M.SC THESIS - DATA SCIENCE AND TECHNOLOGY

AGGREGATION IN PROBABILISTIC DATABASES

Implemented for **DUBIO**

LE THI NHI HA

Institute for Data Management & Biometrics Department
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
Enschede, Netherlands

August 2022

Supervisors: DR.IR. M. VAN KEULEN (MAURICE)
EEMCS/Data Management & Biometrics
University of Twente

DR. M. GERHOLD (MARCUS)
EEMCS/Formal Methods and Tools
University of Twente

Advisor: ING. J. FLOKSTRA (JAN)
EEMCS/Data Management & Biometrics
University of Twente

Examiner: DR.IR. M. VAN KEULEN (MAURICE)
DR. M. GERHOLD (MARCUS)

Contents

Acknowledgements

First of all, I want to give a very special thanks to my first supervisor, Dr. Maurice van Keulen. He had brought up an interesting topic and allowed me to work in his database project. In the past nine months, he had attended meeting with me weekly to supervise my work and provide all necessary advice for writing clear, concise thesis. I am so grateful for his feedback through all of my messy drafts. I also genuinely appreciate his patience and understanding for my imperfect English. Next, I would like to thank my advisor, Jan Flokstra. Jan has helped to explain detail about the database implementation, developed several sub functions that I needed to complete mine. Besides, I discovered a cool hobby - Lego technique during our weekly meeting.

I also want to give a big thank to my friend, my housemate Tarun Ravi, he is such a nice fellow student. Even though his major is chemical engineering, his interest in Computer Science topic has no doubt. Explaining the topic and providing background knowledge to him really strengthened my comprehension. He also volunteered to proofread this report.

I'm always grateful to my husband, Le Van Kien. He is a cool husband, awesome co-op game player and experienced programmer. He always gave me helpful keywords to search for when I was stuck in my own program. Without his support, I should have been struggled a lot.

Last but not least, I would like to thank myself for having a great determination to finish my project and my study program through the difficult time.

Abstract

Probabilistic databases lack of the support of aggregate functions, since there is difficulty in representing query result. Specifically, probabilistic databases have exponential number of possible worlds with potentially different answers for each worlds, it is impractical to represent all of them to the user. To tackle the problem, a few probabilistic databases present aggregate queries as expected value or top-k result. In this thesis, the ultimate goal is implementing aggregate function in probabilistic databases, particularly for DuBio, a probabilistic database developed by the University of Twente.

This research includes a methodology to build such aggregate functions in DuBio, experiments to study the feasible number of records these functions can achieve, what is their scalability and how uncertainties of the database affects them. We devise 2 approaches to represent the results of aggregate queries. Firstly, representing all possible answers to the user. Secondly, representing the query result approximately as a histogram or top k. We have implemented 3 aggregate functions in total, i.e. Combination and All possible worlds to return all possible aggregate answers, TopK to return aggregate answers over top K probable worlds. To evaluate these functions, we conduct 5 experiments including changing the database size by increasing number of records, number of random variables, number of alternatives and sentence's complexity.

Our conclusion is as follows: Combination and All possible worlds basically have exponential growth rate. However All possible worlds is favored over Combination because it grows exponentially w.r.t. number of variables while Combination grows exponentially w.r.t. number of records. The maximum number of record Combination and All possible worlds can query safely under 30 seconds on a commodity hardware is 15 and 26, respectively. Despite exponential growth, in practice, user can safely use All possible worlds aggregate queries when the intermediate result before aggregation has max 26 tuples. TopK algorithm has log-linear growth rate, it reaches safely about 350 records on a commodity hardware. Therefore this function is practical for large-sized databases, and scalable on server hardware.

The contribution of this research is providing aggregate functions which can return all possible answers, and answers on Top K probable worlds for DuBio database.

Nomenclature

Φ	The sentence of world
φ	The sentence of record
a	The number of alternatives
n	The number of records
r	The number of random variables
w	The number of possible worlds

Chapter 1

INTRODUCTION

Uncertainty Management In data integration, uncertainty management has been considered as a fundamental aspect. Since it is unavoidable to produce uncertainty data during mapping process, and the generated uncertainty should be represented explicitly as a relevant information source[9]. In information extraction, uncertainty data emerges inherently by the ambiguity of natural language text[5]. In business intelligence, the uncertainty exists in decision making, e.g. because the source has unreliable data collection method[10]. And in data publishing, preserving privacy is usually achieved by adding noise to original data[16] which leads to imprecise data on purpose. As a result, a large number of applications today have to deal with imprecise and uncertain data. Meanwhile, traditional databases are not characterized to manage the uncertainty of information, probabilistic databases are modelled to handle the uncertain data[18].

Probabilistic Databases "An *incomplete database* is a database that allows its instance to be in one of multiple states (worlds); *a probabilistic database* is an incomplete database that, furthermore, assigns a probability distribution to the possible worlds"[11]. A tuple t is uncertain in the databases and described by a random variable, which can be either true if it belongs to the database instance or false if it does not. Each tuple can be either independent or mutual dependent/exclusive from each other[11]. The probability of a variable represents the level of certainty of a tuple, it has a value in between 0 and 1, higher value represents higher confidence in the presence of the record.(The terms tuple and record are used interchangeably in this thesis)

Problem Statement It is expected that probabilistic databases scale and support complex queries as well as traditional databases[11]. In fact, in probabilistic database, it is difficult to represent the aggregate query efficiently. Every tuple is described by a variable, when combining all of those tuples, it is impractical to present all possible answers to the user. For example, given that n records are uncertain in a table T with attribute a, an aggregate query like "`SELECT SUM(a) FROM T`" in the worst case, could have a different possible answer for each possible world, i.e., 2^n possible answers. As a result, we acknowledge there is no such aggregate functions in probabilistic database to date. Hitherto, MayBMS[7] and Trio[4] are 2 prominent probabilistic databases, instead of supporting aggregate functions, these databases offer expected value for aggregation. Nevertheless, this approach could lead to deviated

query answers if the data values and their probabilities follow skewed and non-aligned distributions[19]. The lack of support for aggregates in probabilistic databases is the primary motivation for us to conduct this research. We are aware of the exponential complexity of aggregate in probabilistic databases, however we decided to find out up to which size it is practically possible to aggregate, and different forms of aggregate query(exact answer, histogram answer, top-k answer).

Research Questions The research conducted is to implement aggregates in probabilistic database. Hence, the principal question we try to answer is "*How to implement aggregates in probabilistic databases, particularly for DuBio?*". By referring aggregates here, we concentrate solely on COUNT function, since algorithmically, it is representative of the other functions(SUM, MIN, MAX, AVG, etc.). To elaborate further on the main question, we have the following questions:

1. **RQ1** To present the result of queries exactly, how do we implement a function which returns all possible answers and their probabilities?
2. **RQ2** To present the result of queries approximately, how do we implement a function which returns possible answers and their probabilities distribution as a top-k ranking or histogram?
3. **RQ3** What is the maximum number of records our aggregate functions can reach under a reasonable time, particularly in DuBio(for both exact and approximate count)?
4. **RQ4** What is the complexity of our algorithms?
5. **RQ5** How the uncertainties in probabilistic database affects our algorithms. Speaking of uncertainties in DuBio, it is number of possible worlds the database can present. This number is varied by following factors in DuBio which we would like to measure:
   a) **RQ5.1** How the number of random variables affects our algorithms?
   b) **RQ5.2** How the number of alternatives per random variable affects our algorithms?
   c) **RQ5.3** How the complexity of sentences(propositional formulas) affects our algorithms?

Implementation Particularly, we build aggregates for DuBio [1], the probabilistic database being developed by the University of Twente, based on possible worlds theory and Binary Decision Diagram structure. The certainty of a tuple in Dubio is represented by a sentence(propositional formula of random variable). We developed aggregate functions in PL/SQL, which are *exact aggregate*(return all possible answers), *histogram aggregate*(return answers grouped into buckets) and *topk*(return answers in top highest probability worlds). In the scope of this thesis, we focus our algorithmic work on exact and topk only.

[1] `https://github.com/utwente-db/DuBio`

Methodology We designed 5 experiments to evaluate the performance of the aggregate algorithms, specifically in the database size and the number of database instances. We have generated synthetic data, and different number of random variables, number of alternatives, different number of variables in a sentence. The aggregate query is attached along with *explain analyze* to collect the running time. Code for aggregate and experiments in the thesis is available at [2]

Structure The thesis is organized as follows: in chapter 2 we describe the background knowledge about probabilistic databases, Binary Decision Diagrams and DuBio. In chapter 3, we explain our algorithms and their complexity, which are Combination, All possible worlds and TopK. Then in chapter 4, we detail our experiment setup and describe main results. Next in chapter 5, we discuss about optimization and the limitations of this research. Finally in chapter 6, we draw our conclusion and talk about future work.

[2]Code for aggregate and experiments is available at `https://github.com/stubbornfox/dubio/`

Chapter 2

BACKGROUND & RELATED WORK

This chapter includes 4 sections, in section 2.1 we give high-level definition of probabilistic database according to its most general form, i.e. *Possible Worlds Model*. Next, in section 2.2 we provide background on Binary Decision Diagram, a data structure used to represent a sentence in DuBio. Afterwards, we explain thoroughly about DuBio and how does it adopt possible world theory in its implementation(section 2.3). Finally, different approaches of aggregation have been developed over multiple probabilistic databases(section 2.4).

2.1 Probabilistic Database

In general, a probabilistic database is a probability space over the possible instance of the database. I denotes a database instance of the probabilistic database *PWD*. A probabilistic database is a probability space *(W, P)*, where[18]:

- W is a non empty set, i.e. $W = \{I_1, I_2, \ldots I_n\}$, which represents all possible instances.
- P is a function, i.e. $P : W \longrightarrow (0, 1]$, which assigns probabilities to the instances I such that $\sum_{j=1,n} \mathbf{P}(I_j) = 1$

In the scope of this research, we restrict the probabilistic databases to have a finite set of possible worlds. By definition, there is a practical representation of probabilistic database called *probabilistic conditional tables*[11], in which, each tuple is annotated with a propositional formula over random variables. Then the probability space is assigned over probability distribution of these random variables. Using that representation, the correlation of tuples in a table is expressed, which can be either:

- Independence, e.g. $\langle t_1, X \longrightarrow 1\rangle, \langle t_2, Y \longrightarrow 1\rangle$ (X, Y is independent)
- Mutual dependence, e.g. $\langle t_1, X \longrightarrow 1\rangle, \langle t_2, X \longrightarrow 1\rangle$

- Mutual exclusive, e.g. $\langle t_1, X \rightarrow 1\rangle, \langle t_2, X \rightarrow 2\rangle$

Hitherto, we explain probabilistic database at high-level conceptual definition, for further examples and detailed representation of the databases, we describe it particularly in Dubio section.

2.2 Reduced Ordered Binary Decision Diagrams

DuBio uses Binary Decision Diagram(BDD) structure to represent "sentence" of a record. To understand what is BDD and why it is applicable to present sentences, this section provides knowledge of BDD, Reduced BDD, Ordered BDD and their properties.

A Binary Decision Diagram (BDD) is a rooted directed acyclic graph which represents a propositional formula effectively. In which[12]:

- Leaf node is logical value of the formula, which is either False or True.
- Non-leaf node is proposition, from which there are 2 outgoing edges, *false edge* is denoted by a dotted line($\dashrightarrow$), *true edge* is denoted by a solid line($\rightarrow$)
- All paths from root to leaf have distinct propositions.

Figure 2.1 is a binary decision diagram for formula $\phi = p \vee (q \wedge r)$, from which we can identify all possible interpretations of the value of ϕ. For example, the branch that goes left, left, left w.r.t the interpretation $\mathscr{I}(p,q,r) = \{false, false, false\}$, and leaf label is false so we can conclude value of ϕ is false under this interpretation.

