Query Processing of Schema Design Problems for Data-driven Renormalization

by

Xiao Meng

M.Eng., Harbin Institute of Technology, 2010
B.Eng., Harbin Institute of Technology, 2008

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2017

Approval

Name:	**Xiao Meng**
Degree:	**Master of Science (Computing Science)**
Title:	**Query Processing of Schema Design Problems for Data-driven Renormalization**
Examining Committee:	**Chair:** Jiannan Wang Assistant Professor
	Jian Pei Senior Supervisor Professor
	Arrvindh Shriraman Supervisor Associate Professor
	Nick Sumner Internal Examiner Assistant Professor
Date Defended:	**October 16, 2017**

Abstract

In the past decades, more and more information has been stored or delivered in non-relational data models—either in NoSQL databases or via a Software as a Service (SaaS) application. Users often want to load these data sets into a BI application or a relational database for further analysis. The *data-driven renormalization framework* is often used to transform non-relational data into relational data. In this thesis, we explore how to help users to make design decisions in such a framework. We formally define two kinds of queries—the point query and the stable interval query—to help users making design decisions. We propose two index structures, which can represent a list of FDs concisely but also process the queries efficiently. We conduct experiments on two real datasets and show that our algorithms greatly outperform the baseline method when processing a large set of FDs.

Keywords: functional dependencies; database design; renormalization; NoSQL

To my family

"I may not have gone where I intended to go, but I think I have ended up where I needed to be."

by Douglas Adams, (1952 – 2001)

Acknowledgements

I would like to express my deep gratitude to my senior supervisor Dr. Jian Pei for his continuous support and guidance during my graduate studies. He always explains things crystally clear even on very difficult topics and goes right to the core of problems, which always encourage me to write more clearly and think harder. He gave me various opportunities to explore my way in academia and industry and helped me out in difficult situations. More importantly, he teaches me to think more strategically in my life and work. I was fortunate to have such a wise and thoughtful mentor. I am a slow learner, but I certainly benefit from what I learned from him in my future career.

My gratitude also goes to my supervisor Dr. Arrvindh Shriraman for his many insightful suggestions in my research, especially on the topic of NoSQL databases. I am very thankful to Dr. Nick Sumner for serving as the examiner and many suggestions to improve this thesis. Special thanks go to Dr. Jiannan Wang for serving as the chair and many helpful discussions for this thesis.

I am also grateful to many great people—the R&D group, the Memphis team, and the driver's team—in Simba Technologies (now a subsidiary of Magnitude Software) during my internship there. George Chow, who was my mentor and a great friend, shared a lot of knowledge in industry—making design decisions, leading teams and identifying business values. It was a fun experience and also motivated the work of this thesis.

I have been fortunate to meet many great people during my time at SFU. Our `IDEAL` (Intelligence and Data Engineering and Analytics Lab) is a large family now. My acknowledgment goes to the current members—Xiangbo Mao, Juhua Hu, Chuancong Gao, Yu Yang, Zicun Cong, Linyang Chu, Yanyan Zhang, Yajie Zhou and Xia Hu—and a long list[1] of the alumnus—Zijin Zhao, Xiaojian Wang, Dr. Guanting Tang and many others. It is an honor to be part of the family. Also, I would like to thank many other friends in SFU—most of them have already graduated. They make my graduate studies more enjoyable.

I would like to thank my dear friend Jaime Zhao who continuously encouraged me and lit up my darkest moments. Finally, I would like to thank my parents for their love and supports accompanying me along my journey.

[1] `https://www.cs.sfu.ca/~jpei/students.htm`

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past decades, more and more information has been stored or delivered in non-relational data models. Many companies use NoSQL databases (e.g., Couchbase [1] and MongoDB [2]) as their data management systems. Also, Software as a Service (SaaS), which delivers information via non-relational data models (e.g., JSON and XML), has become a mainstream software licensing and delivery model. The industry has embraced the idea of *APIs as a product* [3] and exposes internal and external data according to their APIs. *ProgrammableWeb*, a leading source about Internet-based APIs, has collected 17,406 APIs as of May 2017 [4].

Those data sets have potential value waiting for discovery. Among many approaches of data exploration, two widely used ones are loading data into a Business Intelligence (BI) system, and loading data into a relational database. BI tools, such as Tableau [5] and PowerBI [6], offer intuitive graphical interfaces and are popular among non-technical users. Relational databases provide the support of a mature declarative query language—SQL—which is common knowledge for data analytics practitioners. Also, users can benefit from the query performance of relational databases, which has evolved and been improved over decades.

The challenge here is to transform non-relational data into a more rigid tabular format. It is a non-trivial task and requires significant engineering and human efforts. Recent works [7, 8] tackle one aspect of this technical challenge via exploring functional dependencies from data. We use the term *data-driven renormalization* to refer such a framework. This thesis aims to support critical design decision queries—*BCNF Test* and *Primality Test*—in such a framework.

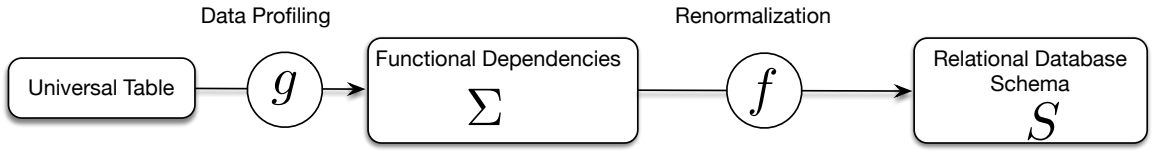


Figure 1.1: The Data flow of generating a database schema from a universal table

1.1 Data-driven Renormalization

We illustrate the data-driven renormalization framework in this section. Figure 1.1 shows the data flow of this process. We only introduce technical terms informally. We refer readers not familiar with the relational data model to the preliminary section of Chapter 3.

First, the data-driven renormalization framework assumes that the initial input can be transformed into a *universal table*. Using JSON data as an example, we can achieve such a goal by flattening the tree structure of JSON into a set of attributes associated with values.[1] Figure 1.2 gives an example of flattening JSON documents. We refer the set of flattening attributes as the *universal attribute set* and the wide table as the *universal table*.

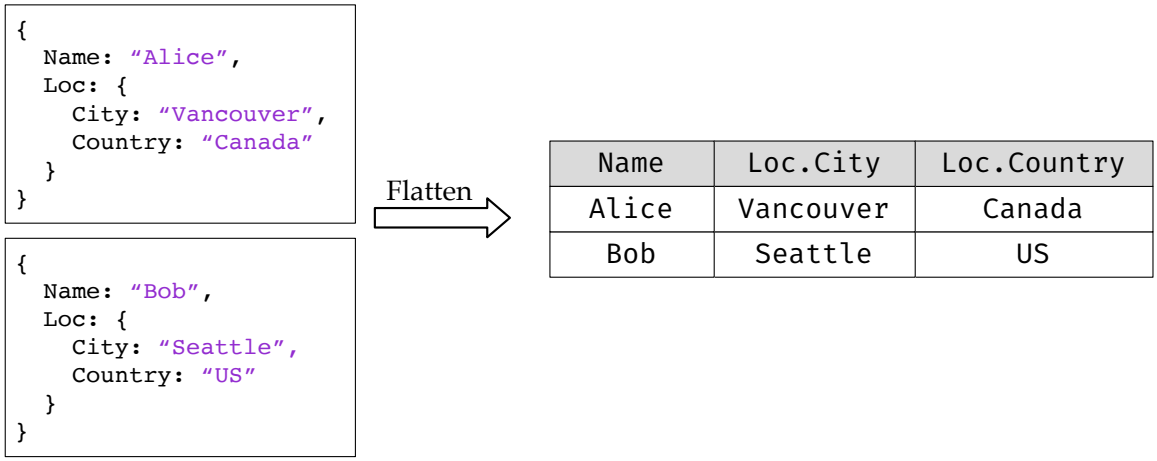


Figure 1.2: JSON documents and the corresponding flatten table

Given a universal table, the data-driven renormalization process consists of two phases, as shown in Figure 1.1.

1. **Data profiling.** A data profiling procedure g finds a set of Functional Dependencies (FDs) Σ with the universal attribute set U on the data instance I, denoted by $(U, \Sigma) = g(I)$. An FD is in the form of $X \to Y$, where X and Y are both sets of attributes. It implies that, as long as two rows have the same value of X, they must share the same value of Y. Informally, we say that X determines Y.

[1] If a JSON document contains a nested array, we can either treat the array as a binary object or an independent table.

2. **Renormalization.** A normalization procedure f generates a relational database schema S based on the set of FDs Σ discovered in the previous phase, denoted by $S = f(U, \Sigma)$. The relational database schema is represented by a collection of attribute sets, and the union of attribute sets in such a collection equals the universal attribute set.

In practice, however, the FDs discovered in the first phase—data profiling—may not always be *correct* or *accurate*. *Why?* First, data profiling may produce incorrect FDs or miss correct FDs due to poor data quality. The data source may have dirty data—given that there is no enforcement of any integrity constraints, it is likely to happen—and thus produce incorrect FDs or miss semantically correct FDs. Second, the FDs discovered may be inaccurate due to the choice of FD discovery algorithms. Since discovering the exact FDs is expensive[2], we may use an approximate FD discovery algorithm. Given these practical problems, many data profiling algorithms [9, 10, 11] use a ranking function to indicate the plausibility of a discovered FD. Thus, we model the output of the data profiling phase as *a ranked list of FDs*.

How do we proceed to the second phase—renormalization—under the assumption that the data profiling phase outputs a ranked list of FDs? One approach, as used by [7], is to introduce a parameter k and only use the top-k FDs as FDs of interest to generate the normalized database schema.[3] However, renormalization is hardly a fully automatic procedure and often involves human interventions. Also, users may want to pick different k and compare the resulted schema to make better design decisions. In this thesis, instead of generating a database schema automatically, we study the query answering aspect of renormalization and support two *core design decision problems* [12], which we will illustrate shortly, for an arbitrary k. The process is independent of the schema generating algorithms and thus can be used in the data-driven renormalization framework without any constraint of the schema generating algorithm employed. The answers to these design decision problems offer guidelines for schema design.

However, most of the design decision problems are known to be NP-complete [13]. It may be expensive to answer these decision problems on the fly.

In this thesis, we study how to handle two fundamental design decision problems—*BCNF Test* and *Primality Test*—by pre-computing a compact representation of FDs and maintaining index structures for query processing.

BCNF Test. Given an attribute set X, does X satisfy Boyce-Codd Normal Form (BCNF)? That is, for any subset $Y \subseteq X$, can Y either determine itself or attribute set X? The

[2] For example, it takes about 61 hours to discover the 13 million FDs from the TPC-H dataset with 6 million records and 52 attributes in

[3] More precisely, they use a numeric threshold on the score function of FDs.

BCNF test helps users to evaluate whether a schema design is good in the sense of data redundancy.

Primality Test. Given an attribute set X and an attribute $p \in X$, does attribute p belong to some key of X? The primality test helps users to make decisions on the primary key of a relation schema.

They can be answered by the *point queries* in this thesis—a user specifies the value of k, a 'point' in the possible range of k.

Besides, a technically interesting extension of these problems is to study *the stability often these answers on* k—the number of FDs we choose. The answer to these problems is an interval. Thus, we refer to such queries as the *stable interval queries* and will formulate them in Chapter 3. *Why is it interesting?* Such an interval indicates the stability of a property—BCNF or primality—on k. Users can inspect the interval to make design decisions—whether the FDs at the interval boundaries are semantically correct and whether to decompose a relation or choose an attribute as part of a primary key. We will see it in action shortly in Example 1.

How does our work fit into a real-world system? We give a use case to show how to process a dataset and handle various real-world situations. Imagine a collection of JSON documents and a data-driven renormalization system.

First, we want to flatten the JSON documents into a universal relational table. In practice, a JSON document may contain complicated structures such as arrays. We can handle it in two ways. If the array only contains values of the primitive types (e.g., number and string), we can concatenate the primitive values into a string. If the array contains a list of nested objects, we can introduce a foreign key and make the array as an auxiliary table linked via the foreign key. Then, we can treat the auxiliary table as an independent instance of our problem. Also, if a JSON document has missing values, we can use a special value `NULL` to represent that some value is missing.

Next, the data profiling component produces a ranked list of FDs based on the data instance. These FDs give users clues on the relationship between attributes. Users can choose the top-k FDs—the value of k can be a rough guess—and use well-known normalization algorithms to generate candidate database schemas. Then, users further inspect the database schema to find the relation schemas that do not meet their expectations. Users can trace back the cause of such a relation schema (either due to the data profiling algorithm and due to the data source quality) and fix the issues by tuning the data profiling algorithms or repairing errors in the data source. The stable interval query is helpful in such a scenario, which provides the candidate FDs that may lead to the issues.

We wrap up this section with a complete example.

Example 1. *Consider a* `Course` *database, the universal attribute set is*

$$R(X) = (\texttt{instructor}, \texttt{department}, \texttt{faculty}, \texttt{course}, \texttt{term}).$$

A sample instance of such a database is shown in Figure 1.3.

Instructor	Department	Faculty	Course	Term
Alice	Math	Science	Algebra	2017 Spring
Bob	CS	Applied Science	Database	2017 Spring
Carl	CS	Applied Science	OS	2017 Spring
David	Math	Science	Algebra	2017 Summer
Bob	CS	Applied Science	Python	2016 Fall

Figure 1.3: A sample `Course` database

Assume that we have discovered the following ranked list of FDs from data

$$f_1 : \texttt{instructor} \rightarrow \texttt{department}$$
$$f_2 : \texttt{department} \rightarrow \texttt{faculty}$$
$$f_3 : \texttt{course}, \texttt{term} \rightarrow \texttt{instructor}$$

The BCNF test asks that, if we take the top-k FDs, does $R(X)$ conform to BCNF? For example, when we take $k = 3$, $R(X)$ does not conform to BCNF. Why? *The attribute* `instructor` *can determine at most three attributes* (`instructor`, `department`, `faculty`)*, which do not equal to the universal attribute set. In other words, there exist data redundancies for the attribute set* (`instructor`, `department`, `faculty`)*. The answer—$R(X)$ does not conform to BCNF—guides users to decompose the relation if users want to get a redundancy-free schema.*

The primality test asks whether an attribute belongs to some key in a relation schema. For instance, does `instructor` *belong to some key of the schema $R(X)$? Among all super-keys*[4] *of $R(X)$, the minimal one containing* `instructor` *is* (`instructor`, `course`, `term`)*. However,* (`instructor`, `course`, `term`) *is not a key since* (`course`, `term`) *is its subset and is also a super-key.*

Next, we give examples of interval queries.

For the BCNF test of the universal attribute set, the stable interval is $[1, 3]$. That is, it does not conform to BCNF when $k \in [1, 3]$. Consider another schema (`instructor`, `department`, `faculty`)*. It does not conform to BCNF when $k = 1$. But it conforms to*

[4] We say an attribute set is a super-key of $R(X)$ if it can determine all attributes in X.

BCNF when $k = 2, 3$. Users can carefully examine whether the turning point—FD f_2—is semantically correct. If it is a valid FD, we can conclude that we do not need to decompose such a relation for any $k \geq 2$.

For the primality test, attribute `instructor` *belongs to a primary key in the interval* $1 \leq k \leq 2$.

- *when* $k = 1$, (`instructor`, `faculty`, `course`, `term`) *is a key;*
- *when* $k = 2$, (`instructor`, `course`, `term`) *is a key;*

With the involvement of f_3, attribute `instructor` *is no longer a prime attribute since the attribute can be determined by* (`course`, `term`). *The answer gives users guideline on choosing the prime key on different k.*

1.2 Contributions

In this thesis, we aim to support two core design decision problems—BCNF Test and Primality Test—for the top-k FDs in a ranked list of FDs.

- We formally define two types of queries—the point query and the interval query—under the data-driven renormalization framework.
- We design efficient index structures and query processing algorithms to support the queries.
- We conduct experiments on two real datasets and study the performance of our query processing algorithm.

As far as we know, this is the first work on supporting these two core design decision problems in the context of the data-driven renormalization framework.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 reviews related work. Chapter 3 formally defines two type of queries—the point query and the interval query. Chapter 4 proposes two index structures to store intermediate results efficiently. Chapter 5 presents the query processing algorithms on our index structures. Chapter 6 concludes the thesis.

Chapter 2

Related Work

In this chapter, we review the related work and divide them into two categories—*computational problems of schema design* and *data-driven renormalization*.

2.1 Computational Problems of Schema Design

The relational data model was first described by Edgar F. Codd in 1969 [14]. A lot of research work was devoted to the area in 70's and 80's. In particular, Beeri and Bernstein [13] proposed the first linear-time algorithm for computing the closure of an attribute set and applied it to schema design problems. Besides, they also showed that the BCNF test for sub-schema and the primality test are NP-complete. Mannila and Räihä [15] studied the structure of Armstrong relations—a representation of FDs with a sample database instance of minimal size. They drew the connection between the *family of closed sets* and the *family of maximal sets* for a set of FDs, which are the major structures we study in this thesis. For a complete review of the major results in this period, please refer to [16, 17].

Gottlob et al. [12] studied three schema design problems (primality test, 3NF test, and BCNF test) whose sets of FDs have bounded tree width—the relational schema can be represented by a hypergraph, and the treewidth is defined on the hypergraph. They developed sophisticated polynomial-time algorithms to solve these particular instances of design problems. The work was mostly theoretical and focused on the result of computational complexity.

Köhler and Link [18] re-examined the schema design problem in the SQL data model. The SQL data model is different from the standard relational data model—it allows duplicate tuples and uses a special null value to process incomplete information. They re-defined the FDs and BCNF under the SQL data model. We only tackle design problems under the relational model in this thesis and will discuss the extension to the SQL data model with the future work in Chapter 7.

All these works assume that the FDs are available as the prior knowledge—users devote a lot of time to identify the FDs hidden in an application context to minimize potential changes in the future. The paradigm is also known as *schema-on-write* [19] in recent years.

Our work, however, is conducted in a different context. Users use a non-relational data model, which often does not require a rigid schema definition in prior, to boost the speed of iterative developments. The paradigm is also known as *schema-on-read* [19]. The schema is unknown until we profile the datasets and discover the FDs from data. Since the discovered FDs may not always be semantically correct, this thesis aims to guide the users to re-design the schema in the relational data model.

2.2 Data Driven Renormalization

The data driven renormalization framework was proposed under the *schema-on-read* paradigm mentioned in the previous section. We have introduced the framework in Chapter 1 and present the related work in this section.

DiScala and Abadi [7] proposed a three-phase algorithm that automatically transforms nested JSON documents into relational tables. In the first phase, they discovered the possible FDs—each FD is associated with a score of strength—from the data sets flatten from the JSON documents. Then, they decomposed the relation schema based on the top-k FDs selected by a threshold. In the second phase, they identified the relation schemas that may represent the same entity and merge them into one relation schema. In the third phase, they fine-tuned the generated schemas with various heuristics. Although this approach is fully-automatic, it heavily relies on observations from real-world datasets and may not work well on an arbitrary dataset.

Papenbrock and Naumann [8] presented an end-to-end solution of data-driven schema renormalization. In particular, they treated the normalization as an iterative process and proposed several quality measures to help users to choose FDs and decompose a relation schema into small relation schemas.

A large body of work was devoted to functional discovery algorithms in the data profiling phase [20]. Papenbrock et al. [21] conducted an experimental evaluation of seven FD discovery algorithms. They showed that none of those state-of-the-art algorithms scale to datasets with hundreds of columns or millions of rows. Bleifuß et al. [22] presented an algorithm that approximately discovers FDs—the discovered FDs may be not correct—to process large datasets.

This thesis, as discussed in Chapter 1, tackles a different aspect compared with all the above work—the query answering aspect of the data-driven renormalization.

Chapter 3

Problem Definition

In this chapter, we formalize the problems discussed in Chapter 1. We first review the relational data model. Then we define two types of queries—the *point query* and the *stable interval query*. Finally, we sketch the major technical ideas in this thesis.

3.1 Preliminary

In this section, we give the definitions of the relation schema, functional dependencies (FDs) and their inference rules. Based on these definitions, we introduce important terms in this thesis—*closure*, *BCNF*, and *primality*. We use the `Course` database in Example 1 to illustrate them when necessary.

Definition 1 (Relation Schema [17])**.** *A **relation schema**, denoted by $R(U)$, consists of a name R and a set U of attributes. A **tuple** for a relation schema $R(U)$ is a set of* (`attribute`, `value`) *pairs where each attribute is unique and belongs to the attribute set U. A set of tuples on a given relation schema is called a **relation instance**. Given a tuple t on $R(U)$, the **projection** of t on attribute set X, denoted by $t[X]$, consists of the* $(attribute, value)$ *pairs for the attributes in X. A **sub-schema** of $R(U)$ is a relation schema $R(X)$ such that $X \subseteq U$. We often write $R(X)$ in the form of $R[X]$ to indicate that it is a sub-schema.*

For example, `Course(instructor, department, faculty, course, term)` in Example 1 is a relation schema, where the relation name is `Course`, and the attribute set is `{instructor, department, faculty, course, term}`. The first tuple in the `Course` database is ⟨`(instructor, Alice)`, `(department, Math)`, `(faculty, Science)`, `(course, Algebra)`, `(term, 2017 Spring)`⟩. We often assume a total order on the attribute set and abbreviate such a tuple as ⟨`Alice`, `Math`, `Science`, `Algebra`, `2017 Spring`⟩. The projection of this tuple on ⟨`insctructor`, `department`⟩ is ⟨`Alice`, `Math`⟩, which is a tuple for the sub-schema R[`insctructor`, `department`].

