and each $\mathsf{RIM}_{\mathsf{L}}(\boldsymbol{\sigma}_i, \Pi_i, \lambda_i)$ is the labeled RIM model of a particular session. This query identifies pairs of candidates c_1, c_2 such that some voter v has the same gender as c_1 and the same education level as c_2, and prefers c_1 to c_2. The output of Q contains one tuple per value edu, along with its probability. In our example in Figure 1, the head variable is $edu \in \{\mathtt{BS}, \mathtt{MS}, \mathtt{JD}\}$. To evaluate Q, we instantiate three Boolean queries, one for each binding of the head variable. Consider $edu = \mathtt{BS}$, supported by the sessions corresponding to `Ann` and `Bob`.

- For `Ann` we construct the pattern g_1 with the label c_1, assigned to `Clinton` (that is, $c_1 \in \lambda_{\mathtt{Ann}}(\mathtt{Clinton})$ since `Clinton`

has the same gender as Ann), preferred to the label c_2, assigned to Trump and Sanders (the same education as Ann).
- Similarly, for Bob we construct the pattern g_2 with the label c_1, assigned to Trump, Sanders, and Rubio, preferred to c_2, assigned to Trump and Sanders.

Finally, we retrieve preference models $\mathsf{MAL}(\boldsymbol{\sigma}_1, \phi_1)$ for session (Ann, Oct-5) and $\mathsf{MAL}(\boldsymbol{\sigma}_2, \phi_2)$ for session (Bob, Oct-5) from **Polls**, and generate the corresponding labeled models $\mathsf{MAL}_\mathsf{L}(\boldsymbol{\sigma}_1, \phi_1, \lambda_{\mathsf{Ann}})$, and $\mathsf{MAL}_\mathsf{L}(\boldsymbol{\sigma}_2, \phi_2, \lambda_{\mathsf{Bob}})$. Following that, the system generates two inference requests containing the generated patterns and labeled Mallows models, and passes them to the Optimizer, described next.

3.2 Optimizer

This module groups and orders inference requests, and then invokes an inference algorithm (such as LTM, described in Section 4) on each group of requests. Optimizer is also responsible for reconciling the probabilities returned by each invocation of the inference algorithm to compute the final probability of each tuple in the result of Q. Suppose that inference requests for (Ann, Oct-5) and (Bob, Oct-5) return probabilities p_1 and p_2, respectively. By the session independence assumption, the probability of $Q(\mathsf{BS}) = 1 - (1 - p_1)(1 - p_2)$, and similarly for $Q(\mathsf{MS})$ and $Q(\mathsf{JD})$.

Multi-ϕ optimization. Inference requests $(g_i, \mathsf{RIM}_\mathsf{L}(\boldsymbol{\sigma}_i, \Pi_i, \lambda_i))$ and $(g_j, \mathsf{RIM}_\mathsf{L}(\boldsymbol{\sigma}_j, \Pi_j, \lambda_j))$ are *identical* if $g_i = g_j$, $\boldsymbol{\sigma}_i = \boldsymbol{\sigma}_j$ and $\Pi_i = \Pi_j$. The execution engine will invoke the inference algorithm once for every set of identical requests, and will keep track of their multiplicity to correctly compute the final probability.

In the special case of Mallows, requests $(g_i, \mathsf{MAL}_\mathsf{L}(\boldsymbol{\sigma}_i, \phi_i, \lambda_i))$ and $(g_j, \mathsf{MAL}_\mathsf{L}(\boldsymbol{\sigma}_j, \phi_j, \lambda_j))$, where MAL_L is defined similarly to RIM_L, are *identical up to* ϕ, if $g_i = g_j$, $\boldsymbol{\sigma}_i = \boldsymbol{\sigma}_j$ and $\phi_i \neq \phi_j$. The execution engine will invoke the inference algorithm once for every such set of requests, passing in a *list* of dispersion parameters ϕ to be processed in a batch. We refer to this as the *multi-ϕ optimization*, and will demonstrate that it improves performance in Section 5.3.

Top-K Sessions. Our query engine supports top-K queries that return, for a given CQ Q and a given integer K, the set of sessions that have the highest probability of generating a ranking that satisfies Q. We implement two methods for evaluating these queries. The first is straightforward: evaluate Q, generating and invoking an inference request for each session, and then selecting the top-K. The second method uses an upper-bound optimization. The basic idea is to first quickly compute an upper bound of the probability for each session. To do so, we remove one or more labels from a label pattern, making the pattern less restrictive. The probability of a pattern with one or several labels removed is no lower than the probability of the original pattern. Because the resulting patterns are smaller, their probabilities are also faster to compute, as we will demonstrate in Section 5.2. In our implementation we greedily remove a label that has the highest number of candidate items.

Having computed an upper bound on the probability for each session, we then sort sessions in decreasing order of this upper bound. Finally, we compute the exact probability for sessions in sorted order, stopping once (a) the probability for K sessions is known and (b) the upper bound for the next session is no higher than the K^{th} highest exact probability computed so far.

4 LIFTED TOP MATCHING (LTM)

While the TM algorithm [26] of Kenig et al. has polynomial time data-complexity (when the pattern is considered fixed), it is highly impractical in practice, as reflected in our experimental analysis. In what follows, we present a novel inference algorithm for labeled RIM that carries two major contributions. First, the new algorithm has superior complexity guarantees (i.e., is more efficient asymptotically). Specifically, if we let L denote the number of item variables in the query, then the complexity of TM is $O(m^{2L})$, while that of LTM is $O(2^L m^L)$. Second, the superior complexity guarantees translate to dramatic performance improvements in practice. Notably, in our experimental analysis we show that evaluating itemwise CQs with the new inference algorithm results in a considerable improvement in performance, when compared to TM.

We begin with an overview of the proposed approach, then highlight the differences from TM [26], and then give the details of LTM. For the sake of readability, and convenient comparison with the original algorithm for evaluating itemwise CQs, we adopt the notation of Kenig et al. [26]. Proofs are deferred to the appendix.

Strategy. In this section we fix a model $\mathsf{RIM}_\mathsf{L}(\boldsymbol{\sigma}, \Pi, \lambda)$ and a pattern g where $\boldsymbol{\sigma} = \langle \sigma_1, \ldots, \sigma_m \rangle$. We devise an algorithm for computing $\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda)$, as defined in (1). We assume that g is acyclic and all nodes of g are in the image of λ; otherwise, $\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda) = 0$. Let l be a node of g. A *parent* of l is a node l' such that g has the edge $l' \rightarrow l$. We denote by $pa_g(l)$ the set of all parents of l.

Naturally, the challenge in computing $\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda)$ is that the number of rankings in our space (and in particular, those that have an embedding of g) can be factorial in m. We overcome this challenge by *partitioning* the space of all rankings $\boldsymbol{\tau} \in rnk(\boldsymbol{\sigma})$ into a polynomial number of pairwise-disjoint subspaces, using the notion of a *top embedding*. We compute the probability of each partition. Since the number of subspaces is polynomial in m, we arrive at an algorithm with polynomial time data complexity.

Let $\boldsymbol{\tau}$ be a random ranking. We recall the notion of an embedding of a graph pattern g in $\boldsymbol{\tau}$. Let δ_1 and δ_2 be two embeddings of g in $\boldsymbol{\tau}$. We denote by $\delta_1 \succeq_{\boldsymbol{\tau}} \delta_2$ the fact that $\delta_1(l) \leq \delta_2(l)$ for all nodes $l \in \text{nodes}(g)$. An embedding $\delta \in \Delta(g, \boldsymbol{\tau})$ is said to be a *top embedding of g in $\boldsymbol{\tau}$* if $\delta \succeq_{\boldsymbol{\tau}} \delta'$ for all embeddings $\delta' \in \Delta(g, \boldsymbol{\tau})$. Lemma 4.2 below shows that if there is any embedding of g in $\boldsymbol{\tau}$, then there is a *unique* top embedding of g in $\boldsymbol{\tau}$.

Example 4.1. Consider again the RIM model $\mathsf{RIM}_\mathsf{L}(\boldsymbol{\sigma}, \Pi, \lambda)$, pattern g, and the ranking $\boldsymbol{\tau} = \langle \sigma_1, \sigma_5, \sigma_4, \sigma_3, \sigma_2 \rangle$ from Example 2.2. The reader may verify that the top embedding of g in $\boldsymbol{\tau}$ is the following: $\delta_1 = \{X \mapsto 1\ (\sigma_1), Y \mapsto 3\ (\sigma_4), Z \mapsto 5\ (\sigma_2)\}$. □

Lemma 4.2. *For all random rankings $\boldsymbol{\tau}$, if $(\boldsymbol{\tau}, \lambda) \models g$ then there is precisely one top embedding of g in $\boldsymbol{\tau}$.*

Intuitively, Lemma 4.2 allows us to partition the set of rankings that have an embedding of g, according to their top embedding. This enables us to reduce the inference task to the one of calculating the probability of the set of possible top embeddings.

Let $\delta : \text{nodes}(g) \rightarrow \{1, \ldots, m\}$ be a function. We denote by $img(\delta)$ the set of positions $\{\delta(l) \mid l \in \text{nodes}(g)\}$. We say that a ranking $\boldsymbol{\tau}$ *realizes* δ if δ is a top embedding of g in $\boldsymbol{\tau}$. We denote by $rnk(\boldsymbol{\sigma}, \delta)$ the set of all rankings $\boldsymbol{\tau} \in rnk(\text{items}(\boldsymbol{\sigma}))$ that realize δ.

By Lemma 4.2 every ranking realizes a single function, therefore:

$$\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda) = \sum_{\substack{\delta:\text{nodes}(g)\rightarrow \\ \{1,\ldots,m\}}} \sum_{\substack{\boldsymbol{\tau} \in \\ rnk(\boldsymbol{\sigma},\delta)}} \Pi_{\boldsymbol{\sigma}}(\boldsymbol{\tau}) \,. \tag{3}$$

We will show how to efficiently compute $p(\delta)$, the probability of generating a ranking that realizes δ, for every function from nodes(g) to $\{1, \ldots, m\}$. Formally:

$$p(\delta) \stackrel{\text{def}}{=} \sum_{\boldsymbol{\tau} \in rnk(\boldsymbol{\sigma},\delta)} \Pi_{\boldsymbol{\sigma}}(\boldsymbol{\tau}) \tag{4}$$

We denote by $\mathcal{R}$ the set of all embeddings $\delta : \text{nodes}(g) \mapsto \{1, \ldots, m\}$. From (3), and (4) we conclude that:

$$\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda) = \sum_{\delta \in \mathcal{R}} p(\delta) \,. \tag{5}$$

The algorithm is based on the following lemma that characterizes the conditions under which a function δ is a top embedding.