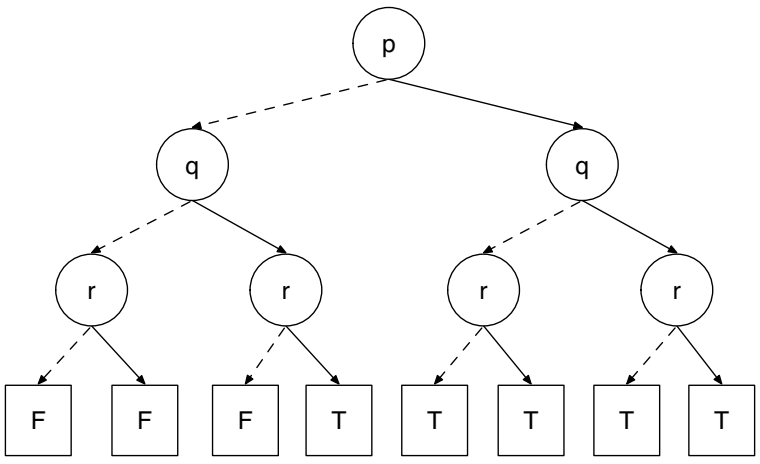

Figure 2.1: A binary decision diagram for formula $\phi = p \vee (q \wedge r)$

Reduced BDD A BDD can become more concise without loosing any interpretations by applying reductions. A BDD is reduced if there is no further reduction can be done. To reduce a BDD[12]:

- Remove duplicate leaves, redirect all edges to 2 remaining leaves.
- Delete a node and redirect as long as:
    1. It has 2 edges pointing to the same node. After that, redirect its coming node to its child node.
    2. It is identical to another node(same proposition label, same sub-BDDs child). After that, redirect its incoming edges to the other node.

Figure 2.2 is a reduced BDD for ϕ, as we can see the tree is concise now with 4 paths(the full BDD in Figure 2.1 has 8 paths)

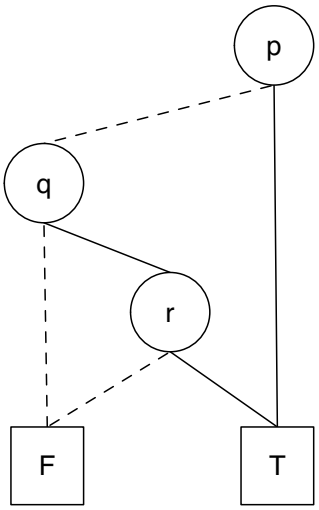

Figure 2.2: A reduced binary decision diagram for formula $\phi = p \vee (q \wedge r)$

In a BDD, each path represents an interpretation, if there is no indication about order of propositions, the appearance of propositions in each path could be different. Figure 2.3 is an example of unordered BDD. We can see the order of propositions is different between left branch(p < q < r) and right branch(p < r < q).

Ordered BDD An Ordered BDD is a BDD where the order of propositions in all branches is compatible[12]. For example, the BDD(in Figure 2.2) is ordered as p < q < r. It has four branches where the orderings of atoms are $\{pq, pqr, p\}$, thus, the orderings are compatible.

Properties If a BDD is ordered, its reduced BDD is also ordered. Two equivalent formulas will have structurally identical reduced BDD under an ordering of proposition[1]. From a reduced ordered BDD(ROBDD) of formulas ϕ_1, ϕ_2, we can deduce:

- ϕ is valid(always true) iff ROBDD(ϕ) is a single leaf True
- ϕ is unsatisfiable(always false) iff ROBDD(ϕ)is a single leaf False
- if ROBDD(ϕ_1) and ROBDD(ϕ_2) are identical, $\phi_1 \equiv \phi_2$

The size of reduced order BDD depends strongly on orderings of propositions so choosing efficient orderings helps construct efficient reduced ordered BDD.

We can also combine two BDDs using connectives “and”, “or” or “not” by using the Apply operator [12]. If the BDDs are both ordered with the same atoms ordering, there exists an efficient merge algorithm for Apply. This makes BDD powerful since we are not required to construct the full BDD again when performing logic operator between 2 formulas.

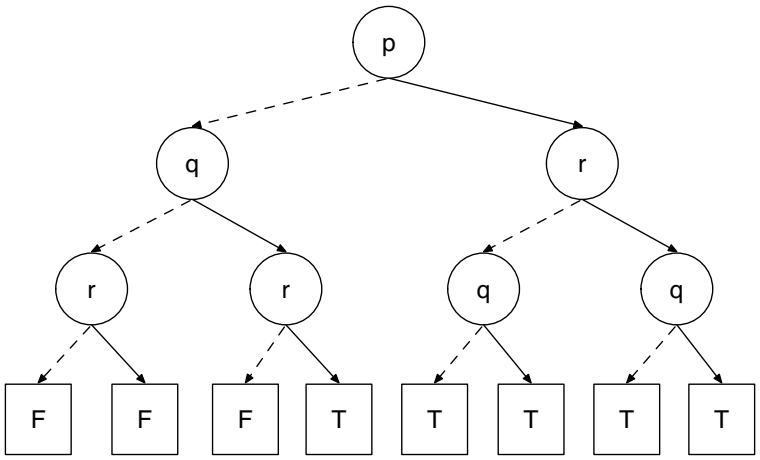

Figure 2.3: A BDD for formula $\phi = p \vee (q \wedge r)$ with different orders of atoms in branches

2.3 DuBio Database

We illustrate the probabilistic database DuBio through the examples of “Big cats” in Figure 2.4. DuBio is a scalable and powerful probabilistic database developed by the University of Twente using possible worlds semantic. To represent the possible worlds, DuBio uses the concept of “sentences”[13], which are propositional formulas with random variable assignments, also called “alternatives”, as atoms. We use the following notation:

- **Z** A random variable represents the existence of a specific bit of uncertainty. It is a probability distribution of a number of mutually exclusive alternatives. For example, random variable X represents the alternative species of “Mufasa”.
- **Z = a** A random variable assignment(rva) represents such an alternative. For example, the random variable X has 3 alternatives X=1, X=2, X=3, which describes that “Mufasa” could either be a Leopard, a Jaguar or a Cheetah, respectively.
- φ A sentence represents a propositional formula constructed from alternatives i.e rva, using binary operations $\wedge$(and), $\vee(or)$, and the unary operation $\neg$ (not) . For examples, $(\neg X = 1), (Y = 2) \wedge (C = 3), (Y = 1) \vee (Y = 2)$

V

variable	possible values	description
A	1,2,3,4	Simba's age range
B	1,2,3,4	Mufasa's age range
C	1,2,3,4	Scar's age range
X	1,2,3	Mufasa's species
Y	1,2	Scar's species
F	1,2	Simba's lineage

A

cat	age(years)	φ	p
Simba	[0-2]	A=1	0.8
Simba	[3-6]	A=2	0.2
Mufasa	[3-6]	B=2	0.5
Mufasa	[7-10]	B=3	0.5
Scar	[7-10]	C=3	0.7
Scar	[11-14]	C=4	0.3

S

cat	species	φ	p
Mufasa	Leopard	X=1	0.8
Mufasa	Jaguar	X=2	0.1
Mufasa	Cheetah	X=3	0.1
Scar	Leopard	Y=1	0.7
Scar	Jaguar	Y=2	0.3
Simba	Leopard	(F=1 $\wedge$ X=1) $\vee$ (F=2 $\wedge$ Y=1)	0.75
Simba	Jaguar	(F=1 $\wedge$ X=2) $\vee$ (F=2 $\wedge$ Y=2)	0.2
Simba	Cheetah	$F = 1 \wedge X = 3$	0.05

L

cat	dad	φ	p
Simba	Mufasa	F=1	0.5
Simba	Scar	F=2	0.5

Figure 2.4: Big cat Observation Database. Tables Species(cat, species) and Age(cat, age) describe (possibly uncertain) big cat species and their age range, respectively. Lineage(cat, dad) L represents relationship of those cats.

- $(\mathbf{t}, \boldsymbol{\varphi})$ A pair represents the possible attributes and their corresponding sentence. For example,
  ($\langle$cat = "Scar", species = "Leopard", age = [7-10]$\rangle$, $Y = 1 \wedge C = 3$).
- $\mathscr{D} = \{(\mathbf{t_1}, \boldsymbol{\varphi_1}), .., (\mathbf{t_n}, \boldsymbol{\varphi_n})\}$ A probabilistic database is a set of pairs (t_i, φ_i). Without loss of generality, we use a representation of a database $\mathscr{D}$ as containing one relation to keep the formulas simple. For our example database consisting of two relations, we denote this as a database storing $S \bowtie A$.
- $\mathscr{W} = \{\mathbf{t_1}, .., \mathbf{t_n}\}$ A world is induced by a random variable assignment θ_w to each random variable. A tuple t_i exists in world $\mathscr{W}$ if φ_i is true for θ_w; For example, $\theta_w = \{A \rightarrow 1, B \rightarrow 2, C \rightarrow 3, X \rightarrow 1, Y \rightarrow 2, F \rightarrow 1\}$ produces:
  $\mathscr{W} = \{\langle$"Mufasa", "Leopard", [3-6]$\rangle$, $\langle$"Scar", "Jaguar", [7-10]$\rangle$, $\langle$"Simba", "Leopard", [0-2]$\rangle\}$
- $\mathscr{PWS} = \{\mathscr{W}_1, ..., \mathscr{W}_m\}$ $\mathscr{PWS}$ is a set of all possible worlds. Semantically, a query is performed over probabilistic database $\mathscr{D}$: $Q(\mathscr{D}) = \cup_{W \in PWS} Q(W)$.

The query over probabilistic database translated in relational algebra with the extension of sentence propagation:

- **Projection** $\pi_a(R)$

$$(< a_i >, \varphi) \in \pi_a(R) \iff \begin{cases} (< a_1, .., a_n >, \varphi) \in R \\ a_i = a \end{cases}$$

- **Selection** $\sigma_p(R)$

$$(t, \varphi) \in \sigma_p(R) \iff \begin{cases} (t, \varphi) \in R \\ p(t) = true \end{cases}$$

- **Cartesian product** $R \times S$

$$(t_1 t_2, \varphi_1 \wedge \varphi_2) \in R \times S \iff \begin{cases} (t_1, \varphi_1) \in R \\ (t_2, \varphi_2) \in S \end{cases}$$

- **Join** $R \bowtie_p S$

$$R \bowtie_p S = \sigma_p(R \times S)$$

For example, the query of Leopard under 6 years old resulted as $Q_1$2.5. The query plan is interpreted as: from the result of joining relations S and A, joined tuples with age smaller than "6 years" are selected. Projection is applied to the resulting tuples to leave only the desired attributes.

$Q_1 = \pi_{cat,species}(\sigma_{age<6,species=Leopard}(S \bowtie_{s.cat=a.cat} A))$

cat	species	φ	p
Mufasa	Leopard	$X = 1 \wedge B = 2$	0.4
Simba	Leopard	(F=1 $\wedge$ X=1) $\vee$ (F=2 $\wedge$ Y=1)	0.75

Figure 2.5: The result query for under 6-year-old Leopard

The probability of each tuple can then be calculated from sentence as follows[17]:

$$\begin{aligned}
P(X = a \wedge Y = b) &= P(X = a) \times P(Y = b) && X \neq Y \\
P(X = a \wedge X = b) &= 0 && a \neq b \\
P(X = a \vee X = b) &= P(X = a) + P(X = b) && a \neq b \\
P(X = a \vee Y = b) &= P(X = a) + P(Y = b) - P(X = a \wedge Y = b) && X \neq Y
\end{aligned}$$

In *Big cat* database, example of calculating some probabilities:

- Mufasa is not Leopard:
  $P(\neg X = 1) = 1 - 0.8 = 0.2$
- Mufasa and Scar are both Leopard:
  $P(X = 1 \wedge Y = 1) = 0.8 \times 0.7 = 0.56$
- Mufasa is either Leopard or Jaguar:
  $P(X = 1 \vee X = 2) = 0.8 + 0.1 = 0.9$

On an engineering level, DuBio was built on top of PostgreSQL. To describe rva, the database adds new data type called “dictionary”, where the key is rva and the value is probability. To represent a sentence and calculate the truth value of the sentence, DuBio offers a type that represents a Binary Decision Diagram(BDD)[2]. For detailed implementation of BDD, we refer to [14], [15].