Definition 2 (Functional Dependency (FD) [17])**.** *Given a relation schema $R(U)$, a **functional dependency (FD)** over U is an expression in the form of $X \to Y$ where $X, Y \subseteq U$.*

Given a relation instance r on $R(U)$, an FD $X \to Y$ ***holds*** *in r, denoted by $r \vdash X \to Y$, iff $\forall t_1, t_2 \in r$, $t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y]$. An FD $X \to Y$ is* ***trivial*** *if $Y \subseteq X$.*

For example, the FD `instructor` $\to$ `department` holds in the sample `Course` database in Figure 1.3. But the FD `instructor` $\to$ `course` does not hold since Bob teaches both `Python` and `Database`. One trivial FD is `course`, `instructor` $\to$ `course`.

In practice, functional dependencies are expressed as *integrity constraints* and are part of the relational data model. We use the term *relational schema* (Definition 3) to refer the schema expressed within the relational model. It is different from the term *relation schema*.

Definition 3 (Relational Schema [17])**.** *A relational schema, denoted by $(R(U), \Sigma)$, consists of a relation schema $R(U)$ and a set Σ of FDs.*

Definition 4 (Armstrong's Axioms and Derived FDs [17])**.** ***Armstrong's Axioms*** *are three FD inference rules: (1) Reflexivity. If $Y \subseteq X$, then $X \to Y$; (2) Augmentation. If $X \to Y$, then $XZ \to YZ$; (3) Transitivity. If $X \to Y$ and $Y \to Z$, then $X \to Z$. Given a set Σ of FDs, an FD f can be* ***derived from*** *Σ, denoted by $\Sigma \vDash f$, if there exists a finite sequence of applying Armstrong's axioms to generate the FD f from Σ.*

For example, consider the three FDs in Example 1:

$$f_1 : \texttt{instructor} \to \texttt{department}$$
$$f_2 : \texttt{department} \to \texttt{faculty}$$
$$f_3 : \texttt{course}, \texttt{term} \to \texttt{instructor}$$

An FD `course`, `term` $\to$ `faculty` can be derived from $\Sigma = \{f_1, f_2, f_3\}$ by applying the transitivity inference rule twice—applying on f_3 and f_1 to get `course`, `term` $\to$ `department` and then applying with f_2.

Definition 5 (Closure, Key and Closed Set [17])**.** *Consider a relational schema $(R(U), \Sigma)$ and an attribute set $X \subseteq U$. The* ***closure*** *of the attribute set $X \subseteq U$ under Σ, denoted by X^+, is the set of attributes that are determined by X. Formally,*

$$X^+ = \{a \in U \mid \Sigma \vDash X \to a\}.$$

An attribute set $Y \subseteq X$ is ***closed*** *on the attribute set X under the FDs Σ iff $Y^+ \cap X = Y$, and we say that the attribute set Y is a* ***closed set*** *on the attribute set X. The* ***family of closed sets*** *on the attribute set X is denoted by $\texttt{closed}(X)$. Formally,*

$$\texttt{closed}(X) = \{Y \subseteq X \mid Y^+ \cap X = Y\}.$$

For a closed set $Y \in \texttt{closed}\,(U)$, we have $Y^+ = Y$.

For example, the closure of the attribute set `instructor` is (`instructor`, `department`, `faculty`). Also, the attribute set (`instructor`, `department`, `faculty`) is closed on the attribute set U in the `Course` database. Consider an attribute set $X = \{$`instructor`, `department`, `course`, `term`$\}$—it excludes the attribute `faculty` from the attribute set U. The attribute set $\{$`instructor`, `department`, `term`$\}$ is a closed set on X, though it is not closed on the attribute set U.

The following theorem connects the closure structure with FD inferences.

Theorem 1 ([17]). *Given a relational schema $(R[U], \Sigma)$ and two attribute sets $X, Y \subseteq U$, we have $\Sigma \vDash X \to Y \iff Y \subseteq X^+$.*

Definition 6 (Key and Prime [17]). *Given a relational schema $(R(U), \Sigma)$, an attribute set X is a **key** of the attribute set U if $\Sigma \vDash X \to U$, and for any attribute $a \in X$, $\Sigma \nvDash X \backslash a \to U$. An attribute $a \in U$ is prime if $\exists X \subseteq U$ such that $a \in X$ and X is a key of the attribute set U.*

For example, (`course`, `term`) is a key for the `Course` database. Thus, both attribute `course` and attribute `term` are prime.

Definition 7 (Boyce-Codd Norm Form (BCNF) [17]). *A relational schema $(R(U), \Sigma)$ conforms to BCNF if for any non-trivial FD $X \to a$ such that $\Sigma \vDash X \to a$, we have $\Sigma \vDash X \to U$.*

For example, the relational schema of the `Course` database does not conform to BCNF, since $\Sigma \vDash$ `instructor` $\to$ `department` but $\Sigma \nvDash$ `instructor` $\to U$.

3.2 Problem Definition

Given an attribute set U and an ordered list of FDs $\Sigma = \langle f_1, f_2, \ldots, f_n \rangle$ on U, where $n = |\Sigma|$ is the number of FDs in Σ, let $\Sigma_k = \langle f_1, f_2, \ldots, f_k \rangle$ be the top-k FDs of Σ.

We have seen how the point query and the stable interval query can guide users on schema design in Example 1 (Chapter 1). The point query helps users to check the property—BCNF or primality—of a schema design under the relational schema $(R(U), \Sigma_k)$ for a specific k. The stable interval query helps users to inspect the stability of a property—BCNF or primality—near k. Users can inspect the interval to make design decisions—whether the FDs at the interval boundaries are semantically correct and whether to decompose a relation or choose an attribute as part of a primary key.

We first define the point query and the stable interval query independently of the particular design decision question. Here we treat a design decision question as a black box that outputs a boolean value (true or false) based on the property of a relational schema.

Such a definition enables us to support more design decision questions in the future, as we will discuss the future work in Chapter 7.

Definition 8 (Point Query)**.** *Consider a relational schema* $(R(U), \Sigma)$*. Given an integer* k *such that* $1 \leq k \leq n$*, the point query of a design decision problem* $\mathcal{D}$*, denoted by* $\texttt{PQuery}(k, \mathcal{D})$*, outputs a boolean value—true or false—to the decision problem* $\mathcal{D}$ *under the relational schema* $(R(U), \Sigma_k)$*.*

Definition 9 (Stable Interval Query)**.** *Consider a relational schema* $(R(U), \Sigma)$*. Given an integer* k *such that* $1 \leq k \leq n$*, the stable interval query of a design decision problem* $\mathcal{D}$*, denoted by* $\texttt{SInterval}(k, \mathcal{D})$*, outputs a maximum length interval* $[i, j]$ *such that* $i \leq k \leq j$*, and* $\forall i \leq u \leq j, \texttt{PQuery}(u, \mathcal{D}) = \texttt{PQuery}(k, \mathcal{D})$*.*

Next, we define the two design decision problems—the BCNF test and the primality test—under the relational schema $(R(U), \Sigma)$.

Definition 10 (BCNF Test)**.** *Given an attribute set* $X \subseteq U$ *under a relational schema* $(R(U), \Sigma_k)$*, the BCNF test on the attribute set* X *asks whether there exists a non-trivial FD* $Y \to Z$ *where* $Y, Z \subseteq X$ *such that* $\Sigma_k \vDash Y \to Z$ *but* $\Sigma_k \nvDash Y \to X$*.*

Definition 11 (Primality Test)**.** *Given an attribute set* $X \subseteq U$ *and an attribute* $x \in X$ *under a relational schema* $(R(U), \Sigma_k)$*, the primality test asks whether* x *belongs to some key of* X*.*

We have given the examples of those queries in Example 1. Readers can re-examine Example 1 if desired.

3.3 Technical Ideas

In this section, we sketch the major technical ideas of this thesis and pave the way for our solution in the next few chapters.

First, many computation steps in the two design decision problems—the BCNF Test and the Primality Test—boil down to computing the closure of an attribute set. Both the two design decision problems require checking whether some FD $X \to Y$ can be derived from a set Σ of FDs. By Theorem 1, such a problem (checking whether an FD can be inferred from a set of FDs) can be reduced to checking whether $Y \subseteq X^+$. Thus, we want to support querying the closure of an attribute set efficiently.

A straightforward approach is to memorize the answers to all closure queries in advance and look up the answer to the closure query of an attribute set. However, the space cost is high.[1]

[1]We will analyze the space cost in Chapter 4.

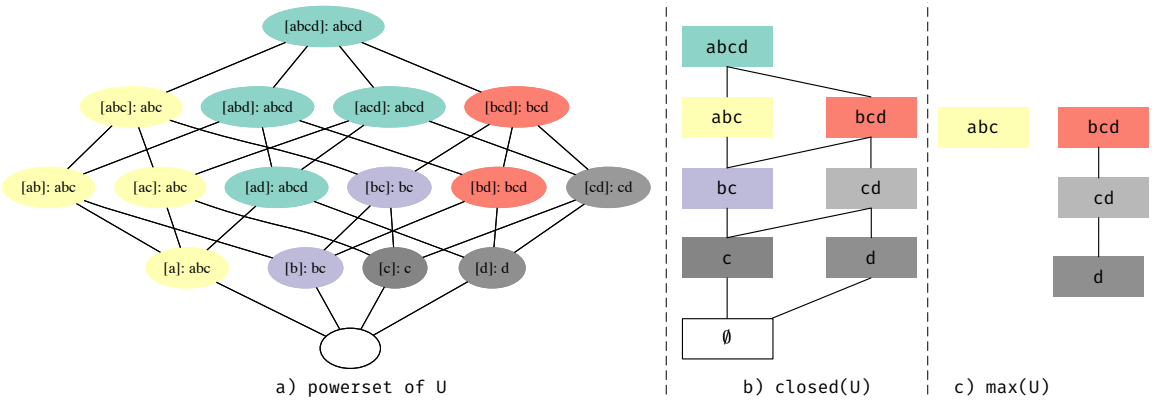


Figure 3.1: The powerset lattice vs. the quotient set lattice vs. the maximal set on $U = \{a, b, c, d\}, \Sigma = \{a \to b, b \to c\}$. Each ellipse shows the attribute set and its closure in the format of [`attribute set`] : `closure`.

Can we find a compact representation of these answers and also provide efficient query support? Fortunately, formal concept analysis (FCA) [23, 24] offers guidelines on this problem, which inspires the solutions in this thesis. It is beyond the scope of this thesis to introduce the whole area. We focus on its application under our context and illustrate it using an example.

Example 2. *Consider a relational schema* $(R(U), \Sigma)$ *where* $U = \{a, b, c, d\}$ *and* $\Sigma = \langle a \to b, b \to c \rangle$. *We want to compute the closures of all possible attribute sets. That is, we are interested in the collection of* (X, X^+) *pairs where* $X \subseteq U$. *As a special case, let the closure of an empty set be itself.*

Figure 3.1 (a) shows the powerset of the universal attribute set, denoted by 2^U, *and their closures. The attribute set forms a partial order based on the inclusive relationship between attribute sets. Formally, such a structure is known as the powerset lattice.*

Figure 3.1 (b) shows a compact representation of the powerset lattice—the quotient set lattice. *The key observation is that* ***attribute sets may share the same closure****, which is color-coded in the figure. Then we can partition the powerset* 2^U *into equivalent classes based on the closure* X^+ *for each attribute set* $X \subseteq U$. *That is, two attribute sets belong to the same equivalent class iff their closures are the same. And we use a representative—the attribute set that is closed (see Definition 5)—to summarize each equivalent class. It turns out that those equivalent classes also form a lattice structure, known as the* ***quotient set lattice****. The family of attribute sets in the quotient set lattice is exactly the family of closed sets* `closed`(U) *(see Definition 5).*

Figure 3.1 (c) shows a more compact representation of the quotient set lattice. The key observation is that an attribute set in `closed`(U) *may be generated by intersecting other attribute sets in* `closed`(U). *For example, the closed set* $\{b, c\}$ *can be generated by two closed sets* $\{a, b, c\}$ *and* $\{b, c, d\}$. *Based on the observation, we can achieve a more compact*

representation known as the family of maximal sets, denoted by $\max(U)$*. We leave the technical details to the next chapter.*

In summary, we get a compact structure to represent a set of FDs.

How does the structure help us solving the schema design problems? We present two running examples—one for the BCNF test and the other for the primality test—on the relational schema $(R(U), \Sigma)$*. We show* ***how to*** *use the family of maximal sets (Figure 3.1 (c)) to answer the schema design problems. More specifically, for each design decision problem, we show a sufficient and necessary condition*[2] *to answer the decision problem using the family of maximal sets.*

Consider the BCNF test for the schema $R(U)$ *with the set* Σ *of FDs. Does* $R(U)$ *conform to BCNF? We only need to check the following condition: for any attribute set* $W \in \max(U)$*, does* $\Sigma \nvDash W \setminus \{x\} \rightarrow \{x\}$ *for any attribute* $x \in W$*? If the answer is yes,* $R(U)$ *conforms to BCNF. Otherwise,* $R(U)$ *does not conform to BCNF. Consider the attribute set* $W = \{a, b, c\}$*. Here if we remove the attribute* c *from* $W = \{a, b, c\}$*, we have* $\Sigma \vDash \{a, b\} \rightarrow c$*. Thus it does not conform with BCNF.*

Consider the primality tests for some attribute $x \in U$ *under the relational schema* $(R(U), \Sigma)$*. Does the attribute* x *belong to a key of the relation schema* $R(U)$*? We only need to check the following condition: is there an attribute set* $W \in \max(U)$ *such that no superset of* W *belongs to* $\max(U)$*, and* $x \notin W$*? If the answer is yes, then the attribute* x *is prime. Otherwise, the attribute* x *is not prime. Here, we have two attribute sets* $\{a, b, c\}$ *and* $\{b, c, d\}$ *satisfied the checking condition—they both belong to* $\max(U)$*, but any of their supersets does not belong to* $\max(U)$*. It follows that, for all attributes in* U*, only the attribute* a *and the attribute* d *are prime—*$a \notin \{b, c, d\}$ *and* $d \notin \{a, b, c\}$*.*

The example shows how to handle the point queries on the relation schema $R(U)$ using the family of maximal sets, which is the cornerstone of answering more involved queries—the point queries regarding sub-schemas and the stable interval queries. The family of maximal sets is the core structure studied in this thesis. The crux of our problem is *how to maintain* $\max(U)$ *as an index structure* (the topic of Chapter 4) and *how to support query processing using the index structure* (the topic of Chapter 5).

[2]We leave the proof to Chapter 5.

Chapter 4

Index Structures

In this chapter, we study how to maintain a family of maximal sets using an index structure. First, we investigate the characteristics of attribute set closures and introduce the family of maximal sets. Then, we present two index structures—a simple list-based `SIndex` structure and a trie-based `TIndex` structure—for maintaining the family of maximal sets. Finally, we present the algorithms for constructing the index structures.

4.1 The Characteristics of Attribute Set Closures

This section formalizes the observations in Example 2. We repeat the example in Figure 4.1. Given a relational schema $(R(U), \Sigma)$, we study the collection of all attribute sets $X \subseteq U$ and their closures X^+. That is, we are interested in the collection $\mathcal{X} = \{(X, X^+) \mid X \subseteq U\}$ under the relational schema $(R(U), \Sigma)$.

Powerset lattice

The powerset lattice $(2^U, \subseteq)$ is a straightforward approach to represent the collection $\mathcal{X}$. Here we associate each attribute set X in the lattice with a value—its closure X^+. As we stated in Chapter 3, the space cost is high if we store such a powerset lattice completely without optimization. Both an attribute set X and its closure X^+ require at least $|U|$-bits to represent. We have $2^{|U|}$ possible (X, X^+) pairs in the powerset lattice and need at least $2^{|U|} \cdot 2|U|$ bits space to store them for each relational schema $(R(U), \Sigma_k)$. To store the powerset lattices for all possible k where $1 \leq k \leq |\Sigma|$, we need at least $|\Sigma| \cdot 2^{|U|+1} \cdot |U|$ bits. That is, even with $|U| = 32$ and $|\Sigma| = 10$, we need around 320 GB storage space.

Quotient set lattice and the family of closed sets

The powerset lattice $(2^U, \subseteq)$ can be represented by a *quotient set lattice* $(\texttt{closed}(U), \subseteq)$, where $\texttt{closed}(U)$ is the family of closed sets (Definition 5) on the attribute set U. We construct such a lattice as follows.

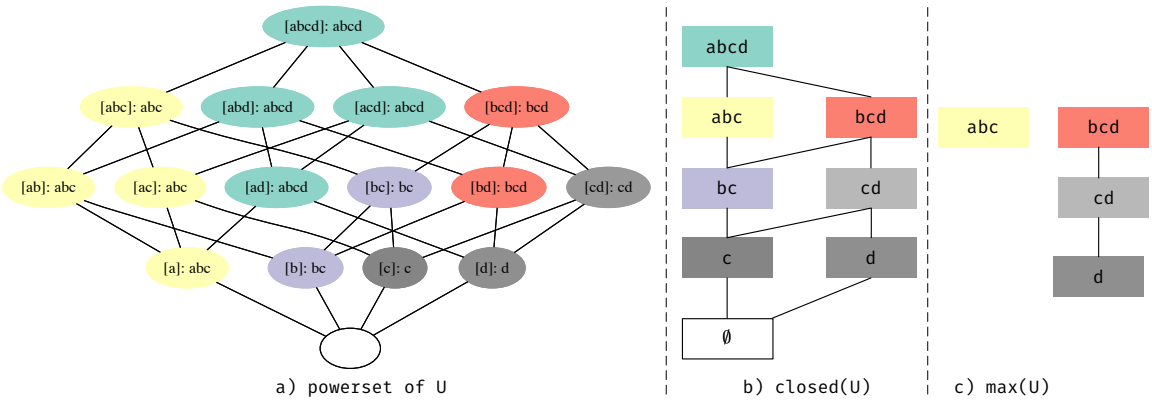


Figure 4.1: The powerset lattice vs. the quotient set lattice vs. the maximal set on $U = \{a, b, c, d\}, \Sigma = \{a \to b, b \to c\}$. Each ellipse shows the attribute set and its closure in the format of [`attribute set`] : `closure`.

First, we partition the powerset 2^U into *equivalence classes* based on the value of closure X^+ for each attribute set $X \subseteq U$. Two attribute sets X and Y are in the same equivalence class, denoted by $X \sim Y$, iff $X^+ = Y^+$ under the relational schema $(R[U], \Sigma)$. Let $2^U / \sim$ be the collection of equivalence partitions of the powerset 2^U on the equivalence relation $\sim$. Formally, $2^U / \sim$ is the collection $\{C \subseteq 2^U \mid \forall X, Y \in C, X^+ = Y^+\}$. The collection $2^U / \sim$ is also known as the *quotient set*.

Then, for each equivalent class, we choose an attribute set that is closed (i.e., the attribute set whose closure is itself) as the representative of the equivalent class. Thus, the quotient set $2^U / \sim$ can be represented by the closed set $\texttt{closed}(U)$.

Lemma 1. *Given an equivalence class $C = \{X_1, \ldots, X_m\}$ of $2^U / \sim$, there is an attribute set $Z \in C$ such that $Z = Z^+$.*

Proof. Consider an attribute set $X_i \in C$, and let Z be ${X_i}^+$. We prove that $Z \in C$. First, by the definition of the equivalence class, we have ${X_1}^+ = \cdots = {X_m}^+ = Z$. Second, by the definition of the attribute set closure (Definition 5), we have $(X_i^+)^+ = X_i^+$ and thus $Z^+ = Z$. As a result, $Z^+ = X_i^+$. It follows that $Z \sim X_i$. Therefore, $Z \in C$. □

Last, the family of closed sets also forms a lattice, and we call it *the quotient set lattice*. The correctness of the above procedures is established by Theorem 2.

Theorem 2. $(\texttt{closed}(U), \subseteq)$ *is a lattice.*

Proof. We only need to prove that, for any two attribute sets $X, Y \in \texttt{closed}(U)$, there exist a least upper bound and a greatest lower bound.

First, since $\emptyset \in \texttt{closed}(U)$ and $U \in \texttt{closed}(U)$, there always exist an upper bound (the universal attribute set U) and a lower bound (the empty attribute set $\emptyset$) for the two attribute sets X and Y.

Next, we prove that there exists a least upper bound for the attribute sets X and Y. Consider the collection $C = \{W \in \texttt{closed}(U) \mid X, Y \subseteq W \text{ and } \nexists W' \in \texttt{closed}(U) \text{ s.t. } W' \subsetneq W\}$. Assume there does not exist a least upper bound for X and Y. Then, there must exist at least two distinct attribute sets $Z_1, Z_2 \in C$. Apparently, $X, Y \subseteq Z_1 \cap Z_2$. Since $Z_1, Z_2 \in \texttt{closed}(U)$, we have $Z_1 \cap Z_2 \in \texttt{closed}(U)$. Since $Z_1 \neq Z_2$ and $Z_1 \not\subseteq Z_2$, we have $Z_1 \cap Z_2 \subsetneq Z_1$. That is, there exists an attribute set $W' = Z_1 \cap Z_2$ such that $W' \in \texttt{closed}(U)$ and $W' \subsetneq Z_1$. It leads a contradiction on the condition $Z_1 \in C$.