Lemma 4.3. *Let $\boldsymbol{\tau} \in rnk(\boldsymbol{\sigma})$ such that $(\boldsymbol{\tau}, \lambda) \models g$. Then $\boldsymbol{\tau}$ realizes the mapping $\delta : \text{nodes}(g) \mapsto \{1, \ldots, m\}$ if and only if for every $l \in \text{nodes}(g)$ and item $\sigma \in \{\tau_i : i < \delta(l)\}$ at least one of the following conditions is met.* (a) *The item σ is not associated with l. That is, $l \notin \lambda(\sigma)$.* (b) *$pa_g(l) \neq \emptyset$ and $\boldsymbol{\tau}^{-1}(\sigma) < \max_{u \in pa_g(l)} \delta(u)$.*

Example 4.4. Consider the model $\text{RIM}_\text{L}(\boldsymbol{\sigma}, \Pi, \lambda)$, where $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_4\}$, g is the label pattern of Figure 2b, and the mapping λ is given by $\lambda(\sigma_1) = \{l_1, l_2\}$, $\lambda(\sigma_2) = \{l_1\}$, $\lambda(\sigma_3) = \{l_2, l_3\}$, and $\lambda(\sigma_4) = \{l_3\}$. Consider any embedding δ such that $\delta(l_1) < \delta(l_2) < \delta(l_3)$. For example: $\delta(l_1) = 1, \delta(l_2) = 3, \delta(l_3) = 4$. Now, consider the ranking $\boldsymbol{\tau} = \langle \sigma_1, \sigma_2, \sigma_3, \sigma_4 \rangle$. By Lemma 4.3, δ is not a top embedding for $\boldsymbol{\tau}$. Specifically, item σ_1, which is positioned before $\delta(l_2) = 3$ in $\boldsymbol{\tau}$, is associated with label l_2 (i.e., $l_2 \in \lambda(\sigma_1)$), and $pa_g(l_2) = \emptyset$. Indeed, the mapping δ', given by $\delta'(l_1) = 1$, $\delta'(l_2) = 1$, and $\delta'(l_3) = 3$, is the top embedding of g in $\boldsymbol{\tau}$, so $\boldsymbol{\tau}$ realizes δ'. Furthermore, by Lemma 4.2, δ' is the unique top embedding of g in $\boldsymbol{\tau}$.

Comparison with Top Matching. Let $\text{RIM}_\text{L}(\boldsymbol{\sigma}, \Pi, \lambda)$ denote a RIM model with reference ranking $\boldsymbol{\sigma} = \langle \sigma_1, \ldots, \sigma_m \rangle$, and let g denote a pattern with L nodes. Kenig et al. [26] use the notion of a *top matching* to partition the space of rankings, as follows. Similar to the notion of embedding, a matching of g in a ranking $\boldsymbol{\tau}$ is a mapping γ from the nodes of g to the *items* in $\boldsymbol{\tau}$ such that *(1)* labels match: $l \in \lambda(\gamma(l))$, *and (2)* edges match: for all edges $l \rightarrow l'$ of g it holds that $\boldsymbol{\tau}^{-1}(\gamma(l)) < \boldsymbol{\tau}^{-1}(\gamma(l'))$, where $\boldsymbol{\tau}^{-1}(a)$ is the position of item a in $\boldsymbol{\tau}$. Given two matchings γ and γ' of g in $\boldsymbol{\tau}$, we denote by $\gamma \succeq_{\boldsymbol{\tau}} \gamma'$ the fact that $\gamma(l) \succeq_{\boldsymbol{\tau}} \gamma'(l)$ for all nodes l of g. A matching γ is a *top matching* of g in $\boldsymbol{\tau}$ if $\gamma \succeq_{\boldsymbol{\tau}} \gamma'$ for all matchings γ' of g in $\boldsymbol{\tau}$. Every ranking that is consistent with g has a single top matching. Therefore, top matchings can be used to partition the space of rankings that have a matching in g.

TM iterates over all possible mappings γ from the nodes of g to the items of $\boldsymbol{\sigma}$, and calculates the probability of generating a ranking whose top matching is γ. For the latter, TM enumerates the space of possible mappings ν from the set of items in the range of γ to their positions in the ranking. Overall, TM considers the space of pairs $\langle \gamma, \nu \rangle$ where γ is a top matching and ν is the positioning of the items, that belong to the range of γ. This leads to a state space of size $O(m^{2L})$ where m is the size of $\boldsymbol{\sigma}$, and $L = |\text{nodes}(g)|$.

LTM, on the other hand, considers only the *positions* of the items that constitute the top matching, *disregarding the specific items in these positions*. Basically, we are able to avoid the expensive step of enumerating the set of top matchings. The main idea behind the algorithm is that at every iteration t of the RIM process, we only require the following information: (1) the positions of the items constituting the top matching, and (2) which one of these items has an index (in the reference ranking $\boldsymbol{\sigma}$) that is less than t. This leads to a significantly smaller state space, which is reflected in both the complexity guarantees, and in the performance gains that are evident from the experimental analysis.

Lifted Top Matching. Algorithm LTM follows the insertion process of RIM, but starts with a nonempty ranking. The modified RIM starts with a subranking $\mathbf{b} = \langle b_1, \ldots, b_k \rangle$ of *placeholder items*, disjoint from items($\boldsymbol{\sigma}$). Every item $b \in \mathbf{b}$ represents a set of nodes (or labels) denoted $\lambda(b)$. The items of $\mathbf{b}$ hold the positions that will eventually (i.e., at the end of the modified RIM process) make up the top embedding of g inside the generated ranking.

The item positions that constitute the top embedding are maintained as mappings $\delta : \text{nodes}(g) \mapsto \{1, \ldots, m\}$. During the algorithm execution, the placeholder items of $\mathbf{b}$ are replaced by "real" items from $\boldsymbol{\sigma}$, while maintaining the invariant that δ is a top embedding for the prefix generated by the RIM process. By maintaining this invariant, at the end of the algorithm, every consistent mapping δ is associated with its probability $p(\delta)$ (see (4)).

Outline. We begin this section with a formal definition of *valid* placeholder rankings. Only valid placeholder rankings can lead (via insertions of items from $\boldsymbol{\sigma}$) to a ranking consistent with g. We then describe the mechanism by which we apply the conditions of Lemma 4.3 during the insertion process, thus maintaining the invariant that the mapping δ is a top embedding. Next, we explain how to update the insertion function Π to consider the placeholder items that are not part of the reference ranking $\boldsymbol{\sigma}$. The challenge here lies in proving that despite the fact that our algorithm uses an insertion function different than the original, the computed probability $p(\delta)$ corresponds to the original RIM distribution. This proof is quite involved and therefore deferred to the Appendix. Finally, we describe the algorithm pseudo code followed by its analysis, and an additional optimization.

Algorithm Initialization: Placeholder Subrankings. Every placeholder $b \in \mathbf{b}$ represents a set of labels $\lambda(b)$. The pattern induced by the initial subranking $\mathbf{b} = \langle b_1, \ldots, b_k \rangle$, is the graph $g^{\mathbf{b}}(L, E \cup E^{\mathbf{b}})$, where $E^{\mathbf{b}} = \{l_1 \rightarrow l_2 : l_1 \in \lambda(b_i), l_2 \in \lambda(b_j), i < j\}$. We can assume that $g^{\mathbf{b}}$ is acyclic. Otherwise, the precedence constraints induced by $E^{\mathbf{b}}$ are inconsistent with those of g. Thus, the subranking represented by $\mathbf{b}$ cannot be an embedding of g for any ranking.

Definition 1 (valid placeholder ranking). *A subranking $\mathbf{b} = \langle b_1, \ldots, b_k \rangle$ of placeholder items is* valid *if:* (a) *For every $1 \leq i < j \leq k$ the placeholders b_i, and b_j represent disjoint sets of labels. That is, $\lambda(b_i) \cap \lambda(b_j) = \emptyset$;* (b) *The set of placeholders represent an exhaustive set of labels. Formally,* $\text{nodes}(g) = \cup_{i=1}^{k} \lambda(b_i)$*;* (c) *For every $b \in \text{items}(\mathbf{b})$ the set of labels in $\lambda(b)$ are incomparable in g. That is, for every pair of labels $l_1, l_2 \in \lambda(b)$ there is no directed path from l_1 to l_2 in g.* and (d) *The label pattern $g^{\mathbf{b}}(L, E \cup E^{\mathbf{b}})$ induced by $\mathbf{b}$ is acyclic.*

Example 4.5. Consider the label pattern of Figure 2b. We consider a few possible placeholder rankings (Def. 1). The first is $\mathbf{b}_1 = \langle b_1, b_2 \rangle$ where $\lambda(b_1) = \{l_1, l_2\}$, and $\lambda(b_2) = \{l_3\}$. There is no directed path between l_1 and l_2, and the ranking does not induce any new edges. Since $\mathbf{b}_1$ meets all of the conditions of the definition, it is a valid placeholder ranking. Another valid placeholder ranking is $\mathbf{b}_2 = \langle b_1, b_2, b_3 \rangle$ where $\lambda(b_i) = \{l_i\}$ for $i \in \{1, 2, 3\}$. Another one that is valid is $\mathbf{b}_3 = \langle b_2, b_1, b_3 \rangle$ where $\lambda(b_i) = \{l_i\}$ for $i \in \{1, 2, 3\}$.

The algorithm begins with the complete set of valid placeholder rankings (Def. 1), each represented by a mapping $\delta : \text{nodes}(g) \mapsto \{1, \ldots, k\}$. We denote this set of initial mappings by $\mathcal{R}_0$. For every $\delta \in \mathcal{R}_0$, and every label $l \in \text{nodes}(g)$, we have that $\delta(l) = i$ if l is associated with placeholder b_i (i.e., $l \in \lambda(b_i)$). Note that placeholder items correspond to positions in $img(\delta)$. Since we assume that the placeholder ranking is valid, then every label is mapped to exactly one position (item 1 of Def. 1), making δ well defined.

Execution: Inserting Items. The algorithm iterates through the items of $\boldsymbol{\sigma}$ in order, from σ_1 to σ_m, and at each iteration t inserts σ_t into a prefix that contains $\{\sigma_1, \ldots, \sigma_{t-1}\}$ along with the placeholder items that have not yet been replaced by items from $\boldsymbol{\sigma}$. The positions of the prefix that constitute its top embedding are maintained by δ.

Inserting σ_t into the prefix can take one of two forms. The first is that σ_t replaces a placeholder item b. Assume that b is mapped, via the function δ, to position j. Item σ_t can replace b if $\lambda(\sigma_t) \supseteq \lambda(b)$, and its insertion into position j meets the conditions of Lemma 4.3, with respect to the mapping δ. Following the insertion of σ_t into position j, we say that the set of labels $\lambda(b)$ have been *assigned*.

In the second case, item σ_t does not replace any placeholder item. That is, it is inserted into a position j that is not part of $img(\delta)$, resulting in a new mapping δ_{+j} defined as follows. For every label $l \in \text{nodes}(g)$ such that $\delta(l) < j$, then $\delta_{+j}(l) = \delta(l)$. Otherwise, if $\delta(l) \geq j$, then $\delta_{+j}(l) = \delta(l) + 1$. Position j is legal for σ_t if it meets the conditions of Lemma 4.3 with respect to the mapping δ_{+j}.

Throughout the execution, we keep track of the set of placeholder items that have been replaced. We do this by associating with every mapping δ a mapping $v : img(\delta) \mapsto \{0, 1\}$ that represents the status of placeholders. That is, $v(j) = 1$ if the placeholder item b, at position j, has been replaced, and 0 otherwise. We denote by $v^{-1}(0)$ the indices of v, or placeholder items, that have not yet been replaced by items from $\boldsymbol{\sigma}$, and the cardinality of this set by $|v^{-1}(0)|$.

