Tractable Queries with Inequalities on Probabilistic Databases

Rasmus Wissmann
University College
Supervisor: Dr. Dan Olteanu

Dissertation submitted in partial fulfilment of the degree of Master of Science in Computer Science
Computing Laboratory, University of Oxford

September 2009

Abstract

The problem of efficient evaluation of queries on probabilistic databases has received a great attention in recent years. In this thesis, we introduce novel classes of queries with inequalities ($<, \leq$) that admit tractable evaluation, in the context where query evaluation is #P-hard in general. We also introduce a new kind of hard query that is not captured by previous characterizations. As an application outside database theory, we show that particular tractable instances of our problem capture previously-open problems of counting vertex covers in bipartite chain and convex graphs, for which we can now easily derive efficient algorithms. Our approach to both query evaluation and counting vertex covers is based on a novel syntactical characterization of classes of k-DNFs that capture the lineage of queries from our tractable classes and that can be compiled in at most quadratic time into binary decision diagrams.

Acknowledgements

During the last year I have benefitted from the kind help and support of many people and I want to use this possibility to thank them.

First and foremost, I want to thank Dan Olteanu for being an incredibly inspiring and encouraging supervisor throughout the whole process of the creation of this dissertation. He has introduced me to interesting research in the field of probabilistic database theory and spent many hours discussing and reflecting about new ideas and techniques. This dissertation would not have been possible without his guidance and continuous feedback.

Ani Calinescu has supervised my coursework during the first two terms and provided helpful answers to many questions, for which I want to express my gratitude. I also want to thank Prof. George Gottlob and Prof. Michael Bennedikt for their enlightening lectures about database theory and relational databases, which have provided me with much of the theoretical background for this thesis.

University College and the Oxford University Computing Laboratory have been institutional in making the last year in Oxford a great experience. This includes the academic life as well as the countless social activities organized by various student bodies, in particular the University College WCR.

Finally, I want to thank my family and friends for their constant support and encouragement.

This work was supported by a joint Hölderlin-scholarship from the Studienstiftung des Deutschen Volkes and Siemens Management Consulting.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Problem Statement

Recent years have seen increased interest in the general theory and applications of probabilistic databases [7, 1, 3, 23, 12, 25, 26], i.e. databases which contain uncertain data that is only true with some probability. The standard model for this case is the *tuple-independent probabilistic database*, in which independent Boolean random variables are added to each tuple of a relational database to represent uncertainty [7]. Practicable applications of these databases include scientific databases, data integration, data cleaning and handling of sensor data [7, 1, 3, 23, 12, 25, 26]. A number of projects at major universities and research institutions currently aims to develop database management systems for probabilistic databases [4, 31, 12, 30]. Oxford is involved via the database management system MayBMS [2, 11] (joint project with Cornell University) and the query engine SPROUT (Scalable Query PROcessing in Probabilistic Databases) [17].

Operations on tuple-independent probabilistic databases are in principle easy to define by modifying the semantics of the original operators. For example, a Join operator (on two different, independent relations) would execute a join of the underlying data and then AND the corresponding Boolean variables. Correspondingly, duplicate elimination would keep one data entry and replace its variable by a disjunctive combination of the variables of all equal tuples.

While the operational semantics are relatively easy to define, a basic difficulty is the calculation of the final probabilities of the answer tuples. Due to joins and projections, formulas with complex combinations of Boolean variables can arise. Evaluating these formulas is in general #P-complete [7, 6].

Common approaches used to tackle this problem either resort to probabilistic algorithms [22, 16, 11] or try to identify tractable classes of queries and implement efficient algorithms for answering them [7, 8, 15, 16]. A tractable query in this sense is a query which can be evaluated in polynomial time w.r.t data complexity (the query itself being fixed).

Despite the question of which queries are tractable being a very central and foundational one, it has only been completely solved for conjunctive queries involving only equalities [7, 8, 6]. The tractability question for queries with inequalities remaines largely unknown, despite first promising results and algorithms [15, 16].

In this project, we aim to extend current results, describe new algorithms and provide further insights into the tractability map of queries on probabilistic databases. Our central problem is:

> Given a conjunctive query Q with inequalities ($<$) over a tuple-independent probabilistic database, is Q tractable?

1.2 State of the Art

Dalvi and Suciu completely answered the tractability problem for Boolean conjunctive queries with equalities by showing that a query is either PTIME or #P-complete, depending on whether it is hierarchical or not. [7, 6, 8] They also gave an optimization algorithm that computes the tractable queries as well as approximate methods for the intractable ones.

Olteanu and Huang extended these results to larger classes of queries, in particular those with inequalities ($<$) [15, 16]. They proved tractability for Boolean conjunctive queries with inequalities and without self-joins for which at most one query variable from each relation occurs in the inequalities. They also gave extensions towards certain non-Boolean and mixed equality-inequality queries and generalized the concept of hierarchical queries to include queries with inequalities [16].

1.3 Outline and Contributions

This thesis is structured as follows:

- Chapter 2 gives a brief overview over query answering on tuple-independent probabilistic databases, disjunctive normal forms (DNFs) and binary decision diagrams (BDDs). It introduces basic definitions and notation.

- Chapter 3 presents polynomial-time algorithms which compile two newly defined classes of DNFs - linear-inclusion k-DNFs and cofactor-seperable 2-DNFs - into equivalent BDDs.

- Chapter 4 shows how BDD compilation for DNFs and query evaluation on tuple-independent databases are connected. It introduces the general framework and an algorithm for probability computation on BDDs.

- Chapter 5 presents new tractability results for queries with inequalities. The results are obtained by relating the query lineages to the DNF classes and algorithms from Chapter 3. It also discusses further extensions of these techniques.

- Chapter 6 shows why query evaluation on tuple-independent probabilistic databases is in general hard and describes known hard queries. It also introduces a new kind of hard query and show possible directions towards a more general picture.

- Chapter 7 contains applications of the BDD compilation algorithms to counting problems on graphs. It establishes new tractability results for counting vertex covers on bipartite chain and convex bipartite graphs by relating them to queries from Chapter 4.

- Chapter 8 summarizes the thesis work and contributions. It also describe open questions and gives possible directions for future work.

The main contributions of the thesis are:

1. Previous approaches for confidence computation of queries with inequalities on tuple-independent probabilistic databases were based on OBDD construction [15, 16]. We extend this method towards general BDDs and introduce a new technique for BDD compilation of query lineages which creates polynomial BDDs by keeping track of a polynomial number of parallel paths.

2. We define two classes of DNFs - linear-inclusion k-DNFs and cofactor-seperable 2-DNFs - and show how they can be compiled in polynomial time into BDDs.

3. We extend known tractability results for conjunctive queries with inequalities ($<$). So far, only tree-shaped queries with at most one query variable per relation participating in inter-subgoal inequalities were tractable [15, 16]. We show that the considerably larger class of acyclic queries with at most one variable per relation participating in out-going inequalities is tractable, which subsumes the previous results.[1]

4. Based on the results for cofactor-separable 2-DNFs, we show tractability for queries beyond acyclic max-one queries and introduce ideas towards tractability of even more general queries.

[1] An inequality $R.X < S.Y$ is out-going from relation R and in-going to relation S.

5. We introduce a new kind of hard query which subsumes previous hard queries and creates structurally more complicated lineages. Combined with our tractability results this reveals more insights about the tractability frontier for conjunctive queries with inequalities on tuple-independent probabilistic databases.

6. We apply the algorithms for BDD construction to counting problems on graphs. This allows us to show that counting vertex covers and counting independent sets on bipartite chain and convex bipartite graphs is tractable. To the best of our knowledge, this has not been show before.

Parts of this thesis are in preparation for submission to 13th International Conference on Database Theory (ICDT 2010) under the title "On Efficiently Evaluating Inequality Queries on Probabilistic Databases and Counting Vertex Covers".

Chapter 2

Preliminaries

2.1 Tuple-Independent Probabilistic Databases

Let $\mathbf{X}$ be a finite set of independent Boolean random variables and P : $\mathbf{X} \to [0, 1]$ a function s.t.h $Pr(\mathbf{x})$ is the probability of of $x \in \mathbf{X}$ being true. A (finite) *event* or *formula* e over $\mathbf{X}$ is a (finite) combination of random variables from $\mathbf{X}$ using the Boolean operators $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT) and brackets. For example, $e = x_1 \wedge x_2$ represents the conjunction of random variables $x_1, x_2 \in \mathbf{X}$. We will also use $\cdot$, $+$ and $\overline{x}$ to represent $\wedge$, $\vee$, and $\neg x$ resp.

A *probabilistic relation* R over a (data) schema $\tilde{R}$ and variable set $\mathbf{X}$ is a relation over $(\tilde{R}, \mathbf{E})$ where the column $\mathbf{E}$ contains events over $\mathbf{X}$. R is called *tuple-independent* if all events are pairwise independent. In this case, each event can be considered a different random variable. This als means that the functional dependency $\tilde{R} \to \mathbf{E}$ holds. The set of random variables of R is denoted by $Vars(R) \subseteq \mathbf{X}$.

A (tuple-independent) *probabilistic database* D is a set of (tuple-independent) probabilistic relations. A tuple is *certain* if its associated random variable is true with unit probability. A relation C is certain if all its random variables are certain. This subsumes the case of standard (non-probabilistic) relations.

Example 2.1. *Consider the probabilistic database in Figure 2.1. The relation R is tuple-independent, because all events in column $R.\mathbf{E}$ are independent. In contrast, relation S is not tuple-independent, because y_1 and $y_1 \wedge y_2$ are not independent (unless one of them is always true or false). The sets of relation variables are $Vars(R) = \{x_1, x_2, x_3\}$ and $Vars(S) = \{y_1, y_2\}$.*

The interpretation of such a probabilistic database follows the *possible world semantics.* A *total truth assignment* for a database D is a truth assignment of all its random variables, i.e. a function $\sigma : Vars(D) \to \{0, 1\}$. The probability of a total truth assignment is given by the product

R	A	B	E
	1	1	x_1
	1	2	x_2
	2	2	x_3

S	C	E
	1	y_1
	3	$y_1 \wedge y_2$

Figure 2.1: Example probabilistic database D

R	A	B
	1	1
	2	2

S	C
	1

Figure 2.2: Possible database instance D_σ

of the probabilities of the truth values for the single random variables. In mathematical terms,

$$Pr(\sigma, D) = \prod_{x \in Vars(D)} Pr(x = \sigma(x)).$$

Each total truth assignment σ defines a *possible world*. The probability of this possible world is thus the same as the probability of the total truth assignment. The corresponding database instance D_σ is derived from D by removing all tuples with random variables/events which evaluate to false.

It is possible for several possible worlds to contain the same database instance D'. Thus, the probability of a distinct database instance is the sum of the probabilities of all possible worlds which contain this instance:

$$Pr(D') = \sum_{\sigma : D_\sigma = D'} Pr(\sigma).$$

For tuple-independent databases, two possible worlds can only have equal database instances if the probabilistic database contains duplicate tuples. It is also worth pointing out that a probabilistic database D encodes an exponentially large set of database instances, because there are in general $2^{|Vars(D)|}$ different total truth assignments.

Example 2.2. *Continuing the example from Figure 2.1, let* $Pr(x_1) = Pr(y_1) = 1/2$*,* $Pr(x_2) = Pr(x_3) = 1/4$ *and* $Pr(y_2) = 1/8$ *and fix an assignment* σ *with* $\sigma(x_1) = \sigma(x_3) = \sigma(y_1) = 1, \sigma(x_2) = \sigma(y_2) = 0$*. The corresponding database instance* D_σ *is* $R = \{(1,1),(2,2)\}$*,* $S = \{(1)\}$*. The probability of this instance is* $Pr(D_\sigma) = Pr(\sigma) = Pr(x_1) \cdot (1 - Pr(x_2)) \cdot Pr(x_3) \cdot Pr(y_1) \cdot (1 - Pr(y_2)) = 1/2 \cdot 3/4 \cdot 1/4 \cdot 1/2 \cdot 7/8 = 21/512$*, because* σ *is the only variable assignment that leads to this database instance.* D_σ *is shown in Figure 2.2.*

2.2 Conjunctive Queries with Inequalities

Using datalog notation, conjunctive queries with inequalities ('<', '>') can be written as

$$Q(\overline{X}) : -R_1(\overline{X}_1), ..., R_n(\overline{X}_k), \Lambda$$

where $\overline{X}$,$\overline{X}_1, ..., \overline{X}_k$ are lists of distinct query variables and Λ is a conjunction of inequalities between query variables. Not that the query variables are used to encode connections between different columns and are different from the random variables in $\mathbf{X}$.

If not specified otherwise, in the following by "query" we mean a Boolean conjunctive query over tuple-independent probabilistic databases with inequalities, but without equalities ('=') and inequalities of the from '≤','≥','≠'. We will also restric ourselves to Boolean queries, i.e. queries with $\overline{X} = \emptyset$. Non-boolean queries can be treated using their Boolean reducts:

Definition 2.3 (Boolean reduct)**.** *[16] The* Boolean reduct *of a query*

$$Q(\overline{X}) : -R_1(\overline{X}_1), ..., R_n(\overline{X}_k), \Lambda$$

is defined as

$$Q' : -R_1(\overline{X}_1 - \overline{X}), ..., R_n(\overline{X}_k - \overline{X}), \Lambda'$$

where Λ' is Λ without the inequalities involving query variables from $\overline{X}$.

The Boolean reduct reduces a non-Boolean query to a Boolean query for each fixed value of the head variables $\overline{X}$. This leads to:

Proposition 2.4. *[16][Tractability of Non-Boolean Queries]*
Non-Boolean queries are tractable if their Boolean reducts are tractable.

To visualize queries and characterize tractable subclasses, we will use the following graph representation:

Definition 2.5 (Query Graphs)**.** *The* query graph *associated with a query $Q() : -R_1(\overline{X}_1), ..., R_n(\overline{X}_k), \Lambda$ is the graph $G_Q = (V_Q, E_Q, R_Q)$, where V_Q is the set of query variables involved in inequalities with query variables from other subgoals, $(X_i, X_j) \in E_Q$ if and only if Λ contains the inequality '$X_i < X_j$' and $R_Q = \{\overline{X}_1 \cap V_Q, ..., \overline{X}_k \cap V_Q\}$.*

The sets in R_Q describe which query variables occur in the same relation. Graphically, they can be visualized by *relation nodes*, which are clusters of query variable nodes.

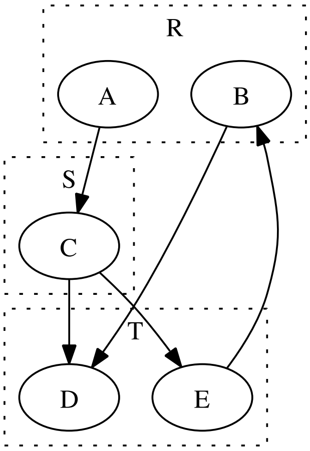


Figure 2.3: Example query graph

Example 2.6. *Consider the query*

$$Q() :- R(A,B), S(C), T(D,E), A < B, A < C, B < D, C < D, C < E, E < B.$$

Its query graph is shown in Figure 2.3. The sets of nodes, edges and relation nodes are

$$\begin{aligned} V_Q &= \{A,B,C,D,E\}, \\ E_Q &= \{(A,C),(B,D),(C,D),(C,E),(E,B)\}, \\ R_Q &= \{\{A,B\},\{C\},\{D,E\}\}. \end{aligned}$$

The inequality $A < B$ is not represented in the graph, since it only involves query variables from the same relation. Such inequalities can always be trivially dealt with using a pre-processing with selections. In this example, we only need to replace $R(A,B)$ with $R'(A,B) \equiv \sigma_{A<B}(R(A,B))$. (Note that $\sigma_{A<B}$ here represents a standard relational selection and not a truth assignment.)