Similarly, we can prove that there exists a greatest lower bound for the attribute sets X and Y. Consider the collection $C = \{W \in \texttt{closed}(U) \mid X, Y \supseteq W \text{ and } \nexists W' \in \texttt{closed}(U) \text{ s.t. } W' \subsetneq W\}$. Assume there does not exist a greatest lower bound for X and Y. Then there exists at least two distinct attribute sets $Z_1, Z_2 \in C$. Obviously, $Z_1 \cap Z_2 \in \texttt{closed}(U)$ and $Z_1 \cap Z_2 \subsetneq Z_1$, which contradicts with the condition $Z_1 \in C$. □

The family of maximal sets

The quotient set lattice $(\texttt{closed}(U), \subseteq)$ can be represented by a subfamily of $\texttt{closed}(U)$—the family of maximal sets $\max(U)$. The key observation is that *the intersection of closed sets is also closed*. If a subfamily $\mathcal{P} \subseteq \texttt{closed}(U)$ can generate $\texttt{closed}(U)$ by taking interactions of the attribute sets in $\mathcal{P}$, we call $\mathcal{P}$ a *generator* of $\texttt{closed}(U)$. In particular, let the intersection of an empty collection be U.[1] Among all possible generators of $\texttt{closed}(U)$, there exists a generator of minimum size—the family of maximal sets (Definition 12). The correctness is established by Theorem 3.

Definition 12 (The Family of Maximal Sets)**.** *Consider a relational schema $(R(U), \Sigma)$. Given an attribute $x \in U$ and an attribute set $Y \subseteq U$ such that $\Sigma \nvDash Y \to x$, we say the attribute set Y is **maximal** on the attribute x iff $\Sigma \vDash Y \cup \{b\} \to x$ for any attribute $b \in U \setminus Y$. We define the family of maximal sets on the attribute x under the relational schema $(R[U], \Sigma)$ as*

$$max(U, x) = \{Y \subseteq U \mid \Sigma \nvDash Y \to x \wedge \forall b \in U \setminus Y, \Sigma \vDash Y \cup \{b\} \to x\}$$

The family of maximal sets, denoted by $\max(U)$, is the union of the family of maximal sets for all attributes. That is, $\max(U) = \bigcup_{x \in U} \max(U, x)$.

Also, we use $\max_k(U)$ to denote the family of maximal sets under the relational schema $(R(U), \Sigma_k)$.

We illustrate the concept using an example.

Example 3. *Consider a relational schema $(R(U), \Sigma)$ where $U = \{a, b, c, d\}$ and $\Sigma = \{a \to b\}$. What is $\max(U, b)$? Let $Y \in \max(U, b)$. Apparently, $a \notin Y$ and $b \notin Y$, since $\Sigma \nvDash Y \to a$.*

[1]The attribute set U is always closed. By this convention, we can represent the attribute set U implicitly.

Consider a candidate attribute set $Y = \{c, d\}$. Apparently, the attribute set Y cannot derive the attribute a—it satisfies the condition $\Sigma \not\models Y \to a$. However, adding either the attribute a or the attribute b to the attribute set Y can derive the attribute a. Thus, $\{c, d\} \in \max(U, b)$. There are no other maximal sets in $\max(U, b)$ since all possible candidate attribute sets are subsets of $\{c, d\}$.

Similarly, we can get the family of maximal sets on the other attributes. In summary, we have the following families of maximal sets:

$$\max(U, a) = \{\{b, c, d\}\}$$
$$\max(U, b) = \{\{c, d\}\}$$
$$\max(U, c) = \{\{a, b, d\}\}$$
$$\max(U, d) = \{\{a, b, c\}\}$$

The family of maximal sets $\max(U)$ is the union of the above collection. That is, $\max(U) = \{\{b, c, d\}, \{c, d\}, \{a, b, d\}, \{a, b, c\}\}$.

The collection $\max(U)$ is also a generator of the collection $\texttt{closed}(U)$. Here, we have $\texttt{closed}(U) = \{abcd, abc, abd, bcd, ab, bc, bd, cd, b, c, d, \emptyset\}$.[2] Any attribute set in $\texttt{closed}(U)$ can be generated from $\max(U)$. For example, the attribute set $\{a, b\}$ can be generated by intersecting two maximal sets $\{a, b, d\}$ and $\{a, b, c\}$.

Theorem 3 (Paraphrased from [15])**.** *The family of maximal sets $max(U)$ is a minimal generator of $\texttt{closed}(U)$.*

In conclusion, we have a concise representation $\max_k(U)$ to represent the collection $\mathcal{X} = \{(X, X^+) \mid X \subseteq U\}$ under the relational schema $(R(U), \Sigma_k)$. We move on and focus on how to efficiently construct and maintain index structures for the sequence $\langle \max_1(U), \ldots, \max_n(U) \rangle$.

4.2 Index Structures

This section studies how to represent the sequence $\langle max_1(U), \ldots, max_n(U) \rangle$ where $max_k(U)$ is the family of maximal sets under the relational schema $(R(U), \Sigma_k)$. We answer the following two questions first.

How to represent an attribute set? An attribute set $X \subseteq U$ is represented by a bitset B_X of length $|U|$. Assume that the attributes in the attribute set U are indexed from 1 to $|U|$ by some total order (e.g., lexicographic order). Then, $B_X[i] = 1$ iff $U[i] \in X$. For example, consider a universal attribute set $U = \{a, b, c, d\}$. An attribute set $X = \{a, b, c\}$ can be represented by a bitset $B_X = \langle 1110 \rangle$. It is easy to transform a bitset back to an

[2]For brevity, we abbreviate a collection in the form of $\{a, b, c, d\}$ to a string *abcd*.

attribute set. Thus, we do not distinguish an attribute set X with its bitset representation B_X when it is clear from the context. That is, we often refer the bitset B_X as the attribute X directly for brevity.

How to represent an attribute set shared by multiple maximal sets? Given an attribute set $X \subseteq U$, consider the set of integers $I = \{i \mid X \in \max_i(U), 1 \leq i \leq n\}$. Then, the set I only consists of a sequence of consecutive integers. It is because, if we remove an attribute set from the family of maximal sets, say $\max_i(U)$, we will never add it back to a family of maximal sets $\max_j(U)$ where $j > i$. The proposition is implied by Theorem 4. We will revisit it in the next section along with Theorem 4. Now we only need two integers $I_{\min}$ (the minimum value of I) and $I_{\max}$ (the maximum value of I) to represent the set I. That is, $I = \{i \mid I_{\min} \leq i \leq I_{\max}\}$.

Now we are ready to represent the sequence $\langle max_1(U), \ldots, max_n(U)\rangle$ as an index structure. Next, we present the `SIndex` structure using an example.

Example 4. *Consider the following families of maximal sets under the relational schema $(R(U), \Sigma)$ where $U = \{a, b, c, d\}$ and $\Sigma = \langle a \to b, b \to c, \{c, d\} \to a\rangle$:*

$$max_1(U) = \{\{a, b, c\}, \{a, b, d\}, \{b, c, d\}, \{c, d\}\}$$
$$max_2(U) = \{\{a, b, c\}, \{b, c, d\}, \{c, d\}, \{d\}\}$$
$$max_3(U) = \{\{a, b, c\}, \{b, c\}, \{c\}, \{d\}\}$$

We can represent them as a set of triples (X, l, r) where $X \subseteq U$ is an attribute set, and l and r are two integers such that $X \in max_i(U)$ iff $l \leq i \leq r$. Figure 4.2 shows the representations for the relational schema $(R(U), \Sigma_2)$ and the relational schema $(R(U), \Sigma_3)$.

	bitset	l	r
abc	1110	1	2
abd	1011	1	1
bcd	0111	1	2
cd	0011	1	2
d	0001	2	2

(a) $\langle \max_1(U), \max_2(U)\rangle$

	bitset	l	r
abc	1110	1	3
abd	1011	1	1
bcd	0111	1	2
cd	0011	1	2
d	0001	2	3
bc	0110	3	3
c	0010	3	3

(b) $\langle max_1(U), max_2(U), max_3(U)\rangle$

Figure 4.2: The representation of a set of triples under $(R(U), \Sigma_2)$ and $(R(U), \Sigma_3))$

The `SIndex` *structure is a variant of such a representation. Figure 4.3 shows the* `SIndex` *structure for the relational schema $(R(U), \Sigma_2)$ and the relational schema $(R(U), \Sigma_3)$. It uses an additional integer k to represent the current size of FDs along with a list of triples. The list of triples is similar to the one in Figure 4.2. The difference is that, for each triple (X, l, r) in the list, we also allow the value of r to be a special value ∞—a value that is larger than any integer—such that $X \in max_i(U)$ iff $l \leq i \leq \min(k, r)$.*

k = 2

	bitset	l	r
abc	1110	1	∞
abd	1011	1	1
bcd	0111	1	∞
cd	0011	1	∞
d	0001	2	∞

(a) $\langle \max_1(U), \max_2(U) \rangle$

k = 3

	bitset	l	r
abc	1110	1	∞
abd	1011	1	1
bcd	0111	1	2
cd	0011	1	2
d	0001	2	∞
bc	0110	3	∞
c	0010	3	∞

(b) $\langle max_1(U), max_2(U), max_3(U) \rangle$

Figure 4.3: The improved representation of a set of triples under $(R(U), \Sigma_2)$ and $(R(U), \Sigma_3)$)

Such a representation reduces the cost of maintaining the index structure incrementally. For example, consider the index structure for the relational schema $(R(U), \Sigma_2)$ and $(R(U), \Sigma_3)$. Two attribute sets $\{a, b, c\}$, $\{d\}$ are both in $\max_2(U)$ and $\max_3(U)$. Compared Figure 4.2a with Figure 4.2b, we have to update the value of r for both the two attribute sets. On the contrast, compared Figure 4.3a with Figure 4.3b, we only need to update the value of k.

We define the above `SIndex` structure (simple list-based index structure) precisely as follows.

Definition 13 (`SIndex`)**.** *Given a relational schema $(R(U), \Sigma)$, a* `SIndex` *structure consists of an integer k that indicates the number of FDs in Σ and a list of triples. Each triple in the list consists of a bitset representation of an attribute set $X \subseteq U$, an integer l and a field r that can be either an integer or ∞ (a value that is larger than any integer) such that for any integer $i \leq k$ and $l \leq i \leq r$, we have $X \in max_i(U)$.*

The core component of the `SIndex` structure is to represent a collection of bitsets of fixed length. We present another index structure—the `TIndex` (trie-based index structure)—to represent the collection of bitsets. The idea is to use a compressed prefix-tree data structure (also known as radix tree or crit-bit tree [25]) to represent the collection of bitsets. We present the `TIndex` structure using an example.

Example 5. *Consider the bitsets for the sequence $\langle max_1(U), max_2(U) \rangle$ where*

$$max_1(U) = \{0011, 0111, 1011, 1110\}$$
$$max_2(U) = \{0001, 0011, 0111, 1110\}$$

The `TIndex` *structure consists of an integer k, which records the current size of FDs, and a tree structure T, as shown in Figure 4.4b. The equivalent* `SIndex` *structure is shown in Figure 4.4a for comparison. The tree has two types of nodes—internal nodes (Ⓐ, Ⓑ, Ⓒ and Ⓓ) and external nodes (marked with a gray box with two fields).*

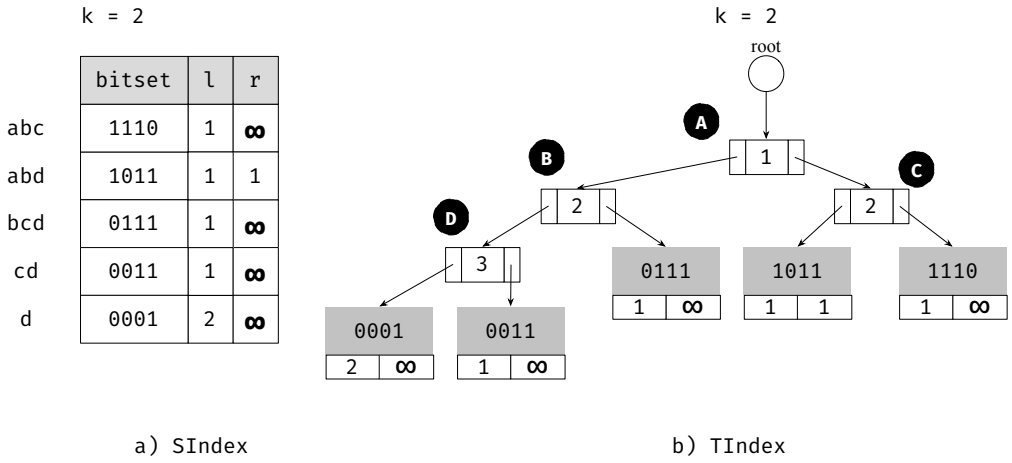


Figure 4.4: The `SIndex` structure and the `TIndex` structure under the relational schema $(R(U), \Sigma)$ where $U = \{a, b, c, d\}$ and $\Sigma = \langle a \to b, b \to c \rangle$.

Similar to the `SIndex` *structure, each external node consists of a bitset representation of an attribute set X, an integer l and a filed r (either an integer or ∞) such that for any integer i with $l \leq i \leq \min(r, k)$, we have $X \in max_i(U)$. Here we have five unique bitsets in $max_1(U) \cup max_2(U)$, and thus we have five external nodes.*

Each internal node consists of an integer and two pointers—a left pointer to the left-subtree and a right pointer to the right-subtree. The integer, denoted by pos, indicates the first position where the bitsets of the node's left-subtree and the bitsets of the node's right-subtree differs. For any bitset in the left-subtree, the bit at pos is 0. For any bitset in the right-subtree, the bit at pos is 1. For example, consider the internal node Ⓑ. Its left subtree consists of a collection of two bitsets $\{\langle 0001 \rangle, \langle 0011 \rangle\}$, and its right subtree consists of a collection of one bitset $\{\langle 0111 \rangle\}$. The common prefix of these two collections is $\langle 0 \rangle$, and the first position where two collections differ is the second bit (i.e., $pos = 2$). In other words, the subtree rooted at node Ⓑ represents the collection of bitsets with the prefix $\langle 0 \rangle$. The left-subtree of node Ⓑ represents the collection of bitsets with the prefix $\langle 00 \rangle$. The right-subtree of node Ⓑ represents the collection of bitsets with the prefix $\langle 01 \rangle$.

We define the above `TIndex` structure precisely as follows.

Definition 14 (`TIndex`)**.** *Given a relational schema $(R(U), \Sigma)$, a* `TIndex` *structure consists of an integer k that indicates the number of FDs in Σ and a tree structure T. The root of the tree is either an external node or an internal node.*

Each external node consists of a bitset representation of an attribute set $X \subseteq U$, an integer l and a field r that can be either an integer or either an integer or ∞ (a value that is larger than any integer) such that for any integer $i \leq k$ and $l \leq i \leq r$, we have $X \in max_i(U)$.

Each internal node consists of an integer pos, a pointer to the left subtree and a pointer to the right subtree. For any external node (B_1, l_1, r_1) *in the left subtree and any external node* (B_2, l_2, r_2) *in the right subtree, we have* $B_1[pos] = 0$, $B_2[pos] = 1$, *and the longest common prefix of* B_1 *and* B_2 *is of length* $(pos - 1)$.

The `SIndex` structure stores the essential information of the family of maximal sets. Compared with the `TIndex` structure, the `SIndex` structure basically consists of the external nodes in the `TIndex` structure. The `TIndex` structure organizes the family of maximal sets into a tree structure to allow traversing the family of maximal sets in various ways. For instance, in Example 5, we may want to traverse all maximal sets not containing the attribute a. The `TIndex` structure (Figure 4.4b) allows us to prune the left subtree completely—the one rooted at node ❽.

4.3 Constructing the Index Structures

This section presents how to compute the sequence $\langle \max_1(U), \ldots, \max_n(U) \rangle$ incrementally and how to construct the `SIndex` structure and the `TIndex` structure.

4.3.1 The foundation: update the family of maximal sets incrementally

How to compute $\max_k(U)$ *incrementally on* k*?* We answer the following three questions.

What attribute sets should we remove when getting $\max_{k+1}(U)$ **from** $\max_k(U)$**?** Theorem 4 is a sufficient and necessary condition of removing an attribute set from the family of maximal sets.

What attribute sets should we add when getting $\max_{k+1}(U)$ **from** $\max_k(U)$**?** Theorem 5 is a necessary condition of adding an attribute set to the family of maximal sets. With Theorem 6, we can validate whether we should add an attribute set to the family of maximal sets.

How to compute the base case $\max_1(U)$**?** Theorem 7 gives the result of a special case $\max_0(U)$ where the set of FDs is empty. We can compute the base case $\max_1(U)$ from $\max_0(U)$.

What attributes to remove

The following theorem gives a necessary and sufficient condition on whether to remove an attribute set from the family of maximal sets after adding a new FD to the relational schema.

Theorem 4. *Given an attribute set* $X \in \max_k(U)$, *then* $X \notin \max_{k+1}(U)$ *iff* $f_{k+1}.lhs \subseteq X$ *but* $f_{k+1}.rhs \not\subseteq X$.

We prove Theorem 4 by combining the following two lemmas.

Lemma 2. *Given an attribute set $X \in \max_k(U)$, then $X \notin \max_{k+1}(U)$ iff X is not closed under the relational schema $(R(U), \Sigma_{k+1})$.*

Proof. $\Longleftarrow$: Trivial since a maximal set must be closed.

$\Longrightarrow$: Consider an attribute set X such that $X \in \max_k(U)$ but $X \notin \max_{k+1}(U)$. Assume that X is closed under the relational schema $(R(U), \Sigma_{k+1})$.

Since $X \in \max_k(U)$, there exists an attribute $a \in U$ such that $X \in \max_k(U, a)$. Then, we have $\Sigma_k \nvdash X \to a$, and thus $a \notin X$. Since X is closed under $(R(U), \Sigma_{k+1})$ and $a \notin X$, we have $\Sigma_{k+1} \nvdash X \to a$.

Given that $\Sigma_{k+1} \nvdash X \to a$ and $X \notin \max_{k+1}(U)$, there must exist an attribute set X' such that $X \subsetneq X'$ and $X' \in \max_{k+1}(U, a)$. Otherwise, by the definition of maximal sets, we have $X \in \max_{k+1}(U, a)$, which contradicts with the condition $X \notin \max_{k+1}(U)$. Since $X' \in \max_{k+1}(U, a)$, X' is closed under the relational schema $(R(U), \Sigma_{k+1})$. As a result, the attribute set X' is also closed under the relational schema $(R(U), \Sigma_k)$. However, given that $X \in \max_k(X)$, a superset of X cannot be closed under the relational schema $(R(U), \Sigma_k)$, which leads a contradiction. □

Lemma 3. *Given an attribute set $X \in \max_k(U)$, then X is not closed under the relational schema $(R(U), \Sigma_{k+1})$ iff $f_{k+1}.lhs \subseteq X$ but $f_{k+1}.rhs \not\subseteq X$.*

Proof. $\Longleftarrow$: Consider an attribute set $X \in \max_k(U)$ such that $f_{k+1}.lhs \subseteq X$ and $f_{k+1}.rhs \not\subseteq X$. Since $f_{k+1}.lhs \subseteq X$, we have $f_{k+1}.rhs \subseteq X^+$ under $(R(U), \Sigma_{k+1})$. Then, we have $X \subsetneq X \cup \{f_{k+1}.rhs\} \subseteq X^+$. Thus, X is not closed under $(R(U), \Sigma_{k+1})$.

$\Longrightarrow$: Consider an attribute set $X \in \max_k(U)$ such that X is not closed under $(R(U), \Sigma_{k+1})$. Since $X \in \max_k(U)$, X is closed under $(R(U), \Sigma_k)$. Thus, $X^+ = X$ under $(R(U), \Sigma_k)$. We study how the closure of X will change after adding an FD f_{k+1} to the set Σ_k of FDs. According to the closure computation algorithm [17], if $f_{k+1}.lhs \not\subseteq X$, then we will not add any attributes to the closure of X; if $f_{k+1}.rhs \subseteq X$, then adding $f_{k+1}.rhs$ to the closure of X does not change its value. In both cases, we still have $X^+ = X$ under $(R(U), \Sigma_{k+1})$. It leads a contradiction of the condition that X is not closed under $(R(U), \Sigma_{k+1})$. □

Remark. When discussing how to represent an attribute shared by multiple maximal sets in the last section, we claim that if we remove an attribute set from the family of maximal sets, say $\max_i(U)$, we will never add it back to the family of maximal sets $\max_j(U)$ where $j > i$. The proposition is implied by Lemma 2. If we remove an attribute set X from $\max_i(U)$, then X is not closed under $(R(U), \Sigma_i)$. Adding more FDs to Σ_i will not make the attribute set X closed.

What attributes to add

The following theorem gives a necessary condition on whether to add a new attribute set from the family of maximal sets after adding a new FD to the relational schema.