Example 4.6. Consider the label pattern of Figure 2b and the placeholder ranking $\mathbf{b}_1 = \langle b_1, b_2 \rangle$ with $\lambda(b_1) = \{l_1, l_2\}$, and $\lambda(b_2) = \{l_3\}$. Here, $\lambda(\sigma_1) = \{l_1, l_2\} \supseteq \lambda(b_1)$. Hence, in the first iteration, item σ_1 can replace the placeholder b_1, and is assigned to labels l_1 and l_2. At this point, δ is associated with the subranking $\langle \sigma_1, b_2 \rangle$. □

Events $\langle \delta, v \rangle$. For every iteration $t \in \{0, \ldots, m\}$, we denote by $\mathcal{R}_t$ the set of pairs $\langle \delta, v \rangle$, where δ is a mapping consistent with g, and v is a binary vector indicating the state of the placeholder items. By slight abuse of notation, we use $\langle \delta, v \rangle$ to denote the event of generating a ranking $\boldsymbol{\tau} \in rnk(\{\sigma_1, \ldots, \sigma_t\})$, from the distribution $\mathbf{RIM}(\boldsymbol{\sigma}, \Pi)$, such that inserting placeholder items into the positions $\boldsymbol{s} = \{j \in img(\delta) \mid v(j) = 0\}$ of $\boldsymbol{\tau}$, results in a prefix that realizes δ.

Example 4.7. Consider the pattern g of Figure 2a, and the pair $\langle \delta, v \rangle \in \mathcal{R}_3$ defined as follows. $\delta = \{X \mapsto 2, Y \mapsto 1, Z \mapsto 4\}$, and $v = \{2 \mapsto 1, 1 \mapsto 0, 4 \mapsto 0\}$ (note that the domain of v is $img(\delta)$). The ranking $\boldsymbol{\tau} = \langle \sigma_3, \sigma_1, \sigma_2 \rangle$ is an event of type $\langle \delta, v \rangle$ because the prefix $\boldsymbol{\tau}' = \langle b_1, \sigma_3, \sigma_1, b_4, \sigma_2 \rangle$ that results from inserting placeholders b_1 and b_4 into $\boldsymbol{\tau}$, realizes δ (i.e., δ is a top embedding for $\boldsymbol{\tau}'$). To see this, note that b_1 (representing Y), σ_3 (representing X), and b_4 (representing Z) form a top embedding for g in $\boldsymbol{\tau}'$. □

Algorithm $\text{LTM}(\text{RIM}_{\text{L}}(\boldsymbol{\sigma}, \Pi, \lambda), g)$

for $i = 1, \ldots, m$ **do**
  for all $\langle \delta, v \rangle \in \mathcal{R}_{i-1}$ **do**
    $\langle \text{range}_r, \text{range}_{ib} \rangle := \text{Range}(\sigma_i, \langle \delta, v \rangle, g)$
    for all $j \in \text{range}_r \cup \text{range}_{ib}$ **do**
      $v' := v$
      if $j \in \text{range}_r$ **then**
        $\delta' := \delta$
        $v'[j] := 1$
      else
        $\delta' := \delta_{+j}$
      $q_i(\langle \delta', v' \rangle) \mathrel{+}= q_{i-1}(\langle \delta, v \rangle) \times \Pi'(i, j, \langle \delta, v \rangle)$
      $\mathcal{R}_i := \mathcal{R}_i \cup \{\langle \delta', v' \rangle\}$
return $\sum_{\{\langle \delta, v \rangle \in \mathcal{R}_m \mid\mid v^{-1}(0)|=0\}} q_m(\langle \delta, v \rangle)$

Subroutine $\text{Range}(\sigma_i, \langle \delta, v \rangle, g)$

$\text{range}_r := \{k \in img(\delta) \mid v(k) = 0\}$
$\text{range}_{ib} := \{1, \ldots, i + Z(v)\} \setminus \text{range}_r$
for $k \in \text{range}_r$ **do**
  $L_k := \{l \in \text{nodes}(g) \mid \delta(l) = k\}$
  if $\lambda(\sigma_i) \supseteq L_k$ **then**
    for $\{l \in \text{nodes}(g) \mid k < \delta(l)$ AND $l \in \lambda(\sigma_i)\}$ **do**
      if $pa_g(l) = \emptyset$ or $k \geq \max_{u \in pa_g(l)} \delta(u)$ **then**
        $\text{range}_r := \text{range}_r \setminus \{k\}$
        break
  else
    $\text{range}_r := \text{range}_r \setminus \{k\}$
for all $\{l \in \lambda(\sigma_i)\}$ **do**
  $k := 0$
  if $pa_g(l) \neq \emptyset$ **then**
    $k := \max_{u \in pa_g(l)} \delta(u)$
  $\text{range}_{ib} := \text{range}_{ib} \setminus \{k+1, \ldots, \delta(l)\}$
return $\langle \text{range}_r, \text{range}_{ib} \rangle$

Figure 4: An algorithm for computing $\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda)$

So, consider a pair $\langle \delta, v \rangle$ in $\mathcal{R}_t$. We describe how to compute the probability $p(\langle \delta, v \rangle)$. Recall that $|v^{-1}(0)|$ is the number of zero bits in v. Since every unset bit in v corresponds to a placeholder item, the pair $\langle \delta, v \rangle$ represents a prefix that contains precisely $t + |v^{-1}(0)|$ items: $\sigma_1, \ldots, \sigma_t$ from $\boldsymbol{\sigma}$, and the placeholder items that have not yet been replaced. Furthermore, every placeholder that is part of the prefix at time t can be replaced only by an item whose index in $\boldsymbol{\sigma}$ is strictly greater than t. This allows us to define the insertion function Π' pertaining to the extended prefix.

$$\Pi'(t, j, \langle \delta, v \rangle) = \Pi(t, j - |\{k < j \mid k \in img(\delta), v(k) = 0\}|) \quad (6)$$

Example 4.8. Consider a RIM model with labels l_1, l_2, l_3. Let $\langle \delta, v \rangle \in \mathcal{R}_5$ where $\delta(l_1) = 1$,$\delta(l_2) = 3$, and $\delta(l_3) = 4$. Assume that $v(1) = 0$, and $v(3) = v(4) = 1$. Then $\Pi'(6, 2, \langle \delta, v \rangle) = \Pi(6, 1)$. □

From Modified RIM to Complete State Space Generation. Recall that our goal is to compute $\Pr(g \mid \boldsymbol{\sigma}, \Pi, \lambda)$, which amounts to generating all possible mappings, and calculating their probability (see (5)). Therefore, at every iteration $t \in \{1, \ldots, m\}$, the algorithm examines *all* combinations of mappings $\delta \in \mathcal{R}_{t-1}$, and insertion positions j that are *legal* for σ_t, according to the characterization of Lemma 4.3, with respect to δ. For every combination, the algorithm generates a new mapping $\delta' \in \mathcal{R}_t$, and computes its probability. A central property of the algorithm, and one that is used to prove its correctness (see Section B of the appendix) is that in the last iteration (i.e., $t = m$), the event $\langle \delta, v \rangle$ where $|v^{-1}(0)| = 0$, and the event δ, as defined in (4), are unified. That is $p(\langle \delta, v \rangle) = p(\delta)$.

Pseudo code. The algorithm pseudo code is presented in Figure 4. We assume that the set $\mathcal{R}_0$ contains all pairs $\langle \delta, v \rangle$, where all bits in v are set to zero (since no placeholder has been replaced), and the mapping δ represents the initial placeholder ranking.

The subroutine Range receives an item σ_i, a pair $\langle \delta, v \rangle \in \mathcal{R}_{i-1}$, and the pattern g, and returns two sets of indexes denoted range_r, and range_{ib}. Both sets contain legal positions for item σ_i with respect to δ, according to the characterization of Lemma 4.3. In addition, the set range_r contains placeholder positions j such that $v(j)$ is defined (i.e., $j \in img(\delta)$), and $v(j) = 0$. The set range_{ib} contains non-placeholder positions.

At each iteration $i \in \{1, \ldots, m\}$, LTM generates the pairs $\mathcal{R}_i$ by considering every combination of $\langle \delta, v \rangle \in \mathcal{R}_{i-1}$ and legal insertion position j for σ_i (lines 2 and 4). If item σ_i replaces some placeholder (i.e., $j \in \text{range}_r$, line 6) then the mapping remains unchanged (i.e., $\delta' = \delta$), and we set the appropriate bit in v (lines 7 and 8), indicating that the placeholder item has been replaced. Otherwise, $j \in \text{range}_{ib}$, and then no placeholder was replaced, thus $v' = v$. We do, however, need to update the mapping following the insertion of σ_i into position j (line 10). Once we have generated the pair $\langle \delta', v' \rangle \in \mathcal{R}_i$, its probability is updated in line 11 and it is added to $\mathcal{R}_i$. Once the set $\mathcal{R}_m$ has been generated, we aggregate the probabilities of all pairs $\langle \delta, v \rangle \in \mathcal{R}_m$ for which all placeholders have been replaced.

Analysis. Consider the size of the set $\mathcal{R}_t$, for $t > 0$. The number of possible mappings δ is in $O(m^q)$, and the number of possible binary vectors v is in $O(2^q)$. The overall complexity of the algorithm is $O((2m)^q)$, a significant improvement when compared to the TM algorithm of Kenig et al. [26] whose complexity is in $O(m^{2q})$.

Optimization. LTM not only yields superior complexity guarantees, but also enables introducing an optimization that leads to significant performance gains in practice. The idea is that at each iteration $t \in \{1, \ldots, m\}$, we prune the set of pairs in $\mathcal{R}_t$ that will not lead to the generation of a pair $\langle \delta, v \rangle \in \mathcal{R}_m$ such that $|v^{-1}(0)| = 0$ (see line 13 of LTM). Specifically, consider a placeholder b, and let t be the largest index such that $\lambda(\sigma_t) \supseteq \lambda(b)$. Clearly, if σ_t does not replace b at iteration t, then no other item in $\{\sigma_{t+1}, \ldots, \sigma_m\}$ can replace it. Therefore, any pair $\langle \delta, v \rangle$ that results from inserting σ_t into a position different than that of b, can be pruned.