Remark 2.7. *Inequalities of the form 'X > Y' can always be rewritten as 'Y < X'. In the following we will thus restrict the inequalities to the latter type.*

The structure of connections between relations plays an important role in the context of query answering tuple-independent probabilistic databases. One particularly important definition is:

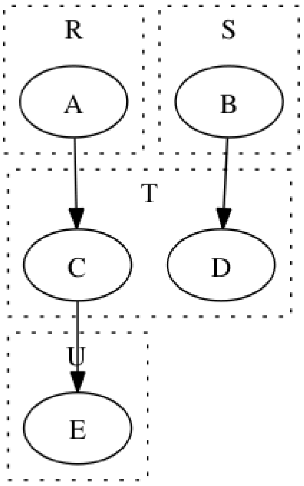


Figure 2.4: Example acyclic max-one query

Definition 2.8 (Max-one property)**.** *A query with subgoals* $R_1, ..., R_k$ *and corresponding sets* $\overline{X}_1$*, ...,*$\overline{X}_k$ *of query variables has the* max-one *property if at most one query variable from each subgoal participates in out-going inequalities with query variables from other sets.*

Remark 2.9. *The max-one property is an immediate generalization of the max-one property defined by Olteanu and Huang [16]. They demanded that only one query variable from each subgoal participates in inequalities with other subgoals. Our condition is a weaker, globally directed variant.*

We call queries which have the max-one property *max-one queries.*

Example 2.10. *The query shown in Figure 2.4 has the max-one property.*

Finally, we need a definition of acyclicity of a query with a broken query graph:

Definition 2.11. *A query* Q *is called acyclic, if its broken query graph is acyclic and contains no cycles of of length* ≥ 1 *broken nodes.*

Example 2.12. *The query in Figure 2.3 is cyclic. For example,* $R.B < T.D$ *and* $T.E < R.B$*. We could easily make it acyclic by removing the inequality between* B *and* E*. In contrast, the query in Figure 2.4 is acyclic.*

2.3 Query Semantics

Query evaluation on probabilistic databases is governed by the possible world semantics: $Q(D)$ is a probability distribution over possible answer

$$\begin{array}{lcl} \sigma'_\lambda\left((\tilde{r}, e_r)\right) & = & (\sigma_\lambda(\tilde{r}), e_r) \\ \pi'_A\left((\tilde{r}, e_r)\right) & = & (\pi_A(\tilde{r}), e_r) \\ (\tilde{r}, e_r) \times' (\tilde{s}, e_s) & = & (\tilde{r}, \tilde{s}, e_r \wedge e_s) \\ \mathrm{DupEl}(\{(r_1, e_1), ..., (r_n, e_n)\}) & = & \bigcup\limits_{\substack{1 \leq i \leq n, \\ r_i \neq r_j \forall 1 \leq j < i}} \left\{ \left(r_i, \bigvee_{1 \leq k \leq n : r_k = r_i} e_k \right) \right\} \end{array}$$

Figure 2.5: Modified relational operators

tuples, where the probability of a tuple t is the sum of probabilities of all total truth assignments σ such that Q applied to D_σ returns a tuple set containing t. In mathematical notation,

$$Pr(t) = \sum_{\substack{\sigma : Vars(D) \to \{0,1\} \\ s.th.\, t \in Q(D_\sigma)}} Pr(\sigma). \tag{2.1}$$

Note that the query answer is not defined as the probability distribution P over all possible result sets of tuples, because the number of distinct result sets is in general exponential in the number of distinct answer tuples [7]. For Boolean queries, $t \in \{true, false\}$ and thus the query result is a single real number p_Q which gives the probability that the query returns true.

Equation 2.1 means that semantically the query is evaluated in every possible world and the result is weighted by the associated probability. Since the number of possible worlds is exponential in the number of random variables, the naive algorithm which computes the query result on each possible world has exponential runtime. Consequently, more sophisticated approaches are necessary and have been successfully investigated. [7, 15] They are based on an equivalent description of the possible world semantics on the database D using modified query evaluation operators. These follow standard semantics on the data columns and additionally combine the event columns according to the specific relational operation.

Figure 2.5 shows the modified versions of the relational operators relevant to the kinds of queries investigated in this thesis. For example, when two tuples with events e_1 and e_2 are combined during duplicate elimination ('DupEL'), the resulting event is $e_1 \vee e_2$, since the result tuple is in the output relation when at least one of the events is true. Similar reasoning leads to the formulas of the other operators [7]. (Note, that the projection operator in Figure 2.5 does not use set semantics.)

The query answer is now a probabilistic relation with possible very complex events associated with its tuples. For Boolean queries, the answer relation has schema (**E**) - i.e. no data columns - and thus contains only one tuple. We call the event $\phi_{Q(D)}$ associated with this tuple *query lineage.* The probability $Pr(\phi_{Q(D)})$ is the probability of $Q(D)$ being true, i.e.

$$Pr(\phi_{Q(D)}) = Pr(Q(D)). \tag{2.2}$$

Each relation R_i in D contains data columns and the event coulmn with independent Boolean random variables $x_1^{(i)}, ..., x_{n_i}^{(i)}$. Furthermore, let $I_{max} \equiv \{(i_1, ..., i_k)|1 \leq i_1 \leq |R_1|, ..., 1 \leq i_k \leq |R_k|\}$. Since the inequality joins in Q are semantically equivalent to selections on the cross-product $R_1 \times ... \times R_k$ of all relations in Q, the linage $\phi_{Q(D)}$ for each output tuple is of the form

$$\phi_Q(D) = \sum_{(i_1,...,i_N)\in I} x_{i_1}^1 x_{i_2}^2 \cdot ... \cdot x_{i_k}^k, \quad I \subseteq I_{max}. \tag{2.3}$$

In particular, $I = I_{max}$ corresponds to the sum of all possible products of Boolean random variables from different relations.

Remark 2.13. *To simplify out notation, we will in the following often write ϕ_Q instead of $\phi_{Q(D)}$, where the database D is clear from the context.*

The query lineage ϕ_Q can be rewritten as:

$$\begin{aligned}
\phi_Q &= \sum_{(i_1,...,i_k)\in I} x_{i_1}^1 x_{i_2}^2 \cdot ... \cdot x_{i_k}^k \\
&= \sum_{i_1:\exists(i_1,...,i_k)\in I} x_{i_1}^1 \sum_{(i_1,...,i_k)\in I} x_{i_2}^2 \cdot ... \cdot x_{i_k}^k \\
&= \sum_{i_1:\exists(i_1,...,i_k)\in I} x_{i_1}^1 \cdot f(x_{i_1}^1) \\
&= \sum_{i_1:\exists(i_1,...,i_k)\in I} x_{i_1}^1 \sum_{i_2:\exists(i_1,...,i_k)\in I} x_{i_2}^2 \cdot f(x_{i_1}^1 x_{i_2}^2) \\
&= \ldots .
\end{aligned}$$

This leads to the definition of the central concept of *cofactors*:

Definition 2.14 (Cofactors)**.** *The partial sums $f(x_{i_1}^1)$, $f(x_{i_1}^1 x_{i_2}^2)$, ... are called* cofactors*. As it can be seen from the calculation above, the cofactor of $x_{i_1}^1 x_{i_2}^2 \ldots x_{i_{l-1}}^{l-1}$ is given by*

$$f(x_{i_1}^1 x_{i_2}^2 \ldots x_{i_{l-1}}^{l-1}) = \sum_{(i_1,...,i_k)\in I} x_{i_l}^l \cdot ... \cdot x_{i_k}^k$$

for any $1 \leq l \leq k$ and $1 \leq i_j \leq |X_j| \; \forall 1 \leq j \leq l-1$.

2.4 Disjunctive Normal Forms

In the last section we introduced query lineages of conjunctive queries over probabilistic databases. They are directly connected to a very well-known normal form of Boolean formulas:

Definition 2.15 (k-DNF)**.** *A Boolean formula ϕ is a* k-disjunctive normal form (k-DNF) *if it is a disjunction of conjunctions of k literals.*

In our mathematical notation, k-DNFs are sums of products of k Boolean variables. Dually, we interpret them as a set of conjunctions of k Boolean variables.

Example 2.16. *The general query lineage in Equation 2.3 is a k-DNF.*

Additionally to being of size k, all the products in query lineages fulfill an order on their literals: The variable at the l-th position is always from the variable set X_l of relation R_l. We capture this property with the following definition:

Definition 2.17 (Multipartite k-DNF)**.** *A k-DNF ϕ is* multipartite *with variable sets $X_1, \ldots, X_k$ if $X_i \cap X_j = \emptyset \; \forall 1 \leq i < j \leq k$ and all conjunctions in ϕ can be ordered such that for $1 \leq l \leq k$ the l-literal is always in X_l.*

To capture queries with self-joins, we additionally define a larger class of k-DNFs, where the variable sets are not necessarily distinct:

Definition 2.18 (Semi-multipartite k-DNF)**.** *A k-DNF ϕ is* semi-multipartite *with variable sets $X_1, \ldots, X_k$ if all conjunctions in ϕ can be ordered such that for $1 \leq l \leq k$ the l-literal is always in X_l.*

Remark 2.19. *Every k-DNF ϕ is semi-multipartite with the variable sets $X_1 = \ldots = X_k = Vars(\phi)$, where the latter denotes the set of all variables in ϕ. It is thus clear that the variable sets are an important part of the characterization of a semi-multipartite k-DNF.*

From the last section we get

Proposition 2.20. *Let Q be a query with subgoeals $R_1, \ldots, R_k$. Then the lineage ϕ_Q is a semi-multipartite k-DNF with variable sets $Vars\{R_1\}$, …, $Vars\{R_k\}$. If Q has no self-joins, then ϕ_Q is a multipartite k-DNF.*

Example 2.21. *An example of a multipartite k-DNF is the query lineage from Figure 6.2:*

$$\phi_{Q'}(D) = x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_3y_3.$$

The variable sets are $X_1 = \{x_1, x_2, x_3\}$ and $X_2 = \{y_1, y_2, y_3\}$.

A k-DNF which is not multipartite but semi-multipartite with non-trivial variable sets is

$$\phi = x_1y_1x_1 + x_2y_1x_2 + x_2y_2x_1.$$

Here, $X_1 = X_3 = \{x_1, x_2\}$ and $X_2 = \{y_1, y_2\}$.

Definition 2.22. *The size of a k-DNF ϕ is defined as*

$$|\phi| = k \cdot n,$$

where n is the number of conjunctions in the disjunction.

Example 2.23. *For the k-DNFs in Example 2.21 we get $|\phi_{Q'}(D)| = 2 \cdot 5 = 10$ and $|\phi| = 3 \cdot 3 = 9$.*

Definition 2.24. *Let ϕ, ϕ' be a k-DNF, c be a conjunction of k Boolean variables, X be a set of Boolean variables and $\sigma : X \to \{0,1\}$ a truth assignment. Then*

- *$c \in \phi$ denotes that c is one of the conjunctions in the disjunction of ϕ,*
- *$\sigma(\phi)$ denotes the DNF that arises from ϕ when all Boolean variables in $x \in X \cap Vars(\phi)$ are replaced by their assigned value $\sigma(x)$,*
- *$\phi \setminus X$ is defined as $\sigma(\phi)$ with $\sigma : X \to \{0,1\}, \sigma \equiv 0$,*
- *$\phi \cap X$ denotes the set of conjunctions in ϕ which have at least on variable from X in them,*
- *$\phi \subseteq \phi'$ denotes that $c \in \phi'$ for every conjunction $c \in \phi$.*

Example 2.25. *Let $\phi = x_1y_1 + x_1y_2 + x_2y_2$, $c = x_1y_1$, $c' = x_2y_1$ and $X = \{y_2\}$. Then $c \in \phi$, $c \notin \phi$, $\phi \setminus X = x_1y_1 \subseteq \phi$ and $\phi \cap X = \{x_1y_2, x_2y_2\}$. Furthermore, let $\sigma : \{x_1, x_2\} \to \{0,1\}$ with $\sigma(x_1) = 1, \sigma(x_2) = 0$. Then $\sigma(\phi) = y_1 + y_2$.*

Remark 2.26. *Even though inclusion for k-DNFs is defined on a syntactic level, it can easily be interpreted semantically. If $\phi \subseteq \phi'$, then $\phi = 1$ implies $\phi' = 1$, since all conjunctions from ϕ are in the sum of ϕ' and at least one of them is true. Conversely, $\neg\phi' \Rightarrow \neg\phi$.*

2.5 Binary Decision Diagrams

Binary decision diagrams (BDDs) are graphical representations of Boolean formulas. Mathematically, a BDD is a rooted, directed and acyclic graph which represents a Boolean function ϕ with (independent) variables $x_1, x_2, ...$ based on the *Shannon expansion*

$$\begin{aligned}
\phi &= x_1 \cdot \phi|_{x_1=1} + \overline{x}_1 \cdot \phi|_{x_1=0} \\
&= x_1 \cdot x_2 \cdot \phi|_{x_1=1,x_2=1} + x_1 \cdot \overline{x}_2 \cdot \phi|_{x_1=1,x_2=0} \\
&\quad + \overline{x}_1 \cdot x_2 \cdot \phi|_{x_1=0,x_2=1} + \overline{x}_1 \cdot \overline{x}_2 \cdot \phi|_{x_1=0,x_2=0} \\
&= ...
\end{aligned}$$

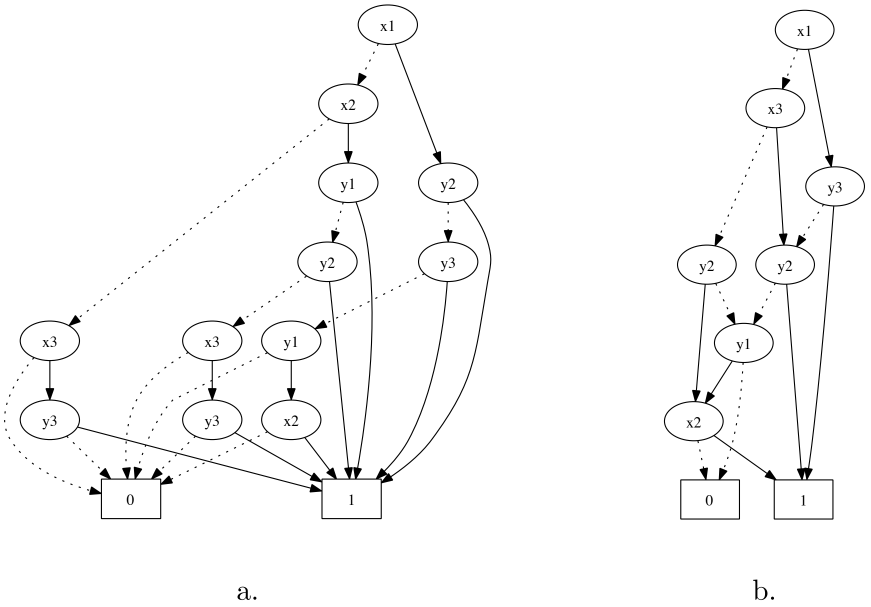


Figure 2.6: Example BDDs for $\phi = x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_3y_3$

The node-set of the graph consists of two terminal nodes - 0 and 1 - which have no out-going edges, and nodes for the random variables of the function, each of which has two out-going edges - high and low. Following standard notation, high (low) edges are visualized by solid (dotted) arrows. The high edge represents the 'true/1' case and the low edge represents the 'false/0' case. Since the BDD is rooted, there exists one node which has no in-going edges.