2.4 Related Work

Theoretically, a query on a probabilistic database is a set of all possible answers, which are evaluated over all possible worlds. The set of answers could be huge and it is impractical to present all of them to the user. Thus, to date, there are 2 approaches considerably good to represent the query answer: ranking tuples and aggregation over imprecise values[18].

2.4.1 Ranking tuples

In this approach, the system computes all possible answer tuples, then ranks them in the decreasing order of their probabilities. Afterwards, top k tuples are chosen to be displayed to the user. Christopher Ré et.al came up with *Efficient Top-k Query Evaluation on Probabilistic Database*[6]. To speed up the query performance, instead of naively computing all probabilities then selecting top k; they approximate probabilities only to the degree needed to guarantee that the top k answers are the correct ones, and the ranking of these top k answers is correct[6]. The algorithm is called *multisimulation*. In general, it runs many simulation steps to identify and rank k answers, and only a few simulation steps to the others to ensure that they are not in the top k. *Multisimulation*'s running time scales almost linearly with respect to k. The fewer answers the user requests, the faster they are retrieved [6]. However this approach is suitable if the user is only interested in the top highest probabilities.

2.4.2 Aggregation over imprecise values

In this approach, the value of aggregate query is interpreted as expected value. It means the answer is defined to be the expected value of the aggregation operator over instances database of probabilistic database. With this approach, most aggregate functions can be computed easily using the linearity of expectation[18]. For example, expected count can be computed by a single pass over the set of tuples: $ECOUNT = \sum_{i=1}^{n} Prob(\varphi_i)$. Applying this representation, MayBMS[7] and Trio[4] probabilistic databases support expected sum and expected count operators. Nevertheless, this approach could lead to unintuitive query answers if the data values and their probabilities follow skewed and non-aligned distributions[19].

2.4.3 Histograms on Probabilistic Data

In tradional database management systems, *Histogram*[3] is a special type of column statistic that represents the distribution of data. It sorts values into a specific range of values, which is typically called "bins" or "buckets". In probabilistic database, Cormode and Garofalakis used histogram method to compress the large amount of data down to concise data synopses while retaining their statistical characteristics[8]. It then becomes feasible to evaluate queries on these summaries to give an approximate answer in a reasonable time. In general terms, similar tuples are divided into the same bucket, and the bucket boundaries are optimized to reduce the within-bucket dissimilarity (using the given error functions such as sum-squared-error, sum-absolute-error, etc.)

Chapter 3

ALGORITHMS

As we have mentioned earlier, we would like to have 3 representations for the result of aggregate query, which are exact aggregate(return all possible answers), histogram aggregate(return answers grouped into buckets) and topk(return answers in top highest probability worlds). Within the scope of this thesis, we dedicated our effort to algorithms for exact aggregate and topk. Regarding exact aggregate, we developed two functions called Combination and All Possible Worlds. For topk, we built a function call Topk Worlds using max heap data structure.

3.1 Exact aggregate

3.1.1 Notation and definition

Exact aggregate includes all possible aggregate query answers. In support of small sized table, this representative is still reasonable and accountable.

Notation Given relation R contains n tuples in probabilistic database presenting set of possible worlds $\mathscr{PWS} = \{(W_1, \phi_{W_1}), (W_2, \phi_{W_2}), \ldots, (W_p, \phi_{W_p})\}$, here, each sentence $\phi_{W_j} = (X = x) \wedge (Y = y) \wedge \cdots \wedge (Z = z)$ represents world W_j. A query for number of rows over R has form:

$$count(R) = \{(i, \Phi_i)\}, \text{ where } \begin{cases} i \in 0, 1, \ldots, n \\ \Phi_i = \vee \phi_{W_j} \mid count(R \in W_j) = i \end{cases}$$

3.1.2 Method 1 - Combination of records

Intuition Every record has 2 statuses, either exist or non-exist in a world. Therefore, all subsets of n records presents all possible outcomes. We started with this brute force algorithm to simply guarantee to find the correct solution and know how far this inefficient approach could lead to.

Syntax SQL query form:

```
SELECT count, STRING_AGG(bdd, '|')
FROM dicts, COMBINATION('select bdd from R', dict)
WHERE dicts.name='mydict'
GROUP BY count;
```

The Algorithm is shown in Algorithm 1. The function is written in PL/SQL, its argument is a query string which selects record sentences. We generate all subsets of the array of sentences, then looping through the subsets to build its sentence and calculate its size. Finally, we group all the subsets with the same size, and using `STRING_AGG` to unite their sentences. DuBio provides `AGG_OR` function to aggregate BDD through operator "or". To group those sentences, `AGG_OR` requires plenty of time to concatenate a large BDD so we simply used `STRING_AGG` instead.

Complexity The complexity of this algorithm is exponential, i.e. 2^n because it loops through all subsets of n records. Basically, this algorithm depends strongly on the number of records. The uncertainties of the database such as the number of random variables or number of alternatives are expected not to affect its performance.

Discussion This approach is straight forward to implement and easy to understand, however it is slow and inefficient. Because of its complexity, we consider it as a good solution for the database with few records but rich in uncertainty.

Algorithm 1: COMBINATION(query, dict)

Data: query = SELECT bdd FROM R
Result: $[(0,\phi_0),(1,\phi_1),..,(n,\phi_n])$
Initialize
sentences = EXECUTE query; // $[\varphi_1...\varphi_n]$
all_combinations ← sentences.powerset; ; // returns all subsets of an array
for *comb in all_combinations* **do**
  $\phi \leftarrow Bdd(1)$;
  $size \leftarrow 0$;
  for φ *in sentences* **do**
    if $\varphi \in comb$ **then**
      $\phi \leftarrow \phi \wedge \varphi$;
      $size \leftarrow size + 1$;
    else
      $\phi \leftarrow \phi \wedge \neg\varphi$;
    end
  end
  if *not isfalse*(ϕ) *and prob(dict,*$\phi) > 0$ **then**
    RETURN NEXT $size, \phi$
  end
end

Figure 3.1: PseudoCode of the Combination of records Algorithm.

3.1.3 Method 2 - All possible worlds

Intuition This approach is intuitively suitable with possible worlds theory. We simply build all possible worlds from dictionary, then aggregate on each world as usual.

Syntax SQL query form:

```
SELECT (q.c).count, STRING_AGG((q.c).bdd, '|')
FROM (
    SELECT UNNEST(POSSIBLE_WORLDS_COUNT(R.bdd, dicts.dict)) AS c
    FROM R, dicts WHERE dicts.name='mydict'
) q
GROUP BY count;
```

The Algorithm We implemented a custom aggregate `POSSIBLE_WORLDS_COUNT`, its state transition function is `POSSIBLE_WORLDS_ACCUM`("The state transition function takes the previous state value and the aggregate's input value(s) for the current row, and returns a new state value"[21]). We described the state transition function in Algorithm 2. Here, the input value is "row's sentence", the state value is an array of world's sentence and their current count. For each row, we extract its sentence variables, collect all alternatives of the variables, then extend the current list of worlds with this new variables. Next we update the aggregate result in the new list of worlds. Finally, we group all the worlds that have the same aggregate result, the sentence of each world is grouped by `STRING_AGG`.

Complexity The complexity of this algorithm is also exponential. Basically, it is the size of possible worlds, if the probabilistic database has v variables, each variable has a alternatives, the complexity is a^v.

Discussion In comparing to the latter algorithm, this one should perform more efficiently. It depends strongly on the uncertainties of probabilistic databases, i.e., the number of possible worlds w.

Algorithm 2: POSSIBLE_WORLDS_ACCUM(φ, listOfWorlds)

```
Data: record sentence φ, listOfWorlds[world sentence φ, count c] loW
Result: new loW
Initialize
array vars ← φ.vars ; // random variables in sentence φ
array worlds ← build_worlds(vars) ; // return worlds includes
 those variables
array newLoW ← []; // store new worlds and count result
for world in worlds do
    for φ, count in loW do
        φ ← φ∧ world;
        if not isfalse(φ∧ φ) then
            count ← count + 1;
        end
        newLoW.add(φ, count);
    end
end
return newLoW;
```

Figure 3.2: PseudoCode of the All possible worlds Algorithm.

3.2 Approximate aggregate

Approximate aggregate answers aggregate query approximately by grouping the answers in buckets or returns answers in top k probable worlds. The approximate representation is expected to support larger sized table.

3.2.1 Notation and definition

Buckets aggregate divides all the possible count values into buckets. Using this representation, we can see roughly the peaks, skews or outliers over the distribution values of count.

Notations Each histogram bucket $b = [s, e]$ consists of a start point s and end point e and covers $|b| = e - s + 1$ possible counts. Technically, the sentence of bucket b is the union of all worlds where count on this world i belongs to intervals $[s, e]$

$$b\_count(R, b) = \{([s_i - e_i], \Phi_i)\}, \text{ where } \begin{cases} i \in 1, \ldots, b \\ \Phi_i = \cup \phi_{W_j} \mid count(R \in W_j) \in [s_i - e_i] \end{cases}$$

Top k aggregate returns aggregate in the top k probable worlds.

Notations Given relation R contains n tuples in probabilistic database presenting set of possible worlds $\mathscr{PWS} = \{(W_1, \phi_{W_1}), (W_2, \phi_{W_2}), \ldots, (W_p, \phi_{W_p})\}$, here, each sentence

$\phi_{W_j} = (X = x) \wedge (Y = y) \wedge \cdots \wedge (Z = z)$ represents world W_j, its probability is defined as $P(W_j)$. A top k query aggregate over R has form:

$$top\_count(R,k) = \{(i,\Phi_i)\}, \text{ where } \begin{cases} i \in 0,\ldots,n \\ \Phi_i = \cup\phi_{W_j} \mid count(R \in W_j) = i \\ \forall j \in [1\ldots k], \forall p > k, P(W_p) \leq P(W_j) \end{cases}$$

3.2.2 Method 3 - Top k ranking

Intuition Natural representation to users by restricting the number of worlds to k, since large number of worlds have low probability, users are only interested in top highest probabilities.

Syntax SQL query has form:

```
SELECT (q.c).count, STRING_AGG((q.c).bdd, '|')
FROM(
    SELECT UNNEST(TOPK_WORLDS_COUNT(TOPK_WORLDS(dicts.dict, k), R.bdd))
    AS c FROM R, dicts WHERE dicts.name='mydict'
) q
GROUP BY count;
```

The Algorithm Firstly, we coded a function `TOPK_WORLDS` to find top k probable worlds, which is described in detail in Algorithm 3. It is assumed that the alternatives of each random variable are sorted by probability. We begin by choosing the highest probable alternative of each random variable to build the first world. Next, by replacing each random variable's alternative with its successive alternative, we have new v worlds. We keep those worlds sorted by using max heap data structure. The root always gives the greatest probable world.

After having topk worlds, we do the aggregation in this k worlds by implementing a custom aggregate function called `TOPK_WORLDS_COUNT`, and its state transition function is `TOPK_WORLDS_ACCUM` (Algorithm 4). Finally, we also group the worlds with the same count result and world's sentences are grouped by `STRING_AGG`.

Complexity The complexity of `TOPK_WORLDS` is log-linear. Specifically, the cost for building max heap is $O(xlogx)$, x is the number of nodes in the heap. Here, we loop k times to collect top k worlds, and in each loop we add v worlds to the the heap. Thus, there are totally $v * k$ nodes in the heap, and the complexity is $O((vk)log(vk))$.