5 EXPERIMENTAL EVALUATION

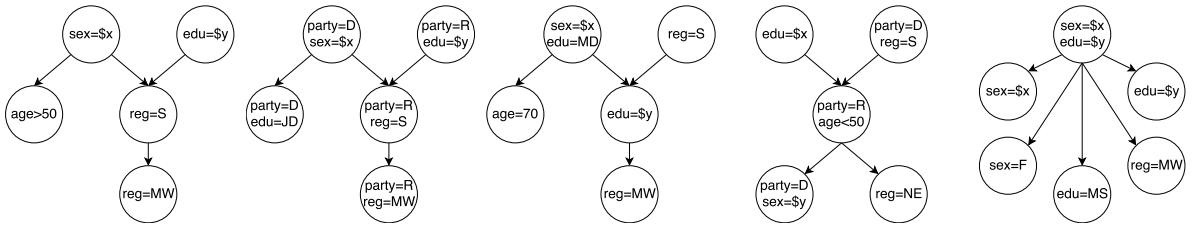
We evaluated the performance of our query engine for probabilistic preferences using a PostgreSQL 9.5.9 database. The *Top Matching* (TM) algorithm of [26], which we implemented based on its published description, our novel *Lifted Top Matching* (LTM) algorithm (Section 4), and the Preprocessor (Section 3.1) and Optimizer (Section 3.2) modules of our system were all implemented in Java 8. All exeperiments were executed on an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, with 512GB of RAM, and with 48 cores on 4 chips, running 64-bit Ubuntu Linux. We report most experimental results on 48 cores. In several experiments, we limit the number of cores to as few as four, to show that our methods scale on commodity hardware. These results are presented in the Appendix.

5.1 Experimental Dataset and Workload

Datasets. We experiment with real and synthetic datasets. Our first real dataset is drawn from MovieLens (www.grouplens.org). In line with prior work [36], we use 200 (out of roughly 3900) most frequently rated MovieLens movies, and the ratings of the 5980 users (out of roughly 6000) who rated at least one of these. Integer ratings from 1 to 5 are converted to pairwise preferences in the obvious way, and no preference is added for ties. We mined a mixture of 16 Mallows models $\mathbf{MAL}_1(\boldsymbol{\sigma}_1, \phi_1), \ldots, \mathbf{MAL}_{16}(\boldsymbol{\sigma}_{16}, \phi_{16})$ from MovieLens using publicly-available tools [45]. Movies have associated information such as genre and director, but voter demographics are unavailable. In our experiments, we use Relation **Movies** (movie, genre, sex, age), that stores movie attributes title and genre, and the sex and age group of the lead actor.

Our second real dataset is CrowdRank [46], which consists of movie rankings collected with Amazon Mechanical Turk. The dataset includes 50 Human Intelligence Tasks (HITs), with 20 movies and 100 users per HIT. We mined a mixture of Mallows from each HIT using publicly-available tools [45], and selected a HIT with the highest number of Mallows—seven reference rankings and three ϕ values per ranking, for a total of 21 distinct models. CrowdRank includes movie attributes and user demographics. We used a publicly-available tool [41] to generate synthetic user profiles that are statistically similar to those of the original 100 users. We treated a user's preference model as a categorical variable, and generated 200,000 user profiles (sessions) in which demographics correlated with preference model assignment. We use CrowdRank to show scalability in the number of sessions.

We also evaluated performance over a synthetic database that models political candidates, voters and polls in the 2016 US presidential election. We generated this database following the running example of [26]. The database schema and some of the tuples are in Figure 1. We generated an instance as follows. We inserted tuples into **Candidates** and **Voters**, generating each tuple independently, and drawing values for each attribute independently at random. Attributes party and sex have cardinality 2, reg (geographic region) has cardinality 5, while edu and *age* both have cardinality 6. For age, we assigned values between 20 and 70 in increments of 10, with each value denoting the start of a 10-year age bracket: $[20, 30)$, $[30, 40)$ etc. In summary, among 1,000 voters, there are 72 distinct demographic profiles — sets of individuals with an identical assignment of values to demographic attributes.



(a) Q_0 **(b)** Q_1 **(c)** Q_2 **(d)** Q_3 **(e)** Q_4

Figure 5: Templates of a label patterns for queries in the experimental workload over synthetic data.

Finally, we instantiate **Polls** as follows. Each voter from **Voters** is associated with one of two poll dates, and is assigned a Mallows model according to his demographic profile. We generate nine distinct models $\mathbf{MAL}_1(\boldsymbol{\sigma}_1, \phi_1), \ldots, \mathbf{MAL}_9(\boldsymbol{\sigma}_3, \phi_3)$ for each profile, associating one of three distinct random rankings of candidates $\boldsymbol{\sigma}_1$, $\boldsymbol{\sigma}_2$ and $\boldsymbol{\sigma}_3$ with one of three values of $\phi \in \{0.2, 0.5, 0.8\}$. In all our experiments over this dataset, **Voters** and **Polls** each contains 1,000 tuples. **Candidates** has 20 tuples in the majority of our experiments. When a different number of candidates is used, we state it explicitly.

Query workload. We evaluate performance of our system on a workload that is made up of 7 itemwise CQ. Recall from Section 3 that an itemwise CQ is translated to a set of label patterns. All patterns of a given query have the same structure—the same set of label nodes and edges, but they differ in the possible assignment of values to a label, and thus result in a different set of candidate items that match a label. The label patterns of the queries are presented in Figure 5. For readability, the labels are not variable names (as in our translation in Section 3.1), but rather the conjunctive condition (e.g., $party = \mathtt{D}$, $sex = \$x$) that a candidate must satisfy to be mapped to the label. Bindings of $\$x$ that occur in conditions are determined during query evaluation. Let Q be a query with label pattern g. In what follows we refer to the cartesian product of candidate items mapped to the labels of g as the *cartesian product of* Q.

Consider query Q_0, with label pattern template in Figure 5a.

$$Q_0(sex, edu) \leftarrow P(v, \_; c_1; c_3), P(v, \_; c_1; c_4), P(v, \_; c_2; c_4),$$
$$P(v, \_; c_4; c_5), V(v, sex, \_, edu), C(c_1, \_, sex, \_, \_, \_),$$
$$C(c_2, \_, \_, \_, edu, \_), C(c_3, \_, \_, age, \_, \_), C(c_4, \_, \_, \_, \_, \mathtt{S}),$$
$$C(c_5, \_, \_, \_, \_, \mathtt{MW}), age > \mathtt{50}$$

Depending on the demographics of the voter, attributes *sex* and *edu* will take on different values in the label pattern derived from the template in Figure 5a, resulting in a different number of candidate items per label. For Q_0, this parameter is between 1,296 and 13,608. Consider next query Q_1, with label pattern template in Figure 5b.

$$Q_1(sex, edu) \leftarrow P(v, \_; c_1; c_3), P(v, \_; c_1; c_4), P(v, \_; c_2; c_4),$$
$$P(v, \_; c_4; c_5), V(v, sex, \_, edu), C(c_1, \mathtt{D}, sex, \_, \_, \_),$$
$$C(c_2, \mathtt{R}, \_, \_, edu, \_), C(c_3, \mathtt{D}, \_, \_, \mathtt{JD}, \_), C(c_4, \mathtt{R}, \_, \_, \_, \mathtt{S}),$$
$$C(c_5, \mathtt{R}, \_, \_, \_, \mathtt{MW})$$

The structure of the label pattern template of Q_1 is similar to that of Q_0. However, the size of the Cartesian product is much lower for Q_1, and falls between 16 and 112.

The next query, Q_2, is shown below, with the corresponding label pattern template in Figure 5c. This query gives rise to label patterns in which the size of the Cartesian product of the items is between 162 and 1,512, falling in the intermediate range.

$$Q_2(sex, edu) \leftarrow P(v, \_; c_1; c_3), P(v, \_; c_1; c_4), P(v, \_; c_2; c_4),$$
$$P(v, \_; c_4; c_5), V(v, sex, \_, edu), C(c_1, \_, sex, \_, \mathtt{MD}, \_),$$
$$C(c_2, \_, \_, \_, \_, \mathtt{S}), C(c_3, \_, \_, \mathtt{70}, \_, \_), C(c_4, \_, \_, \_, edu, \_),$$
$$C(c_5, \_, \_, \_, \_, \mathtt{MW})$$

The next query in our workload, Q_3, has the same number of label nodes as the other queries, but a different topology, as seen in Figure 5d. The size of the Cartesian product of the candidate items is similar to that of Q_2, and is between 144 and 1,764.

$$Q_3(sex, edu) \leftarrow P(v, \_; c1; c3), P(v, \_; c_2; c_3), P(v, \_; c_3; c_4),$$
$$P(v, \_; c_3; c_5), V(v, sex, \_, edu), C(c_1, \_, \_, \_, edu, \_),$$
$$C(c_2, \mathtt{D}, \_, \_, \_, \mathtt{S}), C(c_3, \mathtt{R}, \_, age, \_, \_), C(c_4, \mathtt{D}, sex, \_, \mathtt{MD}, \_),$$
$$C(c_5, \_, \_, \_, \_, \mathtt{NE}), age > \mathtt{50}$$

The next query, Q_4, is challenging: the size of the Cartesian product of candidate items is between 1,440 and 40,320, and few order constraints are introduced, giving rise to 5! linear extensions.

$$Q_4(sex, edu) \leftarrow P(v, \_; c_1; c_2), P(v, \_; c_1; c_3), P(v, \_; c_1; c_4),$$
$$P(v, \_; c_1; c_5), P(v, \_; c_1; c_6), V(v, sex, \_, edu),$$
$$C(c_1, \_, sex, \_, edu, \_), C(c_2, \_, sex, \_, \_, \_), C(c_3, \_, \_, \_, edu, \_),$$
$$C(c_4, \_, \_, \_, \_, \mathtt{MW}), C(c_5, \_, \_, \_, \mathtt{MS}, \_), C(c_6, \_, \mathtt{F}, \_, \_, \_)$$

Query Q_5 is evaluated over MovieLens, and computes the probability that a romance movie is preferred to an adventure.

$$Q_5() \leftarrow P(\_, \_; m_1; m_2), M(m_1, \mathtt{romance}, \_, \_), M(m_2, \mathtt{adv}, \_, \_)$$

Finally, Q_6 is evaluated over CrowdRank to computes the probability that a voter prefers a movie in which the leading actor is of the same gender to one in which the leading actor is of the same age.

$$Q_6(sex, age) \leftarrow P(v, \_; m_1; m_2), V(v, sex, age),$$
$$M(m_1, \_, sex, \_), M(m_2, \_, \_, age)$$

5.2 Performance of LTM

We start by presenting results of an experiment that establishes scalability of the *Lifted Top Matching* (LTM) algorithm, and shows that this algorithm clearly outperforms the baseline — the *Top Matching* (TM) algorithm of [26]. In this experiment, we execute a parallel version of each algorithm over 48 threads. (An experiment

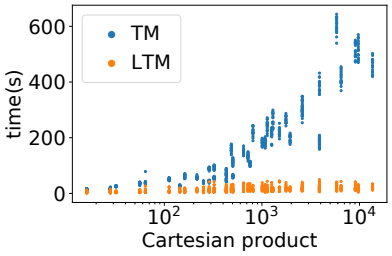


Figure 6: Running times vs. size of Cartesian product of items mapped to labels. Q_0, Q_1, Q_2, Q_3, 20 candidates.

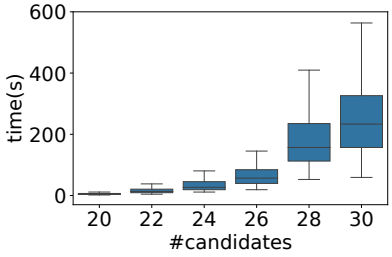


Figure 7: Running time of LTM vs. number of candidates (size of σ), for Q_1, over synthetic data.