Theorem 5. *Consider an attribute set $W \subseteq U$ such that $W \notin \max_k(U)$. If $W \in \max_{k+1}(U)$, then there exist two attribute sets X and Y in $\max_k(U)$ such that $W = X \cap Y$, $X \notin \max_{k+1}(U)$, $Y \in \max_{k+1}(U)$ and $f_{k+1}.lhs \not\subseteq Y$.*

Proof. First, we have the following known theorem.

Theorem ([15])**.** *Consider a relational schema $(R(U), \Sigma)$ and an attribute set $W \in \max_{k+1}(U, a)$ where $a \in U$. Then, either $W \in \max_k(U, a)$ or there exist two attribute sets X and Y such that $W = X \cap Y$, $X \in \max_k(U, a)$, $Y \in \max_k(U, b)$ and $b \in f_{k+1}.lhs$.*

Consider an attribute set $W \in \max_{k+1}(U, a)$ where $a \in U$. Let X and Y be the two attribute sets such that $W = X \cap Y$, $X \in \max_k(U, a)$, $Y \in \max_k(U, b)$ and $b \in f_{k+1}.lhs$, as stated in the above theorem.

The attribute sets X and Y cannot both belong to $\max_{k+1}(U)$. Otherwise, $W \in \max_{k+1}(U)$ can be expressed as the intersection of the two attribute sets in $\max_{k+1}(U)$, which leads a contradiction that $\max_{k+1}(U)$ is a generator of minimum size for the family of closed sets (Theorem 3).

Now we only need to prove that $Y \in \max_{k+1}(U)$. Since $Y \in \max_k(U, b)$, we have $b \notin Y$. Along with $b \in f_{k+1}.lhs$, we have $f_{k+1}.lhs \not\subseteq Y$. By Theorem 4, we also have $Y \in \max_{k+1}(U)$. □

To check whether the attribute set W in Theorem 5 is indeed a maximal set, we use the following theorem to validate.

Theorem 6. *Consider an attribute set W that is closed under the relational schema $(R(U), \Sigma_k)$. Then, $W \in \max_k(U)$ iff $C \setminus W \neq \emptyset$ where $C = \bigcap_{b \in U \setminus W} (W \cup \{b\})^+$.*

Proof. $\Longrightarrow$: Since $W \in \max_k(U)$, there exists an attribute $a \in U$ such that $W \in \max_k(U, a)$. Then, we have $\Sigma_k \nvDash W \to a$ and $\Sigma_k \vDash W \cup \{b\} \to a$ for any attribute $b \in U \setminus W$. By the definition of the closure (Definition 5), we have $a \notin W$ and $a \in C$. Thus, $C \setminus W \neq \emptyset$.

$\Longleftarrow$: Consider an attribute $a \in C \setminus W$. Given that $a \notin W$ and W is closed, we have $\Sigma_k \nvDash W \to a$. For any attribute $b \in U \setminus W$, since $a \in C$, we have $\Sigma_k \vDash W \cup \{b\} \to a$. By the definition of maximal sets, $W \in \max(U, a) \subseteq \max(U)$. □

Base Case

The base case gives the result for the family of maximal sets $\max_0(U)$ and helps us to compute $\max_1(U)$.

Theorem 7. *Let $\max_0(U)$ be the family of maximal sets under the relational schema $(R(U), \Sigma_0)$ where $\Sigma_0 = \emptyset$. Then, $max_0(U) = \{U \setminus \{a\} \mid a \in U\}$*

Proof. Consider an attribute $x \in U$. First, we prove that $U \setminus \{x\} \in \max_0(U, x)$. We have $\Sigma_0 \not\vdash U \setminus \{x\} \to x$. Otherwise, the set Σ_0 of FDs cannot be empty. Also, the FD $U \to x$ is trivial. By the definition of maximal sets, we have $U \setminus \{x\} \in \max_0(U, x)$.

Next, we prove that the attribute set $U \setminus \{x\}$ is the only element in $\max_0(U, x)$. Assume that there is another attribute set $Y \in \max_0(U, x)$ and $Y \neq U \setminus \{x\}$. Then, we have $Y \subsetneq U \setminus \{x\}$. Otherwise, we have $x \in Y$, which contradicts with the condition $\Sigma \not\vdash Y \to x$ for the maximal set Y. Given that $U \setminus \{x\} \in \max_k(U)$ and $Y \subsetneq U \setminus \{x\}$, the attribute set Y cannot be maximal.

Finally, $\max_0(U) = \cup_{x \in U} \max_0(U, x) = \{U \setminus \{a\} \mid a \in U\}$. □

4.3.2 Index Construction Algorithm

With the foundation built in the previous subsection, we present the index construction algorithm. The algorithm (Algorithm 1) starts from $\max_0(U)$ (Theorem 7) and incrementally updates the family of maximal sets—either by removing existing attribute sets (Theorem 4) or by adding new attribute sets (Theorem 5 and Theorem 6).

The algorithm computes the family of maximal sets $\max_k(U)$ iteratively. At each iteration (Line 4), we search for two collections of attribute sets: $\mathcal{S}$ (Line 6) and $\mathcal{T}$ (Line 7). Then, we check all pairs (X, Y) where $X \in \mathcal{S}$ and $Y \in \mathcal{T}$. We remove the attribute set X from $\mathcal{S}$ and add the attribute set $W = X \cap Y$ if W is qualified (Line 12). To check whether W is qualified (procedure TestMaxSet), we validate the condition in Theorem 6.

Next, we show how the index construction algorithm works on each index structure using examples.

Example 6 (`SIndex`)**.** *Figure 4.5 shows the index structure of the* `SIndex` *structure under the relational schema* $(R(U), \Sigma)$ *where* $U = \{a, b, c, d\}$ *and* $\Sigma = \langle a \to b, b \to c, \{c, d\} \to a \rangle$.

For the base case of $k = 0$*, we have four attribute sets* $\{a, b, c\}, \{a, b, d\}, \{a, c, d\}$ *and* $\{b, c, d\}$.

To compute $\max_1(U)$*, we first compute the collection* $\mathcal{S}$ *(Line 6 in Algorithm 1) by scanning the* `SIndex` *structure. Given that* $f_1 : a \to b$*, by the condition of Theorem 4—* $a \in \{a, c, d\}$ *but* $b \notin \{a, c, d\}$*, we have* $\mathcal{S} = \{\{a, c, d\}\}$*. Thus, the attribute set* $\{a, c, d\}$ *does not belong to* $\max_1(U)$ *and can be removed.*

Now, consider the collection $\mathcal{T}$ *(Line 7 in Algorithm 1). We have* $\mathcal{T} = \{\{b, c, d\}\}$*. Along with the collection* $\mathcal{S} = \{\{a, c, d\}\}$*, we get a candidate maximal set* $W = \{a, c, d\} \cap \{b, c, d\} = \{c, d\}$*. We have to verify whether the attribute set* $W = \{c, d\}$ *is indeed a maximal set. To do so, we combine* W *with an attribute* $x \in U \setminus W$ *and compute the closure* $(W \cup \{x\})^+$*. Here, we have two closures* $(W \cup \{a\})^+ = \{a, b, c, d\}$ *and* $(W \cup \{b\})^+ = \{b, c, d\}$*. The intersection of these two closures is* $\{b, c, d\}$*. This intersection has shared elements* $\{b\}$ *besides* $W = \{c, d\}$*. Thus, the attribute set* $\{c, d\}$ *is indeed a maximal set and should be added to* $\max_1(U)$.

Algorithm 1 Construct an index structure $\mathbb{S}$ on the relational schema $(R(U), \Sigma)$

1: **procedure** ConstructIndex($R(U), \Sigma$)
   Input: a relational schema $(R(U), \Sigma)$ where $\Sigma = \langle f_1, \ldots, f_n \rangle$
   Output: an index structure $\mathbb{S}$ for the sequence $\langle \max_1(U), \ldots, \max_n(U) \rangle$
2: $\max_0(U) \leftarrow \{U \setminus \{x\} \mid x \in U\}$
3: $k \leftarrow 0$
4: **for** $i \leftarrow 1 \ldots n$ **do**
5: $\max_i(X) \leftarrow \max_k(X)$
6: $\mathcal{S} \leftarrow \{X \in \max_k(U) \mid f_i.lhs \subseteq X \wedge f_i.rhs \not\subseteq X\}$
7: $\mathcal{T} \leftarrow \{X \in \max_k(U) \mid f_i.lhs \not\subseteq X\}$
8: **for** $X \in \mathcal{S}$ **do**
9: remove X from $\max_i(U)$
10: **for** $Y \in \mathcal{T}$ **do**
11: $W \leftarrow X \cap Y$
12: **if** TestMaxSet(W, i) **then**
13: add W to $\max_i(U)$
14: add $\max_i(X)$ to the index structure $\mathbb{S}$
15: $k \leftarrow i$

16: **procedure** TestMaxSet(W, i)
   Output: Does $W \in \max_i(U)$?
17: $r \leftarrow U \setminus W$
18: **for** $b \in U \setminus W$ **do**
19: $C \leftarrow (W \cup \{b\})^+$ under $(R(U), \Sigma_i)$
20: $r \leftarrow r \cap C$
21: **if** $r = \emptyset$ **then**
22: **return** `false`
23: **return** `true`

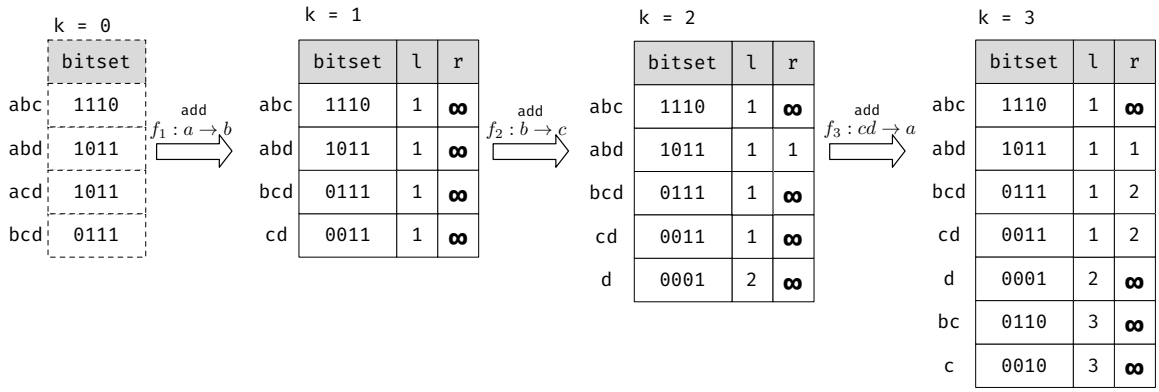


Figure 4.5: Example of index construction on $\Sigma = \langle a \to b, b \to c, cd \to a \rangle$

Figure 4.5 shows how the `SIndex` *structure is updated from* $k = 1$ *to* $k = 3$*. For example, consider how to update the* `SIndex` *after getting* $\max_2(U)$ *from* $\max_1(U)$*. According to the algorithm, we have to remove the attribute set* $\{a, b, d\}$ *and add the attribute set* $\{d\}$*. To remove the attribute set* $\{a, b, d\}$*, we just update the field* r *associated with the attribute set* $\{a, b, d\}$ *to* $k-1$*. To add the attribute set* $\{d\}$*, we add an attribute set with* $l = k$ *and* $r = \infty$*.*

Next, we show an example of the equivalent `TIndex` structure.

Example 7 (`TIndex`)**.** *Figure 4.6 shows the index construction process for the* `TIndex` *structure. The computation of the family of maximal sets is similar to the one in the* `SIndex` *structure. We only show how the tree structure is updated.*

Figure 4.6 shows how the `TIndex` *structure is updated from* $k = 1$ *to* $k = 3$*. Consider how to update the index structure after getting* $\max_2(U)$ *from* $\max_1(U)$*. We have to remove* $\langle 1011 \rangle$ *and add* $\langle 0001 \rangle$ *to the index structure* T_1*.*

To remove $\langle 1011 \rangle$ *from the tree* T_1*, we traverse*[3] *the tree to find the external node with the attribute set* $\langle 1011 \rangle$ *and update the field* r *to* 1*. To add* $\langle 0001 \rangle$*, we traverse the tree to find the external node that shares the longest common prefix with* $\langle 0001 \rangle$*—the external node* $\langle 0011 \rangle$*. The first position where the bitset* $\langle 0001 \rangle$ *and the bitset* $\langle 0011 \rangle$ *differs is* 3*. We create an internal node with* $pos = 3$ *(marked as a red internal node in* T_2*).*

Finally, we do a rough comparison of the two index structures on the space cost and the index construction time.

Consider the space cost of the two index structures. The `SIndex` structure consists of all the external nodes in the `TIndex` structure (see Figure 4.4 as a concrete example). The difference lies in the internal nodes of the `TIndex` structure. Let the number of external nodes in the `TIndex` structure be m. Since the `TIndex` structure is a full binary tree (every internal node has two children), the number of internal nodes in the `TIndex` structure is

[3]The traverse algorithm is same with the crit-bit/compressed-prefix tree [25].

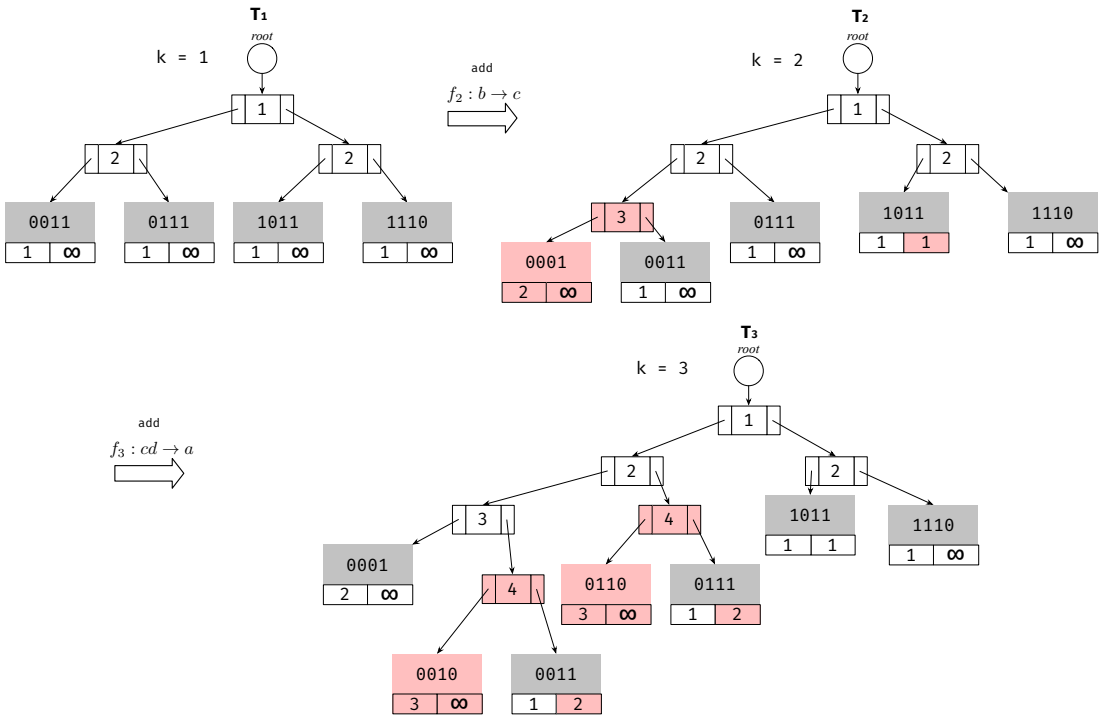


Figure 4.6: The index construction of `TIndex` under the relational schema $(R(U), \Sigma)$ where $U = \{a, b, c, d\}$ and $\Sigma = \langle a \to b, b \to c, \{c, d\} \to a \rangle$. The changes are marked with red.

$m - 1$. Assume that we use 32-bits to represent an integer or a pointer. Each internal node, which consists of an integer and two pointers, requires 96-bits space. Thus, the `TIndex` structure requires $96(m - 1)$-bits additional storage overhead compared with the `SIndex` structure.

Consider the index construction time for the two index structures. As shown in Algorithm 1, the key differences between the two index structures are on how to iterate $\max_k(X)$ for a given k and how to update maximal sets. Again, assume we have m external nodes in the `TIndex` structure—it also implies that we have m entries in the corresponding `SIndex` structure. For the `SIndex` structure, iterating $\max_k(X)$ requires $O(m)$ time; updating maximal sets also requires $O(m)$ time. For the `TIndex` structure, iterating $\max_k(X)$ also requires $O(m)$ time—traversing the whole tree of $2m-1$ nodes. Next, we consider the time of updating maximal sets. Let the height of the `TIndex` structure be h. Since there are at least $2h-1$ nodes and at most $2^{h+1} - 1$ nodes in a full binary tree, we have $2h - 1 \leq 2m - 1 \leq 2^{h+1} - 1$. Then, $\log_2 m \leq h \leq m$. As a result, updating maximal sets in the `TIndex` structure also requires $O(h) = O(m)$ time. Therefore, these two index structures share similar index construction time.

Chapter 5

Query Processing

In this chapter, we present how to leverage the index structures—`SIndex` and `TIndex`—to process the point query and the stable interval query.

5.1 Point Query

Given a relational schema $(R(U), \Sigma)$, a point query (Definition 8) answers a design decision problem—the BCNF test or the primality test—on the relational schema $(R(U), \Sigma_k)$ where $1 \leq k \leq |\Sigma|$. In this section, we first study two core components of the query processing algorithm—computing the closure of an attribute set and computing the projection of maximal sets. Then, we present the query process algorithm for each type of point query.

5.1.1 Core components of point query algorithms

Closure of an attribute set

Given the family of maximal sets $\max_k(U)$*, how to compute the closure of an attribute set* $X \subseteq U$ *under the relational schema* $(R(U), \Sigma_k)$*?*

Theorem 8 builds the connection between the closure of an attribute set and the family of maximal sets.

Theorem 8. *The closure of an attribute set* $X \subseteq U$ *under the relational schema* $(R(U), \Sigma_k)$ *is* $X^+ = \bigcap_{W \in \mathcal{L}} W$ *where* $\mathcal{L} = \{Y \in \max_k(U) \mid X \subseteq Y\}$*. That is, the closure is equal to the intersection of all maximal sets that are the supersets of* X*. Especially, let* $X^+ = U$ *when* $\mathcal{L} = \emptyset$*.*

Proof. Let $P = \bigcap_{W \in \mathcal{L}} W$. We prove that $X^+ = P$ under $(R(U), \Sigma_k)$.

First, we prove that $X^+ \subseteq P$. Consider an attribute set $W \in \mathcal{L}$. Since $X \subseteq W$ and $W \in \max_k(U)$, we have $X^+ \subseteq W^+ = W$. Given that $P = \bigcap_{W \in \mathcal{L}} W$, we have $X^+ \subseteq P$.

Next, we prove that $P \subseteq X^+$. By the definition of the closure (Definition 5), the attribute set X^+ is closed. By Theorem 3, the closed set X^+ can be generated by a collection $\mathcal{L}'$ of maximal sets. That is, $X^+ = \bigcap_{W \in \mathcal{L}'} W$. Since $X \subseteq X^+ \subseteq W$ for each attribute set $W \in \mathcal{L}'$,

we have $\mathcal{L}' \subseteq \mathcal{L}$. Since $P = \bigcap_{W \in \mathcal{L}} W = (\bigcap_{W \in \mathcal{L}'} W) \bigcap (\bigcap_{W \in \mathcal{L} \setminus \mathcal{L}'} W) = X^+ \bigcap (\bigcap_{W \in \mathcal{L} \setminus \mathcal{L}'} W)$, we have $P \subseteq X^+$. □

Algorithm 2 Query the closure X^+ of X under $(R[U], \Sigma_k)$

```
1: procedure Closure(k, X)
2:     L ← {W ∈ max_k(U) | X ⊆ W}
3:     if L = ∅ then
4:         return U
5:     result ← U
6:     for W ∈ L do
7:         result ← result ∩ W
8:     return result
```

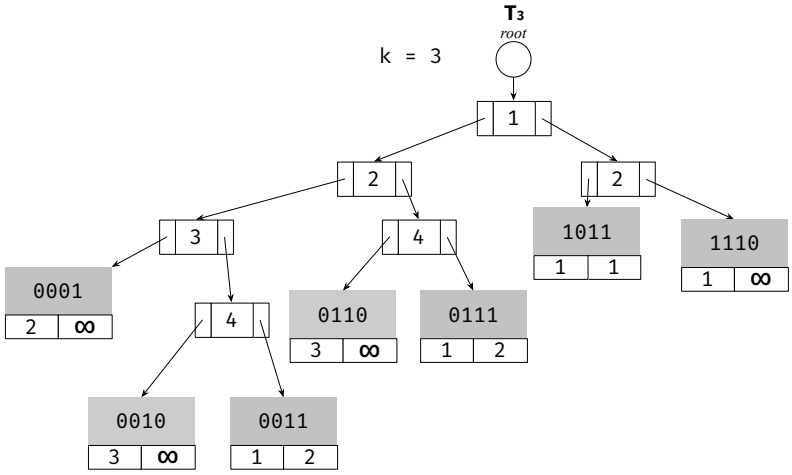
Algorithm 2 applies Theorem 8 to compute the closure of an attribute set. The algorithm searches all supersets of X in $\max_k(U)$ and then takes the intersection of attribute sets in $\mathcal{L}$. If we cannot find a superset of X in our index structure (i.e., $\mathcal{L} = \emptyset$), the algorithm outputs the attribute set U.