Discussion Because of its log-linear complexity, we expect this algorithm to perform well with database that has large number of records and variables. It is worth to mention that the result in TopK Worlds may be less than k answers. For example, if all top k worlds has the same count value, there will be only 1 answer to return. In the last step, we group the worlds by its count, so the final result is not always ordered by probability value.

Algorithm 3: TOPK_WORLDS(dictionary dict, integer k)

```
Data: A dictionary has random variable's alternatives sorted
Result: top k probable worlds
Declare
int v ← dict.vars.size
array alt_size ← [v_0.alts.size, v_1.alts.size, ..., v_v.alts.size]
heap maxHeap ; // max heap sorted by probability
2D-array bestWorlds ← [] ; // store the top k worlds
2D-array seens ← [] ; // store processed node
Initialize
top1world ← [0_1, 0_2, ..., 0_v]
maxHeap.push(top1world)
repeat
    currBestWorld ← maxHeap.heap_pop;
    bestWorlds.add(currBestWorld);
    for int i=0; i<v-1; i++ do
        if alt_size[i] > currBestWorld[i] + 1 then
            nextBestWorld ← currBestWorld;
            nextBestWorld[i] ← nextBestWorld[i] + 1;
        else
            Continue;
        end
        if nextBestWorld ∉ seens then
            maxHeap.heap_push(nextBestWorld);
            seens.add(nextBestWorld);
        end
    end
until bestWorlds.count = k or maxHeap.empty;
return bestWorlds;
```

Figure 3.3: PseudoCode of the TOPK_WORLDS.

Algorithm 4: TOPK_WORLDS_ACCUM(listOfWorlds, topKWorlds, bdd φ)

```
Data: record bdd φ, topKWorlds, listOfWorlds[{world bdd ϕ, count c}]
Result: new listOfWorlds
Initialize
for ϕ in topKWorlds do
    count = listOfWorlds[ϕ];
    if not isfalse(φ ∧ ϕ) then
        listOfWorlds[ϕ].count = count + 1;
    end
end
return listOfWorlds;
```

Figure 3.4: PseudoCode of the TOPK_WORLDS_ACCUM.

Chapter 4

EXPERIMENTAL EVALUATION

4.1 General Settings

Setup Our experiments were run on a commodity hardware. The computer is a MacBookPro 16,1 with 6-Core Intel Core i7 at 2.66 GHz running the macOS Big sur operating system. DuBio Database is installed on Postgres 14.1. The whole set of experiments is built and managed as a web application programmed in Ruby on Rails[1].

Data We evaluate the aggregate functions over synthetically generated data. We call it "cat breed" database. The database has one table which includes cats and their uncertain breeds. We use Faker[2](a library for producing fake data) to generate cat names and their breeds. For each random variable, we generate randomly 2-4 alternatives if not specified. Regrading probability of each alternative in a random variable, the first alternative has probability p_1, which is randomly between $[0..1]$, the second alternative has probability p_2 randomly between $[0..p_1]$ and so on. This condition ensures the alternatives have probabilities in descending order. DuBio dictionary also normalizes automatically the probability value of alternatives to make sure $\sum p_i = 1$.

Measurement To measure the execution time of the aggregate query, we use SQL command `EXPLAIN`[20], i.e.,
`EXPLAIN(ANALYZE TRUE, FORMAT JSON, TIMING FALSE) statement`; where:

- `ANALYZE` causes the statement to be executed. The option is set `TRUE` to get actual execution time(milliseconds).
- `FORMAT` specifies the output format. The format is set to `JSON` so that programs can parse information conveniently.
- `TIMING` includes actual startup time and time spent in each node in the output. We are merely interested in the overall execution time of the statement, so this option is set to `FALSE`. It also reduces timing overhead of `EXPLAIN ANALYZE` compared to executing query normally.

[1] https://github.com/stubbornfox/dubio/
[2] https://github.com/faker-ruby/faker#creature

It's worth noting that, run times measured by `EXPLAIN ANALYZE` do not include the network transmission costs and I/O conversion costs because no output rows are delivered to the client[22]

The ideal execution time for a query is from 1s to 2s, and we have set 30s as the maximum time out for aggregate query. Usually, 30s is the default connection timeout in a system, as well as a decent amount of time for user's patience. We use Class Timeout in ruby to auto-terminate a potentially long-running query if it hasn't finished in 30 seconds.

```
Timeout::timeout(30) {
  ActiveRecord::Base.connection.execute(statement)
}
```

Methodology To ensure fairness to all experiments we run `VACUUM ANALYZE` on the tables before each experiment.

4.2 Experimental Details

Overall, the experiments are used to evaluate the performance of algorithm and how factors such as number of random variables, number of alternatives, complexity of the sentence affect the aggregate functions. The following are details of the 5 experiments, in which, experiments from A to D is conducted for all algorithms separately, regarding TopK algorithm, we use k=5. Experiment E is conducted for TopK only.

4.2.1 Experiment A

Increasing number of tuples and variables gradually. The scale of a probabilistic database is different when compared to a traditional database. Beyond the factors like number of tuples, columns or data type, probabilistic database also accounts for the size of possible worlds this database could present(which is decided by random variables in DuBio).

- **Goal** We use this experiment to find out the feasible largest number of tuples our algorithms can handle. It also helps to observe the trend of execution time on the scale of number of tuples and random variables. By this experiment, we are able to answer research questions **RQ1**, **RQ2**, **RQ3**, **RQ4**.
- **Setup** The detail setup is diagrammed as Figure 4.1
  - DICTS array has 500 random variables, there are uniformly between 2-4 alternatives per variable, probability is chosen randomly based on condition: $1 > p(Z=i) \geq p(Z=j) > 0$, $i < j$.

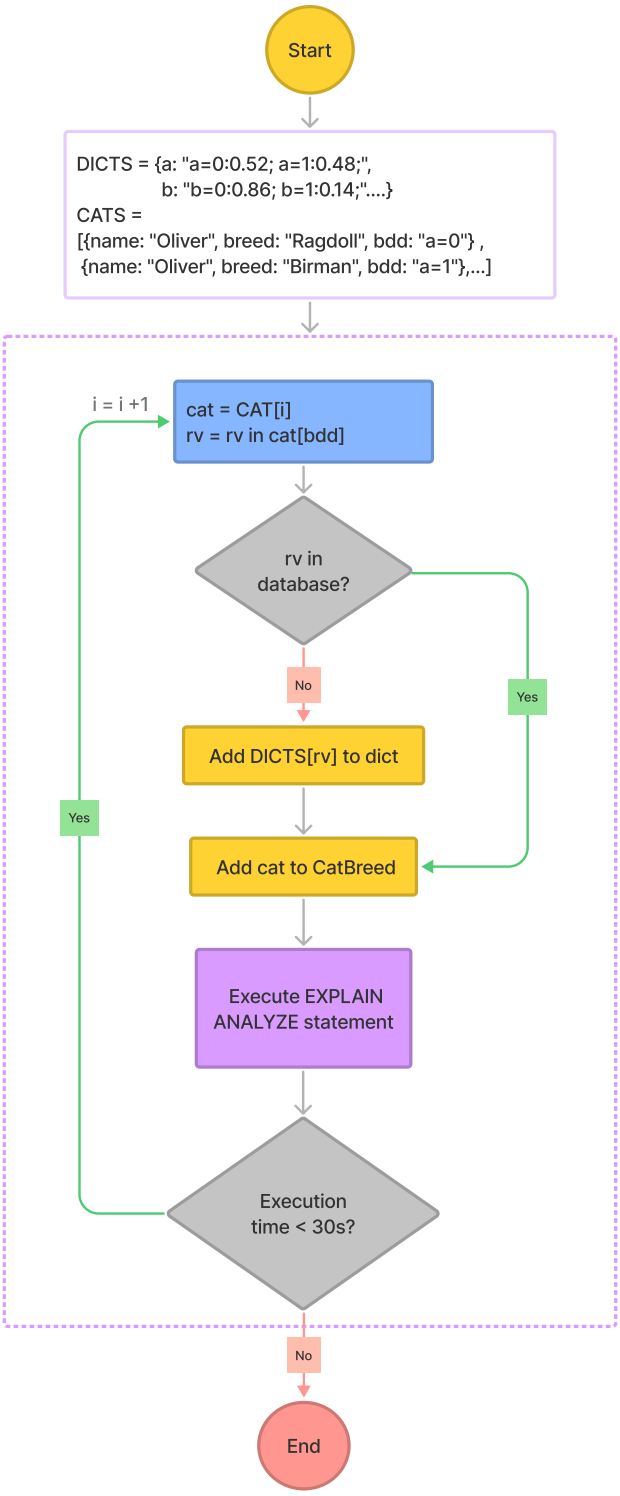

Figure 4.1: Experiment A Setup

– CATS array has 1000 records, represents about 300 cats, each cat has 2-4 possible breeds.
– For each cat in CATS:
    1. If random variable r in cat's sentence is not yet in the database, then add r to dictionary.
    2. Add cat to CatBreed table.

3. Try to get execution time, if it is more than 30s, end loop; else continue

4.2.2 Experiment B

Increasing number of random variables while keeping the number of records constant. As we mentioned earlier, there are 2 dimensions a probabilistic database can expand: possible worlds and quantity of data. This experiment focuses on how the number of possible worlds would affect the aggregate query.

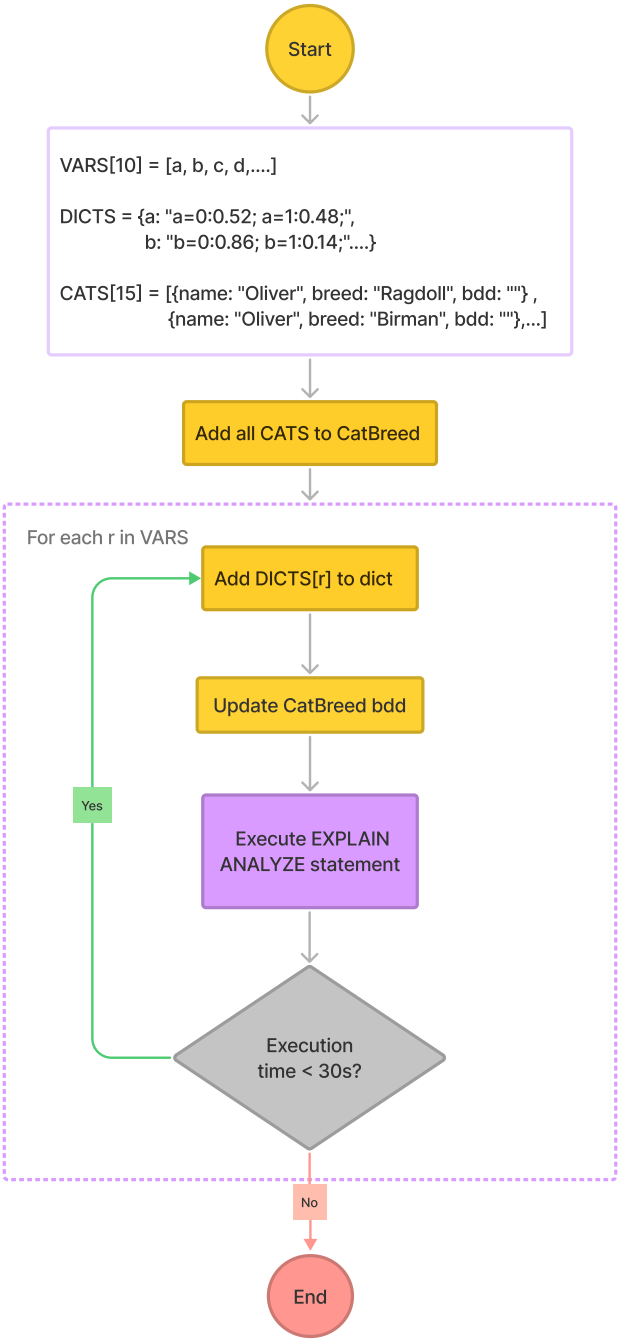

Figure 4.2: Experiment B(n=15, r=1..10, a=2)

- **Goal** We use this experiment to find out how the execution time changes w.r.t the number of random variables. This experiment answers research question **RQ5.1**
- **Setup** The detail setup is diagrammed as Figure 4.2. We setup experiments on 1-10 random variables for a fixed number of 15 tuples.
  - DICTS has 10 random variables, 2 alternatives per variable.
  - Add 15 records to CatBreed.
  - For each random variable r in DICTS:
    1. Add r into dictionary
    2. Assign random variables in dictionary uniformly to tuple's sentence
    3. Get execution time, exit if it takes more than 30s

4.2.3 Experiment C

Increasing number of alternatives per random variables while keeping all tuples constant. This experiment is similar to experiment B, which studies how the number of possible worlds could affect the execution time. In DuBio, possible worlds size could be changed by either adding more random variables or giving more alternatives of each random variable.