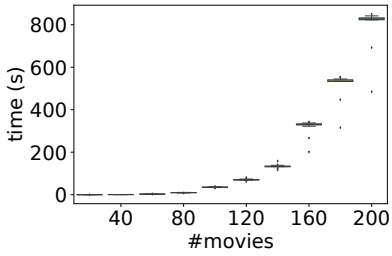


Figure 8: Running time of LTM vs. number of movies (size of σ), for Q_5, over the MovieLens dataset.

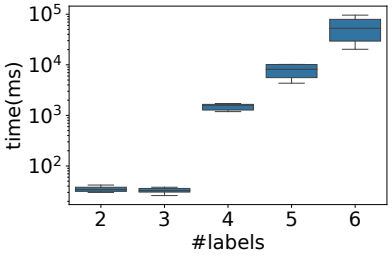


Figure 9: Running times of LTM vs. number labels in g, for Q_4, 20 candidates.

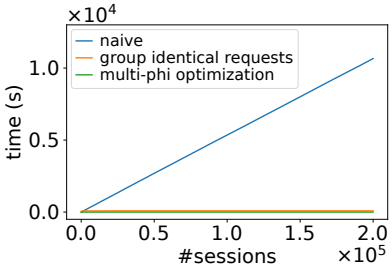


Figure 10: Running time of a workload of LTM requests for query Q_6 over CrowdRank (200,000 sessions).

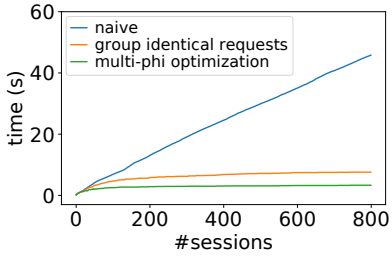


Figure 11: Running time of a workload of LTM requests for query Q_6 over CrowdRank (800 sessions).

that investigates speedup of TM and LTM due to parallelism is presented in Appendix C). Both algorithms are executed with the multi-ϕ optimization enabled. (This optimization was discussed in Section 3.2, and we will evaluate its effect in Section 5.3).

Figure 6 presents the running time of TM and of LTM as a function of the size of the Cartesian product of items mapped to each label, for queries Q_0–Q_3. We observe that the running time of LTM is not impacted by the size of the Cartesian product and remains constant, while that time of TM increases. LTM takes between 2 and 50 sec to complete for the inference requests in our workload, as compared to between 12 and 644 sec for TM. Over-all, LTM outperforms TM by up to a factor of 109 (greater than two orders of magnitude), where strongest improvement is realized in cases that are most challenging for TM. TM outperforms LTM slightly in one case, for a query with Cartesian product size 28, and on which both TM and LTM are efficient: 15.1 sec for LTM vs. 14.4 sec for TM.

Figures 7 and 8 display the execution time of LTM as a function of the number of items (candidates and movies, respectively), and Figure 9 displays the execution time of the algorithm as a function of the number of labels. These charts reflect the expected runtime bounds of the algorithm of $O(2^q m^q)$, where q is the size of the query (i.e., number of labels), and m is the number of items.

Figure 7 presents the running time of query Q_1 on the synthetic dataset of voters and candidates, with between 20 and 30 candidates, in increments of 2. Figure 8 presents the running time of Q_5 over MovieLens, a real dataset described in Section 5.1, with between 20 and 200 items, in increments of 20. A preference dataset with 200 items challenges the state of the art both in inference and in mining preference models, and is, to the best of our knowledge, the largest such dataset considered in the literature [36, 45]. The trends in Figures 7 and 8 are in accordance with the polynomial time data complexity guarantee. Figure 9 shows an exponential trend, as expected by the complexity bound of LTM.

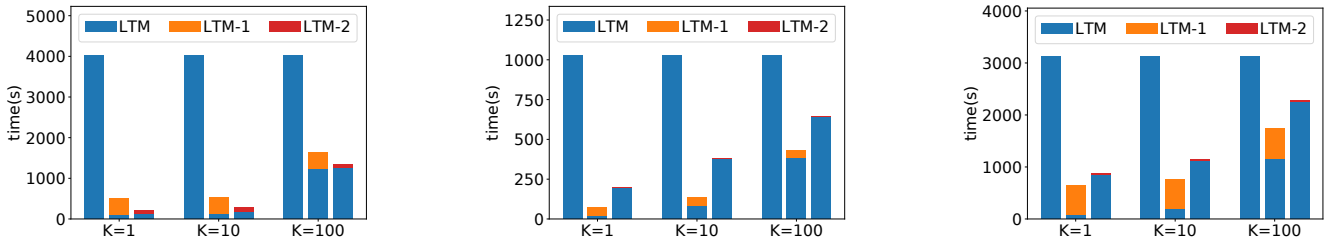
5.3 Workload-based Optimizations

In the next experiment, we demonstrate the benefit of the performance optimization described in Section 3.2, where a single inference request is issued for a set of requests that agree on the label pattern g and on the Mallows reference ranking $\boldsymbol{\sigma}$, but differ on the value of the spread parameter ϕ.

We compare the running time of single-ϕ LTM for queries Q_0–Q_3 to the running time of multi-ϕ LTM. For each query, we select 50 label patterns where $n \in \{2, 3\}$ sessions exist with the same $\boldsymbol{\sigma}$ and different ϕ values, for a total of 200 measurement points per n. For the selected cases, we divide the total running time of multi-ϕ LTM (200 calls) by the total running time of single-ϕ LTM ($200 * n$ calls).

This optimization is very effective, although working with multiple ϕ values concurrently does add overhead. We observe that multi-ϕ LTM is faster (in total) by a factor of 1.86 with two ϕ values, and by a factor of 2.56 with three ϕ values, compared to single-ϕ LTM. We also implemented this optimization in the TM algorithm, where we observe similar performance improvements.

The multi-ϕ optimization allows us to scale to a high number of sessions. To demonstrate this, we evaluate the running time of query Q_6 as a function of the number of sessions over the CrowdRank dataset, with up to 200,000 sessions. Figures 10 and 11 present these results. We observe that naively sending an inference request for



(a) Q_0 (b) Q_1 (c) Q_2

Figure 12: Running time of top-K queries, with the sub-pattern optimization.

each session (blue line) scales linearly with the number of sessions. In contrast, grouping together requests that agree on model parameters and user demographics (orange line) shows linear growth with the growing number of unique requests, and then remains constant. Combining this with the multi-ϕ optimization (green line) has even better performance. Figure 11 zooms in on the left part of Figure 10 to show the difference between the optimizations.

These experiments highlight the benefit of handling multiple related inference requests in a database engine: We are able to send requests that share a label pattern and a $\boldsymbol{\sigma}$, and differ only in the value of ϕ, for concurrent execution because we have knowledge of the full workload of requests, based on voter demographics in relation **Voters** and their Mallows models in relation **Polls**.

5.4 Conjunctive Queries and Top-K Sessions

We now demonstrate the benefit of incorporating probabilistic inference into a database engine. We consider two types of queries. The first type are itemwise CQs, which are translated into a set of calls to LTM to perform probabilistic inference, as discussed in Section 3.1. The second kind are top-K queries: given an itemwise CQ Q and a number K, return the K sessions that have the highest probability of generating a ranking that satisfies Q (see Section 3.2). All experiments described in this section were executed over 48 threads, and with the multi-ϕ optimization enabled.

Itemwise CQs. We present the total running time of executing each query in our workload with LTM (resp. TM). In addition to the total running time, we also list the number of inference requests that were issued for each query. Note that the number of requests differs among the queries. Specifically, for queries Q_1 and Q_4 a lower number of distinct label patterns was produced, and so the engine was able to group together more reference requests.

query	LTM (sec)	TM (sec)	# requests
Q_0	4,022	77,632	215
Q_1	1,027	4,303	179
Q_2	3,118	28,221	215
Q_3	3,198	23,946	215
Q_4	2,860	65,595	161

LTM consistently outperforms TM. Among these queries, improvement was highest for Q_4, where LTM is 23 times faster over-all.

Top-K sessions. We now compare (1) the running time of executing LTM for all sessions and then selecting the top-K by probability, to (2) the running time of an upper-bounds optimization. Figure 12 presents results of this experiment for queries Q_0, Q_1 and Q_2, for $K \in \{1, 10, 100\}$. In our experiment, we remove one label from each pattern and then invoke LTM to compute an upper bound. The running time of this phase is denoted LTM-1 in the plots, and corresponds to orange portions of the bars. Similarly, we can remove two labels and then call LTM to compute an upper bound. This is denoted LTM-2, and depicted in red. The running time of the top-K query is improved significantly when the upper-bounds optimization is used, for all queries and for all values of K. Strongest improvement was achieved by LTM-2 with $K = 1$ for query Q_0, where the total running time was reduced to 221 sec down from 4,023 sec, an improvement by a factor of 18.

Observed improvement is due primarily to the fact that the running time of LTM-1 and LTM-2 is small in all cases: it does not depend on K and is between 5% and 20% of LTM for LTM-1 and between 0.8% and 3% of LTM for LTM-2. Whether LTM-1 or LTM-2 has better performance in scope of an upper-bounds optimization depends on the query — on how far upper bounds are from the true probabilities and on the distribution of the probabilities, determining when the early termination condition can be reached. As is evident from Figure 12, upper bounds are sufficiently accurate to allow early termination in all cases in our experiments.

6 CONCLUSIONS

We embarked on an implementation of a PPD, starting with a query engine for itemwise CQs. We focused on reducing the execution cost of query evaluation, devising LTM, a lifted-inference version of a previous algorithm by Kenig et al. [26]. We also developed workload optimizations, including simultaneous execution for models with a shared reference ranking, and top-K pruning via pattern relaxation. Our experimental study shows significant performance gains.

Future directions include support for additional statistical models of preferences (e.g., [15, 33]), specialized operations on preferences (such as *rank aggregation* [10, 12]), and for essential database services (such as transactions and incremental maintenance).

REFERENCES

[1] Stein Aerts, Diether Lambrechts, Sunit Maity, Peter Van Loo, Bert Coessens, Frederik De Smet, Leon-Charles Tranchevent, Bart De Moor, Peter Marynen, Bassem Hassan, Peter Carmeliet, and Yves Moreau. 2006. Gene prioritization through genomic data fusion. *Nature Biotechnology* 24, 5 (2006), 537–544.

[2] Pranjal Awasthi, Avrim Blum, Or Sheffet, and Aravindan Vijayaraghavan. 2014. Learning Mixtures of Ranking Models. In *NIPS*. 2609–2617.

[3] Suhrid Balakrishnan and Sumit Chopra. 2012. Two of a kind or the ratings game? Adaptive pairwise preferences and latent factor models. *Frontiers of Computer Science* 6, 2 (2012), 197–208.

[4] Ludwig M. Busse, Peter Orbanz, and Joachim M. Buhmann. 2007. Cluster analysis of heterogeneous rank data. In *ICML*. 113–120.

[5] Weiwei Cheng and Eyke Hüllermeier. 2009. A New Instance-Based Label Ranking Approach Using the Mallows Model. In *Advances in Neural Networks – ISNN 2009*, Wen Yu, Haibo He, and Nian Zhang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 707–716.