Example 8. *Consider a relational schema $(R(U), \Sigma)$ where $U = \{a, b, c, d\}$ and $\Sigma = \langle a \to b, b \to c, \{c, d\} \to a \rangle$. The* `SIndex` *structure and the* `TIndex` *structure are shown in Figure 5.1. We consider how to compute the closures of two attribute sets $\{a, c, d\}$ and $\{c\}$ under the relational schema $(R(U), \Sigma_2)$.*

k = 3

	bitset	l	r
abc	1110	1	∞
abd	1011	1	1
bcd	0111	1	2
cd	0011	1	2
d	0001	2	∞
bc	0110	3	∞
c	0010	3	∞

(a) SIndex



(b) TIndex

Figure 5.1: The `SIndex` structure and the `TIndex` structure under the relational schema $(R(U), \Sigma)$ where $U = \{a, b, c, d\}$ and $\Sigma = \langle a \to b, b \to c, \{c, d\} \to a \rangle$.

1. *Consider the attribute set $X = \{a, c, d\}$:$\langle 1101 \rangle$. For the* `SIndex` *structure, we scan the list to find the supersets of X. For the* `TIndex` *structure, starting from the root, we*

only need to traverse the right-hand subtree since the prefix of X is 1. We find that there does not exist an attribute set which is a super set of X (Line 3). As a result, we return the full set $X^+ = \{a, b, c, d\}$.

2. *Consider the attribute set $X = \{c\}{:}\langle 0010\rangle$. Similarly, we traverse the index structure and check whether the associated interval $[l, r]$ satisfies $k \in [l, r]$. There are three supersets of X such that $k \in [l, r]$—$\{a, b, c\}{:}\langle 1110\rangle$ $\{b, c, d\}{:}\langle 0111\rangle$ and $\{c, d\}{:}\langle 0011\rangle$. The intersection of these three attribute sets is intersection is $\langle 1110\rangle \wedge \langle 0111\rangle \wedge \langle 0011\rangle = \langle 0010\rangle$, which represents the attribute set $\{c\}$. We return $X^+ = \{c\}$.*

Projection of maximal sets

Given the family of maximal sets $\max_k(U)$*, how can we compute* $\max_k(X)$ *where* $X \subseteq U$*?* The problem is known as the projection of the family of maximal sets [17]. To solve this problem, we *generate* candidate maximal sets using Corollary 1 and *verify* the correctness using Theorem 9.

To generate the candidate maximal sets of $\max_k(X)$, for an attribute set $W \in \max_k(U)$, we compute the *projection* of W on the attribute set X—that is, the intersection $W \cap X$.

Corollary 1. *Consider a relational schema $(R(U), \Sigma_k)$. Given an attribute set $W \in \max_k(X)$, there exists an attribute set $W' \in \max_k(U)$ such that $W = W' \cap X$.*

Proof. First, we have the following lemma.

Lemma 4 ([17])**.** *Let $W \in \max(X, a)$ for some $X \subseteq U$. Then $W = W' \cap X$ for some attribute set $W' \in \max(U, a)$*

The lemma implies the corollary as follows. Since $W \in \max_k(X)$, there exists an attribute $a \in X$ such that $W \in \max_k(X, a)$. By the above lemma, there exists an attribute set $W' \in \max_k(U, a) \subseteq \max_k(U)$ such that $W = W' \cap X$. □

To verify whether a candidate attribute set $W \subseteq X$ is indeed a maximal set in $\max_k(X)$. We extend Theorem 6—the theorem that checks whether an attribute set belongs to $\max_k(U)$.

Theorem 9. *Given an attribute set $W \subseteq X$ such that $W = W' \cap X$ and $W' \in \max_k(U)$, we have $W \in \max_k(X)$ iff $C \setminus W \neq \emptyset$ where $C = \cap_{b \in X \setminus W}(X \cap (W \cup \{b\})^+)$.*

Proof. $\Longrightarrow$: Since $W \in \max_k(X)$, there exists an attribute $a \in X$ such that $W \in \max_k(X, a)$. Then, we have $\Sigma_k \nvdash W \rightarrow a$ and $\Sigma_k \vDash W \cup \{b\} \rightarrow a$ for any attribute $b \in X \setminus W$. By the definition of the closure (Definition 5), we have $a \notin W$ and $a \in C$. Thus, $C \setminus W \neq \emptyset$.

$\Longleftarrow$: First, we prove that W is closed on the relation schema $R[X]$ by contradiction. Consider an attribute $x \in W^+ \cap X$ and $x \notin W$. Since $W' \in \max_k(X)$, W' is closed and

thus $W^+ \subseteq W'$. Then, $x \in W'$. Now, since $W = W' \cap X$ and $x \notin W$, we have $x \notin X$. It leads a contradiction with $x \in W^+ \cap X$.

Next, consider an attribute $a \in C \setminus W$. Given that $a \notin W$ and W is closed on X, we have $\Sigma_k \nvDash W \to a$. For any attribute $b \in X \setminus W$, since $a \in C$, we have $\Sigma_k \vDash W \cup \{b\} \to a$. By the definition of maximal sets, $W \in \max(X, a) \subseteq \max(X)$. □

We will see an example of applying these two theorems in action shortly when processing the point query for the BCNF test.

5.1.2 BCNF Test

Consider a relational schema $(R(U), \Sigma)$. The BCNF test asks whether a relation schema $R[X]$ conforms with BCNF under the relational schema $(R(U), \Sigma_k)$ where $X \subseteq U$ and $1 \leq k \leq |\Sigma|$.

The naive method to do BCNF test for $R[X]$ is—by Definition 7—to check whether the closure of any attribute set $W \subseteq X$ is either the attribute set W itself or the attribute set X. With the index structures for the family of maximal sets, we leverage the following theorem to process the point query of BCNF test.

Theorem 10 (Rephrased from [17])**.** *A relation schema $R[X]$ conforms with BCNF under the relational schema $(R(U), \Sigma_k)$ iff for any attribute set $W \in \max_k(X)$, there does not exist an attribute set $W' \subsetneq W$ such that $\Sigma_k \vDash W' \to W$—we also say that the attribute set W is non-redundant.*

To verify that the relation schema $R[X]$ conforms with BCNF, we only need to check whether all attribute sets in $\max_k(X)$ are non-redundant. The index structures—`SIndex` and `TIndex`—maintain the family $\max_k(U)$ of maximal sets. We have studied how to compute the projection $\max_k(X)$ of $\max_k(U)$ in the previous section.

How to check whether an attribute set W is non-redundant? We only need to check whether an attribute $a \in W$ can be derived from the other attributes in W. That is, for any attribute $a \in W$, we must verify that $\Sigma_k \nvDash W \setminus \{a\} \to a$. To do so, we resort to the closure query and check that whether $a \notin (W \setminus \{a\})^+$ holds for any attribute $a \in W$ under the relational schema $(R(U), \Sigma_k)$.

Algorithm 3 applies the above idea to process the point query for BCNF test. The algorithm traverses $\max_k(U)$ in the index structures to compute the projection $\max_k(X)$. For each attribute set $W \in \max_k(X)$, it tests whether W is non-redundant (procedure TestNonRedundancy). Let us see an example.

Example 9. *Consider the relational schema $(R(U), \Sigma)$ where $U = \{a, b, c, d\}$ and $\Sigma = \langle a \to b, b \to c, \{c, d\} \to a \rangle$ again. The* `SIndex` *structure and the* `TIndex` *structure are shown in Figure 5.1 (same with Example 8).*

Algorithm 3 BCNF Test

1: **procedure** BCNFTest(k, X)
Input: An attribute set $X \subseteq U$ and an integer k
Output: Does the sub-schema $R[X]$ conform with BCNF under the relational schema $(R(U), \Sigma_k)$?
2: $visited \leftarrow \emptyset$
3: **for** $W \in \max_k(U)$ **do**
4: $W \leftarrow W \cap X$
5: **if** $W \in visited$ **then continue**
6: add W to $visited$
7: **if** $X = U$ or TestMaxSetOnSubschema(W, k, X) **then**
8: **if** TestNonRedundancy(k, W) = `false` **then**
9: **return** `false`
10: **return** `true`

11: **procedure** TestNonRedundancy(k, W)
Input: An attribute set $W \subseteq U$, an integer k
Output: Is the attribute set W non-redundant under the relational schema $(R(U), \Sigma_k)$?
12: **for** $x \in W$ **do**
13: **if** $x \in$ `Closure`$(k, W \setminus \{x\})$ **then**
14: **return** `false`
15: **return** `true`

16: **procedure** TestMaxSetOnSubschema(W, k, X)
Input: An attribute set X, an attribute set $W \subseteq X$ and an integer k
Output: Does $W \in \max_k(X)$ hold?
17: $result \leftarrow X \setminus W$
18: **if** $result = \emptyset$ **then return** `false`
19: **for** $x \in X \setminus W$ **do**
20: $result \leftarrow result \cap$ `Closure`$(k, W \cup \{x\}) \cap X$
21: **if** $result = \emptyset$ **then**
22: **return** `false`
23: **return** `true`

We show how to process the point queries with $k = 2$. That is, given an attribute set $X \subseteq U$, we ask whether the relation schema $R[X]$ conforms to BCNF under the relational schema $(R(U), \Sigma_2)$. Here we consider an attribute set $X = \{a, c, d\}$.

First, we will traverse the index structures on $\max_2(U) = \{0001, 0011, 0111, 1110\}$. *For the* `SIndex` *structure, we traverse the list (Figure 5.1a). For the* `TIndex` *structure, we traverse the tree structure (Figure 5.1b).*

For each attribute set W, we project it on the attribute set $X = \{a, c, d\}$ (Line 4 in Algorithm 3) to get a new value of W.

1. *The projection of $\langle 0001 \rangle : \{d\}$ on X is $W = \{d\}$. Next, we check whether $\{d\} \in \max_2(X)$ (procedure* TestMaxSetOnSubschema*). To do so, we compute $X \cap (W \cup \{x\})^+$ for each attribute $x \in X \setminus W$ where $X \setminus W = \{a, c\}$. We have $X \cap \{a, d\}^+ = \{a, c, d\}$ and $X \cap \{c, d\}^+ = \{c, d\}$. These two attribute sets share a common attribute c in $X \setminus W = \{a, c\}$. Thus, by Theorem 9, it is a maximal set. Last, we check whether the attribute set is non-redundant (procedure* TestNonRedundancy*). Since there are no other attributes we can remove from the maximal sets, it passes the test.*

2. *The projection of $\langle 0011 \rangle : \{c, d\}$ on X is $W = \{c, d\}$. Next, to check whether $\{c, d\} \in \max_2(X)$. With $X \setminus W = \{a\}$, we have $X \cap \{a, c, d\}^+ = \{a, c, d\}$. It contains an attribute a in $X \setminus W = \{a\}$. Thus, it is a maximal set. Finally, we test whether $\{c, d\}$ is non-redundant. To do so, we check whether $x \notin (W \setminus \{x\})^+$ for each attribute $x \in W$. Since $c \notin \{d\}^+$ and $d \notin \{c\}^+$, it passes the test.*

3. *The projection of $\langle 0111 \rangle : \{b, c, d\}$ on X is $W = \{c, d\}$. It has been computed in the previous iteration, and thus we skip it.*

4. *The projection of $\langle 1110 \rangle : \{a, b, c\}$ on X is $W = \{a, c\}$. Next, we check whether $\{a, c\} \in \max_2(X)$. With $X \setminus W = \{d\}$, we have $X \cap \{a, c, d\}^+ = \{a, c, d\}$. It contains an attribute d in $X \setminus W = \{d\}$. Thus, it is a maximal set. Last, we test whether $\{a, c\}$ is non-redundant. Since $c \in \{a\}^+ = \{a, b, c\}$, it is redundant. Therefore, the relation schema $R[\{a, c, d\}]$ does not conform with BCNF.*

5.1.3 Primality Test

Consider the relational schema $(R(U), \Sigma)$. The primality test asks whether an attribute $a \in X$ belongs to some key of the relation schema $R[X]$ where $X \subseteq U$ under the relational schema $(R(U), \Sigma_k)$.

The naive method of primality test is to check whether any attribute set $W \subseteq X$ that contains the attribute $a \in X$ is a key of the relation schema $R(X)$.

We leverage the following theorem to use our index structures for the family of maximal sets.

Theorem 11. *Let the maximal elements of a collection $\mathcal{X}$ w.r.t. the inclusion relationship be* $\texttt{MaxElements}(\mathcal{X}) = \{W \in \mathcal{X} \mid \nexists W' \in \mathcal{X}$ *such that* $W \subsetneq W'\}$. [1] *An attribute $a \in X$ is prime in the relation schema $R[X]$ under the relational schema $(R(U), \Sigma_k)$ iff there exists an attribute set* $W' \in \texttt{MaxElements}(\max_k(X))$ *such that $a \notin W'$.*

Proof. First, we have the following known theorem.

Theorem (Rephrased from [17])**.** *An attribute $a \in X$ is prime in the relation schema $R[X]$ iff $\exists W \in \max(X, a)$ such that $(Wa)^+ = X$ where $(Wa)^+$ is the closure of the attribute set $W \cup \{a\}$ in the relation schema $R[X]$.*

Then, we only need to prove that $\exists W \in \max_k(X, a)$ such that $(Wa)^+ = X \iff \exists W' \in \texttt{MaxElements}(\max_k(X))$ such that $a \notin W'$.

$\Longrightarrow$: Consider an attribute set $W \in \max_k(X, a)$ such that $(Wa)^+ = X$. If $W \in \texttt{MaxElements}(\max_k(X))$, we have already found a qualified attribute set $W' = W$ since $a \notin W$. If $W \notin \texttt{MaxElements}(\max_k(X))$, consider an attribute set $W' \in \texttt{MaxElements}(\max_k(X))$ such that $W \subseteq W'$. Since $W' \in \max_k(X)$, we have $(W')^+ \subsetneq X$. Given that $(Wa)^+ = X$ and $W \subseteq W'$, we have $a \notin W'$, otherwise $W' = X$. Thus, we have a qualified attribute set W'.

$\Longleftarrow$: Consider an attribute set $W' \in \texttt{MaxElements}(\max_k(X))$ such that $a \notin W'$. We prove that $W' \in \max_k(X, a)$ and $(W'a)^+ = X$.

First, we prove that $W' \in \max_k(X, a)$. Since $W' \in \max_k(X)$, the attribute set W' is closed, and thus $\Sigma_k \not\vdash W' \to a$ given that $a \notin W'$. By the definition of maximal sets (Definition 12), there exists a superset W'' of W' and $W'' \in \max_k(X, a)$. Given that $W' \in \texttt{MaxElements}(\max_k(X, a))$, we have $W'' = W'$.

Next, we prove that $(W'a)^+ = X$. Let $X' = (W'a)^+$ and assume that $X' \neq X$. Then, we have $W' \subsetneq X' \subsetneq X$. Consider an attribute $b \in X \setminus X'$. By the definition of attribute set closure (Definition 5), we have $\Sigma_k \not\vdash X' \to b$. By the definition of maximal sets (Definition 12), there exists an attribute set $X'' \supseteq X'$ such that $X'' \in \max_k(X, b)$. Thus, we have an attribute set $W'' \in \max_k(X)$ and $W \subsetneq W''$. It leads a contradiction with $W \in \texttt{MaxElements}(\max_k(X))$. □

Algorithm 4 applies Theorem 11 to process the primality test. Similar to the algorithm for the BCNF test, the algorithm traverses $max_k(U)$ in the index structure to compute the projection $max_k(X)$ of the family of maximal sets. Then, it computes the maximal elements $\mathcal{U}$ of the collection $\max_k(X)$. For each attribute set $W \in \max_k(X)$, it checks whether the attribute $x \in X$ belong to W. Let us see an example under the same setting of Example 9.

[1] The term *maximal element* comes from the order theory. Here, a maximal element of a subset $\mathcal{X}$ of the poset $(2^U, \subseteq)$ is an element that is not "smaller" than (not contained by) any other element in $\mathcal{X}$.

Algorithm 4 Primality Test

1: **procedure** PRIMALITYTEST(x, k, X)
Input: An attribute set $X \subseteq U$, an attribute $x \in X$ and an integer k
Output: Does attribute x belong to some key of $R[X]$ under the relational schema $(R(U), \Sigma_k)$?
2: $\mathcal{U} \leftarrow \{\emptyset\}$ ▷ A set holding the maximal elements.
3: **for** $W \in \max_k(U)$ **do**
4: $W \leftarrow W \cap X$
5: **if** $X = U$ or TESTMAXSETONSUBSCHEMA(W, k, X) **then**
6: $\mathcal{U} \leftarrow \mathcal{U} \cup \{W\}$
7: $\mathcal{U} \leftarrow$ `MaxElements`($\mathcal{U}$)
8: **for** $W \in \mathcal{U}$ **do**
9: **if** $x \notin W$ **then return** `True`
10: **return** `False`

Example 10. *Consider the relational schema* $(R(U), \Sigma)$ *where* $U = \{a, b, c, d\}$ *and* $\Sigma = \langle a \to b, b \to c, \{c, d\} \to a \rangle$ *again. The* `SIndex` *structure and the* `TIndex` *structure are shown in Figure 5.1. We show how to answer whether the attribute* a *is prime for the relation schema* $R[X]$ *where* $X = \{a, c, d\}$ *under the relational schema* $(R(U), \Sigma_2)$.

The procedure of computing the projection of maximal sets is the same as Example 9. We have $\max_2(X) = \{\{d\}, \{c, d\}, \{a, c\}\}$. *The maximal elements of* $\max_2(X)$ *are the collection* $\{\{c, d\}, \{a, c\}\}$. *Since the attribute* $a \notin \{c, d\}$, *we know that the attribute* a *is prime.*

5.2 Stable Interval Query

The stable interval query (Definition 9) is based on the point queries discussed in the previous section. Consider a relational schema $(R(U), \Sigma)$. Given an integer k with $1 \leq k \leq |\Sigma|$, the stable interval query answers how stable of a property—BCNF or primality—near k.

The basic idea is to change the value of k step by step—increase by one or decrease by one—and ask the point query to see whether the answer is different from the original one. Next, we present the query processing algorithm for each type of the stable interval query.

5.2.1 Stable Interval Query for the BCNF Test

The stable interval query for the BCNF test asks what is the maximum length interval $[i, j]$ such that $i \leq k \leq j$, and for any integer u with $i \leq u \leq j$, $R[X]$ conforms with BCNF (in the case that $R[X]$ conforms with BCNF under $(R(U), \Sigma_k)$).

Algorithm 5 presents the query processing algorithm for the BCNF test. The algorithm is straightforward and a variant of the standard linear search algorithm. It asks $O(|\Sigma|)$ BCNF tests in the worst case.

Algorithm 5 Stable interval query for the BCNF test under $(R(U), \Sigma)$

```
1: procedure StableIntervalBCNFTest(k, X)
   Input: An attribute set X ⊆ U, an attribute x ∈ X and an integer k
   Output: The maximum length interval [l, r] such that l ≤ k ≤ r and
   BCNFTest(u, X) = BCNFTest(k, X) for any integer u ∈ [l, r]
2:     q ← BCNFTest(k, X)
3:     l ← k − 1
4:     while 1 ≤ l ≤ n do
5:         if BCNFTest(l, X) = q then l ← l − 1
6:         else break
7:     l ← l + 1
8:     r ← k + 1
9:     while 1 ≤ r ≤ n do
10:        if BCNFTest(r, X) = q then r ← r + 1
11:        else break
12:    r ← r − 1
13:    return l, r
```

5.2.2 Stable Interval Query for the Primality Test

The stable interval query for the primality test asks what is the maximum length interval $[i, j]$ such that $i \leq k \leq j$, and for any integer u with $i \leq u \leq j$, the attribute $a \in X$ is prime in relation schema $R[X]$ under the relation schema $(R(U), \Sigma_u)$.

For the primality test, similar to the BCNF test, we do a linear search on the boundary values of the stable interval (as shown in Algorithm 6).

The algorithm is also a variant of the standard linear search algorithm. It asks $O(|\Sigma|)$ primality queries in the worst case.

Algorithm 6 Stable interval query for the primality test under $(R(U), \Sigma)$

1: **procedure** StableIntervalPrimalityTest(x, k, X)
   Input: An attribute set $X \subseteq U$, an attribute $x \in X$ and an integer k
   Output: The maximum length interval $[l, r]$ such that $l \leq k \leq r$ and PrimalityTest(x, u, X) = PrimalityTest(x, k, X) for any integer $u \in [l, r]$
2: $q \leftarrow$ PrimalityTest(x, k, X)
3: $l \leftarrow k - 1$
4: **while** $1 \leq l \leq n$ **do**
5: **if** PrimalityTest$(x, l, X) = q$ **then** $l \leftarrow l - 1$
6: **else break**
7: $l \leftarrow l + 1$
8: $r \leftarrow k + 1$
9: **while** $1 \leq r \leq n$ **do**
10: **if** PrimalityTest$(x, r, X) = q$ **then** $r \leftarrow r + 1$
11: **else break**
12: $r \leftarrow r - 1$
13: **return** l, r

Chapter 6

Experiments

In this chapter, we conduct experiments on the performance of query processing algorithms of the point queries and the stable interval queries. We run the experiments on a server with the Intel Xeon CPU E7-4830 v4 2.00GHz processor. We evaluate the performance using the JMH benchmark framework[1] under the Java SE running environment 1.8. For each JVM instance, we limit the maximum memory usage to 16GB.