- **Goal** This experiment studies how number of alternatives of random variables affects the execution time, which answers research question **RQ5.2**
- **Setup** The detailed setup is diagrammed as Figure 4.3. We experiment for a fixed number of 15 tuples and 7 random variables. The number of alternatives per random variable varies from 2 to 5.
  - DICTS has 7 random variables, each has 2 alternatives.
  - CATS has 15 cats, 7 random variables assigned among their sentences.
  - Add DICTS and CATS to database
  - For each alternative in 2..5:
    1. Get execution time, exit if it takes more than 30s
    2. Add 1 more alternative to each random variable, its probability is 0.001(try to ensure random variable in dictionary has alternatives in descending order of probability)

4.2.4 Experiment D

Increasing the complexity of sentence in each tuple. As mentioned earlier, DuBio uses concept sentences, which are propositional formulae of random variable assignments. The complexity of a sentence here is the number of different variables in each

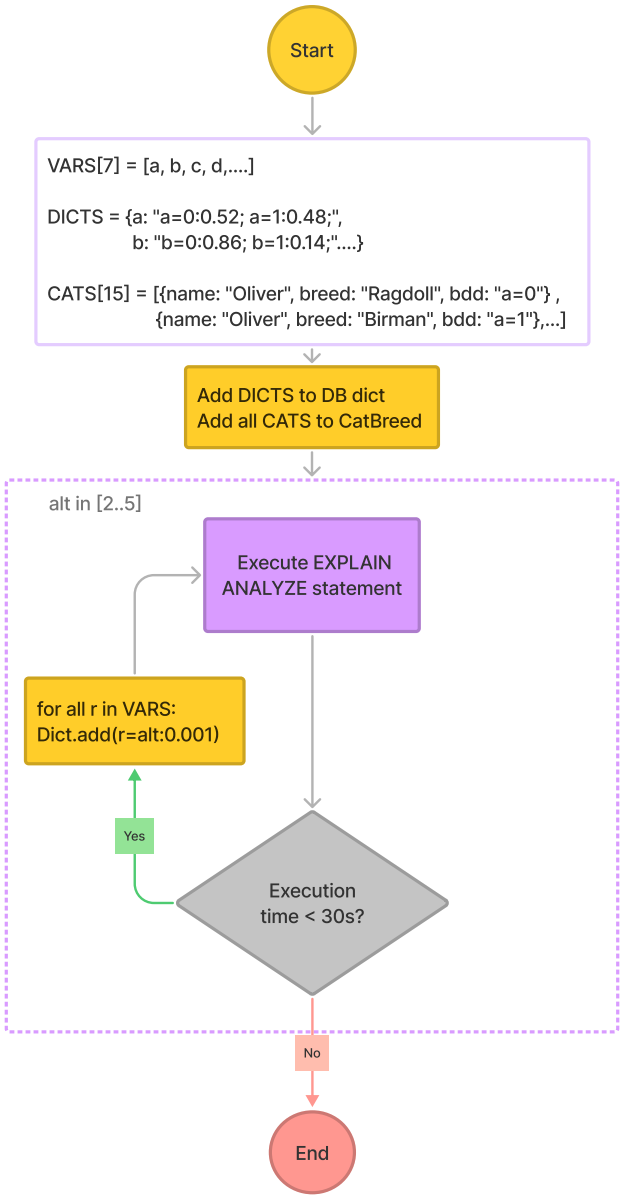

Figure 4.3: Experiment C(n=15, r=7, a=2..5)

sentence. This experiment basically mimics a complex query which includes conditions joining multiple tables. We would like to see how the aggregate function is affected under complicated sentences.

- **Goal** This experiment studies the effects of complexity of the tuple's sentence on the execution time of the algorithms, which answers research question **RQ5.3**
- **Setup** The detailed setup is diagrammed as Figure 4.4. We ran the experiment from 1 to 15 tuples and used in total 5 random variables, the number of variables appearing in each sentence is 1, 2, 3.
  - DICTS has 5 variables, each has 2-4 alternatives
  - SENTENCES has 3 arrays bdds, where the first is list of bdd included 1 random variable, the second is list of bdd included 2 different random vari-

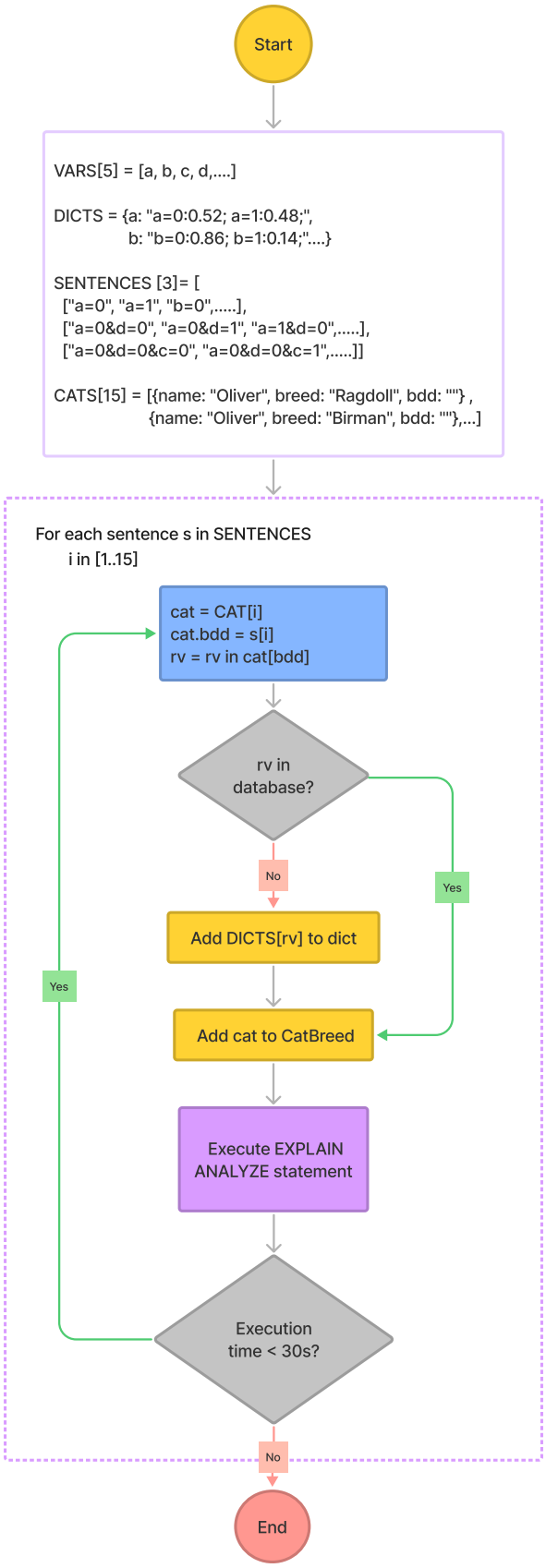

Figure 4.4: Experiment D(n=1..15, 1..3 random variables in a sentence)

ables, and the third one is list of bdd included 3 different random variables.

The operator in those bdds is $\wedge$ to simulate when we join multiple tables.

– For sentences in SENTENCES:
  1. Clear database, assign $i = 0$
  2. Loop until i=15:
     (a) Add variables in the $sentences_i$ to database dictionary
     (b) Assign $sentences_i$ to cat_i, add cat to CatBreed
     (c) Get execution time, exit if it takes more than 30s; else increase i by 1.

4.2.5 Experiment E

Increasing k in Topk. Topk's complexity is a log-linear of k*v, we would like to confirm this dependency of Topk on k.

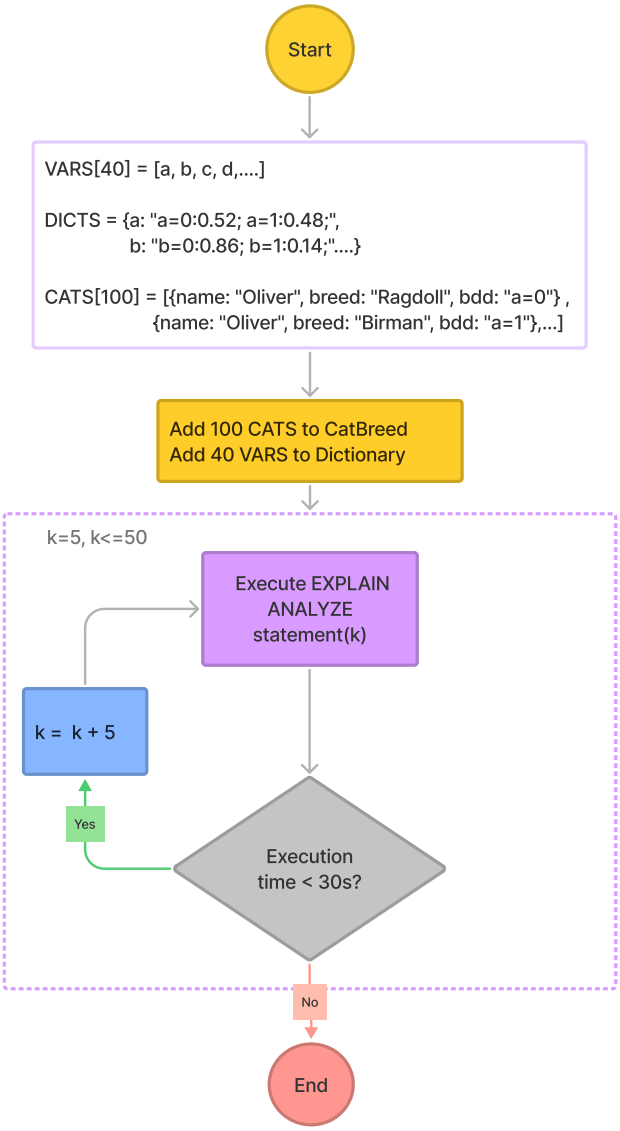

Figure 4.5: Experiment E(n=100, k=[5,10,...,50]

- **Goal** This experiment studies the scalability of TopK on k, which answers research question **RQ2**
- **Setup** The detailed setup is diagrammed as Figure 4.5. We experimented this for a fixed number of 100 records with k varying in [5,10, 15 ... 50]
  - Add 40 random variables and 100 records to database, assign k = 0
  - while k ≤ 50:
    1. k = k + 5
    2. Get execution time for Topk, exit if it takes more than 30s

4.3 Results

4.3.1 Experiment A

The running time of 3 algorithms regarding records is shown in Figure 4.6. On commodity hardware, Combination, All possible worlds and Topk are able to aggregate over **17, 26 and 339** records, respectively under 30s. As the number of records increases, Combinations and All Possible worlds bend upward more steeply. They both have running time increasing exponentially, however we notice that Combination explodes faster than All possible worlds. This happens because time complexity of Combination is 2^n, while All possible worlds is on average $2^{\frac{n}{3}}$ (n records is annotated by $\frac{n}{3}$ variables). Regarding TopK, we observe that it has linearithmic run time. Indeed, its complexity is O(vk*log(vk)). Here, k=5 and v=$\frac{n}{3}$, so its run time complexity over n is O($\frac{5n}{3} * log\frac{5n}{3}$).