Ultimately, all paths in the BDD have to end in either the 1- or 0-terminal node, depending on whether the represented assignment makes the Boolean function true or false. If B is a BDD for ϕ then a path for a truth assignment σ ends in 1 if and only if $\sigma \in SAT(\phi)$, where $SAT(\phi)$ is the set of satisfying assignments of ϕ.

Example 2.27. *Figure 2.6 shows BDDs for the example 2-DNF* $\phi = x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_3y_3$.

The size of a BDD can be measured in terms of the number of nodes, number of edges or a combination of both. It depends heavily on the order of variable elimination along each path and possible recombinations of paths with isomorphic subgraphs.

Example 2.28. *In the he BDD in Figure 2.6.a no paths are recombined on variable nodes (i.e. all non-terminal nodes have at most one in-coming edge), which leads to unnecessary duplication: The* x_3*-*y_3*-subgraphs in the left half of the graph are isomorphic and hence one of them could be dropped.*

In contrast, in BDD in Figure 2.6.b is reduced*, i.e. all isomorphic subgraphs are merged. Additionally, it is* ordered*, which means that the variable order is the same along each path in the BDD. In this case, the order is* $x_1, x_3, y_3, y_2, y_1, x_2$*. Such reduced ordered BDDs (OBDDs) are an important subclass of BDDs [5].*

Chapter 3

BDD Construction for k-DNFs

In this chapter, we introduce two classes of multipartite, monotone k-DNFs - *linear-inclusion k-DNFs* and *cofactor-separable 2-DNFs.* We define them based on syntactic properties and - exploiting those - give polynomial-time algorithms which compile the k-DNFs into equivalent BDDs of polynomial size. We also show that the second class is a strict superclass of the first for $k = 2$.

Our algorithms make use the cofactor structure in the k-DNFs. In the first part, we present BDDLinIncl, which compiles linear-inclusion k-DNFs ϕ in time $O(|\phi|log|\phi|)$ into BDDs of linear size. This is possible by eliminating the Boolean variables in order of the inclusion of cofactors. In the second part of this chapter, we introduce BDDCofSep, which can be applied to cofactor-separable 2-DNFs. Again using a good variable elimination order, it keeps the number of paths in the BDD linear, which results in a quadratical BDD size and run-time.

In Chapter 4 we show how the results in this chapter are connected to query evaluation on probabilistic databases and in Chapter 5 we use them to proof that certain classes of conjunctive queries are tractable. However, the results are not restricted to applications in probabilistic database theory and hold independently of them. For example, in Chapter 7 we apply them to counting problems on graphs and show tractability for counting vertex covers on convex bipartite graphs.

3.1 Linear-Inclusion k-DNFs

3.1.1 Main Results

We start by defining the class of linear-inclusion k-DNFs:

Definition 3.1 (Linear-Inclusion k-DNF). *A multipartite monotone k-DNF ϕ with variable sets $X_1, \dots, X_k$ is a* linear-inclusion *k-DNF if there exist orders $\pi_1, \dots, \pi_k$ for the variable sets such that for any $1 \leq l \leq k$, $1 \leq i'_l < i_l \leq X_l$ and $1 \leq i_m \leq |X_m|$ for $1 \leq m \leq l-1$ it holds*

$$f(x^1_{i_1} \dots x^{l-1}_{i_{l-1}} x^l_{\pi_l(i_l)}) \subseteq f(x^1_{i_1} \dots x^{l-1}_{i_{l-1}} x^l_{\pi_l(i'_l)}) \tag{3.1}$$

or

$$f(x^1_{i_1} \dots x^{l-1}_{i_{l-1}} x^l_{i'_l}) = 0. \tag{3.2}$$

Any such orders are called compatible orders.

Remark 3.2. *For two cofactors f, f' the relation $f \subseteq f'$ means that the index set of f is included in the index set of f' (Definition 2.24). Semantically, it implies that $f \Rightarrow f'$ and $\neg f' \Rightarrow \neg f$ (Remark 2.26).*

Intuitively, cofactor-separability means that if we choose arbitrary instances $x^1_{i_1}, ..., x^{l-1}_{i_{l-1}}$ for the first $l-1$ positions than the corresponding cofactors of the variables in X_l are linearily included in each other. The order of inclusions is specified by π_l, i.e.

$$\begin{aligned} f(x^1_{i_1} \dots x^{l-1}_{i_{l-1}} x^l_{\pi_l(1)}) &\subseteq f(x^1_{i_1} \dots x^{l-1}_{i_{l-1}} x^l_{\pi_l(2)}) \\ &\subseteq f(x^1_{i_1} \dots x^{l-1}_{i_{l-1}} x^l_{\pi_l(3)}) \\ &\subseteq \dots . \end{aligned}$$

Example 3.3. *Let us look at an example linear-inclusion k-DNF:*

$$\begin{aligned} \phi \;=\; & r_1\{s_1\,[t_1(u_1+u_2+u_3)+t_2(u_2+u_3)+t_3(u_3)] \\ & \quad +s_2\,[t_1(u_1+u_2+u_3)+t_2(u_2+u_3)]+s_3\,[t_1(u_1+u_2+u_3)]\} \\ & +r_2\{s_1\,[t_2(u_2+u_3)+t_3(u_3)]+s_2\,[t_2(u_2+u_3)]\} \qquad (3.3) \\ & +r_2\{s_1\,[t_3(u_3)]\} \qquad (3.4) \end{aligned}$$

The variables are already enumerated such that all orders can be choosen to be the trivial order $\pi_{\mathbb{1}}$ with $\pi_{\mathbb{1}}(i) = i$. We have

$$\begin{aligned} f(r_1) \;&=\; s_1\,[t_1(u_1+u_2+u_3)+t_2(u_2+u_3)+t_3(u_3)] \\ &\quad +s_2\,[t_1(u_1+u_2+u_3)+t_2(u_2+u_3)] \\ &\quad +s_3\,[t_1(u_1+u_2+u_3)] \\ \supset f(r_2) \;&=\; s_1\,[t_2(u_2+u_3)+t_3(u_3)]+s_2\,[t_2(u_2+u_3)] \\ \supset f(r_3) \;&=\; s_1\,[t_3(u_3)] \end{aligned}$$

$$\begin{aligned} f(r_1 s_1) &= t_1(u_1+u_2+u_3)+t_2(u_2+u_3)+t_3(u_3) \\ \supset f(r_1 s_2) &= t_1(u_1+u_2+u_3)+t_2(u_2+u_3) \\ \supset f(r_1 s_3) &= t_1(u_1+u_2+u_3) \end{aligned}$$

$$\begin{aligned} f(r_1 s_1 t_1) &= u_1+u_2+u_3 \\ \supset f(r_1 s_1 t_2) &= u_2+u_3 \\ \supset f(r_1 s_1 t_3) &= u_3 \end{aligned}$$

and so on.

In the next section, we introduce the algorithm BDDLinIncl(ϕ, $\pi_1, \ldots, \pi_k$) which constructs a BDD for a linear-inclusion k-DNF ϕ. Our main result is

Theorem 3.4. *Let ϕ be a linear-inclusion k-DNF with variable sets $X_1, \ldots, X_k$ and compatible orders $\pi_1, \ldots, \pi_k$. Then BDDLinIncl(ϕ, $\pi_1, \ldots, \pi_k$) returns in time $O(|\phi| \log |\phi|)$ BDD B_ϕ for ϕ. The size of the BDD is bounded by*

$$|B_\phi| \leq |\phi| \leq k \cdot |X_1| \cdot \ldots \cdot |X_k|. \tag{3.5}$$

This theorem follows immediately from Lemmas 3.6 and 3.7 in the next section.

Remark 3.5. *The variable orders are part of the input and must thus be specified explicitly when calling BDDLinIncl.*

3.1.2 Efficient BDD Construction with BDDLinIncl

The algorithm for BDD construction is described in Figure 3.1. It first sorts ϕ according to the order $\pi_1, \ldots, \pi_k$ and renames the variables such that all orders become the identity order $\pi_{\mathbb{1}}$. Then, starting at the first position, BDDLinIncl creates nodes for the variables in X_1. The low edges are then connected in order and the high edges are connected to the corresponding cofactors. (Figure 3.2) Subsequently, the cofactors are compiled, i.e. nodes are created for X_2 and similar connections are made. This is repeated until all k sets of variables are compiled. (Figure 3.3)

The linear-inclusion property is important, as it allows all low edges from cofactors to be connected to the 0-terminal node: We start by rewriting ϕ

BDDLinIncl(ϕ, $\pi_1, \ldots, \pi_k$)

1. Sort and rename the variables in ϕ.
2. Create the initial structure for the BDD B_ϕ shown in Figure 3.2.
3. Apply step 2 to each of the cofactors $\phi = f(x_1^1)$, ..., $f(x_{n_1}^1)$ (Figure 3.3). Recursively repeat this for the new subproblems until the whole lineage is compiled.
4. Return the B_ϕ.

Figure 3.1: BDDLinIncl

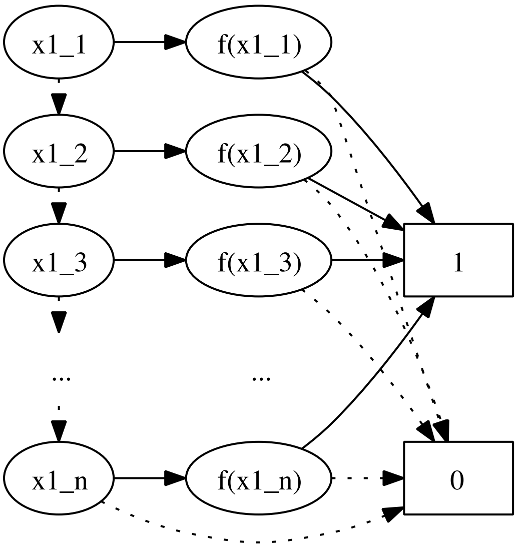


Figure 3.2: Initial structure for B_ϕ in BDDLinIncl at level 1

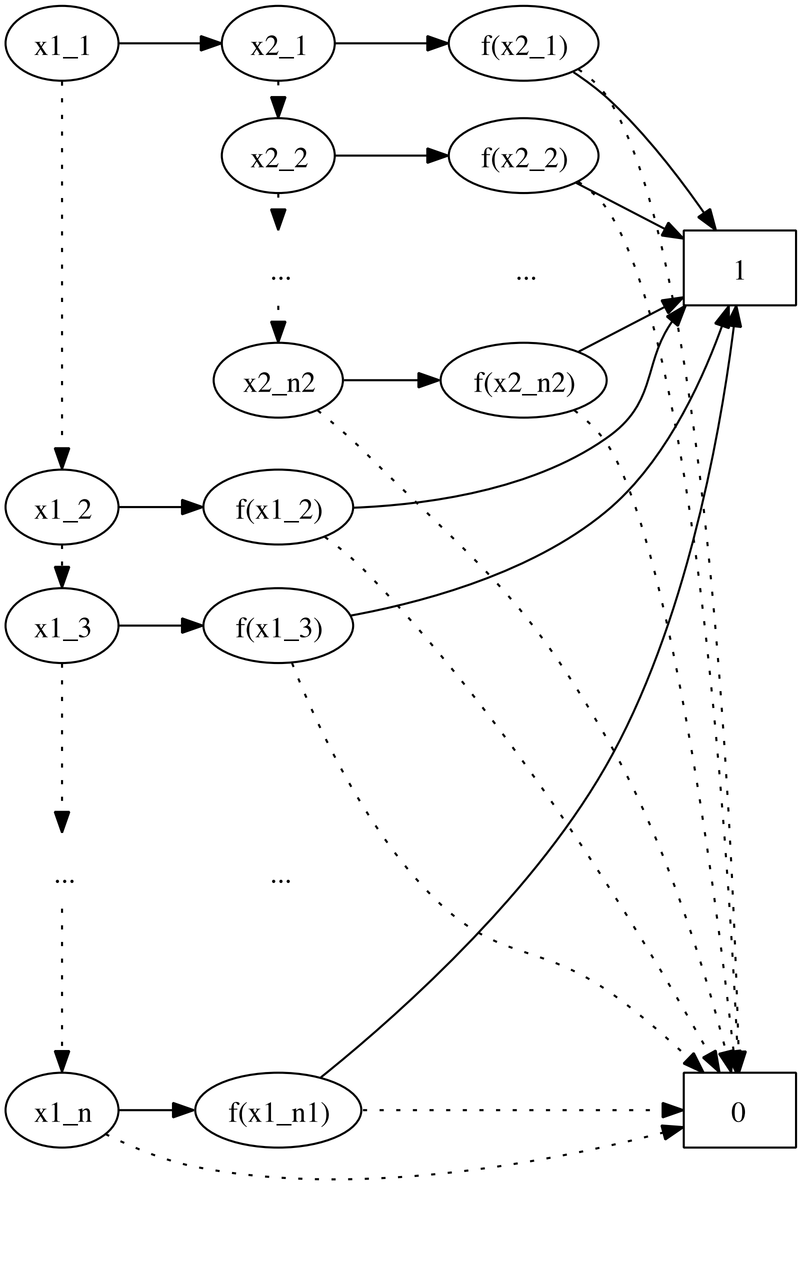


Figure 3.3: Structure for B_ϕ in BDDLinIncl after compiling $f(x_1^1)$

according to partial assignments of the x_i^1-variables using the Shannon expansion:

$$\begin{aligned}\phi &= x_1^1 \cdot \phi|_{x_1^1=1} + \overline{x}_1^1 \cdot \phi|_{x_1^1=0}\\ &= x_1^1 \cdot \phi|_{x_1^1=1} + \overline{x}_1^1 x_2^1 \cdot \phi|_{x_1^1=0,x_2^1=1} + \overline{x}_1^1 \overline{x}_2^1 \cdot \phi|_{x_1^1=0,x_2^1=0}\\ &= \ldots\\ &= x_1^1 \cdot \phi|_{x_1^1=1} + \overline{x}_1^1 x_2^1 \cdot \phi|_{x_1^1=0,x_2^1=1} + \ldots + \overline{x}_1^1 \ldots \overline{x}_{n_1-1}^1 x_{n_1}^1 \cdot\\ &\quad \cdot \phi|_{x_1^1=0,\ldots,x_{n_1-1}^1=0,x_{n_1}^1=1} + \overline{x}_1^1 \ldots \overline{x}_{n_1}^1 \cdot \underbrace{\phi|_{x_1^1=0,\ldots,x_{n_1}^1=0}}_{=0} \qquad (3.6)\end{aligned}$$

Expanding the j-th term in them sum we get

$$\phi|_{x_1^1=0,\ldots,x_j^1=1} = f(x_j^1) + \sum_{k=j+1}^{n_1} x_k^1 \cdot f(x_k^1). \qquad (3.7)$$

The inclusion condition implies that $f(x_j^1) \supseteq f(x_k^1), \forall k \geq j$. This means

$$f(x_j^1) = 0 \Rightarrow f(x_k^1) = 0 \forall k \geq j \qquad (3.8)$$

and thus

$$\phi|_{x_1^1=0,\ldots,x_j^1=1} = f(x_j^1). \qquad (3.9)$$

This allows us to rewrite (3.6) to

$$\phi = x_1^1 \cdot f(x_1^1) + \overline{x}_1^1 x_2^1 \cdot f(x_2^1) + \ldots + \overline{x}_1^1 \ldots \overline{x}_{n_1-1}^1 x_{n_1}^1 \cdot f(x_{n_1}^1), \qquad (3.10)$$

which is exactly what Figure 3.2 represents.