6.1 Dataset and Parameters

We generate two datasets using the following two data sources [7, 22].

1. Flight: The Flight dataset [22] was collected by the US Bureau of Transportation Statistics and consists of flight route information. It contains 109 attributes.

2. Github: The GitHub dataset [7] was collected by the GHTorrent[2] project. Each record in the dataset is a JSON object. We flatten the JSON records into a universal table. It contains 291 attributes.

Then, we generate the set of FDs by the algorithms in [7, 22] and rank the FDs using the following score function [7]. Given a relation instance r on the relation schema $R(U)$ and an FD $X \to Y$ where $X, Y \subseteq U$, the score of the FD $X \to Y$ is

$$score(X \to Y, r) = \frac{|\Pi_X(r)|}{|\Pi_{XY}(r)|}$$

where $\Pi_X(r) = \{t[X] \mid t \in r\}$ is the projection of the relation r on the attribute set X, and $\Pi_{XY}(r) = \{t[XY] \mid t \in r\}$ is the projection of the relation r on the attribute set XY.

The Flight dataset consists of 4097 FDs, and the Github dataset consists of 83810 FDs. Assuming that we are interested in the top-k FDs, to evaluate the performance of our

[1] JMH is "a Java harness for building, running, and analyzing nano/micro/milli/macro benchmarks written in Java and other languages targeting the JVM" [26].

[2] `http://ghtorrent.org/`

algorithms, we choose different scales of k as follows. Let $p = \log_2 n$ where n is the number of FDs in the dataset. We evaluate a range of k of values $2^1, 2^2, \ldots, 2^p$. For the Flight dataset, we have $p = 12$. For the Github dataset, we have $p = 16$.

6.2 Index Construction

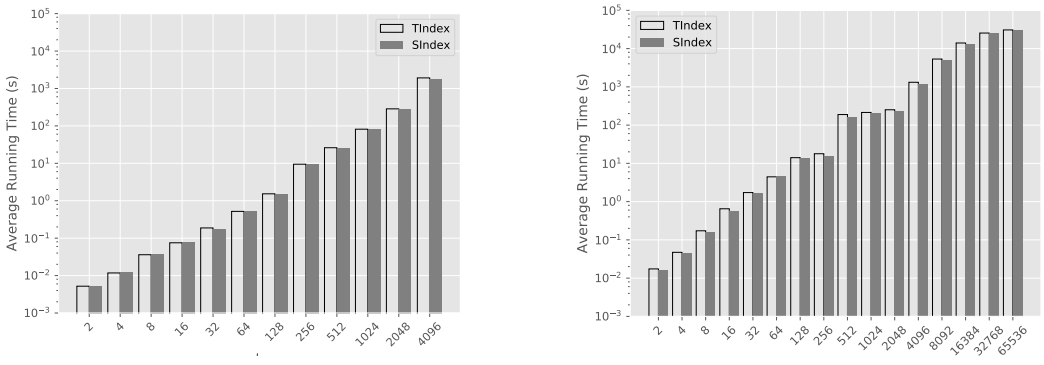
We first evaluate the index sizes and the index construction time on the two datasets.

For the index sizes, we construct the `SIndex` structure and the `TIndex` structure on each dataset and show the sizes in Table 6.1. Their sizes are similar, but the `TIndex` structure has slightly larger size due to the cost of maintaining a tree structure besides the family of maximal sets.

Table 6.1: The size of each index structure on each dataset

Dataset	Index Structure	Size (MB)
Flight	`SIndex`	1.82
	`TIndex`	1.85
Github	`SIndex`	13.12
	`TIndex`	13.18

For the index construction time, Figure 6.1 reports the index construction time for each dataset versus k—the number of FDs involved. On both datasets, the index construction time for both index structures is close. Again, the `TIndex` structure shows slightly larger cost compared to the `SIndex` structure.



(a) Flight (b) Github

Figure 6.1: Index construction time versus k

6.3 BCNF Test

6.3.1 Point Query: BCNF Test

Workload. We randomly generate six attribute sets labeled from X_1 to X_6, each of which is of length 16. For each attribute set X_i, we ask whether the relation schema $R[X_i]$ conforms with BCNF under $(R(U), \Sigma_k)$ where $k = 2, 2^2, \ldots, 2^p$ ($p = 12$ for the Flight dataset, and $p = 16$ for the Github dataset).

Baseline. The baseline method is to enumerate all possible subsets of the attribute set X_i and check whether there exists any BCNF violation.

For each attribute set X_i ($i = 1, \ldots, 6$), we show the query execution time of three algorithms—baseline, `SIndex` and `TIndex`—versus the value of k in a log-log scale. The query execution time is related to the stable intervals of BCNF tests, and we use red vertical lines to indicate the boundaries of these stable intervals. We refer the stable intervals as I_1, I_2, I_3 and so on—in the order of their left boundaries.

Figure 6.2 shows the result on the Flight dataset. Both the `SIndex` method and the `TIndex` method outperform the baseline. The `TIndex` method outperforms the `SIndex` method slightly—the only exception is Figure 6.2*a*. Also, the query execution time of the `SIndex` method and the `TIndex` method are stable with respect to k. It is because that the query processing algorithm mainly iterates the family of maximal sets $\max_k(U)$, whose size is stable (Figure 6.2*g*). One interesting figure is Figure 6.2*d*. It has two stable intervals $I_1 = [1, 382]$ and $I_2 = (382, 4096]$. The query execution time decreases when k changes from 256 to 512. The reason is that, for $k \in I_2$, the answer to the BCNF test is false, and all three algorithms will terminate as long as they found a violation which leads a decrease in query execution time.

Figure 6.3 reports the result on the Github dataset. In this dataset, we observe three intervals (I_1, I_2 and I_3) in most cases—the only exception is Figure 6.3*d*. For $k \in I_1$ and $k \in I_3$, the answer to the BCNF test is true, and the `SIndex` method and `TIndex` method outperform the baseline. The query execution time for the `SIndex` method and the `TIndex` method is stable in the interval I_1, since $|\max_k(U)|$ is also stable in this interval (Figure 6.3*g*). For $k \in I_2$, the answer to the BCNF test is false, and the query execution time depends on when a BCNF violation is discovered. In general, the query execution time for all three algorithms is not monotonic with respect to k and goes upwards and downwards in this interval.

6.3.2 Stable Interval Query: BCNF Test

For each dataset, we reuse the six attribute sets $X_1, X_2, \ldots, X_6$ for the point queries. For each attribute set X_i, we ask what is the stable interval (of BCNF test) for the relation schema $R[X_i]$ under the relational schema $(R(U), \Sigma_k)$. We report the query execution time

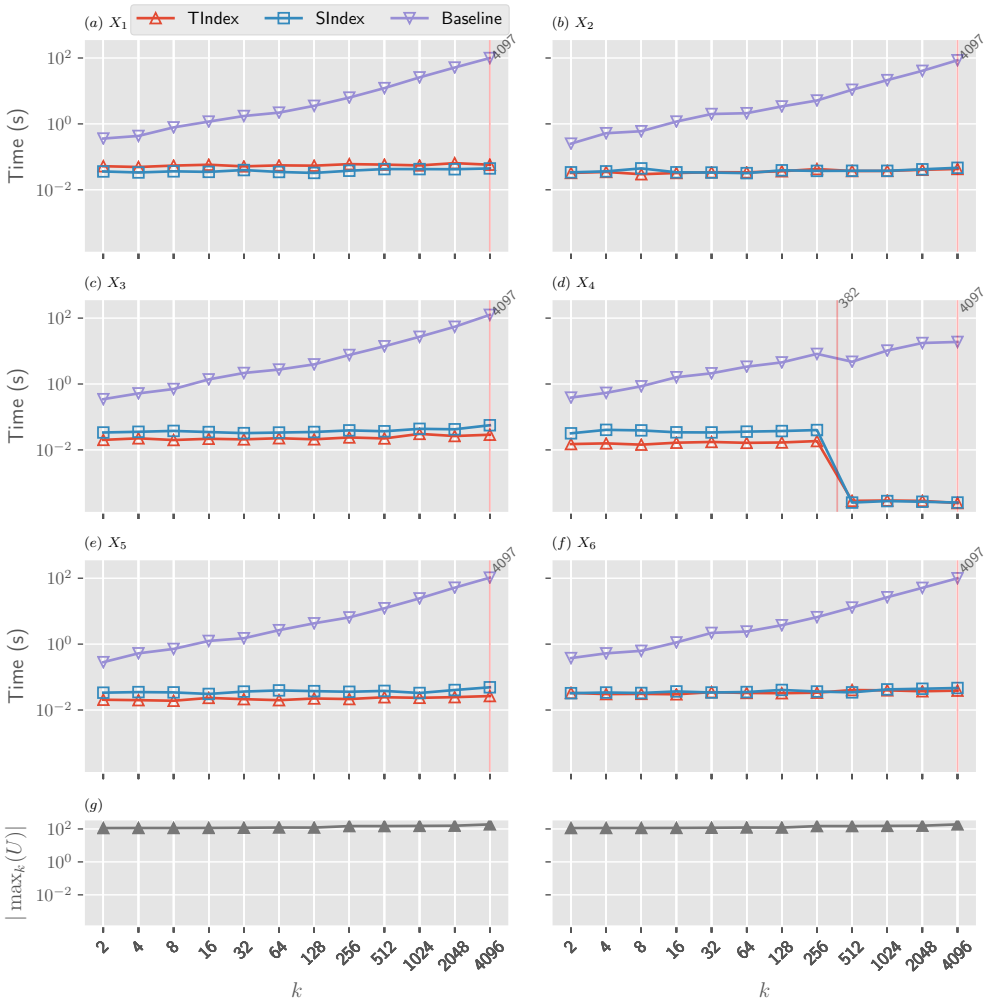


Figure 6.2: Query execution time of the BCNF tests on the Flight dataset

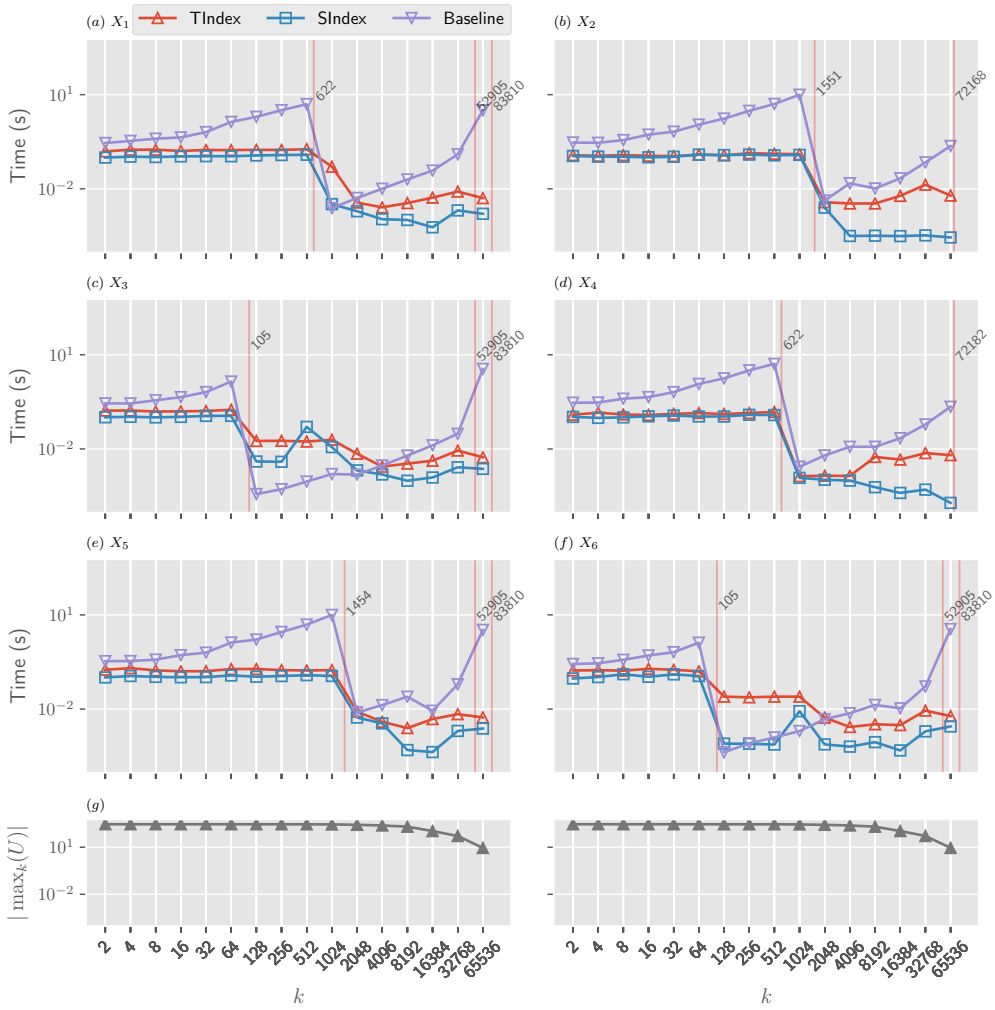


Figure 6.3: Query execution time of the BCNF tests on the Github dataset. The sub-figure (g) shows two identical plots of $|\max_k(U)|$ versus k for easily comparing with the sub-figures in the same column. We follow this convention for all other figures in this chapter.

of the `SIndex` method and the `TIndex` method versus the value of k in a log-log scale.[3] Similar to the plot for the point queries, we also plot the boundaries of stable intervals using red vertical lines.

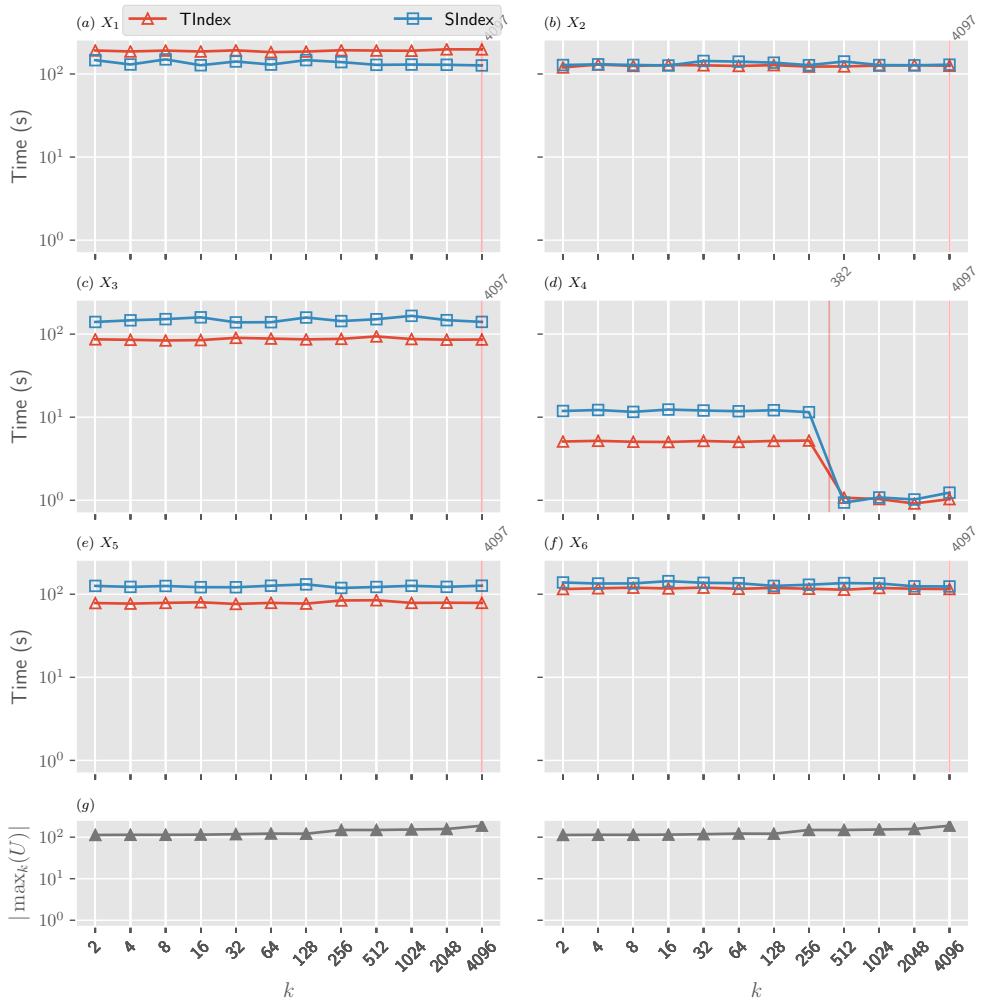


Figure 6.4: Query execution time of the stable interval queries for the BCNF tests on the Flight dataset

Figure 6.4 shows the result on the Flight dataset. The query execution time is stable with respect to k except Figure 6.4d. In Figure 6.4d, we have two stable intervals $I_1 = [1, 382]$ and $I_2 = (382, 4097]$. The query execution time is stable within each interval. *Why?* The execution time of an interval query is roughly equal to the sum of the execution time of all

[3]We do not show the baseline method since it exceeds the time limit (5 minutes) in our experiments.

possible point queries within the corresponding stable interval. For k in the same interval, the set of executed point queries are the same, and thus the query execution time is similar. The result is consistent with the point queries (Figure 6.2). For example, for the attribute set X_4 (Figure 6.4d), the `TIndex` method outperforms the `SIndex` method, and the execution time decreases when k shifts from the stable interval I_1 to the stable interval I_2. These statements are also true for Figure 6.2d.

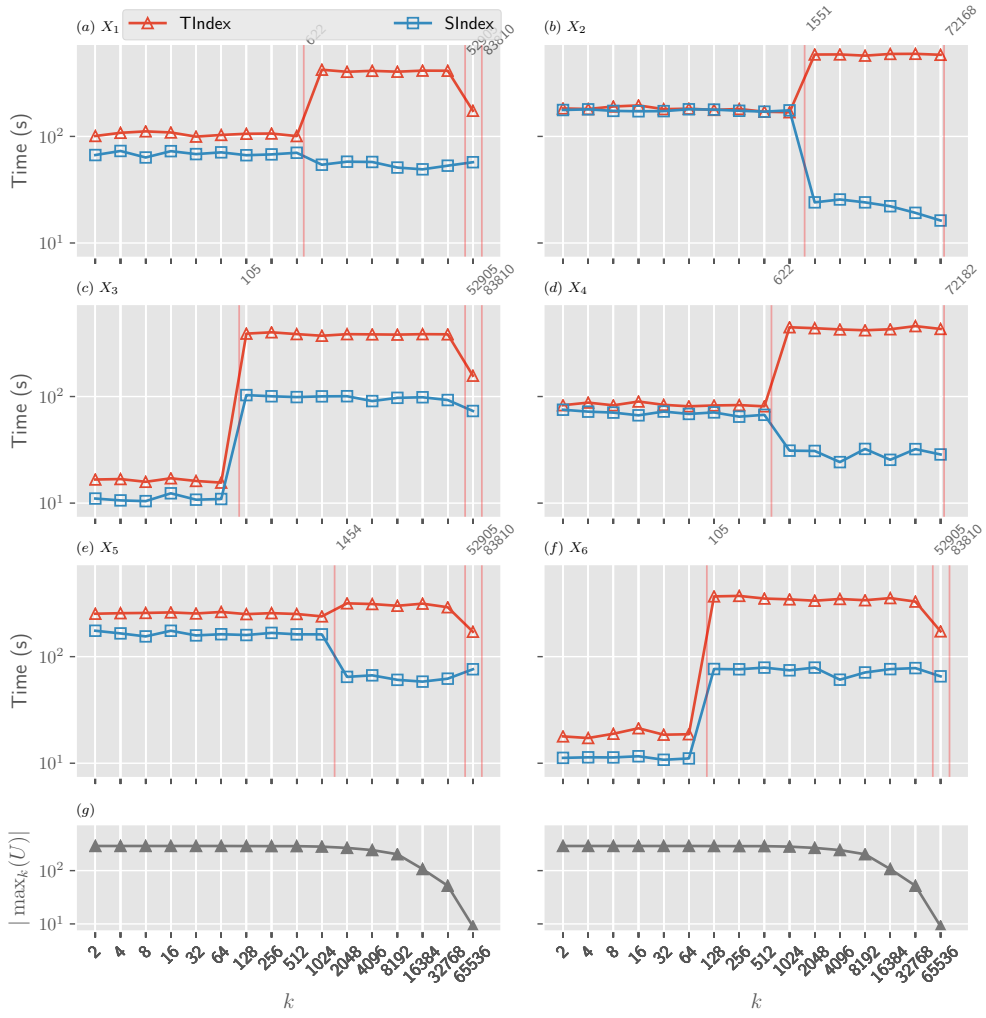


Figure 6.5: Query execution time of the stable interval queries for the BCNF tests on the Github dataset

Figure 6.5 shows the result on the Github dataset. Similar to what we have seen in the Flight dataset, the query execution time for k in the same stable interval is stable. The query execution time is also related to the length of the corresponding stable interval. For

example, for the attribute set X_3 (Figure 6.5c), we have three stable intervals $I_1 = [1, 105]$, $I_2 = (105, 52905]$ and $I_3 = (52905, 83810]$. For the point query (Figure 6.3c), the query execution time for $k \in I_2$ is shorter than the one for $k \in I_1$. However, for the interval query here, the query execution time for $k \in I_2$ is longer than the one for $k \in I_1$. This is due to the length of interval I_2 is much longer than the length of interval I_2.