[6] Nilesh N. Dalvi and Dan Suciu. 2004. Efficient Query Evaluation on Probabilistic Databases. In *VLDB*. Morgan Kaufmann, 864–875.

[7] Persi Diaconis. 1989. A generalization of spectral analysis with applications to ranked data. *Annals of Statistics* 17, 3 (1989), 949–979.

[8] Weicong Ding, Prakash Ishwar, and Venkatesh Saligrama. 2015. Learning Mixed Membership Mallows Models from Pairwise Comparisons. *CoRR* abs/1504.00757 (2015).

[9] Jean-Paul Doignon, Aleksandar Pekeč, and Michel Regenwetter. 2004. The repeated insertion model for rankings: Missing link between two subset choice models. *Psychometrika* 69, 1 (2004), 33–54.

[10] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. 2001. Rank aggregation methods for the Web. In *WWW*. 613–622.

[11] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. 2016. Declarative Cleaning of Inconsistencies in Information Extraction. *ACM Trans. Database Syst.* 41, 1 (2016), 6:1–6:44. https://doi.org/10.1145/2877202

[12] Ronald Fagin, Ravi Kumar, and D. Sivakumar. 2003. Comparing top k lists. In *SODA*. 28–36.

[13] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2014. Conflict resolution with data currency and consistency. *J. Data and Information Quality* 5, 1-2 (2014), 6:1–6:37.

[14] M. A. Fligner and J. S. Verducci. 1986. Distance Based Ranking Models. *Journal of the Royal Statistical Society. Series B (Methodological)* 48, 3 (1986), 359–369.

[15] M. A. Fligner and J. S. Verducci. 1986. Distance Based Ranking Models. *Journal of the Royal Statistical Society B* 43 (1986), 359–369.

[16] Michael A. Fligner and Joseph S. Verducci. 1988. Multistage Ranking Models. *J. Amer. Statist. Assoc.* 83, 403 (1988), 892–901.

[17] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC Data-Cleaning Framework. *PVLDB* 6, 9 (2013), 625–636.

[18] Isobel C. Gormley and Thomas B. Murphy. 2006. A Latent Space Model for Rank Data. In *ICML*.

[19] Isobel Claire Gormley and Thomas Brendan Murphy. 2008. A mixture of experts model for rank data with applications in election studies. *The Annals of Applied Statistics* 2, 4 (12 2008), 1452–1477.

[20] Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. 2014. Understanding the Complexity of Lifted Inference and Asymmetric Weighted Model Counting. In *UAI*. 280–289.

[21] Jonathan Huang, Ashish Kapoor, and Carlos Guestrin. 2012. Riffled Independence for Efficient Inference with Partial Rankings. *J. Artif. Intell. Res. (JAIR)* 44 (2012), 491–532.

[22] Ekhine Irurozki, Borja Calvo, and Jose A. Lozano. 2018. Sampling and Learning Mallows and Generalized Mallows Models Under the Cayley Distance. *Methodology and Computing in Applied Probability* 20, 1 (01 Mar 2018), 1–35. https://doi.org/10.1007/s11009-016-9506-7

[23] Marie Jacob, Benny Kimelfeld, and Julia Stoyanovich. 2014. A System for Management and Analysis of Preference Data. *PVLDB* 7, 12 (2014), 1255–1258.

[24] Toshihiro Kamishima and Shotaro Akaho. 2006. Supervised ordering by regression combined with Thurstone's model. *Artif. Intell. Rev.* 25, 3 (2006), 231–246.

[25] M. G. Kendall. 1938. A New Measure of Rank Correlation. *Biometrika* 30, 1/2 (1938), 81–93.

[26] Batya Kenig, Benny Kimelfeld, Haoyue Ping, and Julia Stoyanovich. 2017. Querying Probabilistic Preferences in Databases. In *PODS*. 21–36.

[27] Kristian Kersting. 2012. Lifted Probabilistic Inference. In *ECAI (ECAI'12)*. IOS Press, 33–38. https://doi.org/10.3233/978-1-61499-098-7-33

[28] Werner Kießling. 2002. Foundations of Preferences in Database Systems. In *VLDB*. 311–322.

[29] Benny Kimelfeld, Ester Livshits, and Liat Peterfreund. 2017. Detecting Ambiguity in Prioritized Database Repairing. In *ICDT*. 17:1–17:20.

[30] Angelika Kimmig, Lilyana Mihalkova, and Lise Getoor. 2015. Lifted graphical models: a survey. *Machine Learning* 99, 1 (01 Apr 2015), 1–45. https://doi.org/10.1007/s10994-014-5443-2

[31] Raivo Kolde, Sven Laur, Priit Adler, and Jaak Vilo. 2012. Robust rank aggregation for gene list integration and meta-analysis. *Bioinformatics* 28, 4 (2012), 573–580.

[32] Guy Lebanon and John D. Lafferty. 2002. Cranking: Combining Rankings Using Conditional Probability Models on Permutations. In *ICML*. 363–370.

[33] Guy Lebanon and Yi Mao. 2007. Non-parametric Modeling of Partially Ranked Data. In *NIPS*. Curran Associates, Inc., 857–864.

[34] Guy Lebanon and Yi Mao. 2008. Non-parametric Modeling of Partially Ranked Data. *Journal of Machine Learning Research* 9 (2008), 2401–2429.

[35] Tyler Lu and Craig Boutilier. 2014. Effective Sampling and Learning for Mallows Models with Pairwise-preference Data. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 3783–3829.

[36] Tyler Lu and Craig Boutilier. 2014. Effective sampling and learning for mallows models with pairwise-preference data. *Journal of Machine Learning Research* 15, 1 (2014), 3783–3829. http://dl.acm.org/citation.cfm?id=2750366

[37] C. L. Mallows. 1957. Non-Null Ranking Models. *Biometrika* 44 (1957), 114–130.

[38] Bhushan Mandhani and Marina Meila. 2009. Tractable Search for Learning Exponential Models of Rankings. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*. 392–399. http://www.jmlr.org/proceedings/papers/v5/mandhani09a.html

[39] John I. Marden. 1995. *Analyzing and Modeling Rank Data*. Chapman & Hall.

[40] Gail McElroy and Michael Marsh. 2010. Candidate Gender and Voter Choice: Analysis from a Multimember Preferential Voting System. *Political Research Quarterly* 63, 4 (2010), pp. 822–833.

[41] Haoyue Ping, Julia Stoyanovich, and Bill Howe. 2017. DataSynthesizer: Privacy-Preserving Synthetic Datasets. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*. 42:1–42:5.

[42] Anish Das Sarma, Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. 2010. Ranking mechanisms in twitter-like forums. In *WSDM*. 21–30.

[43] Shannon Starr. 2009. Thermodynamic limit for the Mallows model on Sn. *J. Math. Phys.* 50, 9 (2009), 095208. https://doi.org/10.1063/1.3156746 arXiv:https://doi.org/10.1063/1.3156746

[44] Hal Stern. 1993. Probability Models on Rankings and the Electoral Process. In *Probability Models and Statistical Analyses for Ranking Data*, Michael A. Fligner and Joseph S. Verducci (Eds.). Springer New York, New York, NY, 173–195.

[45] Julia Stoyanovich, Lovro Ilijasic, and Haoyue Ping. 2016. Workload-driven learning of mallows mixtures with pairwise preference data. In *WebDB*. 8.

[46] Julia Stoyanovich, Marie Jacob, and Xuemei Gong. 2015. Analyzing Crowd Rankings. In *WebDB*. 41–47.

[47] Joshua M. Stuart, Eran Segal, Daphne Koller, and Stuart K. Kim. 2003. A Gene-Coexpression Network for Global Discovery of Conserved Genetic Modules. *Science* 302 (2003), 249–255.

[48] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases*. Morgan & Claypool Publishers.

A ADDITIONAL PROOFS

LEMMA 4.2. *For all random rankings $\boldsymbol{\tau}$, if $(\boldsymbol{\tau}, \lambda) \models g$ then there is precisely one top embedding of g in $\boldsymbol{\tau}$*

PROOF. If $(\boldsymbol{\tau}, \lambda) \models g$ then there exists an embedding of g in $\boldsymbol{\tau}$. This means that there is a mapping from the nodes of g to the items of $\boldsymbol{\tau}$ that is consistent with the edges of g. We show that $\boldsymbol{\tau}$ contains a single top embedding. We prove the claim by induction on $|\text{nodes}(g)|$. If g contains a single node l, then $\delta^*(l) = k$ where $k = \min\{i \in \{1, \ldots, m\} \mid l \in \lambda(\tau_i)\}$. Since $\boldsymbol{\tau}$ matches g, then k exists, and by the minimality condition is unique.

Assume correctness for patterns with n nodes, and we prove the claim for a pattern with $n+1$ nodes. Consider a node $l \in \text{nodes}(g)$ with no outgoing edges in g. Since g is acyclic such a node must exist. Now consider the pattern induced by $\text{nodes}(g) \setminus \{l\}$ (i.e., without l and its incoming edges), defined over n nodes. Since $\boldsymbol{\tau}$ embeds g it must also embed the pattern induced by $\text{nodes}(g) \setminus \{l\}$, denoted g^n. By the induction hypothesis, $\boldsymbol{\tau}$ has a single, unique top embedding δ_n^* corresponding to g^n. Let τ_k be an item in $\boldsymbol{\tau}$ that is associated with l (i.e., $l \in \lambda(\tau_k)$). We define k as follows. If $pa_g(l) \neq \emptyset$ then τ_k is the highest ranked item of type l that appears after $\delta_n^*(u)$ for each $u \in pa_g(l)$:

Algorithm $\mathrm{f}_{\langle\delta,\mathrm{v}\rangle}(\boldsymbol{\tau})$

$\boldsymbol{\tau}' := \boldsymbol{\tau}$, $\mathbf{b} := \{j \in img(\delta) : v(j) = 0\}$
Let $j_1 < j_2 < \cdots < j_k$ denote the indexes of $\mathbf{b}$
for $i = 1, \ldots k$ **do**
  Insert placeholder item b_{j_i} into position j_i of $\boldsymbol{\tau}'$
  for $l = i+1, \ldots, k$ **do**
    j_l += 1
return $\boldsymbol{\tau}'$

Figure 13: An algorithm for computing $f_{\langle\delta,v\rangle}(\boldsymbol{\tau})$

$$k = \min\left\{i > \max_{u \in pa_g(l)} \delta_n^*(u) \mid l \in \lambda(\tau_i)\right\} \tag{7}$$

Otherwise, if $pa_g(l) = \emptyset$, then: $k = \min\{i > 0 \mid l \in \lambda(\tau_i)\}$. Since $\boldsymbol{\tau}$ embeds g, such an index k must exist, and due to the minimality condition, is unique. Define the embedding δ_{n+1} as follows:

$$\delta_{n+1}(u) = \begin{cases} \delta_n^*(u) & \text{if } u \neq l \\ k & \text{otherwise} \end{cases}$$