Now the inclusion condition assures that similar expansions hold for the cofactors of all subsequent levels. Thus, the same compilation technique can be used on them (Figure 3.3). This proves the correctness of the iteration step (step 3 in Figure 3.1). Consequently, BDDLinIncl(ϕ, $\pi_1, \ldots, \pi_k$) level-for-level compiles the formula into an equivalent BDD. This proves

Lemma 3.6. *BDDLinIncl(ϕ, $\pi_1, \ldots, \pi_k$) returns a BDD B_ϕ for ϕ.*

Lemma 3.7. *BDDLinIncl(ϕ, $\pi_1, \ldots, \pi_k$) returns in time $O(|\phi| \log |\phi|)$ a BDD B_ϕ of size*

$$|B_\phi| \leq |\phi| \leq k \cdot |X_1| \cdot \ldots \cdot |X_k|.$$

Proof. The size of B_ϕ is bounded by the size of ϕ: $|B_\phi|$ is the number of variables in the formula of ϕ factored according to the variable set order $X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_k$. This is obviously smaller than $|\phi| = k \cdot |I|$, which is the number of variables in the expanded version ϕ, i.e. the k-DNF form where no variables are factored out. Here, I is the index set of ϕ as introduced in Section 2.3.

ϕ has at most $|X_1| \cdot \ldots \cdot |X_k|$ summands. Since each summand has k variables, $|\phi|$ is bounded by $k \cdot |X_1| \cdot \ldots \cdot |X_k|$.

Figure 3.4 shows a pseudo-code implementation of BDDLinIncl. First, the set of summands in the lineage ϕ is sorted ascendingly, which takes $O(|I_Q| \log |I_Q|)$ steps. This is the same as $O(|\phi| \log |\phi|)$, because k is fixed. Then the array/table-based approach (s. Figure 3.5) needs at most a constant number $c = c(k)$ of operations per summand in ϕ, which adds $O(|\phi|)$ operations. Thus, the dominant term derives from the sorting step and the overall run-time is $O(|\phi| \log |\phi|)$. □

Remark 3.8. *The size of a BDD could also by defined as the number of edges or the sum of edges and nodes. Both measures would only differ by a maximal factor of 2 and 3, resp., because in our BDDs every non-terminal node has exactly two out-going edges.*

Remark 3.9. *The BDD which is returned by BDDLinIncl is ordered but it is not reduced, because isomorphic substructures are not necessarily merged. However, this could be done in linear time in the BDD size in a post-processing step [24].*

```
BDDLinIncl(int k, int NumOfSum,
                        array[1:NumOfSum] Phi of variable[1:k])
   Sort Phi ascendingly and rename variables;
   Create array[0:k+1] nodes of Node;
   BDD B = new BDD();
   Node n0 = new TerminalNode(0);
   Node n1 = new TerminalNode(1);
   B.addNode(n0);
   B.addNode(n1);
   Nodes[k+1] = n1;

   For i=k...1 do
      nodes[i] = new Node(Phi[1][i]);
      nodes[i].high = nodes[i+1];
      B.addNode(nodes[i]);

   For j=2...NumOfSum do
      l = 1;
      While Phi[j][l] = Phi[j-1][l] do l++;

      For i=k...l+1 do
         nodes[i].low = n0;
         nodes[i] = new Node(Phi[j][i]);
         nodes[i].high = nodes[i+1];
         B.addNode(nodes[i]);

      nodes[0] = nodes[l];
      nodes[l] = new Node(Phi[j][l]);
      nodes[l].high = nodes[l+1];
      nodes[0].low = nodes[l];
      B.addNode(nodes[l]);

   For i=1...k do
      nodes[i].low = n0;
   Return B;
```

Figure 3.4: Pseudo-code implementation of BDDLinIncl

j	i=1	2	3	4	
1	r_1	s_1	t_1	u_1	
2	r_1	s_1	t_1	u_2	
3	r_1	s_1	t_2	u_1	
4	r_1	s_1	t_2	u_2	
5	r_1	s_1	t_3	u_2	
6	r_1	s_2	t_2	u_1	←
7	r_1	s_2	t_2	u_2	
8	r_1	s_2	t_3	u_2	
9	r_2	s_1	t_2	u_1	
⋮	⋮	⋮	⋮	⋮	

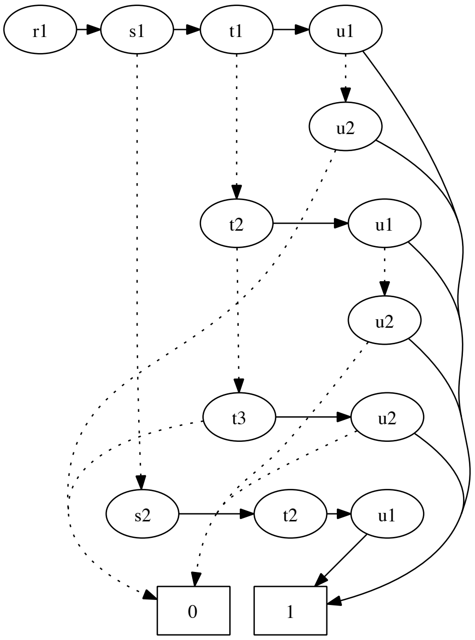


Figure 3.5: Array/table-based approach for BDDLinIncl

3.2 Cofactor-Separable 2-DNFs

3.2.1 Main Results

Let us first introduces two definitions, which will make the notation a lot more compact:

Definition 3.10. *Let ϕ be a semi-multipartite 2-DNF with variable sets $X = \{x_1, ..., x_{|X|}\}$ and Y. Then*

$$\phi[i,j] := \sum_{i \leq l \leq j} x_l \cdot f(x_l). \tag{3.11}$$

In particular, $\phi[1, |X|] = \phi$.

Definition 3.11. *Let ϕ be a semi-multipartite 2-DNF with variable sets $X = \{x_1, ..., x_{|X|}\}$ and Y and $\sigma : Vars(\phi[1,i]) \rightarrow \{0,1\}$ be a partial truth assignment. Then*

$$f(\sigma, i) := \bigcup_{1 \leq j \leq i : \sigma(x_j) = 1} f(x_j). \tag{3.12}$$

Now we can define cofactor-separable 2-DNFs as follows:

Definition 3.12 (Cofactor-Separable 2-DNF)**.** *A multipartite monotone 2-DNF ϕ with variable sets X, Y is a* cofactor-separable *2-DNF if there exist an order π_X for the variable set X such that after renaming the variables according to the order the following holds: For any $1 \leq i \leq |X|$ and any partial truth assignments $\sigma, \sigma' : Vars(\phi[1,i]) \rightarrow \{0,1\}$ which fulfill the following conditions*

$$\sigma(\phi[1,i]) = \sigma'(\phi[1,i]) = 0, \tag{3.13}$$

$$\sigma(x_i^1) = \sigma'(x_i^1) = 1, \tag{3.14}$$

$$f(x_i^1) \setminus f(\sigma, i-1) \neq \emptyset, f(x_i^1) \setminus f(\sigma', i-1) \neq \emptyset \tag{3.15}$$

it follows that

$$\sigma(\phi[i+1,n]) = \sigma'(\phi[i+1,n]). \tag{3.16}$$

Any such order π_X is called compatible order*.*

The intuition behind equation (3.16) is that the assignment of the first $i-1$ variables from X and their cofactors becomes irrelevant once x_i is set true and its cofactor is evaluated. If $f(x_i)$ is true, then ϕ is true. If $f(x_i)$ is false, then we can combine all paths in a BDD corresponding to assignments which meet the above conditions. Thus, the i-th level separates the history - $\phi[1,i-1]|_\sigma$ - and the future - $\phi[i+1,|X|]$ - for all those assignments. Since

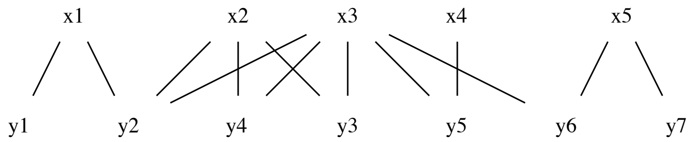


Figure 3.6: Example cofactor-separable 2-DNF

this property depends on the structure of the cofactors, we call such 2-DNFs cofactor-separable.

The conditions in (3.13) to (3.15) can be understood as follows: (3.13) assures that σ does not make ϕ true before the $i+1$-th position. Otherwise, we would not have to consider it anymore. (3.14) means that the cofactor of x_i^1 is relevant for the assignment σ. Condition (3.15) assures that $f(x_i)$ is not already trivially false due to the implications of the first $i-1$ levels, in which would make it irrelevant and hence unnecessary to be considered.

All conditions will become clearer in the next section, because they are naturally related to the BDD construction algorithm for cofactor-separable 2-DNFs.

Example 3.13. *An example cofactor-separable 2-DNF is*

$$\begin{aligned}\phi &= x_1(y_1+y_2)+x_2(y_2+y_3+y_4)+x_3(y_2+y_3+y_4+y_5+y_6)\\ &\quad +x_4(y_5)+x_5(y_6+y_7). \qquad (3.17)\end{aligned}$$

The structure of ϕ is visualized in Figure 3.6. Intuitively, any cofactor separates ϕ into two halves. That ϕ is in fact a cofactor-separable 2-DNF is proven in Section 5.2.

Theorem 3.14. *Let ϕ be cofactor separable 2-DNF with variable sets X, Y and compatible order π_X. Then BDDCofSep(ϕ, π_X) returns in time $O(|\phi|^2)$ a BDD B_ϕ for ϕ. The size of the BDD is bounded by*

$$|B_\phi| \leq 3/8 \cdot |\phi|^2 + |\phi|/2. \qquad (3.18)$$

This theorem follows immediately from Lemmas 3.16, 3.17 and 3.17 in the next section.

As mentioned in the introduction to this chapter, the following theorem connects cofactor-separable and linear-inclusion 2-DNFs:

Theorem 3.15. *Every linear-inclusion 2-DNF is a cofactor-separable 2-DNF.*

Proof. ϕ can be written as

$$\phi = \sum_{1 \leq i \leq n} x_i \cdot f(x_i) \tag{3.19}$$

with $n \in \mathbb{N}$ and $f(x_1) \supseteq f(x_2) \supseteq \ldots$. Let now $1 \leq i \leq n$ and $\sigma : Vars(\phi[1,i]) \rightarrow \{0,1\}$ be arbitrary with $\phi[1,i] = 0$ and $\sigma(x_i) = 1$. Then $f(x_i) = 0$, which implies $\sigma(\phi[i+1,n]) = 0$ independent of the exact choice of σ, because $f(x_i) \supseteq f(x_{i+1}) \supseteq \ldots$. □

3.2.2 Efficient BDD Construction with BDDCofSep

In this section we describe a polynomial-time algorithm that compiles cofactor-separable 2-DNFs into polynomially sized BDDs. The basic idea is to make use of the special structure of cofactor-separable DNFs and keep track of variables in the cofactors which have already been set false along a path in the BDD. This allows recombinations of paths and keeps the BDD polynomial.

A high-level description of BDDCofSep is given in Figure 3.7. In contrast to the algorithm for linear-inclusion k-DNFs it is no longer possible to connect cofactors to the 0-terminal node if they are false, because the cofactors are not linearily included. Thus, we need to continue the path in the BDD while remembering the relevant parts of the assignment so far. Our approach uses the cofactor-seperable property to keep the number of distinct relevant sets of already evaluated (falsified) variables (and hence the number of parallel paths in the BDD) polynomial.

As for BDDLinIncl, the algorithm first sorts and rewrites ϕ according the order given (Figure 3.7, Step 1). We again w.l.o.g. rename the Boolean variables such that the order becomes trivial. This gives

$$\phi = \sum_{1 \leq i \leq |X|} x_i f(x_i)$$

The BDD construction now starts with the structure shown in Figure 3.8: A node for x_1 is created and its high edge is connected to $f(x_1)$. The latter is a sum of variables from Y, which can be implemented as consecutive Y-nodes, which are connected by their low edges and connected to the 1-terminal node via their high edges. The low edges from x_1 and the last node in $f(x_1)$ are then connected to two separate x_2-nodes, so that we can keep track of which variables from Y are already set to false (none for the left path, $f(x_1)$ for the right path).

Subsequently, for each level $i \in \{2, ..., |X|\}$ the BDD is extended as follows: There are incoming paths for subsets s_0, s_1, ... , s_k of Y-variables already set to false. For each of these a x_i-node is created and for the ones with with $s_j \not\supseteq f(x_i)$ its high edge is connected to a sum representing

BDDCofSep(ϕ, π_X)

1. Sort and rename the variables in ϕ.
2. Create the initial structure for the BDD B_ϕ shown in Figure 3.8.
3. For each variable $x_i, i \in \{2, ..., |X|\}$ add the structure in Figure 3.9 to the BDD.
4. Connect all loose edges from the last level to the 0-terminal node.
5. Return the B_ϕ.

Figure 3.7: BDDCofSep

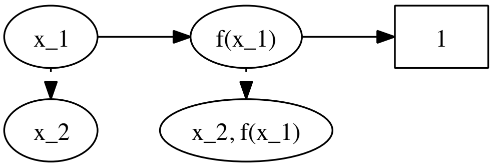


Figure 3.8: Initial structure for BDDCofSep

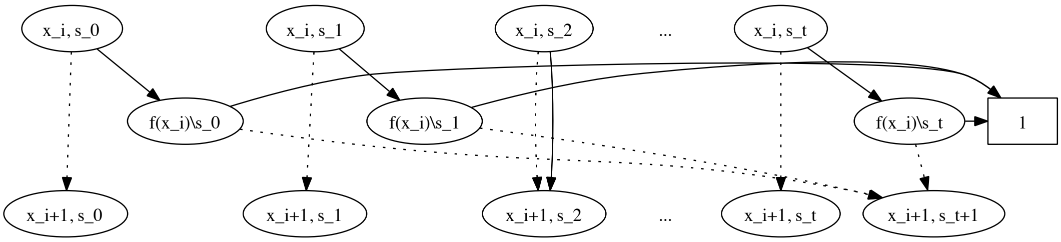


Figure 3.9: Level i of BDDCofSep

$f(x_i) \setminus s_j$. The low edges from all x_i-nodes are forwarded to the next level and will later be connected to x_{i+1}-nodes. The same is done for the high edges of nodes with $s_j \supseteq f(x_i)$. Finally, the high edges from all the sums of Y-variables are connected to the 1-temrinal node and the low edges define a new path with variable set $s_{k+1} = f(x_i)$. This is, where the cofactor-separability is important: It allows that all paths may be combined into a single one, because the history of the paths is no longer relevant. Otherwise, we would set variables multiple times along one path and B_ϕ would not be a BDD.

The whole procedure at level i is shown in Figure 3.9. In this example, $s_0, s_1, s_t \not\supseteq f(x_i)$ and $s_2 \supseteq f(x_i)$.