In general, to query all possible answers, All possible worlds is favored over Combination. Even though All possible worlds reached the limit of 26 records, it can be safely used on the result of a query when the intermediate result before aggregation has max 26 tuples. These circumstances may hold quite often in practice. To query aggregate approximately, TopK with log-linear complexity is adequate to query over three hundred records.

On a server hardware, e.g. 10 times faster than our commodity computer, for All possible worlds and Combination, the increase in size of n is about n + 3 (to be precise, n + $log_2 10$). In fact, the increase in problem size for algorithms with exponential growth rate is by addition of a constant. Therefore, with the server hardware 10 times faster, All possible worlds will be able to process for 29 records, or with hardware 100 times faster, it can only reaches 32 records max. Meanwhile, running TopK on a server hardware will improve the size of n by a multiplicative factor. Top K has log-linear complexity, which means it grows faster than linear(O(n)) but slower than quadratic(O(n^2)). Therefore, on server hardware 10 times faster, TopK will increase size of n by a factor between [$\sqrt[2]{10}, 10$].

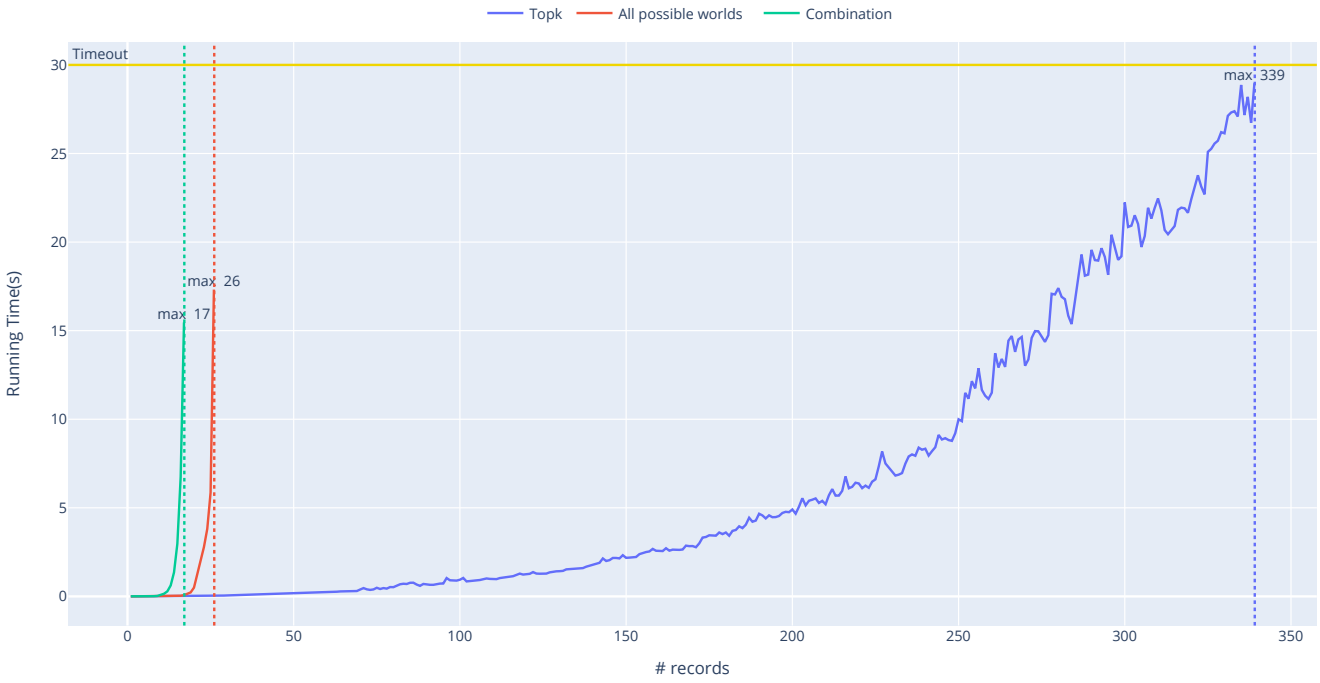

Figure 4.6: Running time of 3 methods in total

4.3.2 Experiment B

Evaluating effect of v on running time is given by the graph in Figure 4.7, which shows running time taken for $n = 15$ as v changes from 1 to 10. As we can see, Combinations and All Possible worlds running time scales exponentially while TopK running time grows linearly as v increases.

All possible worlds complexity is a^v, so the exponential trend proved this hypothesis.

Regarding Combination, we expected it is not affected by increasing v. However, the construction of the BDD in this algorithm, e.g. $\varphi_1 \wedge \varphi_2 \cdots \wedge \varphi_n$ costs exponential growth. When v is small, this world sentence becomes simple, e.g, all records are only annotated by 2 variables A and B, then a world sentence $\Phi = (A = 1) \wedge (B = 1) \wedge \ldots (A = 1) \wedge (B = 1)$ eventually shrinks to $(A = 1) \wedge (B = 1)$. When v is large, the world sentence Φ remains its size and costs time to construct. So the trend here reflects the exponential cost of constructing BDD when v changes.

For TopK, we mentioned its complexity is $vk * log(vk)$, the trend confirmed our expectation. In the implementation of TopK, while being aware of the cost of BDD, we tried to restrict constructing BDD, e.g. to present a world's sentence we use a string "$A = 0 \wedge B = 1 \ldots$". Therefore, the cost for construction of BDDs does not affect much like Combination. Precisely, the number of BDD constructed in Combination is sufficiently large $2^n = 2^{15} = 32768$, while the number of BDD constructed in TopK is $n \times k = 5 \times 15 = 75$(we only build BDD to check if a record belongs to a world, BDD of the world Φ is restricted to only variables appearing in φ).

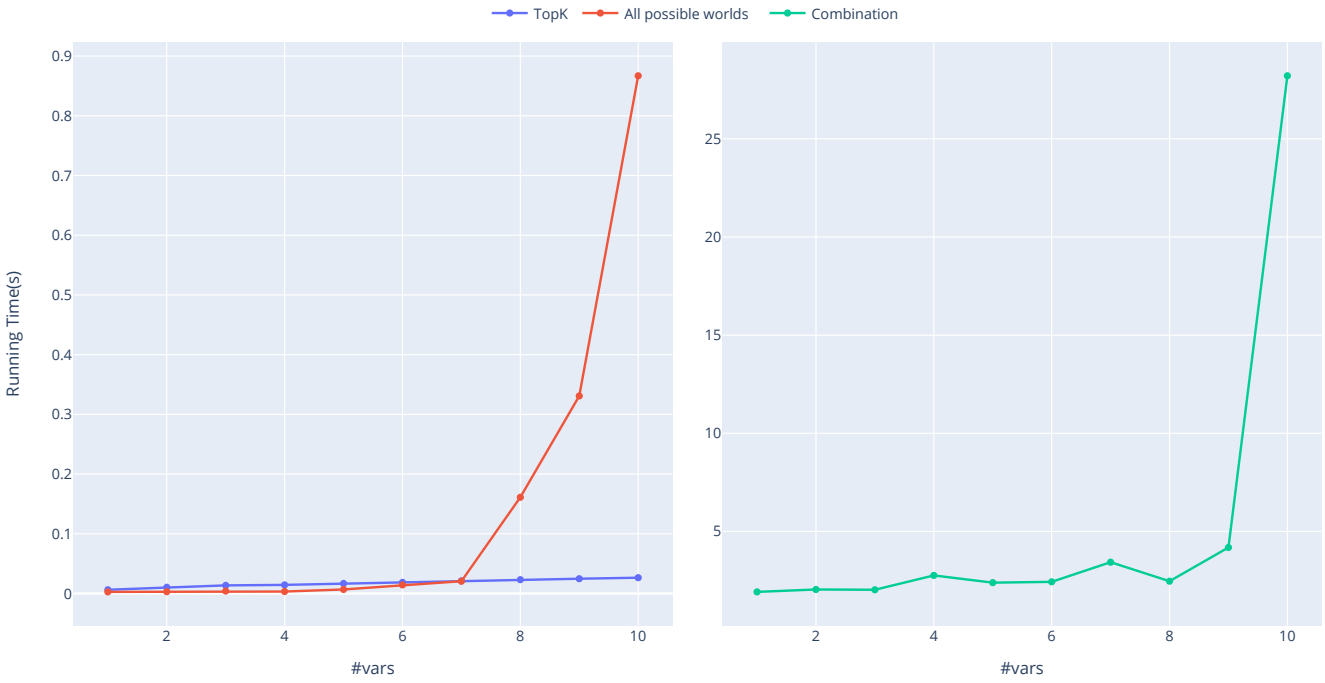

Figure 4.7: Effect of #vars on running time(n=15)

4.3.3 Experiment C

Figure 4.8 shows result of running time with different a, in which, y-axis is logarithmic scale of running time, x-axis is a, changing from 2 to 5. For different a per variable, All possible worlds has ln(running time) increasing linearly, while Combination and TopK stay flat.

As a increases, the running time of All possible worlds scales exponentially. This proves our expectation about scale of All possible worlds to a, because the complexity is a^v, here $v = 7$, so the running time represents the same trend with function $f(a) = a^7$. In a short range of a and the graph $ln(f(a))$ grows linearly.

While Combination and TopK are not affected, because their complexity does not depend on a. Besides, in this experiment, new alternatives added to dictionary and all record's sentence φ is unchanged so any BDD constructed in this 2 algorithms should stay the same even though a changes.

4.3.4 Experiment D

Effect of sentence's complexity on running time is given by the graph in Figure 4.9, which shows running time taken for multiple v in a sentence φ. The solid line shows running time when record's sentence includes 1 variable, dot line shows running time for sentence with 2 different variables and dash line shows running time for sentence with 3 different variables.