We now show that δ_{n+1} is both minimum and unique. By the induction hypothesis δ_n^* is both minimum and unique for the pattern g excluding the node l. Therefore, the top matching for g must be identical to δ_n^* for all nodes in nodes$(g) \setminus \{l\}$. By the minimality and uniqueness of the index k, we get that δ_{n+1} is also minimum and unique, as required. □

Lemma 4.3. *Let $\boldsymbol{\tau} \in rnk(\boldsymbol{\sigma})$ such that $(\boldsymbol{\tau}, \lambda) \models g$. Then $\boldsymbol{\tau}$ realizes the mapping δ : nodes$(g) \mapsto \{1, \ldots, m\}$ if and only if for every $l \in$ nodes(g) and item $\sigma \in \{\tau_i : i < \delta(l)\}$ at least one of the following conditions is met.* (a) *The item σ is not associated with l. That is, $l \notin \lambda(\sigma)$.* (b) *$pa_g(l) \neq \emptyset$ and $\boldsymbol{\tau}^{-1}(\sigma) < \max_{u \in pa_g(l)} \delta(u)$.*

Proof. **The "only if" direction** Assume that δ is a top matching for g in $\boldsymbol{\tau}$. We will show that δ and $\boldsymbol{\tau}$ meet the conditions of the lemma. Assume, by way of contradiction, that this is not the case. That is, there exists a label $l \in$ nodes(g), and an index $i < \delta(l)$ such that *(a)* $l \in \lambda(\tau_i)$, *and (b)* either $pa_g(l) = \emptyset$ or $i > \max_{u \in pa_g(l)} \delta(u)$. In this case, the mapping δ': nodes$(g) \mapsto \{1, \ldots, m\}$ defined as:

$$\delta'(v) = \begin{cases} \delta(v) & \text{if } v \neq l \\ i & \text{if } v = l \end{cases}$$

is an embedding of g in $\boldsymbol{\tau}$ (i.e., $\delta' \in \Delta(g, \boldsymbol{\tau})$), and $\delta' \geq_{\boldsymbol{\tau}} \delta$. Therefore, we arrive at a contradiction that δ is a top embedding for g in $\boldsymbol{\tau}$, and that $\boldsymbol{\tau}$ realizes δ.

The "if" direction Since it is given that $(\boldsymbol{\tau}, \lambda) \models g$ then the set $\Delta(g, \boldsymbol{\tau})$ of embeddings of g in $\boldsymbol{\tau}$ is nonempty. By Lemma 4.2 there exists a single top embedding δ' of g in $\boldsymbol{\tau}$. We show that $\delta' = \delta$.

Assume, by way of contradiction, that $\delta \neq \delta'$. Since the top embedding is unique then there exists a label $l \in$ nodes(g) such that $\delta'(l) < \delta(l)$. We can assume that g is acyclic, since otherwise $\Delta(g, \boldsymbol{\tau})$ is empty. Let $l \in$ nodes(g) be the smallest node in some topological order over g, for which $\delta'(l) < \delta(l)$.

If $pa_g(l) = \emptyset$, then we immediately arrive at a contradiction of the second condition of the lemma. So, we assume that $pa_g(l) \neq \emptyset$. Since δ' is an embedding of g in $\boldsymbol{\tau}$, it holds that $\delta'(l)$ is greater than all indexes $\delta'(l_p)$ where $l_p \in pa_g(l)$. Moreover, the fact that l was chosen to be *first* in some topological order over g, implies that $\delta'(l_p) = \delta(l_p)$ for every node $l_p \in pa_g(l)$. That is, $\delta'(l) > \max_{l_p \in pa_g(l)} \delta(l_p)$. Therefore, the label l and the item $\sigma = \boldsymbol{\tau}(\delta'(l))$ are in violation of the second condition of the lemma, with respect to the embedding δ, bringing us to a contradiction. □

B CORRECTNESS OF LTM

According to (5), proving the correctness of the algorithm LTM requires showing the following.

$$\sum_{\delta \in \mathcal{R}} p(\delta) = \sum_{\{\langle\delta,v\rangle \in \mathcal{R}_m \mid |v^{-1}(0)|=0\}} q_m(\langle\delta, v\rangle)$$

where $\mathcal{R}$ is the set of mappings δ : nodes$(g) \mapsto \{1, \ldots, m\}$ that are consistent with g. In what follows, we show that for every pair $\langle\delta, v\rangle \in \mathcal{R}_m$, where $|v^{-1}(0)| = 0$ it is the case that: $p(\delta) = q_m(\langle\delta, v\rangle)$. Let $\langle\delta, v\rangle \in \mathcal{R}_t$ for some $t \in [1, m]$, and let $\mathbf{b} = \{j \in img(\delta) \mid v(j) = 0\}$ denote the set of positions of placeholder items at time t. Recall that $\langle\delta, v\rangle$ denotes the event of generating a ranking $\boldsymbol{\tau} \in rnk(\{\sigma_1, \ldots, \sigma_t\})$ from $\mathbf{RIM}(\boldsymbol{\sigma}, \Pi)$, such that inserting placeholder items into positions $\mathbf{b}$ of $\boldsymbol{\tau}$ results in a prefix that realizes δ. We define this transformation, denoted $f_{\langle\delta,v\rangle}(\boldsymbol{\tau})$, formally:

$$f_{\langle\delta,v\rangle}(\boldsymbol{\tau}) : rnk(\{\sigma_1, \ldots, \sigma_t\}) \mapsto rnk(\{\sigma_1, \ldots, \sigma_t\} \cup \{b_j \mid j \in \mathbf{b}\})$$

in Algorithm 13. We say that the ranking $\boldsymbol{\tau} \in rnk(\{\sigma_1, \ldots, \sigma_t\})$ *realizes the pair* $\langle\delta, v\rangle$, denoted $\boldsymbol{\tau} \models \langle\delta, v\rangle$, if the prefix $f_{\langle\delta,v\rangle}(\boldsymbol{\tau})$ realizes δ. By this definition, we get that $p(\langle\delta, v\rangle)$ is:

$$p(\langle\delta, v\rangle) = \sum_{\substack{\boldsymbol{\tau} \in rnk(\{\sigma_1,\ldots,\sigma_t\})| \\ \boldsymbol{\tau} \models \langle\delta,v\rangle}} \Pi_{\boldsymbol{\sigma}}(\boldsymbol{\tau}) = \sum_{\substack{\boldsymbol{\tau} \in rnk(\{\sigma_1,\ldots,\sigma_t\})| \\ f_{\langle\delta,v\rangle}(\boldsymbol{\tau}) \models \delta}} \Pi_{\boldsymbol{\sigma}}(\boldsymbol{\tau})$$

Now, consider what happens when $\langle\delta, v\rangle \in \mathcal{R}_m$, and $|v^{-1}(0)| = 0$. In that case, $f_{\langle\delta,v\rangle}(\boldsymbol{\tau}) = \boldsymbol{\tau}$ for every $\boldsymbol{\tau} \in rnk(\{\sigma_1, \ldots, \sigma_m\})$. Furthermore, we get that a ranking $\boldsymbol{\tau} \models \langle\delta, v\rangle$ if, and only if, $\boldsymbol{\tau} \models \delta$. Therefore, if we prove that $p(\langle\delta, v\rangle) = q_t(\langle\delta, v\rangle)$ for every pair $\langle\delta, v\rangle \in \mathcal{R}_t$, and for every $t \in \{1, \ldots, m\}$, then we also get that $p(\delta) = q_m(\langle\delta, v\rangle)$ for all consistent mappings δ : nodes$(g) \mapsto \{1, \ldots, m\}$, and vectors v where $|v^{-1}(0)| = 0$. Therefore, we establish the correctness of the algorithm with the following theorem.

Theorem B.1. *For every $t \in \{0, \ldots, m\}$, and pair $\langle\delta, v\rangle \in \mathcal{R}_t$, we have that $p(\langle\delta, v\rangle) = q_t(\langle\delta, v\rangle)$.*

Proof. We prove the claim by induction on t.

Base case, $t = 0$: By definition, for every $\langle\delta, v\rangle \in \mathcal{R}_0$, we have that $q(\langle\delta, v\rangle) = 1$. All pairs $\langle\delta, v\rangle \in \mathcal{R}_0$ have an empty vector v (i.e., for every $j \in img(\delta)$, $v(j) = 0$). This means that a prefix that realizes the pair $\langle\delta, v\rangle$ contains only placeholder items, and no items from $\boldsymbol{\sigma}$. Therefore, at time $t = 0$, the only ranking $\boldsymbol{\tau}$ such that $\boldsymbol{\tau} \models \langle\delta, v\rangle$ (i.e., $f_{\langle\delta,v\rangle}(\boldsymbol{\tau}) \models \delta$) is the empty ranking whose probability is 1.0. Therefore, for every $\langle\delta, v\rangle \in \mathcal{R}_0$, it is the case that $q(\langle\delta, v\rangle) = p(\langle\delta, v\rangle) = 1$.

Inductive step: Assume correctness for all pairs $\langle \delta, v \rangle \in \mathcal{R}_i$ where $i < t$, and prove correctness for $i = t$. We denote by $(\langle \delta, v \rangle, \sigma_t \mapsto j)$ the event that the pair $\langle \delta, v \rangle$ was generated in iteration t of LTM, *following* the insertion of item σ_t into position j. Given the pair $\langle \delta, v \rangle \in \mathcal{R}_t$, let $t' = t + Z(v)$ be the length of the prefix represented by this pair. By the law of complete probability we have that:

$$p(\langle \delta, v \rangle) = \sum_{j=1}^{t'} p\left(\langle \delta, v \rangle, \sigma_t \mapsto j\right)$$

We consider the event $(\langle \delta, v \rangle, \sigma_t \mapsto j)$, and characterize the set of pairs $\langle \delta', v' \rangle \in \mathcal{R}_{t-1}$ that produce the pair $\langle \delta, v \rangle$ following the insertion of item σ_t into position j. We divide into two cases:

Case 1: $j \in img(\delta)$**.** We denote by $L_j = \{l \in \text{nodes}(g) \mid \delta(l) = j\}$, the set of labels that are mapped to position j by δ. Since j is a legal position for σ_t, then at the beginning of iteration t it must be the case that $v(j) = 0$ (line 1 in Range). Since $\sigma_t \in \text{items}(\boldsymbol{\sigma})$, then it is not a placeholder item. Therefore, at the end of the iteration, $v(j) = 1$, and item σ_t is assigned (via δ) to all labels L_j. Therefore, if σ_t is placed in position j at iteration t, then the only pair $\langle \delta', v' \rangle \in \mathcal{R}_{t-1}$ that led to the generation of $\langle \delta, v \rangle \in \mathcal{R}_t$ is defined as follows: (1) $\delta' = \delta$, and (2) $v'(k) = v(k)$ if $k \neq j$, and $v'(j) = 0$ otherwise. We denote the binary vector v' defined this way as $v \setminus j$.