After repeating this BDD extension for every variable in X, all loose edges from the last level are connected to the 0-terminal node. This completes the BDD compilation of ϕ.

Lemma 3.16. *The graph B_ϕ returned by BDDCofSep(ϕ, π_X) is a BDD.*

The rigorous mathematical proof of Lemma 3.16 can be found in the Appendix.

Lemma 3.17. *BDDCofSep(ϕ, π_X) returns a BDD B_ϕ for ϕ.*

Proof. Let $\sigma : Vars(\phi) \to \{0, 1\}$ be an arbitrary total truth assignment. We have to show that the path representing σ ends in the 1-terminal node if $\sigma \in SAT(\phi)$ and in the 0-terminal node if $\sigma \notin SAT(\phi)$.

"$\sigma \notin SAT(\phi)$": Assume $\sigma \notin SAT(\phi)$, but the path corresponding to σ ends on the 1-terminal node. By construction, this node has only in-going edges from y_j nodes that are set to true. Furthermore, those y_j nodes all occur in sums which follow a x_i node that is set to true and contains y_j in its cofactor. Thus, there exists i, j such that $\sigma(x_i) = 1$, $\sigma(y_j) = 1$ and $y_j \in f(x_i)$. This contradicts $\sigma \notin SAT(\phi)$. Consequently, the path cannot end in the 1-terminal node. Since we have already shown that each path ends in either the 1-terminal or the 0-temrinal node, it follows that the path for σ ends in the 0-terminal node.

"$\sigma \in SAT(\phi)$": Assume $\sigma \in SAT(\phi)$. Then there exists a minimal i and for this i a minimal j such that $\sigma(x_i) = 1$, $\sigma(y_j) = 1$ and $y_j \in f(x_i)$. It follows from the argument for the opposite direction that the path for σ cannot be connected to the 1-terminal at a level $k < i$. It can also not be connected to 0-terminal before the i-th level, because 0 only has in-going edges from the last level. Consequently, the path reaches level i.

At level i, the path goes through a node for x_i and follows the high edge to the corresponding sum of Y-variables. Since y_j is true, it has not already been set along the path. This is true because all high edges from Y-nodes are connected to node 1 and thus y_j cannot have occured along the path. Thus, y_j occurs in the sum. Moreover, the x_l, $l < j$, nodes which (potentially) occur in this sum before y_j all have $\sigma(y_l) = 0$, because j was choosen minimal with regard to i. Consequently, the path reaches the node for y_j and subsequently the 1-terminal node. □

Lemma 3.18. *BDDCofSep(ϕ, π_X) returns in time $O(|\phi|^2)$ a BDD B_ϕ of size*

$$|B_\phi| \leq min\left(3/8 \cdot |\phi|^2 + |\phi|/2, |X|^2 \cdot |Y| + 1/2(|X|^2 + |X|)\right). \qquad (3.20)$$

Proof. The number of X-nodes in B_ϕ is $1/2 \cdot |X| \cdot (|X| + 1)$. The number of Y-nodes is bounded by $|X| \sum_{1 \leq i \leq |X|} |f(x_i)| = |X| \cdot |I| = |X| \cdot |\phi|/2$, where I is the index set of ϕ. Together with $|X| \leq |I| = |\phi|/2$ and $|\phi| \leq 2 \cdot |X| \cdot |Y|$ we get

$$\begin{aligned} B_\phi &\leq 1/2 \cdot |X| \cdot (|X| + 1) + |X| \cdot \phi/2 \\ &\leq |\phi|^2/8 + |\phi|/4 + |\phi|^2/4 \\ &= 3/8 \cdot |\phi|^2 + |\phi|/2 \end{aligned}$$

and

$$\begin{aligned} B_\phi &\leq 1/2 \cdot |X| \cdot (|X| + 1) + |X| \cdot \phi/2 \\ &\leq 1/2(|X|^2 + |X|) + |X|^2 \cdot |Y|. \end{aligned}$$

The pseudo-code implementation of BDDCofSep in Figure 3.10 needs a sorting step with cost $O(|\phi| log |\phi|)$ and $O(|X|^2 \cdot |Y|)$ operations for the BDD creation. This implies that the run-time is $O(|\phi|^2)$. □

Remark 3.19. *We could further reduce the size of the BDD by sharing nodes among the different cofactors for each x_i: Since all cofactors' outgoing edges are recombined into one path, they can share suffixes. In the best case, this can reduce the number of Y-nodes per level to to $|Y|$ (i.e. in particular independent of $|X|$). However, the overall size of the BDD and run-time of the algorithm will still be quadratic.*

Example 3.20. *The BDD for the cofactor-separable 2-DNF from the last section is shown in Figure 3.11. All Y-nodes are connected with their high edge to the 1-terminal node, which we have omitted for simplicity.*

```
BDDCofSep(int NumOfSum,
        array[1:NumOfSum] Phi of variable[1:2], int NumX)
   Sort Phi ascendingly and rename variables;
   BDD B = new BDD(); Node n0 = new TerminalNode(0);
   Node n1 = new TerminalNode(1); B.addNode(n0); B.addNode(n1);
   Array[1:NumX+1] lastX of Node; Node nx; NodeSum ny;
   Array[0:NumX] s of NodeSet; s[0] = new NodeSet();

   lastX[1] = new Node(Phi[1][1]);
   B.addNode(lastX[1]);
   lastX[NumX+1] = n0;
   k = 1;
   For i=1...NumX do
      // Read cofactor of x_i
      s[i] = new NodeSet();
      s[i].addNode(new Node(Phi[k,2]));
      While Phi[k,1] == Phi[k++,1] do
         s[i].addNode(new Node(Phi[k,2]));

       If i < NumX then
          lastX[i+1] = new Node(Phi[k][1]);
          B.add(lastX[i+1]);

      // Create y_j and x_i+1 nodes and connect them
      For j=1...i do
         If i < NumX then
            nx = new Node(Phi[k,1]);
            B.addNode(nx);
         else nx = n0;
         lastX[j].low = nx;

         If s[i]-s[j-1] != emptyset then
            ny = new NodeSum(s[i] - s[j-1]);
            B.addNodeSum(ny);
            lastX[j].high = ny.firstNode();
            ny.lastNode().low = lastX[i+1];
         else
            lastX[j].high = nx;
         lastX[j] = nx;

   Return B;
```

Figure 3.10: Pseudo-code implementation of BDDCofSep

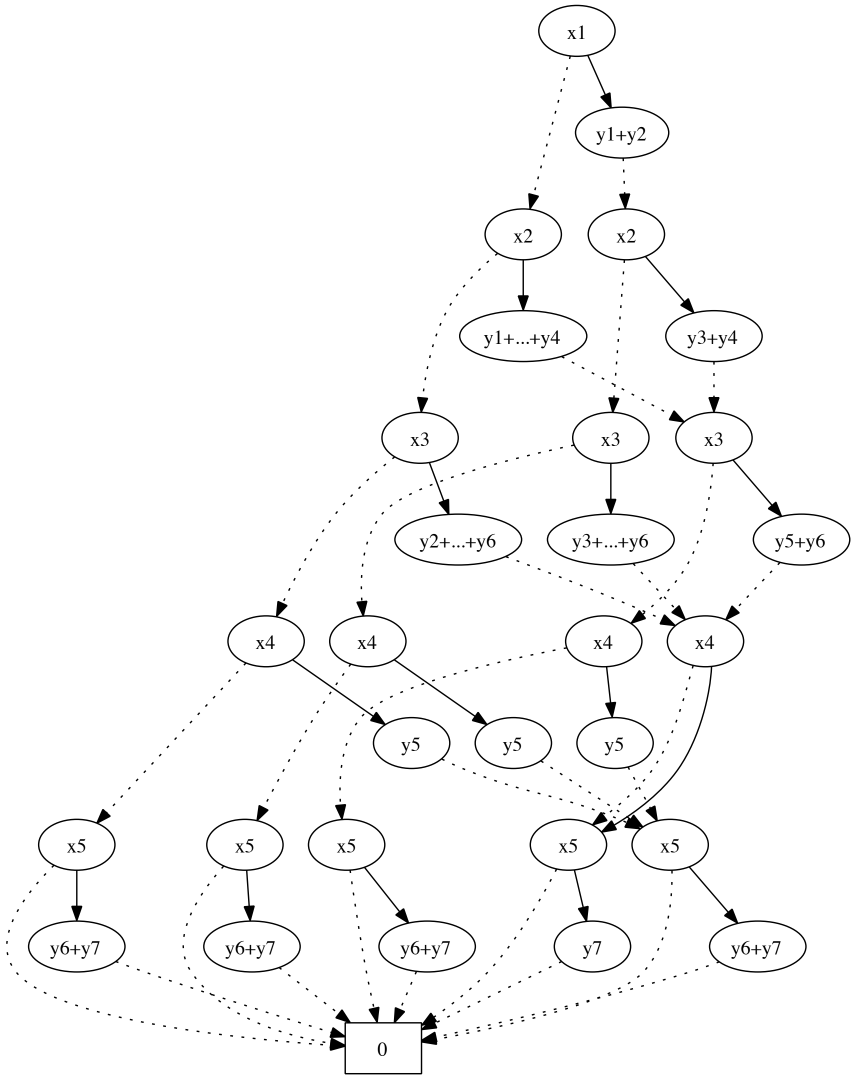


Figure 3.11: B_ϕ for the example DNF in (3.17)

Chapter 4

From BDD Construction To Query Evaluation

There are in principle two approaches for query evaluation on tuple-independent probabilistic databases [7]: Directly via computation of probabilities or indirectly via compilation of complex events. In the first approach, all random variables are replaced with their corresponding probability and those are manipulated during the evaluation of the relational operators. This is very efficient, but it works only for a subset of conjunctive queries called *safe queries* [7]. In the second approach, the query lineage is computed as a complex Boolean formula during query evaluation. The associated probability is then calculated after all relational operations have been applied. This gives the correct result for all queries, but is #P-hard in general [7, 6].

To apply the second approach, tractable classes of queries have to be identifed. For these classes, the lineage probabilites can be computed in polynomial time. One important technique for this is due to Olteanu and Huang [15], who introduced an approach that makes use of binary decision diagrams for the computation of the query probability. They first compile the query lineage into equivalent graphs called *reduced ordered binary decision diagrams* (OBDDs) - which are a subclass of BDDs - and then use a linear time algorithm to calculate the probability of the OBDDs. This leads to efficient state-of-the-art algorithms, which currently outperform the direct ones by up to two orders of magnitude [16]. In the following, we will extend this approach beyond OBDD-based algorithms.

After constructing the decision diagrams, it is important to be able to efficiently calculate their probabilities. This can be done using the surprisingly simple linear-time algorithm shown in Figure 4.1 [15]. We assume that given a node `n`, `n.prob` is the probability of the associated random variable being true and `n.high` and `n.low` point to the child-node along the true/1 and false/0 edge edge. Additionally, `n.p` stores a probability and is initialized to 0 for the 0-terminal node, 1 for the 1-terminal node and -1

```
BDDProb (Node n)
    If (n.p = -1) then n.p = n.prob * BDDprob(n.high)
                                  + (1-n.prob) * BDDprob(n.low);
    return n.p;
```

Figure 4.1: Algorithm BDDProb for probability computation on BDDs [15]

for all non-terminal nodes. Then BDDProb(root) recursively computes the probability of the function represented by the BDD.

Even though the algorithm was originally designed for OBDDs [15], it works just as well on general BDDs. The computation relies on the on the fact that the two branches subsequent to each note represent mutually exclusive assignments.

Proposition 4.1. *BDDprob calculates the probability of a BDD in linear time.*

Example 4.2. *Figure 4.2 shows the probabilities calculated by BDDprob for the example BDD from Figure 2.6.b and the random variable probabilities*

$$\begin{aligned} Pr(x_1) = Pr(x_2) = Pr(y_2) &= 1/2 \\ Pr(x_3) = Pr(y_3) &= 1/3 \\ Pr(y_1) &= 2/3 \end{aligned}$$

The question of whether a query Q is tractable and how it can be evaluated can thus be answered positively for tractable queries by giving a polynomial-time algorithm which creates a polynomially sized BDD for the query lineage. Since the probability of the BDD is the same as the probability of the function represented by it, this approaches allows the computation of $Pr(\phi_Q)$. By equation (2.2) this is the same as $Pr(Q)$. Note, that tractability here only refers to the probability computation for the query lineage, not the construction of the query lineage itself.

Proposition 4.3. *Let Q be a Boolean conjunctive query. If a polynomial-time algorithm can be given that compiles the query lineage ϕ_Q into a BDD, then Q can be evaluated in polynomial time in $|\phi_Q|$.*

A high-level description of the overall algorithm EvalQuery if given in Figure 4. Steps 1 and 3 are already completely specified. For step 2 we will use the BDD construction algorithms introduced in the last chapter. We need to show that a query Q is associated with a query lineage ϕ_Q that is a linear-inclusion or cofactor-separable k-DNF. Additionally, we have to explicitly specify compatible variable orders, because they are part of the input for the BDD construction algorithms.

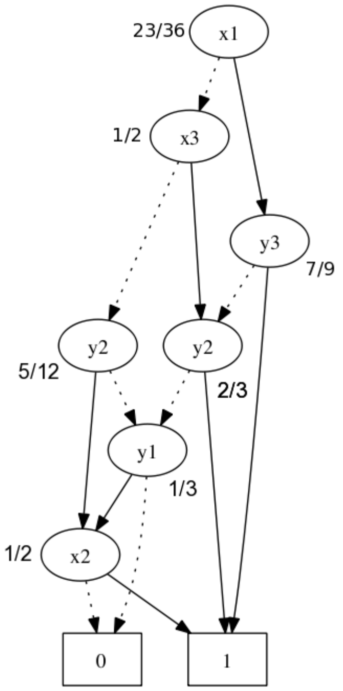


Figure 4.2: Example probability computation with BDDprob

EvalQuery(Query Q)

1. Evaluate the query using the modified operators defined in Figure 2.5 and pre-processing with selections for inequalities within single relations. This returns the query lineage ϕ_Q.
2. Compile ϕ_Q into a BDD.
3. Calculate the probability of the BDD using the BDDProb algorithm from Figure 4.1.

Figure 4.3: High-level description of EvalQuery

The run-time of our BDD compilation algorithms is only polynomial once the number of relations is fixed. This is inevitable, because the number of possible distinct conjunctions in the query lineage scales exponentially with the number of sets of query variables. In terms of complexity of query evaluation, this means that our results capture the *data complexity* for fixed queries. Furthermore, since the run-times are bounded by polynomials in the size of the lineage, combined polynomial complexity is preserved.

Chapter 5

Tractable Queries

In this chapter we introduce classes of tractable Boolean conjunctive queries with inequalities. Our classification of queries is based on structural properties of their associated query graphs, which turns out to be a very natural and clear way of characterizing them. The successive query classes allow for increasingly more complicated query lineages, which we show to be captured by the k-DNF classes introduced in Chapter 3. Consequently, the queries can be evaluated in polynomial-time with the algorithm EvalQuery as described in Chapter 4. Again, evaluation here only refers to the probability computation for the query lineage.

In the first section, we establish tractability for acyclic max-one queries by showing that their query lineages are captured by the class of linear-inclusion k-DNFs. Additionally, the necessary orders for algorithm BDDLin-Incl can be easily inferred from the query structure and database instance. This class of queries already strictly includes all previously known tractable queries with inequality [16].