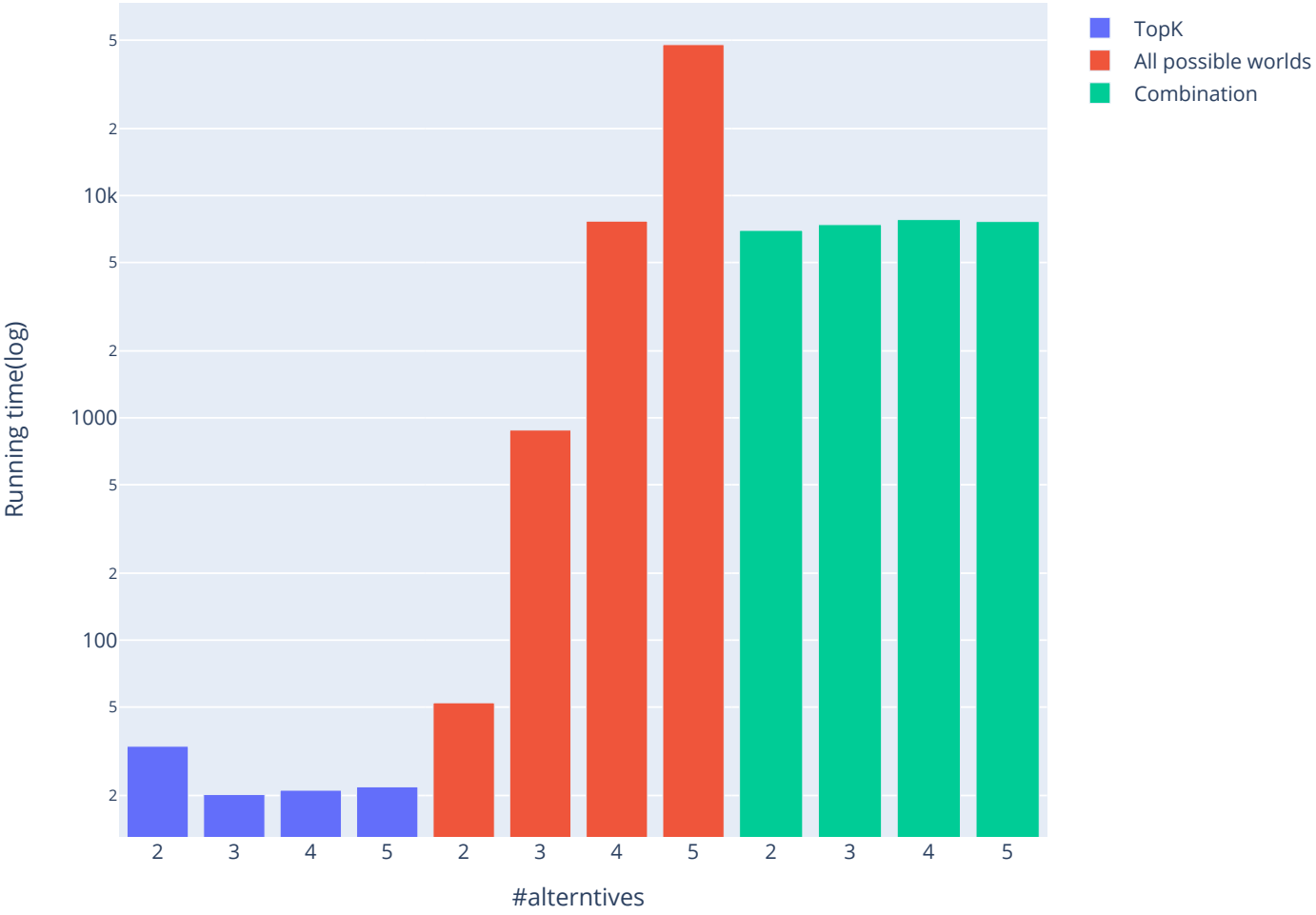

Figure 4.8: Effect of #alternatives on running time(n=15)

TopK has 3 lines twisted together and does not show a clear trend, Combination has 3 lines overriding each other, which means there is no clear effect of different v in a sentence for these 2 algorithms. This happens because Combination depends strongly on n and TopK complexity only depends on k and v.

Regarding All possible worlds, it shows that sentence with 3 variables takes more time than sentence with 2 variables. Similarly, sentence with 2 variables takes more time than sentence with 1 variable. Multiple different variables appearing in a sentence will cost more time. Since in this algorithm, the possible worlds set are expanded across the row, the more variables in sentence of a row, the larger the set will expand.

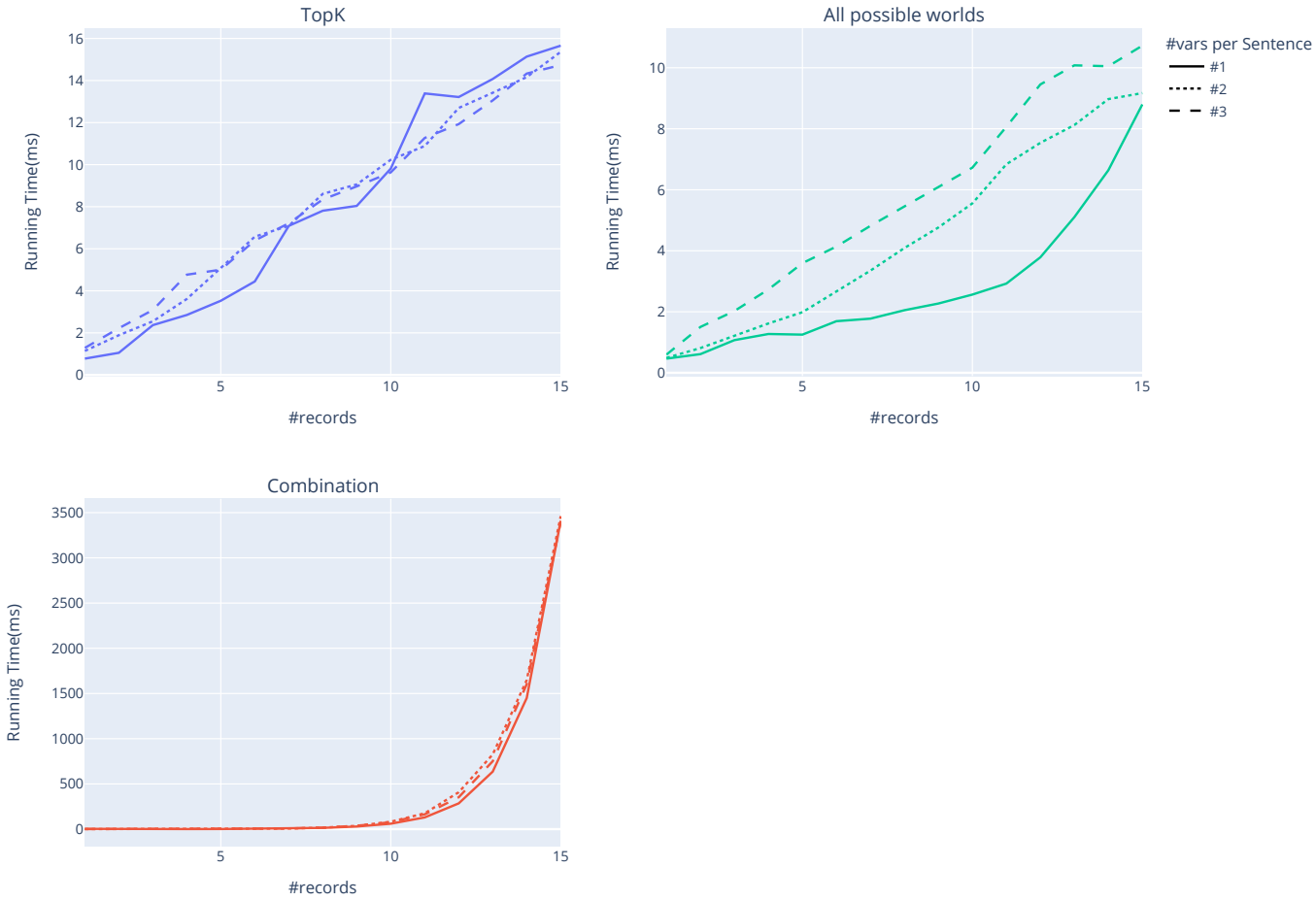

Figure 4.9: Effect of sentence's complexity on running time

4.3.5 Experiment E

Effect of K on running time of TopK is shown in Figure 4.10, the query is executed on 100 records. For top 5 and top 10, it takes 0.7s, and 1.7s respectively. For top 50 of 100 records, it takes 28.6s (almost up to the timeout). We observe that the running time is almost linear. Mathematically, with $v = 31$ the running time should represent trend of the function $f(x) = 31k * log(31k)$. So this result fulfills our expectation.

In this experiment, we also compared between Topk and top k of All possible worlds algorithm. To simulate Topk, we sort possible worlds by their probability and limit to k. The result is shown in Figure 4.11. We spot that with small k (e.g. top1, top5), TopK takes less time than top k of All possible worlds. When $k = 10$, TopK and top k of All possible worlds takes similar time. However, with $k = 15$ TopK takes more time. We also observe that k does not affect top K of All possible worlds.

Even though All possible worlds scales faster, TopK will take more time when k becomes larger. This happens because the O(All possible worlds) formula doesn't have a k in it (confirmed by the experiments to be independent of k) and the O(TopK) formula depends on k by $k * logk$ when keeping v constant, so it grows with this trend(also

confirmed by the experiment that it grows a little bit more than linear).

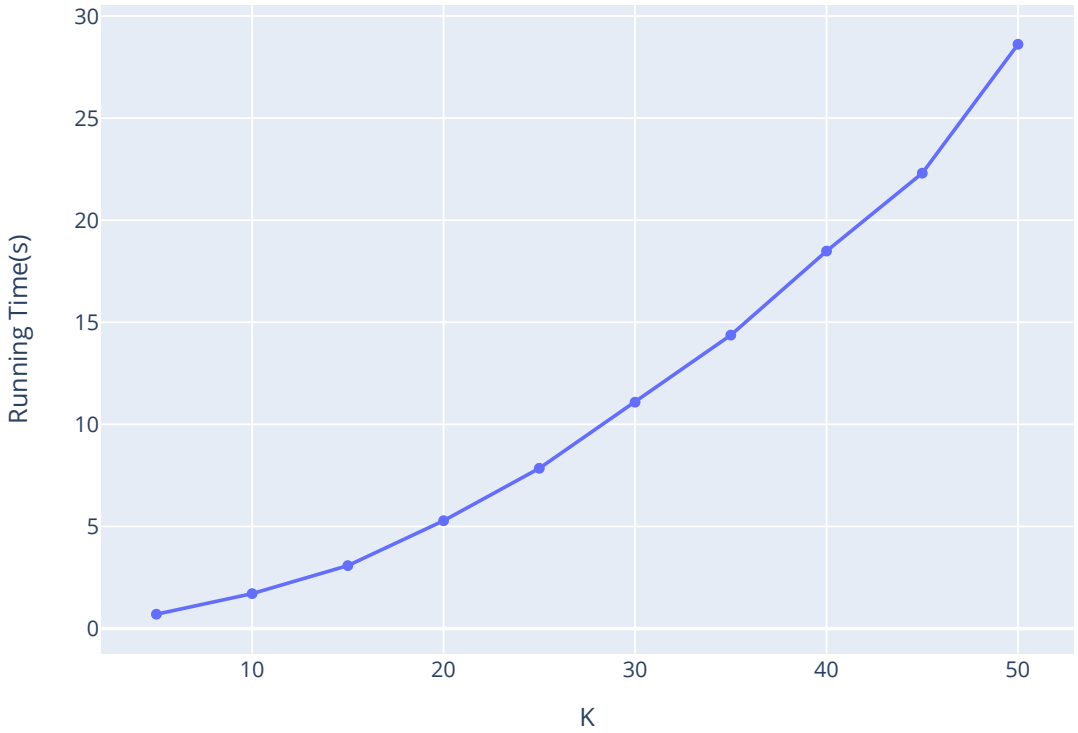

Figure 4.10: Effect of K on running time(n=100)

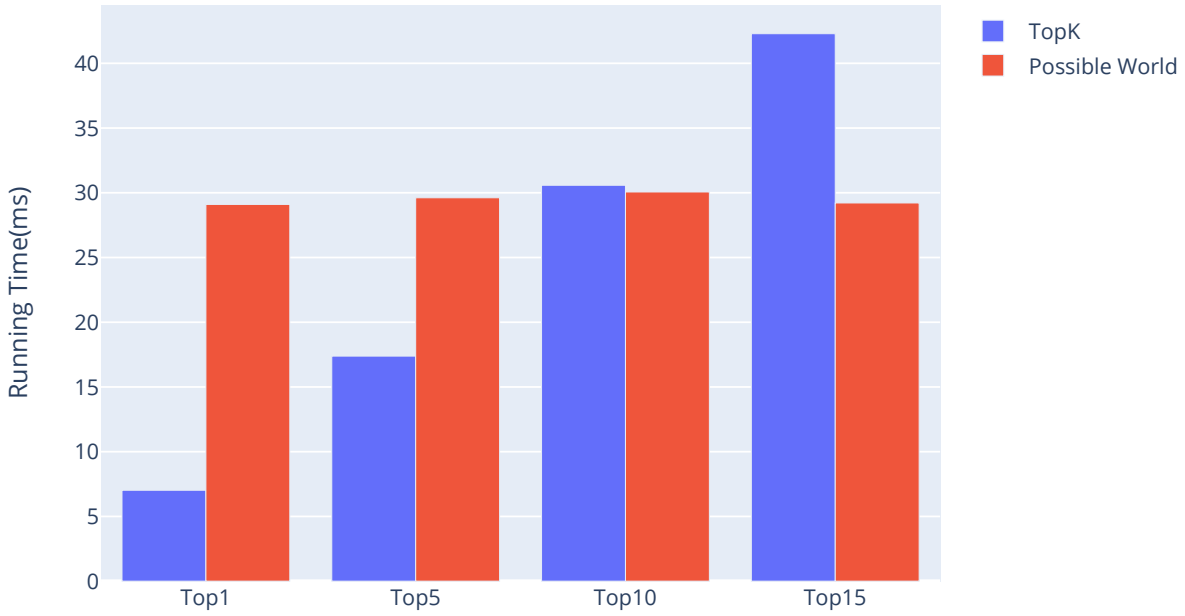

Figure 4.11: Topk versus top k of All possible worlds(n=15)

Chapter 5

DISCUSSIONS

In this chapter, we discuss the ways to optimize and improve our algorithms. Then, we describe the limitations of this research.