6.4 Primality Test

6.4.1 Point Query: Primality Test

Workload. For each dataset, we randomly generate six attribute sets, labeled from X_1 to X_6, each of which is of length 6. For each attribute set X and each attribute $A \in X$, we ask whether the attribute A belongs to some primary key of the relation schema $R[X]$ under the relational schema $(R(U), \Sigma_k)$. Here we only present all the six possible queries for the attribute set X_1.[4] We also plot the boundaries of stable intervals using a red vertical line.
Baseline. The baseline algorithm iterates all subsets containing the attribute $A \in X$ and terminates until we find an attribute set which is a key of the relation schema $R[X]$.

Figure 6.6 shows the result on the Flight dataset. The query execution time of the `SIndex` method and the `TIndex` method are stable with respect to k. This is due to the size $\max_k(X)$ being stable. The `SIndex` method and the `TIndex` method outperform the baseline method when k is large enough ($k \geq 512$ in general). The `SIndex` method performs slightly better than the `TIndex` method.

Figure 6.7 shows the result on the Github dataset. Similar to the Flight dataset, the `SIndex` method and the `TIndex` method outperform the baseline when k is large ($k \geq 2048$ in general). Also, both the query execution time of the `SIndex` method and the `TIndex` method decrease dramatically when k changes from 32768 to 65536. This is because the size of $\max_k(U)$ also greatly decreases when k changes.

[4]There are six possible primality test queries for a given attribute set X. Thus, there are 36 queries in total for all the six attribute sets $X_1, \ldots, X_6$. We leave the full results to the appendix for interested readers.

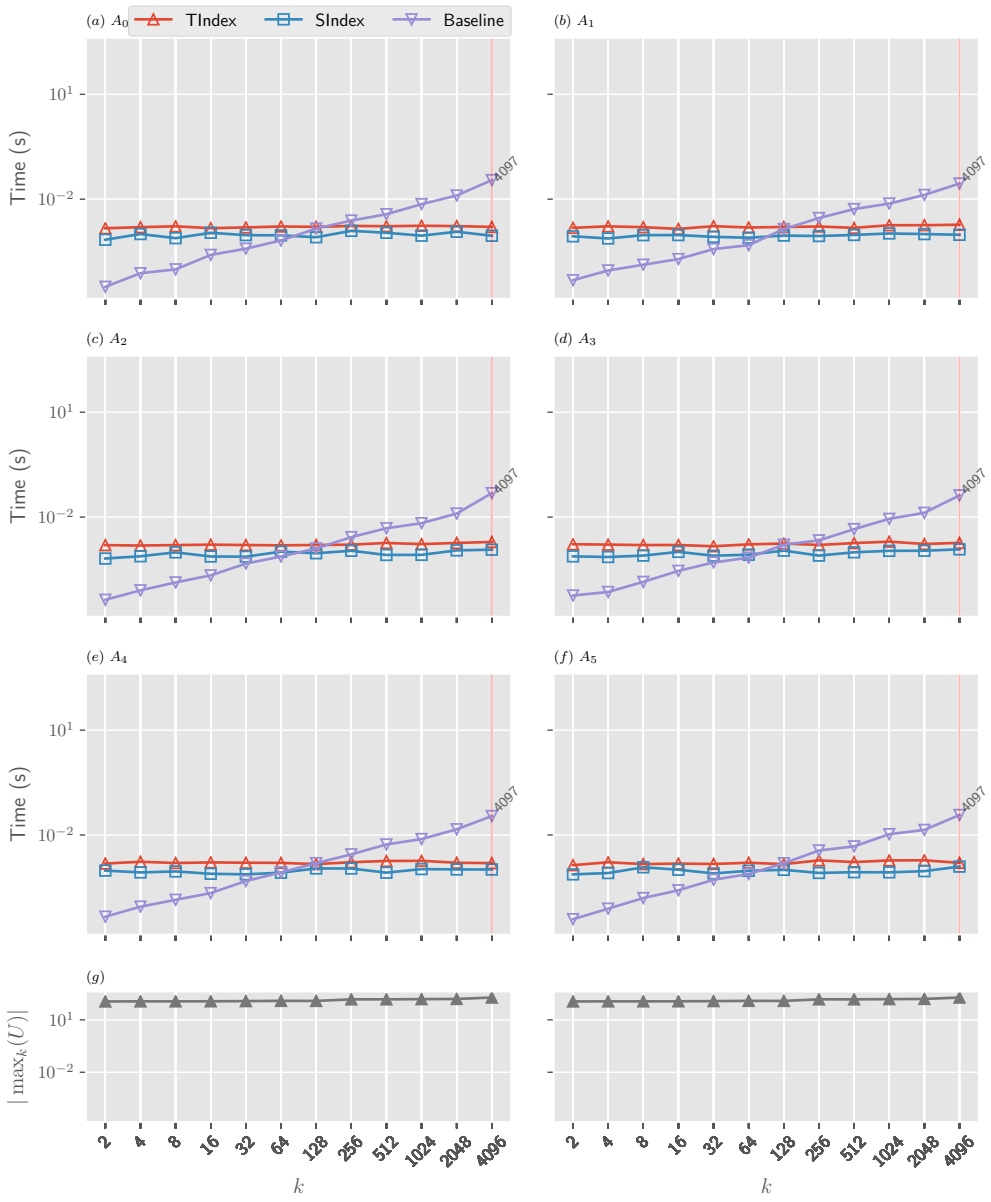


Figure 6.6: Query execution time of the primality tests w.r.t. the attribute set X_1 on the Flight dataset

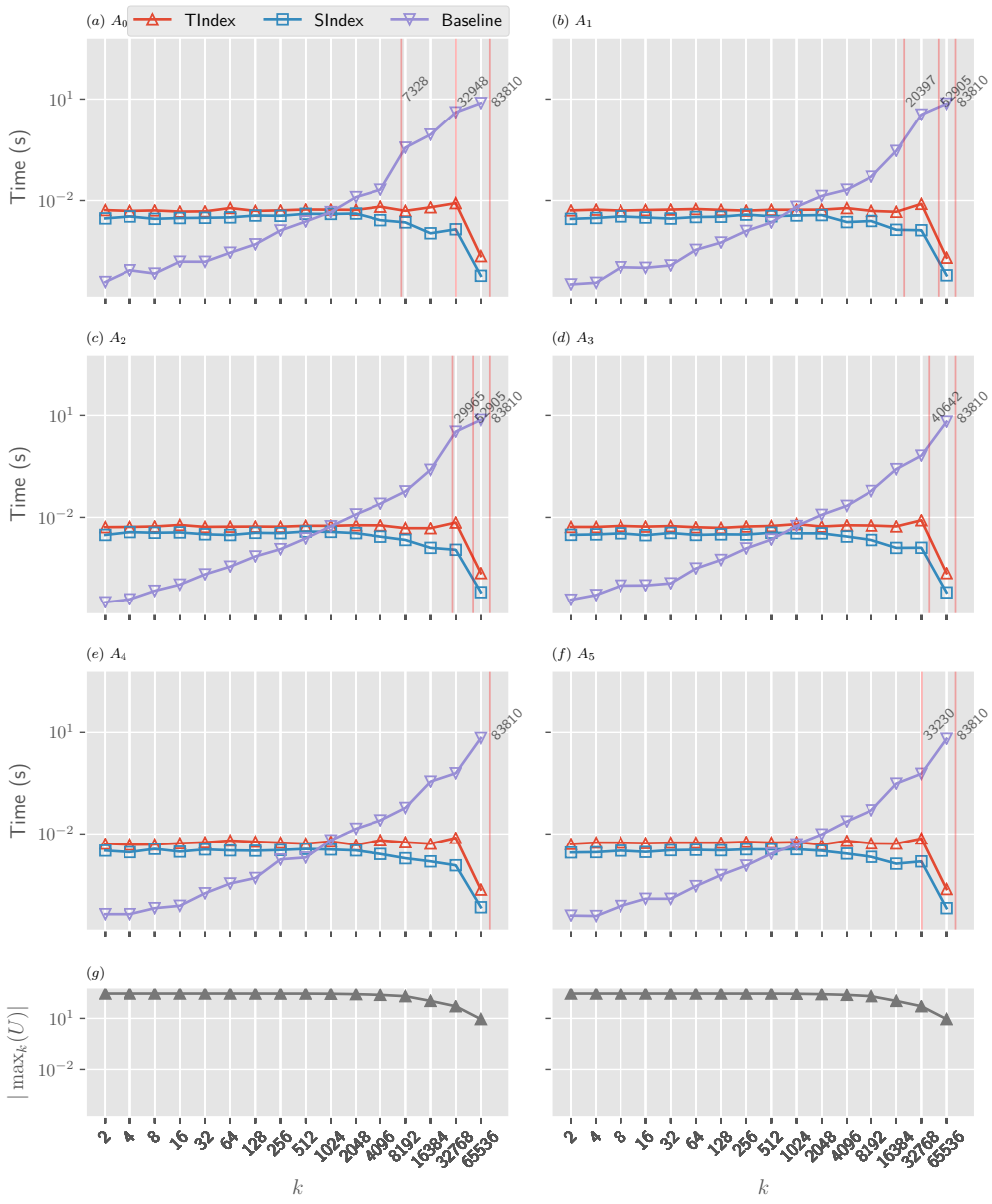


Figure 6.7: Query execution time of the primality tests w.r.t. the attribute set X_1 on the Github dataset

6.4.2 Stable Interval Query: Primality Test

For each dataset, we reuse the same attribute sets $X_1, X_2, \ldots, X_6$ for the point queries. For each attribute set X and each attribute $A \in X$, we ask what is the stable interval for the property that the attribute A belongs to some primary key of the relation schema $R[X]$ for a given k.

For each dataset, we report the query execution time of the `SIndex` method and the `TIndex` method versus k in a log-log scale on the attribute set X_1.[5] The full results are shown in the appendix.

Figure 6.8 shows the result on the Flight dataset. The query execution time for both the `SIndex` method and the `TIndex` method is stable with respect to k. The result is consistent with what we have seen for the point queries (Figure 6.6). The `SIndex` method performs better than the `TIndex` method.

Figure 6.9 shows the result on the Github dataset. Both the `SIndex` method and the `TIndex` method show similar trends with respect to k. Similar to the interval query for the BCNF test, the query execution time for a given k is related to which interval k falls into. For example, consider Figure 6.9a. We have three stable intervals $I_1 = [1, 7328]$, $I_2 = (7328, 32948]$ and $I_3 = (32948, 83810]$. The query execution time for $k \in I_2$ is larger than the one for $k \in I_1$, even though the query execution time of the point query for $k \in I_2$ is smaller than the one for $k \in I_1$ in most cases (Figure 6.7a). This is because the length of I_2 is much larger than the length of I_1, and the algorithms have to do a linear search for all the k in the corresponding interval.

[5]The baseline algorithm does not finish in 5 minutes, and thus we ignore it in the interval primality test.

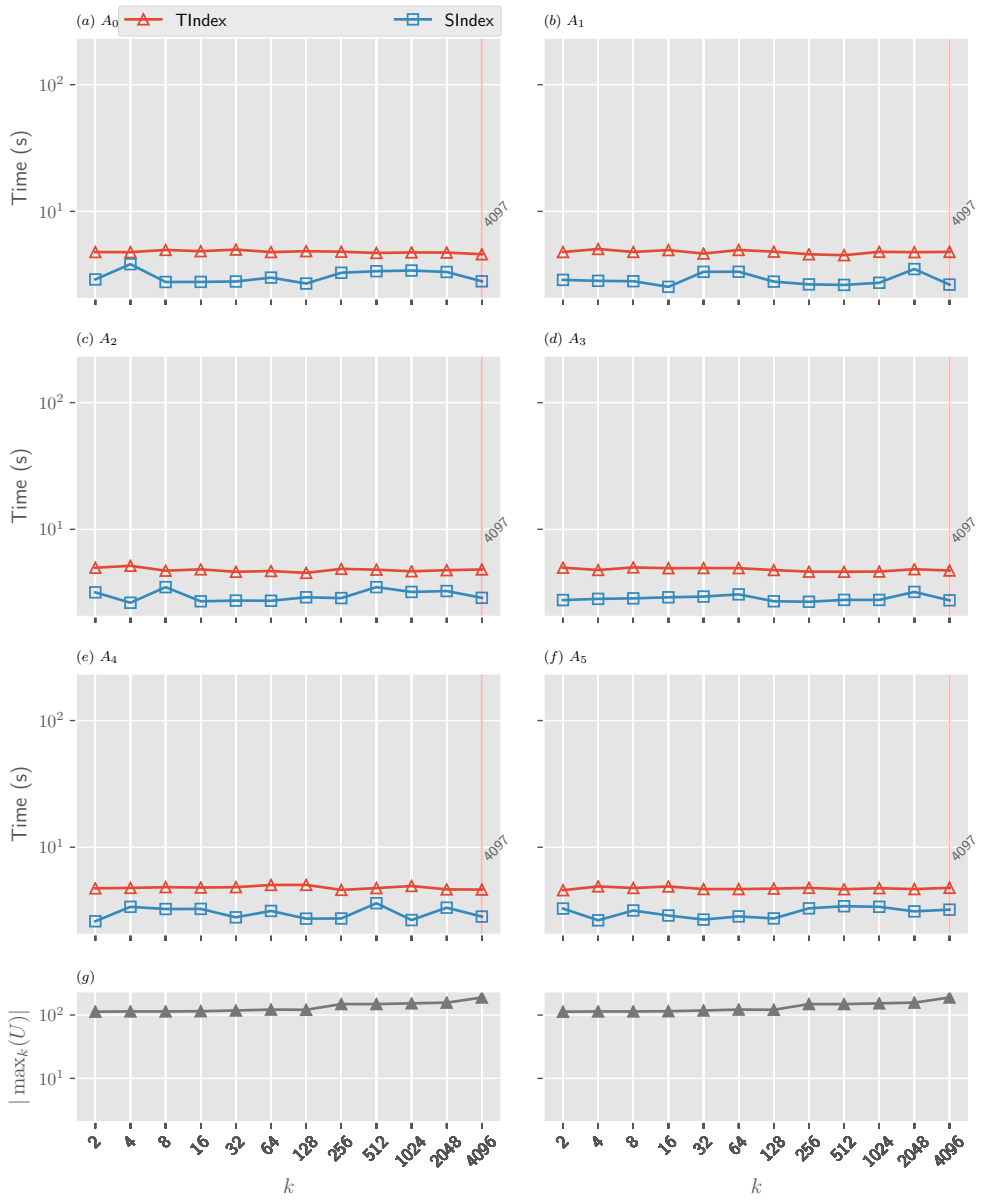


Figure 6.8: Query execution time of the stable interval queries for the primality tests w.r.t. the attribute set X_1 on the Flight dataset

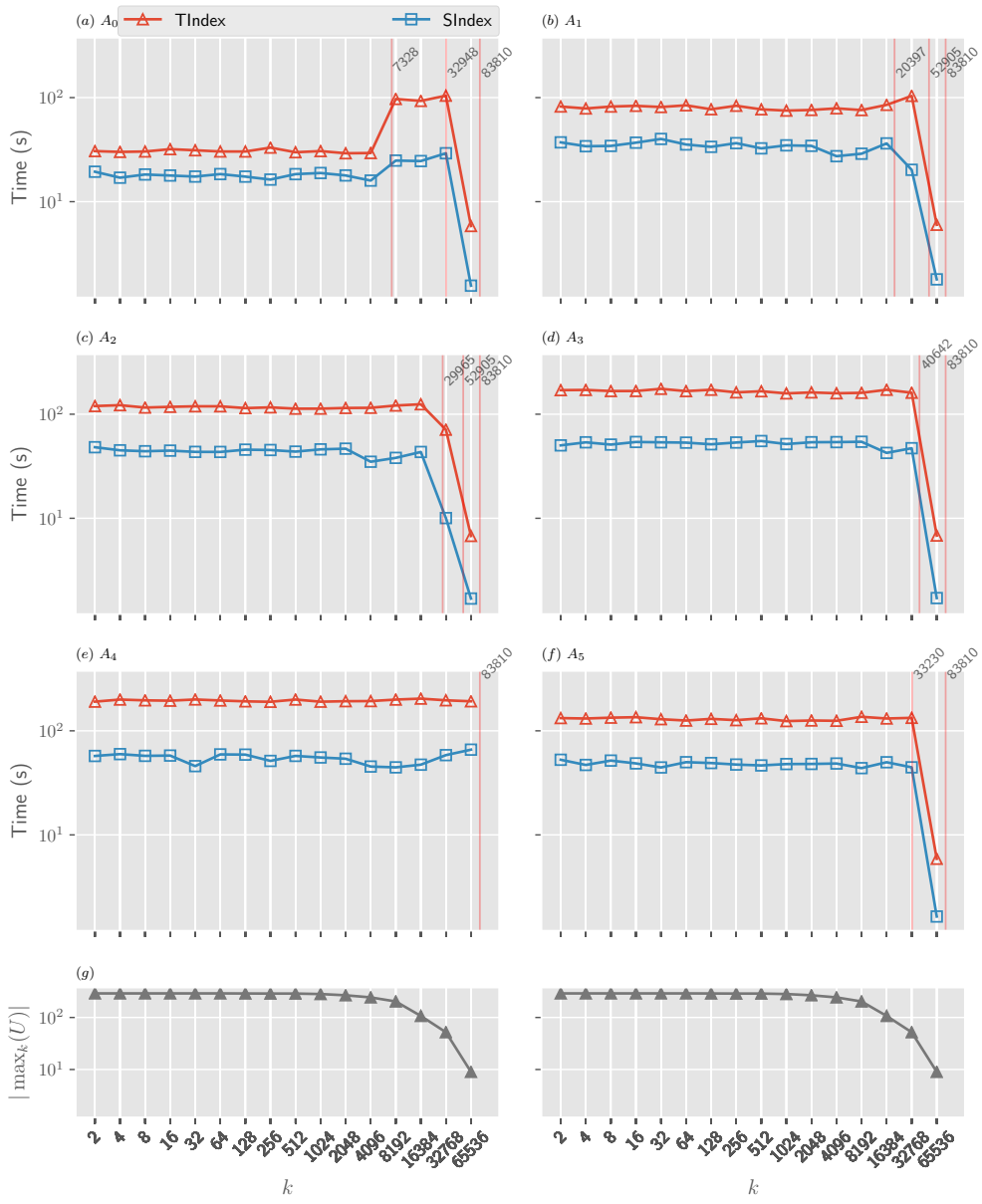


Figure 6.9: Query execution time of the stable interval queries for the primality tests w.r.t. the attribute set X_1 on the Github dataset

6.5 Impact of noisy and sparse data

While this thesis focuses on the performance aspect, we experimentally explore the impact of noise and sparse data and shed light on the quality aspect. Consider the relation schema

$$R(X) = (\texttt{instructor}, \texttt{department}, \texttt{faculty}, \texttt{course}, \texttt{term})$$

introduced in Chapter 1. The relationships among these attributes can be described by the following FDs, which serve as the ground truth.

$$f_1 : \texttt{instructor} \rightarrow \texttt{department}$$
$$f_2 : \texttt{department} \rightarrow \texttt{faculty}$$
$$f_3 : \texttt{course}, \texttt{department}, \texttt{term} \rightarrow \texttt{instructor}$$

We generate a synthetic dataset which exactly satisfies the above FDs. One possible database schema based on the above FDs is as follows, where the underline indicates that an attribute is part of a primary key.

($\underline{\texttt{instructor}}$, department)
($\underline{\texttt{department}}$, faculty)
($\underline{\texttt{course, term, instructor}}$)

6.5.1 Impact of sparse data

We consider two type of sparsity, row sparsity—the dataset has very few rows—and column sparsity—a column has many unknown values.

Row Sparsity

When we have very few rows, the data profiling component can produce false positive FDs.

One extreme case is a dataset with only one row. In such a case, an attribute can determine an arbitrary attribute. As a result, any subset of $R(X)$ conforms with BCNF. A schema generation algorithm based on BCNF will not further decompose the universal relation schema.

To further study the impact of sparse data, we remove one particular row from the synthetic dataset. The data profiling generates an additional FD f_4 besides the ground truth FDs (f_1, f_2 and f_3).

$$f_4 : \texttt{instructor}, \texttt{term} \rightarrow \texttt{course}$$

The reason is that the removed row reveals the fact that *an instructor may teach two courses in the same term*. The schema generation algorithm can still generate a relation schema conforming with BCNF such as (`instructor`, `term`, `course`). However, the algorithm will choose (`instructor`, `term`) as the primary key, which is wrong in semantics.

Column Sparsity

When a column is sparse, the data profiling component can produce both false positive FDs and false negative FDs.

In an extreme case, when the values for the attribute `department` are totally missing, the data profiling component derives the following FDs:

$$\texttt{instructor} \rightarrow \texttt{faculty}$$
$$x \rightarrow \texttt{department} \text{ where } x \in X$$

The schema generation algorithm can generate the same database schema as the one based on the ground truth. However, for the relation schema (`department`, `faculty`), it will choose the attribute `faculty` as the key, which is wrong in semantics.