In the second and third section, we extend the class of tractable queries towards cyclic queries and queries without the max-one property. To this end, we define two specific queries - the Single-Guard Query and the 2-Edge Query - and show that their query lineages are cofactor-separable 2-DNFs. Finally, we discuss possible extensions of our techniques towards other tractable queries, include the observation that all our tractability results hold for queries with smaller-or-equal-than inequalities ($\leq$) as well.

5.1 Acyclic Max-One Queries

We start by considering queries with the max-one property (Definition 2.8), i.e. queries in which at most one query variable per subgoal participates in out-going inequalities with query variables from other relations. Additionally, we demand that the queries be acyclic in the sense of Definition 2.11. An example query was already shown in Figure 2.4.

The following theorem states the main result of this section:

Theorem 5.1. *[Tractability of acyclic max-one queries]*
EvalQuery can evaluate any acyclic max-one query Q without self-joins on tuple-independent probabilistic databases in time $O(|\phi_Q| \log |\phi_Q|)$.

The proof is based on

Lemma 5.2. *Let Q be any acyclic max-one query without self-joins. Then ϕ_Q is a linear-inclusion k-DNF.*

The intuition behind Lemma 5.2 is the following: Since Q is acyclic, we can consider its subgoals - or equivalently its sets of random variables - in topological order. Fixing data values for the first $l-1$ positions leaves only certain tuples in the l-th relation with a possibly non-empty cofactor, namely those which data values fulfill all in-going inequalities.

Because of the max-one property, there is now at most only one column which participates in out-going inequalities with subsequent relations. We can order the tuples in relation R_l ascendingly on their value in this column. Then the cofactor of a tuple t with value v includes the cofactor of a tuple t' with value $v' > v$. This leads to a linear inclusion of cofactors on each position $1 \leq l \leq k$ and for each choice of random variables of the first $l-1$ positions.

A rigorous mathematical proof of Lemma 5.2 can be found in the Appendix.

Proof of Theorem 5.1. It is evident from the proof of Lemma 5.2 that the ascending orders $\pi_1, \ldots, \pi_k$ for the relations in Q are compatible orders in the sense of Definition 3.1. The variable sets $X_1, \ldots, X_k$ are pairwise intersection-free, because Q contains no self-joins and is evaluated on a tuple-independent probabilistic database. Thus, BDDLinIncl(ϕ_Q, $\pi_1, \ldots, \pi_k$) returns in time $O(|\phi_Q| \log |\phi_Q|)$ a BDD B_{ϕ_Q} of size

$$|B_{\phi_Q}| \leq |\phi_Q| \leq k \cdot \left(\max_{1 \leq l \leq k} \{|R_l|\} \right)^k .$$

This together with Proposition 4.3 concludes the proof. □

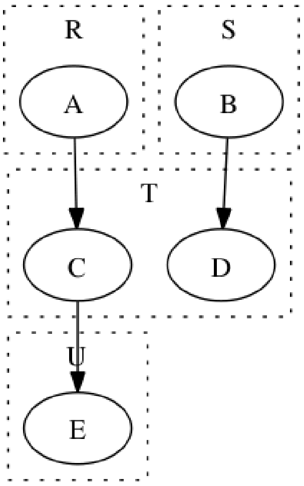


Figure 5.1: Example acyclic max-one query

R	A	**E**
	1	r_1
	2	r_2
	3	r_3

S	B	**E**
	1	s_1
	2	s_2
	3	s_3

T	C	D	**E**
	2	4	t_1
	3	3	t_2
	4	2	t_3

U	E	**E**
	3	u_1
	4	u_2
	5	u_3

Figure 5.2: Example database D

Example 5.3. *Consider again the example query from Section 2.2 (Figure 5.1) and the tuple-independent probabilistic database D is given in Figure 5.2.*

Possible topological subgoal orders for Q are $R \to S \to T \to U$ and $S \to R \to T \to U$. Choosing the first one we can write the query lineage as:

$$\begin{aligned}
\phi_Q \quad = \quad & r_1\{s_1\,[t_1(u_1+u_2+u_3)+t_2(u_2+u_3)+t_3(u_3)] \\
& \qquad +s_2\,[t_1(u_1+u_2+u_3)+t_2(u_2+u_3)]+s_3\,[t_1(u_1+u_2+u_3)]\} \\
& +r_2\{s_1\,[t_2(u_2+u_3)+t_3(u_3)]+s_2\,[t_2(u_2+u_3)]\} && (5.1) \\
& +r_2\{s_1\,[t_3(u_3)]\} && (5.2)
\end{aligned}$$

This is again the lineage from Section 3.1. If follows from Lemma 5.2 that it is in fact a linear-inclusiong k-DNF.

The relations are already sorted ascendingly, so the orders are all the trivial identity order $\pi_R = \pi_S = \pi_T = \pi_U = \pi_{\mathbb{1}}$. The BDD B_ϕ which is returned by BDDLinIncl(Q, $\pi_{\mathbb{1}}, \pi_{\mathbb{1}}, \pi_{\mathbb{1}}, \pi_{\mathbb{1}}$) is shown in Figure 5.4. For simplicity, we have omitted the terminal nodes, but they can easily be inserted: All variable nodes without a dotted out-going edge are connected to 0 and all U-nodes are connected to 1. The BDD has 38 nodes, which is considerably smaller than the worst case size of about $2^{10} = 1024$ nodes.

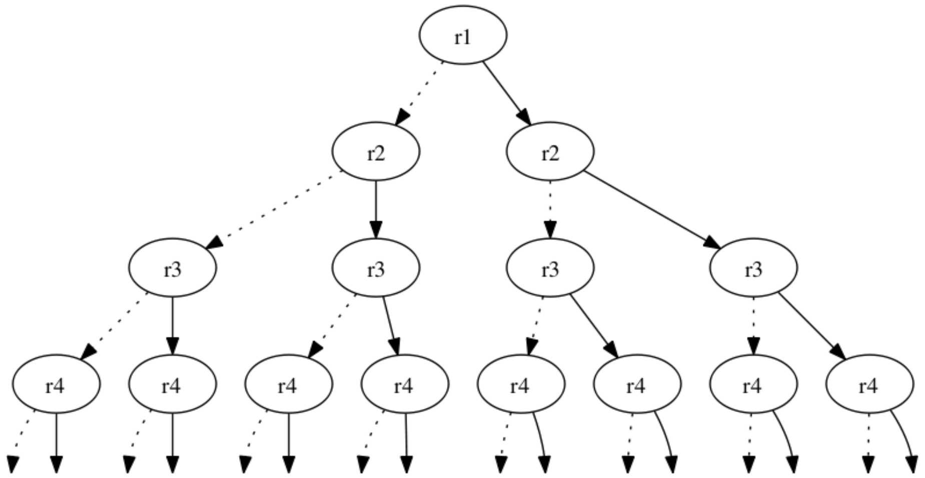


Figure 5.3: Exponentially sized BDD

Intuitively, a good variable generates a smaller BDD, because a lot of paths end in the 0-terminal node after only a few steps. This is dependent on inclusions of cofactors. Compare for example the cofactors of r_2 and r_3: The expression within the '{}'-brackets in line (5.2) is included in (5.1). Thus, if r_2 is true and its cofactor is false, we do not have to take r_3 into account any more and so r_3 does not appear on any path in after where r_2 is true. In contrast, if r_3 was eliminated before r_2, we would have to check the latter along paths for both truth values of r_3. This would for general cofactors and r_1 to r_n lead to an exponential number of r_i nodes. (Figure 5.3)

.

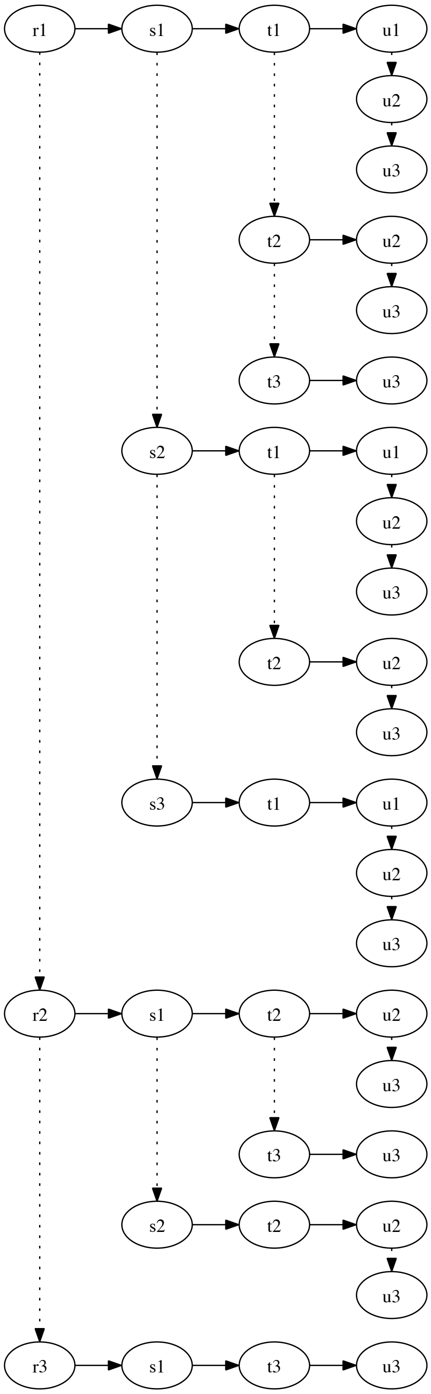


Figure 5.4: BDD for ϕ_Q, π_1

5.2 Single-Guard Query

Cyclic inequalities in conjunctive queries can be used to simulate equalities. Imagine for example two relations $R(A, B)$ and $S(C)$ and the query

$$Q : -R(A, B), S(C), A < C, C < B.$$

We call this a *guard* [16]. We will show in Chapter 6, that for every lineage created from an equality join we can construct a database instance such that this query creates the same lineage. Based upon the #P-results for equality queries this will imply that that queries with multiple guards can be hard [16, 6].

As the simplest case of a non-trivial acyclic query consider the query in Figure 5.5:

$$Q : -R(A, B), S(C), A < C, C < B.$$

We call this the *Single-Guard Query.* In this section we show that it is possible to solve this query in polynomial time.

Theorem 5.4 (Tractability of the Single-Guard Query)**.** *The Single-Guard Query*

$$Q : -R(A, B), S(C), A < C, C < B$$

can be evaluated on tuple-independent probabilistic databases in $O(|\phi_Q|^2)$ *using EvalQuery.*

To prove this theorem, we use the following lemma:

Lemma 5.5. *Let Q be the Single-Guard Query without self-joins. Then ϕ_Q is a cofactor-separable 2-DNF.*

The high-level intuition why the Single-Guard Query creates cofactor-separable 2-DNF is the following: We first sort the relation in R ascendingly first on A and then on B and rename the variables according to that order. We assume that the values in columns A, B and C are $a_1, a_2, \ldots$, $b_1, b_2, \ldots$ and $c_1, c_2, \ldots$, resp. The order for R implies

$$x_i < x_j \Leftrightarrow (a_i < a_j) \vee (a_i = a_j \wedge b_i < b_j).$$

Due to the structure of the inequality, the cofactor of each x_i will be a consequtive sum of y_j's, namely all y_j's with $a_i < c_j < b_i$. An example for this is shown in Figure 5.6.

The x_i's are evaluated from left to right. If an x_i is true, its cofactor is not included in an earlier one and the corresponding cofactor evaluates to false, the linage is separated into a left half (the history) and a right half, which is yet to be evaluated. Due to the order, no remaining cofactor can include y_j-variables from the left. Thus, the history of the path becomes irrelevant.

A rigorous mathematical proof of Lemma 5.5 is given in the Appendix.

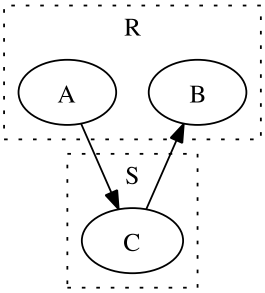


Figure 5.5: Single-Guard Query

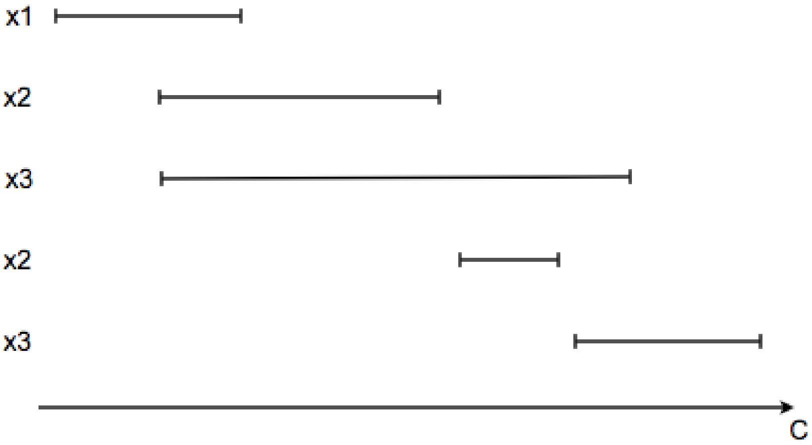


Figure 5.6: Example structure of the Single-Guard Query lineage

R	A	B	E
	0	3	x_1
	1	5	x_2
	1	7	x_3
	4	6	x_4
	5	8	x_5

S	C	E
	1	y_1
	2	y_2
	3	y_3
	4	y_4
	5	y_5
	6	y_6
	7	y_7

Figure 5.7: Example tuple-independent probabilistic database D for the Single-Guard Query

Proof of Theorem 5.4. It follows from the preceding proof that the ascending order π_R is a compatible order for Definition 3.12. The variable sets X and Y intersection-free, because $R \neq S$ and Q is evaluated on a tuple-independent probabilistic database. Thus, Lemma 5.5 together with Theorem 3.14 implies that BDDLinIncl(ϕ_Q, π_R,) returns in time $O(|\phi_Q|^2)$ a BDD B_{ϕ_Q} of size

$$|B_{\phi_Q}| \leq 3/8 \cdot |\phi_Q|^2 + |\phi_Q|/2.$$

This together with Proposition 4.3 concludes the proof. □

Example 5.6. *An example database D for the Single-Guard Query is shown in Figure 5.7. The query lineage on D is the 2-DNF already introduced in equation (3.17), which also proves that this 2-DNF is indeed cofactor-separable. The structure of the lineage is shown in Figure 5.6. The corresponding BDD was shown in Figure 3.11.*

5.3 2-Edge Query

Alternatively to extending the class of acyclic max-one queries with cycles, one can consider extensions towards subsets of queries without the max-one property. In general, this also leads to intractable queries, as will be shown in Chapter 6.

The simplest not max-one query

$$Q : -R(A, B), S(C), T(D), A < C, B < D$$

is similar to a max-one query if we use a reverse topological order for the subgoals. This can be shown by explicit calculation, but it is immediately clear because of the symmetry. Thus, the query shown in Figure 5.8 is probably the simplest acyclic query without the max-one property that cannot be evaluated with BDDLinIncl. We call it *2-Edge Query.* We assume that Q has no self-joins, i.e. the relations in R and S are different.

The main result of this section is:

Theorem 5.7 (Tractability of the 2-Edge Query)**.** *The 2-Edge Query*

$$Q : -R(A, B), S(C, D), A < C, B < D$$

can be evaluated on tuple-independent probabilistic databases in $O(|\phi_Q|^2)$ *using EvalQuery.*

As in the preceding section, the proof is based on showing that ϕ_Q is a cofactor-separable 2-DNF.