5.1 Optimization

In Combination, we blindly generate combination subset of n records. However we can optimize this algorithm by checking correlation between n records, i.e. if 2 records is mutual dependent, it can be treated as 1 element; or 2 mutual exclusive records should never belong in the same subset.

In All possible worlds method, we build the full possible worlds representing all possible answers. However, the number of possible worlds could be refined by using negative proposition. For example, in 3 different worlds, A=1, A=2, A=3 could be reduced to 2 different worlds A=1 and !A=1 (if the table only includes alternative A=1).

In the implementation of all possible worlds, we do not restrict the construction of the BDD. If we apply the optimization like TopK(see subsection 4.3.2), this algorithm could have achieved a better performance.

In TopK implementation, we assumed that our variables have their alternatives sorted by their probabilities. We skipped the implementation of sorted dictionary. This could be always implemented in the future by maintaining sorted dictionary beforehand or sort those alternatives before calling TopK. In our experiments, the synthetic data is generated with alternatives sorted by their probabilities. One thing worth mentioning here is that we assumed the dictionary does not include unused random variables. However if it does, we can also optimize it by refining the dictionary to the table before processing.

5.2 Limitations

In this thesis, we restrict ourselves to a synthetic data with only one table `CatBreed`. The table uses synthetically generated sentences to represent uncertainty of a cat's breed. Also the probabilistic database model we used only handles discrete attribute values of probability. All experiments ran on commodity hardware only. Even though developing approximate aggregation as a histogram is our intention in the beginning, we still haven't figured out any effective algorithm. The TopK algorithm returns the aggregates for the k most probable worlds. An alternative form of topk, which returns exactly the most *m* probable aggregate answers, lets call it TopM. An algorithm could be constructed for this form as well.

Chapter 6

CONCLUSIONS

6.1 Conclusions

Our primary motivation is *to implement aggregates in probabilistic database, particularly DuBio*, which leads us to 5 Research Questions. We implemented 3 algorithms, and set up 5 experiments to evaluate their performance. In this section, we attempt to answer our research questions.

RQ1 *To present the result of queries exactly, how do we implement a function which returns all possible answers and their probabilities?*

To answer this question, we implemented 2 algorithms called Combination and All possible worlds. Combination generates all subsets of records, and group those subsets with the same size. All possible worlds traverses all relevant possible worlds in the database and computes a count for each. The aggregate function processes the row's sentence φ, taking the used random variables and builds the ongoing possible worlds. Next, if φ belongs to a world seen before, increase the count on that world.

RQ2 *To present the result of queries approximately, how do we implement a function which returns possible answers and their probabilities distribution as a top-k ranking or histogram?*

To build topk ranking function we used a sorted dictionary, which means alternatives of each random variables is sorted by their probabilities. Next we build top k worlds by picking the top alternatives of each random variables. To figure out the next best random variable's alternative, we use max heap to store candidate worlds built by picking next alternative of each variable. The world with highest probability is always root of that heap.

Computing histograms as an approach for obtaining approximate aggregate results is very promising. In this research, however, we focused on top-k only. A naive algorithm for computing a histogram is to group the result of an exact aggregate into buckets. This algorithm will have the same complexity as the exact aggregate algorithm.

RQ3 *What is the maximum number of records our aggregate functions can reach under a reasonable time, particular in DuBio(for both exact and approximate count)?*

Under 30s, on a commodity hardware, the number of records that could be aggregated is as follows:

#records		
Combination	All possible worlds	TopK(k=5)
16	26	339

In general, to represent exact query result, All possible worlds is favoured over Combination, because Combination scales w.r.t. the power of n and All possible worlds scales slower w.r.t. the power of v.

For TopK, it performs much better with log-linear complexity. We believe it could reach sufficiently larger number of records by using a server infrastructure.

RQ4 *What is the complexity of our algorithms?*

Algorithm complexity		
Combination	All possible worlds	TopK
$O(2^n)$	$O(a^v)$	$O(vk * log(vk))$

This time complexities are confirmed in our experiment A(Figure 4.6). Both Combination and All possible worlds running time has exponential growth rate, Combination scales faster. TopK has log linear growth rate.

RQ5 *How the uncertainties in probabilistic database affects our algorithms?*

RQ5.1 *How the number of random variables affects our algorithms?*

Combination and All possible worlds scale exponentially. Whilst, TopK scales near linearly with number of random variables

RQ5.2 *How the number of alternatives per random variable affects our algorithms?*

Depending strongly on the number of worlds, All possible worlds scales exponentially with the number of alternatives. While, Combination and TopK running time aren't affected.

RQ5.3 *How the complexity of sentences affects our algorithms?*

The complexity of sentence doesn't affect much the running time of TopK and Combination. However for All possible worlds, it obviously shows the increase in the running time.

Overall, to query all possible answers, All possible worlds is favored over Combination. However each algorithm has advantages and disadvantages varying on different circumstances. Even though All possible worlds can only reach to 26 records, it can be safely used on the result of a query when the intermediate result before aggregation has max 26 tuples. Combination stops at 15 records, but if the database has a huge number of variables and the table has not more than 15 tuples, this algorithm is a reasonable choice. To query aggregate approximately, TopK with log linear complexity is adequate over hundred records. Given the complexity and experiment result on a commodity hardware, we are able to estimate algorithm's performance on a server hardware. Combination and All possible worlds could only handle a couple of more records because of their exponential complexity. Meanwhile, TopK with log-linear complexity can improve the size of records by a multiplicative factor.

6.2 Future work

In discussion part(chapter 5), we show insights used to optimize our algorithms which could be implemented in the future:

- All possible worlds, we could refine the number of possible worlds by using negative propositional formula.e.g. $\{A=1, A=2, A=3, A=4\} \longrightarrow \{A=1, !A=1\}$. This optimization is expected to improve performance when a random variable has many alternatives but the table only relates a few of those alternatives.
- All possible worlds, we are aware it takes time to build BDD with multiple random variables, especially BDD to represent a world Φ. As we mentioned how we optimized it in result(subsection 4.3.2) and discussion part(chapter 5), we can have this algorithm perform better by reducing cost of constructing BDD.
- Implement sorted dictionary for TopK. During our implementation, we assumed that the variables alternatives are sorted by their probabilities. However, in practice, DuBio has alternatives sorted by their added time. We should sort alternatives every time new alternative added. This helps to bring insight when we look at the dictionary, we will notice which alternatives are more likely.

Histogram is a promising representation for aggregation result. We would like to focus on this representation in the future.

Experiments Experiment is run on commodity hardware only, we would like to run it on a server hardware to confirm our estimation. We also want to experiment the query on a real schema data which includes multiple tables and the query is a bit more complicated.

References

[1] Randal E. Bryant. "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams". In: *ACM Comput. Surv.* 24.3 (Sept. 1992), pp. 293–318. ISSN: 0360-0300. DOI: 10.1145/136035.136043. URL: https://doi.org/10.1145/136035.136043.

[2] Henrik Reif Andersen. "An introduction to binary decision diagrams". In: https://www.cs.utexas.edu/~isil/cs389L/bdd.pdf. Department of Information Technology, Technical University of Denmark. 1997.

[3] Yannis E. Ioannidis. "The History of Histograms (abridged)". In: *VLDB*. 2003.

[4] Jennifer Widom. "Trio: A System for Integrated Management of Data, Accuracy, and Lineage". In: *CIDR*. 2005.

[5] Rahul Gupta and Sunita Sarawagi. "Creating Probabilistic Databases from Information Extraction Models." In: Jan. 2006, pp. 965–976.

[6] Christopher Ré, Nilesh Dalvi, and Dan Suciu. "Efficient Top-k Query Evaluation on Probabilistic Data". In: (June 2006). https://www.cs.stanford.edu/people/chrismre/papers/Multisimulation_TR.pdf.

[7] L. Antova et al. "Fast and Simple Relational Processing of Uncertain Data". In: (Apr. 2008), pp. 983–992.

[8] Graham Cormode and Minos Garofalakis. "Histograms and Wavelets on Probabilistic Data". In: *IEEE Transactions on Knowledge and Data Engineering* 22.8 (2010), pp. 1142–1157. DOI: 10.1109/TKDE.2010.66.

[9] Matteo Magnani and Danilo Montesi. "A Survey on Uncertainty Management in Data Integration". In: *J. Data and Information Quality* 2 (July 2010). DOI: 10.1145/1805286.1805291.

[10] Carlos Rodriguez et al. "Toward Uncertain Business Intelligence The Case of Key Indicators". In: *IEEE Internet Computing* 14 (July 2010), pp. 32–40. DOI: 10.1109/MIC.2010.59.

[11] Dan Suciu et al. *Probabilistic Databases (Synthesis Lectures on Data Management)*. Morgan & Claypool Publishers, 2011.

[12] Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. Springer London, 2012.

[13] B. Wanders and Maurice van Keulen. “Revisiting the formal foundation of Probabilistic Databases”. In: *Proceedings of the 2015 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology, IFSA-EUSFLAT 2015*. Advances in Intelligent Systems Research. Netherlands: Atlantis Press, June 2015, p. 47. ISBN: 978-94-62520-77-6. DOI: 10.2991/ifsa-eusflat-15.2015.43.

[14] V. Cincotta. *Design and Implementation of a Scalable Probabilistic Database System*. July 2019.

[15] K.D. van Rijn. *A binary decision diagram based approach on improving probabilistic databases*. July 2020. URL: http://essay.utwente.nl/82217/.

[16] Joonas Jälkö et al. “Privacy-preserving data sharing via probabilistic modeling”. In: *Patterns* 2.7 (2021), p. 100271. ISSN: 2666-3899. DOI: https://doi.org/10.1016/j.patter.2021.100271. URL: https://www.sciencedirect.com/science/article/pii/S2666389921000970.

[17] Maurice van Keulen. “probabilistic DB theory”. In: University of Twente, master course “probabilistic programming”. 2021.

[18] Nilesh Dalvi, Christopher Ré, and Dan Suciu. “Probabilistic Databases: Diamonds in the Dirt”. In: (). https://homes.cs.washington.edu/~suciu/file15_cacm-paper.pdf.

[19] Robert Fink, Larisa Han, and Dan Olteanu. “Aggregation in Probabilistic Databases via Knowledge Compilation”. In: https://www.cs.ox.ac.uk/people/dan.olteanu/papers/fho-vldb12.pdf. Deptartment of Computer Science, University of Oxford.

[20] The PostgreSQL Global Development Group. *SQL Commands - EXPLAIN*. URL: https://www.postgresql.org/docs/current/sql-explain.html.

[21] The PostgreSQL Global Development Group. *User-Defined Aggregates*. URL: https://www.postgresql.org/docs/current/xaggr.html.

[22] The PostgreSQL Global Development Group. *Using EXPLAIN - EXPLAIN*. URL: https://www.postgresql.org/docs/current/using-explain.html.