The data-driven renormalization framework heavily relies on the available data. When we only have very few data, it may produce a very different set of FDs and schema design. We can resort to additional information such as the JSON structure in such a case. We leave the discussion of *integrating with additional information* to Chapter 7.

6.5.2 Impact of noisy data

When the data set is noisy, we can have false negative FDs. To study the impact of noisy data, for the `department` attribute, we introduce two values `CS` and `Computing Science` for the attribute `department`, which are semantically equivalent. The result of discovered FDs is:

$$f_1' : \texttt{instructor} \rightarrow \texttt{faculty}$$
$$f_2 : \texttt{department} \rightarrow \texttt{faculty}$$
$$f_3 : \texttt{course}, \texttt{department}, \texttt{term} \rightarrow \texttt{instructor}$$
$$f_4' : \texttt{instructor}, \texttt{term} \rightarrow \texttt{department}$$

The schema generation algorithm can produce a relation schema (`instructor`, `term`, `department`), which conforms with BCNF based on the detected FDs. However, it does not conform with BCNF under the ground truth (FDs f_1, f_2 and f_3).

To handle noisy dataset, we can leverage various data cleaning tools to repair the dataset and improve the quality of discovered FDs. We leave the discussion to Chapter 7.

6.6 Summary

We have shown our method outperforms the baseline in various tasks—the BCNF test, the interval BCNF test, the primality test and the interval primality test. The `TIndex` method outperforms the `SIndex` method in some dataset, but the `SIndex` method performs slightly better overall. We also empirically study the impact of sparse and noisy data on the generated database schema.

Chapter 7

Conclusion

In this thesis, we explore several schema design problems under the data-driven renormalization framework. We formally define two kinds of queries—the point query and the stable interval query—to help users making design decisions. We propose two index structures `SIndex` and `TIndex`, which can represent a list of FDs concisely—using the family of maximal sets, but also can process various queries efficiently—by leveraging various properties of maximal sets. We conduct experiments on two real datasets and show that our algorithms greatly outperform the baseline method when processing a large set of FDs.

As for future work, we consider the following directions.

- *Support more design decision problems.* There exist many other design decision problems (e.g., 3NF, 4NF and so on). We only support two design decision problems in this thesis—the BCNF test and the primality test. It is possible to support a wider range of design decision problems under the same framework.

- *Extend to the SQL data model.* In practice, SQL data model is different from the idealized relational data model. Recent work [18] extends the theory of the relational model—by redefining FDs and BCNF—to support the SQL data model. It will be interesting to investigate how to incorporate our algorithms under the new theoretical framework.

- *Integrate with additional information.* This thesis uses a data-driven approach to discover the relationships among attributes. However, the JSON document itself often have rich structures, which are valuable especially when the dataset is sparse. We can integrate the additional information as constraints (e.g., a set of attributes must group in one relation schema) on the candidate database schemas. Besides, the programming language communities take a very different approach [27, 28] in the area of program synthesis. It can be a promising direction to integrate with these methods to achieve better results.

- *Quality issue and interactive schema design.* Schema design is usually an iterative and interactive process. The thesis focuses on only one step under the data-driven schema

renormalization framework. The quality issue is an important aspect in a real-world system. Some works [7, 10] introduce various heuristic techniques to improve the quality of the discovered FDs and thus improve the generated database schema. Also, a data cleaning process may further improve the quality of discovered FDs. It will be more helpful to incorporate our framework into the whole process, identify the user requirements, and optimize the whole workflow.

Bibliography

[1] *Couchbase*. URL: https://www.couchbase.com/ (cit. on p. 1).

[2] *MongoDB*. URL: https://www.mongodb.com/ (cit. on p. 1).

[3] *APIs as a product*. URL: https://www.thoughtworks.com/radar/techniques/apis-as-a-product (cit. on p. 1).

[4] *Programmable Web: API Directory*. URL: http://bit.ly/PWeb201705 (cit. on p. 1).

[5] *Tableau*. URL: https://www.tableau.com/ (cit. on p. 1).

[6] *PowerBI*. URL: https://powerbi.microsoft.com (cit. on p. 1).

[7] Michael DiScala and Daniel J Abadi. "Automatic generation of normalized relational schemas from nested key-value data". In: *Proceedings of the 2016 International Conference on Management of Data*. ACM. 2016, pp. 295–310 (cit. on pp. 1, 3, 8, 39, 56).

[8] Thorsten Papenbrock and Felix Naumann. "Data-driven Schema Normalization". In: *Proceedings of the 20th International Conference on Extending Database Technology*. 2017 (cit. on pp. 1, 8).

[9] Jyrki Kivinen and Heikki Mannila. "Approximate inference of functional dependencies from relations". In: *Theoretical Computer Science* 149.1 (1995), pp. 129–149 (cit. on p. 3).

[10] Daisy Zhe Wang et al. "Functional Dependency Generation and Applications in Pay-As-You-Go Data Integration Systems". In: *WebDB*. 2009 (cit. on pp. 3, 56).

[11] Fei Chiang and Renée J Miller. "Discovering data quality rules". In: *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 1166–1177 (cit. on p. 3).

[12] Georg Gottlob, Reinhard Pichler, and Fang Wei. "Tractable database design through bounded treewidth." In: *PODS* (2006), p. 124 (cit. on pp. 3, 7).

[13] C Beeri and Philip A Bernstein. "Computational problems related to the design of normal form relational schemas". In: *TODS* (1979) (cit. on pp. 3, 7).

[14] E. F. Codd. "Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks". In: *IBM Research Report, San Jose, California* RJ599 (1969) (cit. on p. 7).

[15] H Mannila and K Raihä. “Design by example: An application of Armstrong relations”. In: *JCSS* (1986) (cit. on pp. 7, 18, 24).

[16] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983 (cit. on p. 7).

[17] Heikki Mannila and Kari-Jouko Räihä. *Design of Relational Databases*. Addison-Wesley, 1992 (cit. on pp. 7, 9–11, 23, 31, 32, 35).

[18] Henning Köhler and Sebastian Link. “SQL schema design: Foundations, normal forms, and normalization”. In: *Proceedings of the 2016 International Conference on Management of Data*. ACM. 2016, pp. 267–279 (cit. on pp. 7, 55).

[19] Wo L Chang. “NIST Big Data Interoperability Framework: Volume 1, Definitions”. In: *Special Publication (NIST SP)-1500-1* (2015) (cit. on p. 8).

[20] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. “Profiling relational data: a survey”. In: *The VLDB Journal* 24.4 (Aug. 2015), pp. 557–581 (cit. on p. 8).

[21] Thorsten Papenbrock et al. “Functional dependency discovery: An experimental evaluation of seven algorithms”. In: *Proceedings of the VLDB Endowment* 8.10 (2015), pp. 1082–1093 (cit. on p. 8).

[22] Tobias Bleifuß et al. “Approximate Discovery of Functional Dependencies for Large Datasets”. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM. 2016, pp. 1803–1812 (cit. on pp. 8, 39).

[23] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. 1st. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997 (cit. on p. 13).

[24] Claudio Carpineto and Giovanni Romano. *Concept Data Analysis: Theory and Applications*. John Wiley & Sons, 2004 (cit. on p. 13).

[25] D. J. Bernstein. *Crit-bit trees*. 2004. URL: `https://cr.yp.to/critbit.html` (cit. on pp. 20, 27).

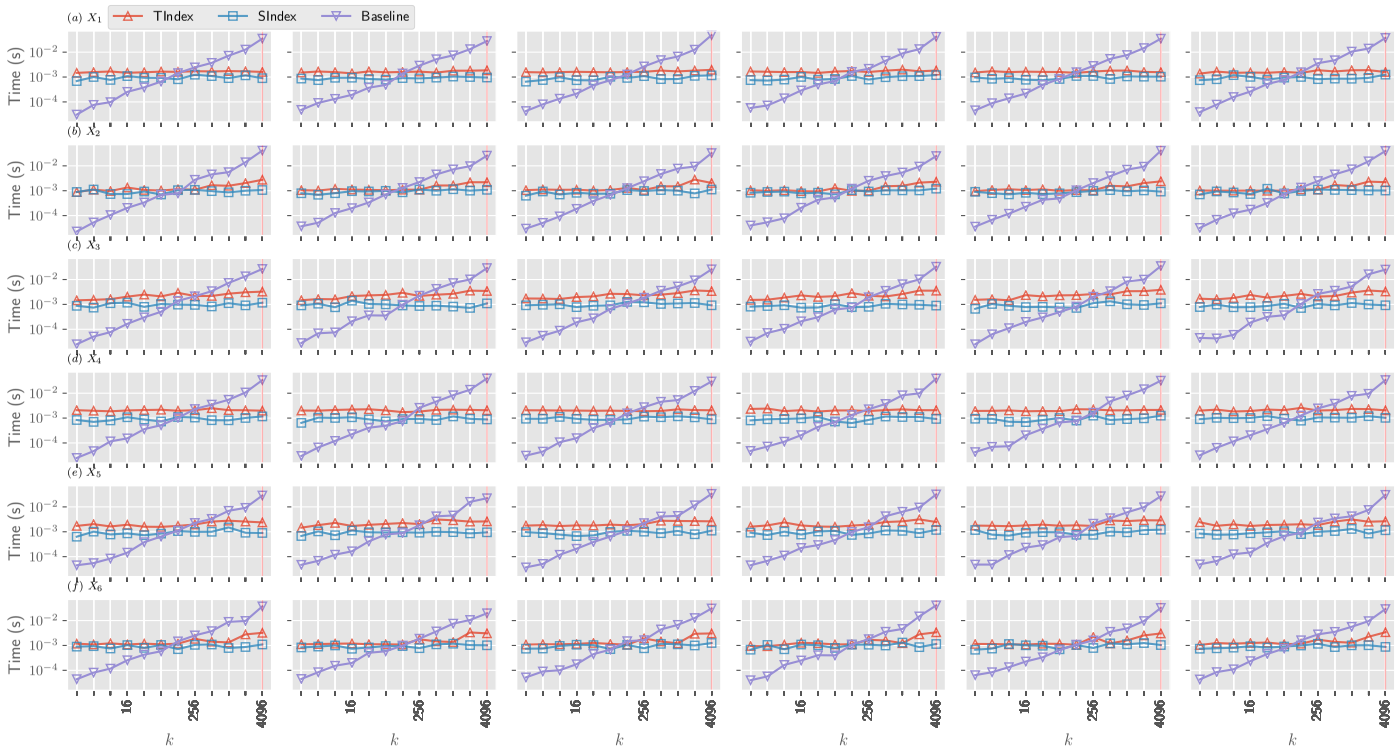
[26] *Code Tools: jmh*. URL: `http://openjdk.java.net/projects/code-tools/jmh/` (cit. on p. 39).

[27] Daniel W Barowy et al. “FlashRelate: extracting relational data from semi-structured spreadsheets using examples”. In: *ACM SIGPLAN Notices*. Vol. 50. 6. ACM. 2015, pp. 218–228 (cit. on p. 55).

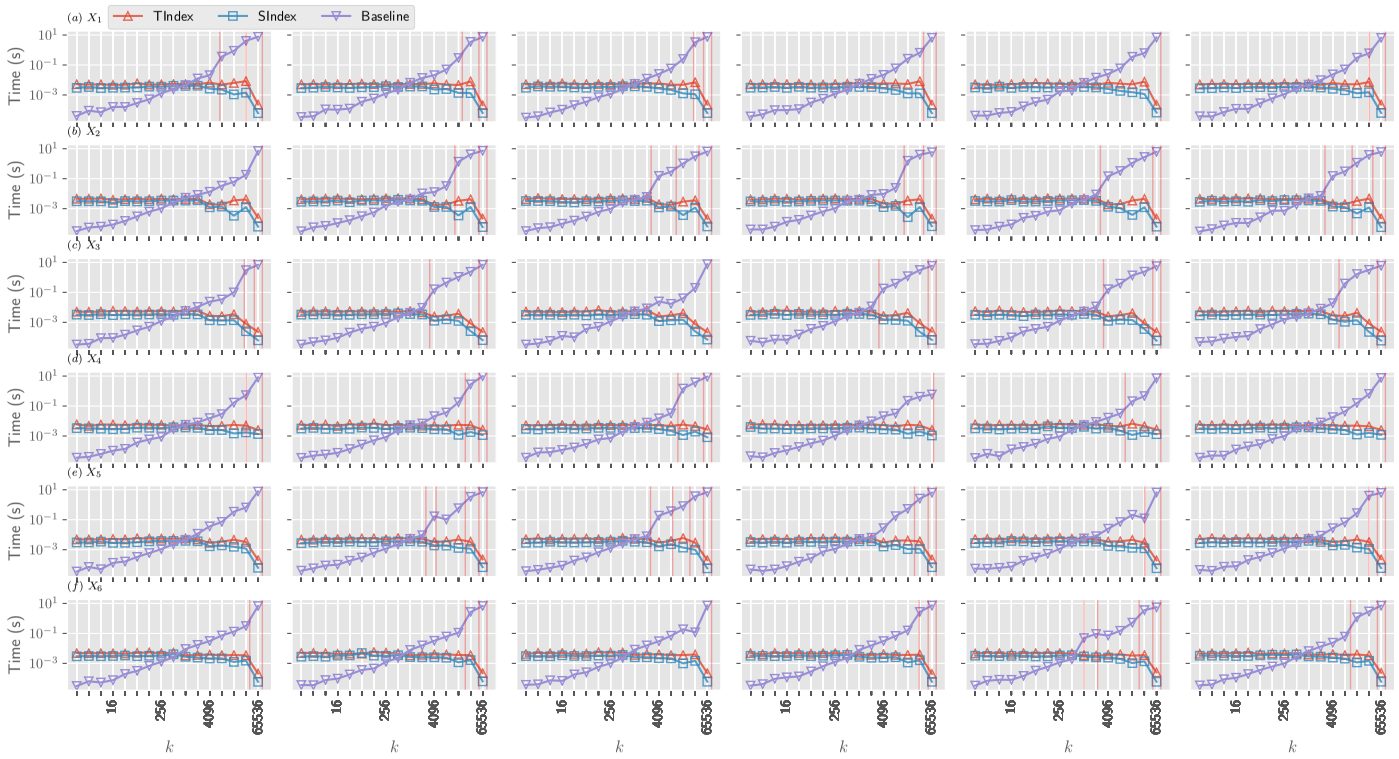
[28] Vu Le and Sumit Gulwani. “Flashextract: A framework for data extraction by examples”. In: *ACM SIGPLAN Notices*. Vol. 49. 6. ACM. 2014, pp. 542–553 (cit. on p. 55).

Appendix A

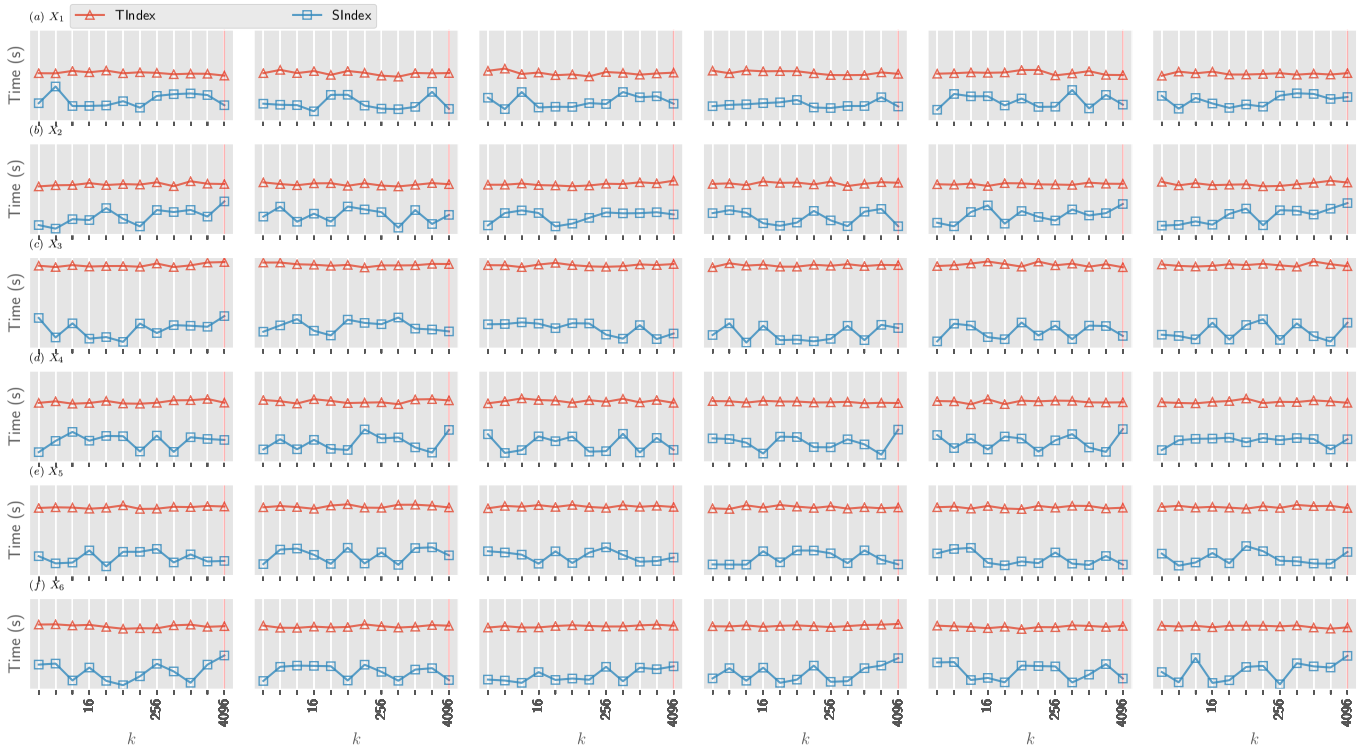
Full Results of Primality Tests



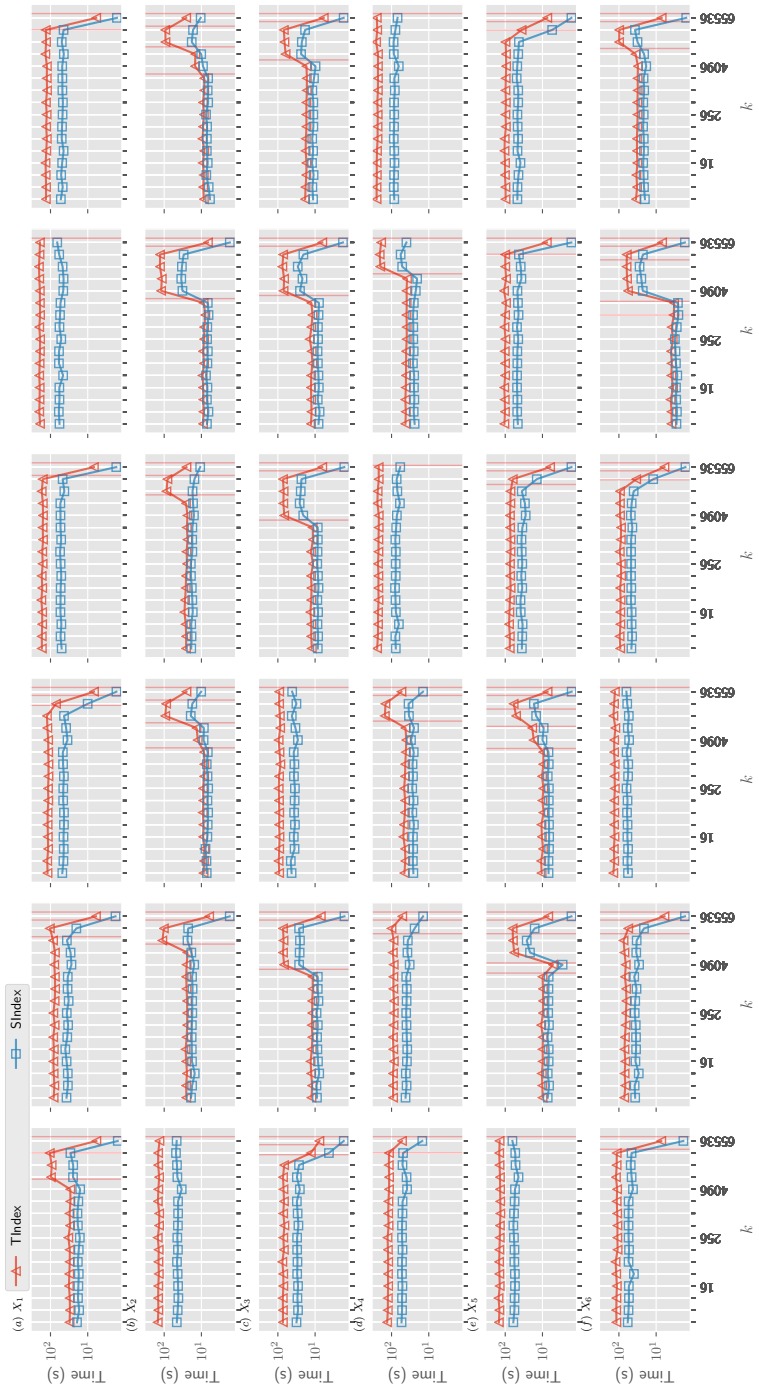
Query execution time of the primality tests on the Flight dataset



Query execution time of the primality Tests on the Github dataset



Query execution time of the stable interval queries for the primality tests on the Flight dataset



Query execution time of the stable interval queries for the primality test on the Github dataset