Lemma 5.8. *Let Q be the 2-Edge Query without self-joins. Then* ϕ_Q *is a cofactor-separable 2-DNF.*

Every cofactor of an x_i in the query lineage associated with Q consists of all y_j's with values $c_j > a_i$ and $d_j > b_i$. Thus, it can be visualized as a rectangle in a two dimensional diagram. (Figure 5.9) If we sort R ascendingly first on A and then on B, we encounter the cofactors top-to-bottom and right-to-left.

It is now easy to understand intuitively, why the cofactors separate history and future of a path σ: Let for σ with subset s_t at level i in BDDCofSep $f(x_i)$ be not included in s_t. Then after elimination of $f(x_i)$, every cofactor for $i' > i$ can only consist of variables which are either in the rectangle representing $f(x_i)$ or to the left of it. This is situation is shown in Figure 5.9.a for $i = 2$ and $i' = 3$. Figure 5.9.b shows the corresponding situation where $s_t \supseteq f(x_i)$. In this case, $f(x_i)$ is already false and not evaluated along σ.

A rigorous mathematical proof of Lemma 5.8 is given in the Appendix.

Proof of Theorem 5.7. The proof is essentially the same as in the last section for Theorem 5.4: It follows from the preceding proof that the ascending order π_R is a compatible for Definition 3.12. The variable sets X and Y

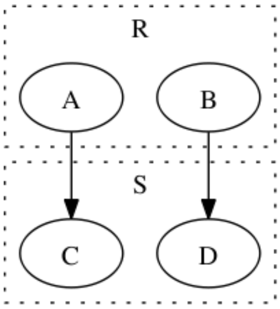
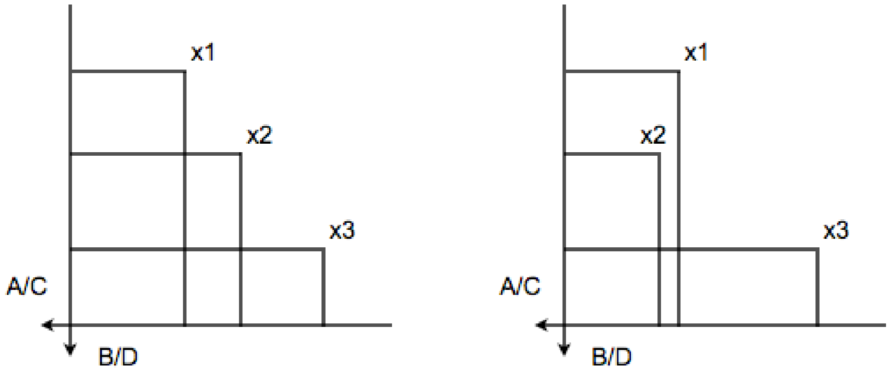


Figure 5.8: 2-Edge Query



a. b.

Figure 5.9: Possible tructures of the 2-Edge Query lineage

R	A	B	E
	1	4	x_1
	2	2	x_2
	2	5	x_3
	3	3	x_4
	4	1	x_5

S	C	D	E
	2	5	y_1
	3	3	y_2
	3	5	y_3
	3	6	y_4
	4	4	y_5
	4	5	y_6
	4	6	y_7
	5	2	y_8
	5	3	y_9
	5	4	y_{10}
	5	5	y_{11}
	5	6	y_{12}

Figure 5.10: Example tuple-independent probabilistic database D for the 2-Edge Query

are intersection-free, because Q contains no self-joins and is evaluated on a tuple-independent probabilistic database. Thus, Lemma 5.8 together with Proposition 3.14 implies that BDDLinIncl(ϕ_Q, π_R) returns in time $O(|\phi_Q|^2)$ a BDD B_{ϕ_Q} of size

$$|B_{\phi_Q}| \leq 3/8 \cdot |\phi_Q|^2 + |\phi_Q|/2.$$

This together with Proposition 4.3 concludes the proof. □

Example 5.9. *An example database D for the 2-Edge Query is shown in Figure 5.10. The corresponding query lineage is*

$$\begin{aligned}\phi_Q \quad = \quad & x_1(y_1 + y_3 + y_4 + y_6 + y_7 + y_{11} + y_{12}) \\ & +x_2(y_2 + \ldots + y_7 + y_9 + \ldots + y_{12}) + x_3(y_4 + y_7 + y_{12}) \\ & +x_4(y_5 + y_6 + y_7 + y_{10} + y_{11} + y_{12}) + x_5(y_8 + \ldots + y_{12}). \end{aligned} \quad (5.3)$$

The BDD returned by BDDLinIncl is shown in Figure 5.11. All Y-nodes are connected with their high edge to the 1-terminal node, which we have omitted for simplicity.

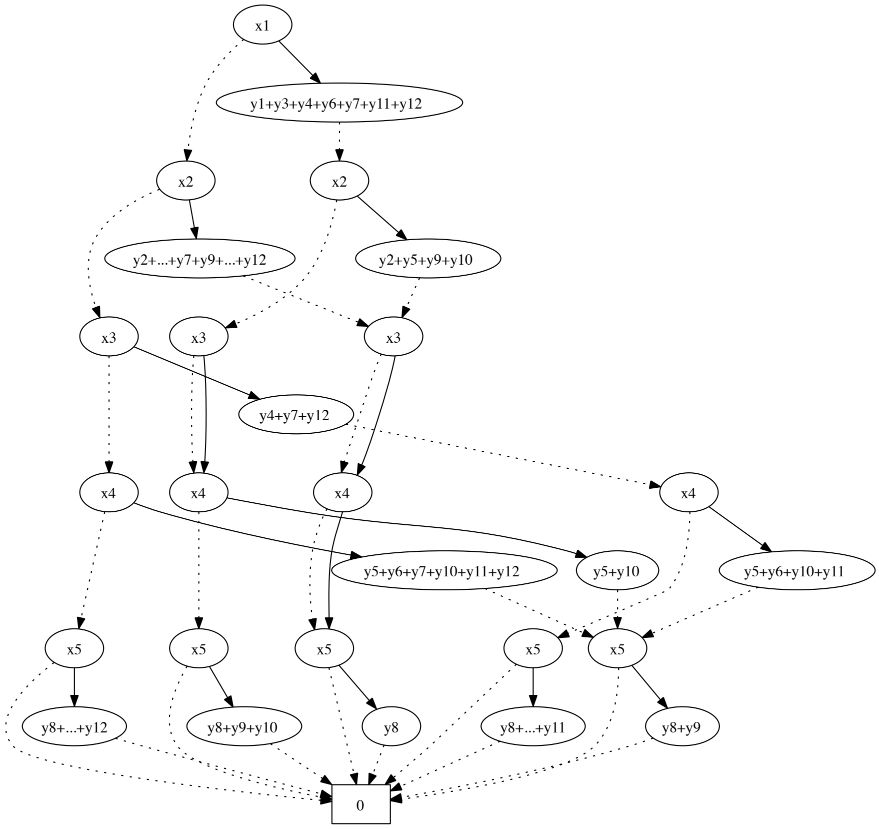


Figure 5.11: B_{ϕ_Q} for the 2-Edge Query with lineage (5.3)

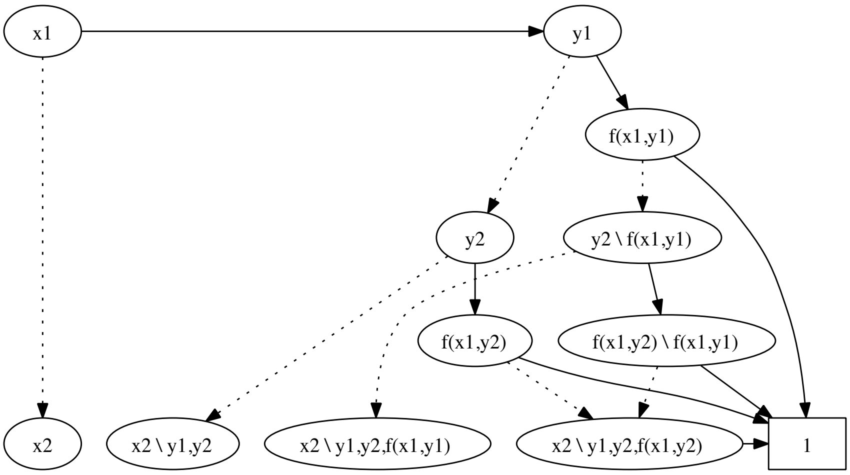


Figure 5.12: Partial structure for a generalization of BDDCofSep

5.4 Extensions

Having seen how certain cyclic and not max-one queries with two relations can be treated, it is natural to ask if and how these techniques can be extended towards such queries with multiple relations. In particular, is there a k-DNF class which is an obvious generalization of cofactor-separable 2-DNFs and can be treated in a similar way?

An approach towards an algorithm for more general k-DNFs is shown in Figure 5.12. On each level, we keep track of which relevant variables from the following levels are already set to false. Given a property similar to the cofactor-separability for 2-DNFs, this would result in a polynomial number of paths once the number of relations is fixed.

Unfortunately, the generalization is not quite so simple. Consider again Figure 5.12. What if y_1 appears in the cofactor of x_2 as well? Along the path through the $x_2 \setminus y_1, y_2, f(x_1, y_2)$-node, y_1 is actually true and not false. Now if $f(x_2, y_1) \not\subseteq f(x_1, y_1)$ the algorithm should test whether the former is true, but it will not, because y_1 is in the set of already falsified variables. Thus, B_ϕ would not correctly represent the k-DNF.

To avoid this problem, we could keep track of which variables are set true along a path. However, this is not feasible, because it would in general result in exponentially many paths. Instead, we need the implication $f(x_1, y_1) = 0 \Rightarrow f(x_i, y_1) = 0 \, \forall i \geq 1$, i.e. we need an additional property on the k-DNF, namely

$$f(x_1, y_j) \supseteq f(x_2, y_j) \supseteq f(x_3, y_j) \supseteq \dots \tag{5.4}$$

for all non-empty cofactors and all y_j. Similar inclusions have to hold on all levels.

Due to the inequalities in our queries, these inclusions in fact hold for many queries. This algorithmic approach should thus be able to answer queries like a Single-Guard Query or 2-Edge Query within an arbitrary acyclic max-one query graph. Additionally, it is likely that queries with multiple such components are tractable, if the components are not directly connected.

In conclusion, structural extensions towards more complicated query classes are likely possible with a generalization of cofactor-separability for k-DNFs which includes the inclusion property described above. Then an algorithmic technique similar to the one in BDDCofSep should result in a polynomial-time algorithm for those queries. They would in particular include combinations of the queries presented in the last three sections. However, the details of these extensions are not yet clear.

Alternatively to extensions of the query structure, one can also consider different connections between query variables. In our case this in particular includes inequalities of the kind $\leq$. In fact, the structure of the query lineage for our tractable queries is not changed significantly if these inequalities are used instead of or in connection with $<$. This gives

Proposition 5.10. *Acyclic max-one queries, the Single-Guard Query and the 2-Edge Query with inequalities ($<, \leq$) are tractable on tuple-independent probabilistic databases.*

Chapter 6

Hard Queries

6.1 Hard Queries with Equalities

For arbitrary Boolean conjunctive queries on tuple-independent probabilistic databases, the complexity of calculating the probability of the result can be very hard. In 2004, Dalvi and Suciu showed that query evaluation can #P-complete even for conjunctive queries with equalities and no self-joins [7]. #P is the complexity class of counting problems associated with decision problems in NP [28]. In particular, if a decision problem is in NP, then the problem of counting the number of positive answers is in #P.

The complexity class #P contains very difficult problems. In fact, NP $\subseteq$ #P, because a problem is solvable if and only if its number of solutions is larger or equal than 1. Furthermore, $P^{\#P}$ - the class of polynomial-time machines with a #P-oracle - includes the entire polynomial hierarchy PH [27].

As for other complexity classes such as NP, one can find #P-complete problems, which lie in #P and are as least as hard as any other problem in #P. The latter means that any problem in #P can be reduced to them in polynomial time, i.e. the solution can be computed in polynomial time from the answer of the #P-complete problem.

For example, consider monotone 2-disjunctive normal forms (monotone 2-DNF), i.e. Boolean formulas of the form $\phi = x_{i_1}y_{j_1} + x_{i_2}y_{j_2} + ... + x_{i_n}y_{j_n}$ over Boolean variables $x_1, ..., x_k,\ y_1, ..., y_l$. Deciding satisfiability (SAT) of a given monotone 2DNF is obviously trivial and hence in P. Somewhat counterintuitively, the counting version of this problem - i.e. the problem #SAT of counting the number of satisfying assignments for monotone 2-DNF - is not only in #P, but is in fact #P-complete [21]. Consequently, it is also at least as hard as any problem in NP. Other well-known #P-complete problems are the computation of the permanent of a 0,1-matrix and the computation of the number of perfect matchings in a bipartite graph [28, 29].

R	A	**E**
	1	x_1
	2	x_2
	3	x_3
	$\vdots$	$\vdots$

S	C	D	**E**
	1	2	z_1
	1	3	z_2
	2	1	z_3
	2	2	z_4
	3	3	z_5
	$\vdots$	$\vdots$	$\vdots$

T	B	**E**
	1	y_1
	2	y_2
	3	y_3
	$\vdots$	$\vdots$

$$\phi_Q(D) = x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_3y_3$$

Figure 6.1: Example of a #P-complete query with equalities

As it turns out, counting the number of satisfying assignments of monotone 2-DNFs and probabilistic databases are fundamentally related [7]: Consider the Boolean conjunctive query

$$Q : -R(A), S(C, D), T(B), A = C, D = B$$

on a tuple-independet probabilistic database D. Let S be a certain relation, i.e. all its random variables are true with probability one. If we omit the trivial random variables from S, which are semantically equal to 1, then this can create arbitrary monotone 2-DNF by pairing corresponding pairs of random variables from R and T. An illustration of this is given in Figure 6.1: The variables $x_1, x_2, \ldots$ in R and $y_1, y_2, \ldots$ in T are associated with tuple values $1, 2, \ldots$. To add the product x_iy_j to ϕ_Q, one adds a tuples with data values $S.C = i, S.D = j$ to S.