Case 2: $j \notin img(\delta)$**.** This means that σ_t does not replace any placeholder item. In this case, the only pair $\langle \delta', v' \rangle \in \mathcal{R}_{t-1}$ that could have led to the generation of $\langle \delta, v \rangle \in \mathcal{R}_t$ is defined as follows: (1) $\delta'(l) = \delta(l)$ if $\delta(l) < j$, and $\delta'(l) = \delta(l) - 1$ otherwise. We denote the mapping δ' defined this way as δ_{-j}. (2) $v' = v$

Fixing the pair $\langle \delta, v \rangle \in \mathcal{R}_t$, every legal position $j \in \{1, \ldots, t'\}$ for item σ_t falls under one of these two cases. Otherwise, j must be illegal for σ_t, making the event $(\langle \delta, v \rangle, \sigma_t \mapsto j)$ impossible.

We denote the set of positions corresponding to case 1 and 2 by $\mathcal{J}_1$, and $\mathcal{J}_2$ respectively. Since $\mathcal{J}_1$ and $\mathcal{J}_2$ are disjoint, and by the above analysis, we have that:

$$\begin{aligned}
p(\langle \delta, v \rangle) &= \sum_{j \in \mathcal{J}_1} p\left(\langle \delta, v \rangle, \sigma_t \mapsto j\right) + \sum_{j \in \mathcal{J}_2} p\left(\langle \delta, v \rangle, \sigma_t \mapsto j\right) \\
&\underbrace{=}_{\substack{\text{cases} \\ \text{1 and 2}}} \sum_{j \in \mathcal{J}_1} p(\langle \delta, v \setminus j \rangle) \cdot \Pi'(t, j, \langle \delta, v \setminus j \rangle)) + \\
&\qquad \sum_{j \in \mathcal{J}_2} p(\langle \delta_{-j}, v \rangle) \cdot \Pi'(t, j, \langle \delta_{-j}, v \rangle)) \\
&\underbrace{=}_{\substack{\text{induction} \\ \text{hypothesis}}} \sum_{j \in \mathcal{J}_1} q(\langle \delta, v \setminus j \rangle) \cdot \Pi'(t, j, \langle \delta, v \setminus j \rangle)) + \\
&\qquad \sum_{j \in \mathcal{J}_2} q(\langle \delta_{-j}, v \rangle) \cdot \Pi'(t, j, \langle \delta_{-j}, v \rangle)) \\
&= \sum_{j \in \mathcal{J}_1} q(\langle \delta, v \rangle, \sigma_t \mapsto j) + \sum_{j \in \mathcal{J}_2} q(\langle \delta, v \rangle, \sigma_t \mapsto j) \\
&= \sum_{j=1}^{t+Z(v)} q(\langle \delta, v \rangle, \sigma_t \mapsto j) = q(\langle \delta, v \rangle)
\end{aligned}$$

This completes the inductive step. □

C THE EFFECT OF PARALLELISM

Our implementation of TM and of LTM exploits CPU parallelism. In this set of experiments, we evaluate the running time improvements as a function of the number of threads. We run our experiments on a 48-core machine, and so execute with at most 48 threads.

We start by presenting results of two experiments in which only four cores were used for the computation, far fewer than in Section 5. Our goal in this experiment is to show that performance of our methods is reasonable on commodity hardware. As expected, the raw running time increases with fewer cores, but the trends remain similar to those at 48 cores.

We start by evaluating the running time of LTM vs. number of candidates (size of $\boldsymbol{\sigma}$), for Q_1, at 4 threads. This result is presented in Figure 14, and should be compared with the result in Figure 7, where the same experiment is executed at 48 threads.

Next, in Figure 15, we present results of the top-K experiment over 4 threads, which should be compared to Figure 12. Observe that the running times increase in all cases, but that the top-K optimization is effective. In the remainder of this section, we systematically evaluate the effects of parallelism on query performance.

Consider Figure 16 that presents speedup achieved by LTM as a function of the number of threads, computed by dividing the running time of the single-threaded execution by the running time of the parallel execution. Different queries are able to realize different degrees of improvement, and we show results separately per query.

Consider first query Q_0 in Figure 16a and observe that adding threads beyond 8 does not improve speedup. This is in contrast to query Q_2 in Figure 16c, which speeds up by a factor of 7 at 14 threads and then plateaus. As another example, Query Q_4 shows close to linear speedup until 12 threads, and improves further all the way to 48 threads. Queries Q_1 and Q_3 exhibit a similar trend as does Q_0: Q_1 achieves linear speed-up at 4 threads, and plateaued at 8 threads, achieving 6-fold speedup, while Query Q_3 plateaued at 8 threads, achieving 4-fold speedup.

We also investigated to what extent TM can leverage CPU parallelism. Recall from Figure 6 that TM does not scale well for label patterns for which the size of the Cartesian product of the number of items per label is large. For this reason, we restrict our experiments with TM to those in which the size of the Cartesian does not exceed 500. Figure 16f shows that TM achieves close to linear speedup for up to 12 threads, and that it benefits from added parallelism for up to 24 threads.

Our implementation of LTM and TM takes advantage of the fact that both algorithms calculate probabilities of disjoint probabilistic

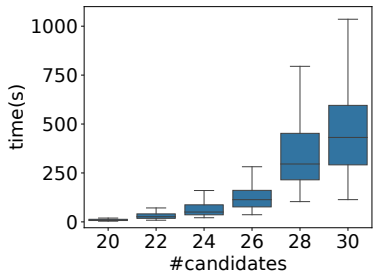
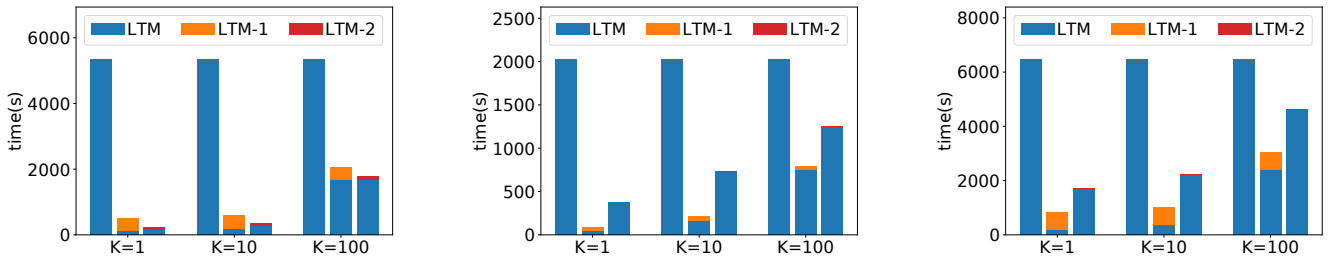
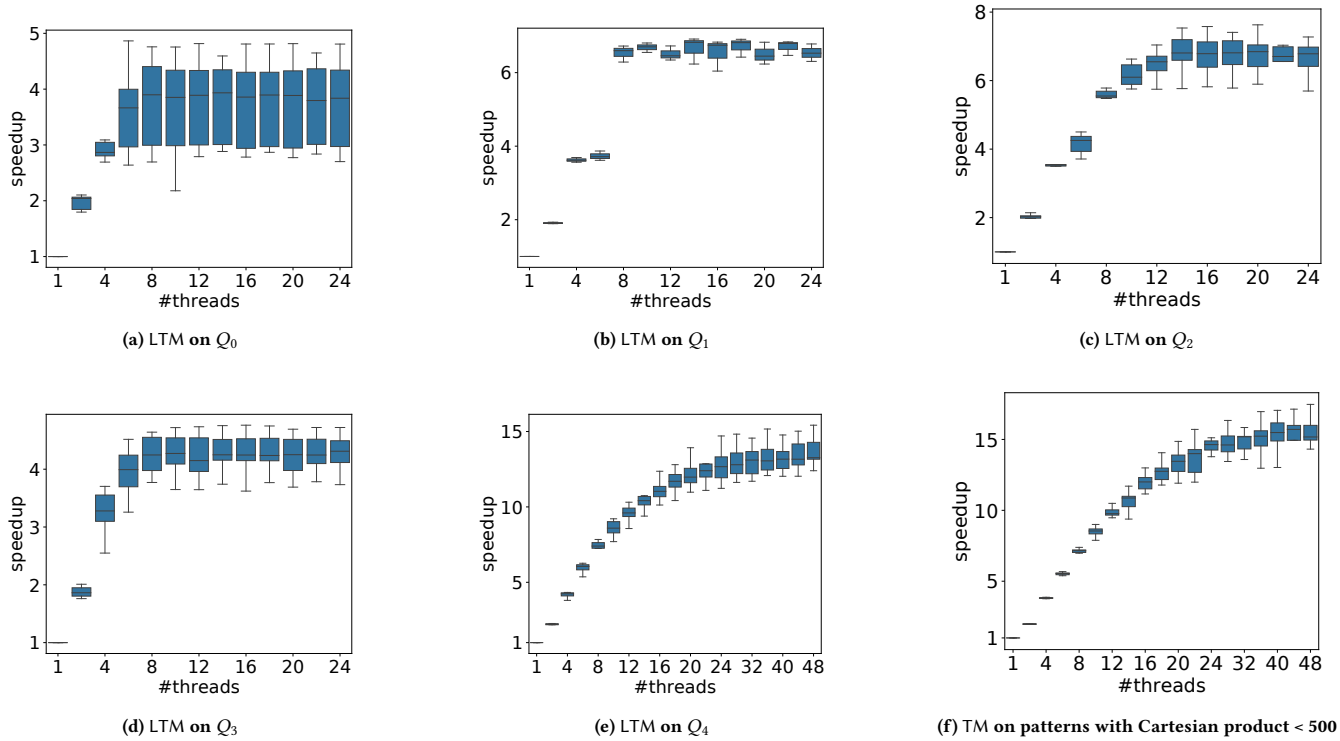


Figure 14: LTM **vs. size of** $\boldsymbol{\sigma}$**,** Q_1**, 4 threads.**



(a) Q_0 (b) Q_1 (c) Q_2

Figure 15: Running time of top-K queries, with the sub-pattern optimization, 4 threads.



(a) LTM **on** Q_0 (b) LTM **on** Q_1 (c) LTM **on** Q_2

(d) LTM **on** Q_3 (e) LTM **on** Q_4 (f) TM **on patterns with Cartesian product < 500**

Figure 16: Speedup of LTM and TM due to parallelism.

events. For TM, the unit of work is the set of possible assignments of items to labels — the size of the Cartesian product of items that correspond to labels, see Figure 6.

For LTM the unit of work is the set of initial mappings from labels to indexes, referred to as R_0 in Section 4. Tasks were distributed to threads based on this number of initial mappings, which was small for most queries in our workload. Query Q_4 has the highest number of initial mappings, and it parallelized best.

The table below presents the relationship between the number of initial mappings (averaged across label patterns for each query) and the maximum speedup. We see that Q_4 has the highest number of initial mappings and so achieves highest speed up. For other queries, the number of initial mappings is much lower. Queries Q_2 and Q_3 give rise to label patterns with a very similar number of initial states, and show comparable speedup. Query Q_1 has fewer initial states than Q_2 and Q_3, but exhibits somewhat better speedup. We believe that this is due to the cost of cross-chip communication: there are 12 cores per chip in our machine, and parallelizing just beyond the boundary of the chip can slightly hurt performance.

query	avg #initial-states	max speedup
Q_0	1.5	4.8
Q_1	7	6.9
Q_2	15.5	7.7
Q_3	14.5	6
Q_4	108	15