If R and T have a total of n random variables and all there probabilities are set to 1/2, then each assignment is equally likely and the probability of the query being true is

$$\begin{aligned} Pr(Q) &= Pr(\phi_Q) \\ &= \frac{\#(\text{satisfying assignments})}{\#(\text{assignments})} \\ &= \frac{\#(\text{satisfying assignments})}{2^n}. \end{aligned} \tag{6.1}$$

Thus, if we can evaluate the query - i.e. calculate $Pr(Q)$ - we can count the number of satisfying assignments. This reduces #SAT(monotone 2-DNF) is to query evaluation, which implies that the later is also #P-hard. This reasoning was originally used by Dalvi and Suciu as part of the proof of

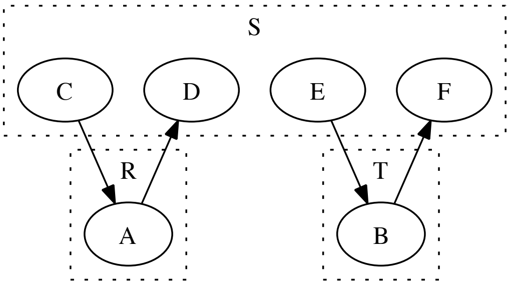
Theorem 6.1. *[7] Evaluation of Boolean conjunctive queries on tuple-independent probabilistic databases is in general #P-complete.*

In fact, Dalvi and Suciu were able to prove an even more fundamental result, namely

R	A	**E**
	1	x_1
	2	x_2
	3	x_3
	⋮	⋮

S	C	D	E	F	**E**
	0.9	1.1	1.9	2.1	z_1
	0.9	1.1	2.9	3.1	z_2
	1.9	2.1	0.9	1.1	z_3
	1.9	2.1	1.9	2.1	z_4
	2.9	3.1	2.9	3.1	z_5
	⋮	⋮	⋮	⋮	⋮

T	B	**E**
	1	y_1
	2	y_2
	3	y_3
	⋮	⋮



$$\phi_{Q'}(D) = x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_3y_3$$

Figure 6.2: Example of a #P-complete query with inequalities

Theorem 6.2. *[7, 6] Let Q be conjunctive queries with equalities on tuple-independent probabilistic databases. Then Q is either in PTIME or #P-complete.*

This implies a dichotomy for conjunctive queries with equalities on tuple-independent probabilistic databases. A query is either in PTIME or #P-complete, where the latter implies that it is fundamentally hard if $P \neq NP$. Furthermore, the original theorem also contains a characterization of all tractable and intractable queries:

Proposition 6.3. *[6] A Boolean conjunctive query with equalities and without self-joins is in PTIME if and only it is hierarchical. Otherwise it is #P-complete.*

Definition 6.4. *[6] A query Q is* hierarchical*, if for any two of its query variables either the sets of subgoals of this query variables are disjoint or one of the sets is included in the other.*

6.2 Hard Queries with Inequalities

The results above hold for conjunctive queries with equalities. However, they can easily be related to queries with inequalities by creating an example query that can be used to generate arbitrary monotone 2-DNF. Consider for example the query [16]

$$Q' : -R(A), S(C, D, E, F,), T(B), C < A, A < D, E < B, B < F. \quad (6.2)$$

This query consists of two Single-Guards (Section 5.2), but importantly they are connect within one relation. The chains of inequalities can be used to simulate equalities: We again associated x_i with i and y_j with j. To pair x_i and y_j, we add a tuple with values $S.C = i - 0.1, S.D = i + 0.1, S.E = j - 0.1$ and $S.F = j + 0.1$ to S. An example is shown in in Figure 6.2, which creates the same lineage as in Figure 6.1. Thus, Q' allows us to create arbitrary monotone 2-DNF.

Proposition 6.5. *Evaluation of Boolean conjunctive queries with inequalities and without self-joins on tuple-independent probabilistic databases is in general #P-complete.*

An important question one can ask is: What kind of properties are necessary so that conjunctive queries with inequalities become hard. Ideally, there would be a theorem similar to Theorem 6.2. So far, we know from Section 5.1 that all acyclic max-one queries are tractable. (Theorem 5.1) This implies that a hard query has to be at least cyclic or not max-one.

Unfortunately, the query (6.2) is cyclic and not max-one, so it does not give new insights towards the minimal requirements. In contrast, the following query is acyclic, not max-one and can generate arbitrary 2-DNFs:

$$Q'' : -R(A, B), S(C, D, E, F), T(G, H), C < A, D < B, E < G, H < F.$$

To proof that $\phi_{Q''}$ can be an arbitrary 2-DNF, we use the following database: The variables x_i in R and y_i in T are associated with data tuples $(i, -i)$, he variables in S are certain. To add a product $x_i y_j$ to the query lineage, we insert a data tuple $(i - 0.1, -i - 0.1, j - 0.1, -j - 0.1)$ into S. (Figure 6.3). Out of all R-variables, this tuple will only be paired with x_i, because $(i, -i, x_i)$ is the only tuple from R with $i - 0.1 < R.A$ and $-i - 0.1 < R.B$. For every other integer $i' \neq i$ we have either $i - 0.1 > i'$ (for $i' < i$) or $-i - 0.1 > -i'$ (for $i' > i$). The same reasoning holds the variables from T.

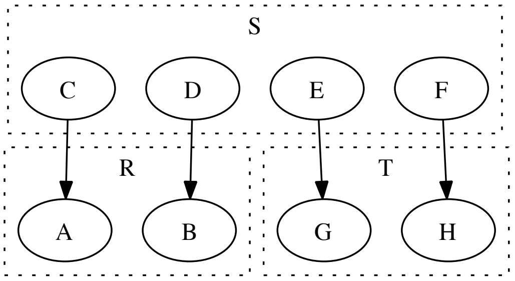
Proposition 6.6. *The query Q'' can create arbitrary 2-DNFs over tuple-independent databases.*

Q'' is related to the 2-Edge Query in Section 5.3 in the same way Q' is related to the Single-Guard Query. Both of them can be answered in

R	A	B	**E**
	1	-1	x_1
	2	-2	x_2
	3	-3	x_3
	⋮	⋮	

S	C	D	E	F	**E**
	0.9	-1.1	1.9	-2.1	z_1
	0.9	-1.1	2.9	-3.1	z_2
	1.9	-2.1	0.9	-1.1	z_3
	1.9	-2.1	1.9	-2.1	z_4
	2.9	-3.1	2.9	-3.1	z_5
	⋮	⋮	⋮	⋮	⋮

T	B	C	**E**
	1	-1	y_1
	2	-2	y_2
	3	-3	y_3
	⋮	⋮	



$$\phi_{Q''}(D) = x_1y_2 + x_1y_3 + x_2y_1 + x_2y_2 + x_3y_3$$

Figure 6.3: Example of an acyclic #P-complete query with inequalities

PTIME on there own, but not if two are connected within one relation. It also follows that a combined version

$$Q''' : -R(A,B), S(C,D,E,F), T(G), C < A, D < B, E < G, G < F.$$

is in general #P-complete. In the next section we additionally show that the 2-Edge Query strictly subsumes the Single-Guard Query in the sense that every lineage generated by the latter can also be generated using the former, but not the other way round (Theorem 7.13).

Interestingly, no hard max-one query is currently known. Furthermore, the discussion in Section 5.4 implies that every query with a combination of cofactor-separability and sufficient inclusions of cofactor should be tractable. Restricting the connections between subgoals to obey the max-one property should be enough to assure that all the necessary order are compatible. This leads us to believe that all max-one queries are tractable:

Conjecture 6.7. *All max-one queries without self-joins are tractable.*

It must be stressed that this is just a conjecture, as the algorithms presented in Chapter 5 are not able to evaluate all max-one queries in PTIME and the ideas of Section 5.4 have yet to be turned into provably correct algorithms. Nonetheless, Conjecture 6.7 provides an interesting question for further research.

As is for example proven by the tractability of the 2-Edge Query, the converse of Conjecture 6.7 - i.e. that all not max-one queries are intractable - is not true. Thus, concerning hard queries we can state:

Proposition 6.8. *Let Q be an intractable conjunctive query with inequalities and without self-joins over tuple-independent probabilistic databases. Then Q is cyclic, not max-one or both.*

Conjecture 6.9. *Let Q be an intractable conjunctive query with inequalities and without self-joins over tuple-independent probabilistic databases. Then Q is not max-one.*

For comparison, we mention an alternative approach towards characterizing hard queries, which is based on guards and equivalence classes of query variables [16]. A pair X, Y of variables guards another variable Z if the interval condition $X < Z < Y$ holds and X, Y appear in the same subgoal or are connected in a structurally equivalent way. The equivalence class of a query variable is then given by the query variable itself, its guards and all variables sharing guards with it.

It is now possible to extend the concept of *hierarchical queries*, which is used in the concept of equality queries [6], to inequalities queries: A query is hierarchical if the sets of subgoals of any two equivalence classes of its query variables are either disjoint or one is included in the other [16]. Non-hierarchical queries can then be used to create arbitrary monotone 2-DNF, which immediately gives

Proposition 6.10. *[16] Non-hierarchical queries are #P-complete.*

This approach does not capture the fact that queries with multiple 2-Edge Queries can be #P-complete, as was shown above. A possible combination of both techniques should include a more generalized concept of equivalence classes, such that Proposition 6.10 includes such queries.

Chapter 7

Counting Vertex Covers in Graphs

7.1 Introduction

In the last chapter we showed how the problem #SAT(monotone bipartite 2-DNF) of counting the number of satisfying assignments of monotone bipartite 2-DNFs - monotone multipartite k-DNFs with $k = 2$ - can be reduced to query-evaluation on tuple-independent probabilistic databases. Counting the number of satisfying assignments is an interesting problem in its own right, but it is also related to important problems on graphs: Let $p \propto p'$ denote that a problem p is polynomial-time reducible to a problem p', i.e. the answer of p can be computed from the answer of p' in polynomial time. If $p \propto p'$ and $p' \propto p$ we write $p \simeq p'$. Provan and Ball showed [21]

Theorem 7.1. *[21]*

$$\begin{aligned}\#VC(\textit{general graphs}) &\propto \#VC(\textit{bipartite graphs})\\ &\propto \#SAT(\textit{monotone bipartite 2-DNF}),\end{aligned}$$

where #VC is the problem of counting the number of vertex covers on a graph.

Definition 7.2 (Vertex Cover)**.** *Let $G = (V, E)$ be an (undirected) graph. $S \subseteq V$ is a* vertex cover *of G if for each edge $e \in E$ at least one of endpoints of e is in S.*

Counting vertex covers on general graphs is #P-complete [9]. Thus, Theorem 7.1 proofs that #VC(bipartite grahps) and #SAT(monotone bipartite 2-DNF) are #P-complete as well [21]. This also implies that the reductions in Theorem 7.1 hold both ways, i.e. the problems are polynomial-time equivalent (which we denote by '$\simeq$').

The connection between bipartite graphs and #SAT is as follows [21]: Given a bipartite graph $G = (X, Y, E)$, create a monotone bipartite 2-CNF

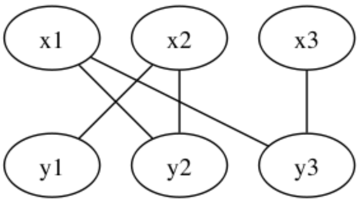


Figure 7.1: Bipartite graph for Example 7.3

(conjunctive normal form) ϕ with random variables for X and Y and $x_i \vee y_j \in \phi$ if and only if (x_i, y_j) is an edge in G. Then $S \subseteq X \cup Y$ is a vertex cover of G exactly if the assignment $\sigma : X \cup Y \to \{0,1\}$ with $\sigma(z) = 1 \Leftrightarrow z \in S$ is in $\mathrm{SAT}(\phi)$.

ϕ is a monotone bipartite 2-CNF, but we can easily create a related monotone bipartite 2-DNF ϕ', if we exchange conjunctions and disjunctions. Then $\sigma \in \mathrm{SAT}(\phi)$ if and only if $\neg\sigma \notin \mathrm{SAT}(\phi')$ (i.e. $\neg\sigma$ is in the set $\mathrm{NSAT}(\phi')$ of non-satisfying assignments of ϕ'). Since the number of all total truth assignment of $|X \cup Y|$ Boolean variables is $2^{|X \cup Y|}$, we can easily relate #SAT and #NSAT.

From now on, let $G(\phi)$ denote the bipartite graph associated with a monotone bipartite 2-DNF ϕ with variable sets X and Y. It is given by $G(\phi) = (X, Y, E)$ with $(x_i, y_j) \in E$ if and only if $x_i y_j \in \phi$.

Example 7.3. *The graph $G(\phi)$ for the 2-DNF*

$$\phi = x_1 y_2 + x_1 y_3 + x_2 y_1 + x_2 y_2 + x_3 y_3$$

from Example 2.27 is shown in Figure 7.1.

Before we state our results about #VC on graph classes, we introduce another kind of subsets of graph vertices that is closely related to vertex covers:

Definition 7.4 (Independent Set)**.** *Let $G = (V, E)$ be an (undirected) graph. $S \subseteq V$ is an* independent set *of G, if no two vertices from S are connected by an edge $e \in E$.*

It is easy to see that S is an independent set if and only if $V \setminus S$ is a vertex cover. This implies that there is a bijective mapping between independent sets and vertex covers and hence the corresponding counting problems are equivalent. In conclusion, we get

Proposition 7.5.

$$\begin{aligned} &\#\mathit{VC(general\ graphs)} \simeq \#\mathit{VC(bipartite\ graphs)} \\ &\simeq \#\mathit{SAT(monotone\ bipartite\ 2\text{-}DNF)} \\ &\simeq \#\mathit{IS(general\ graphs)} \simeq \#\mathit{IS(bipartite\ graphs)}. \end{aligned}$$

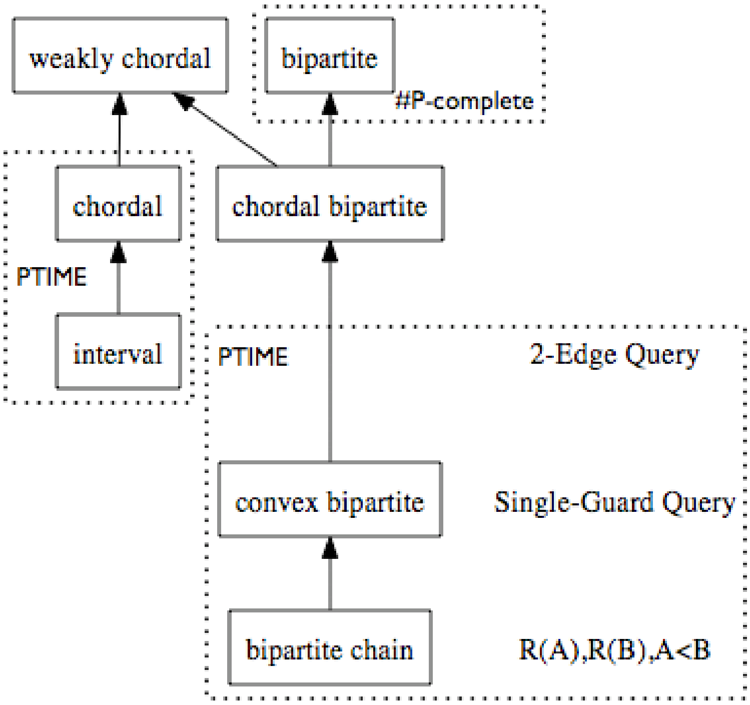


Figure 7.2: Complexity of #VC for related graph and query classes

All these problems are in general #P-complete and so their is little hope of finding a polynomial time algorithm to solve there unrestricted version. Consequently, efforts have been made to find solutions for restricted sub-problems. For example, #VC/#IS is known to be in linear time for chordal graphs and interval graphs [13, 14]. In contrast, it has been shown to be #P-hard even for very restricted graphs such as 3-regular bipartite planar graphs [32].

Common approaches to exact model counting for SAT make use of bounded structural graph parameters such as (hyper)tree-width, clique-width or existence of "backdoors" [10]. To our knowledge, none of these existing restrictions are applicable to our DNFs.

7.2 Main Results

Based on our results from the earlier chapter we are now able to proof tractability for *bipartite chain* and *convex bipartite* graphs, which to the knowledge of the author has not been shown before:

Definition 7.6 (Bipartite Chain Graph)**.** *[18] A bipartite graph $G = (X, Y, E)$ is a bipartite chain graph if and only if for both X and Y the neighbourhoods of the nodes can be ordered linearly w.r.t. inclusion (subset or equal).*

Theorem 7.7. *$\#VC$ is tractable for bipartite chain graphs.*

Definition 7.8 (Convex Bipartite Graph)**.** *[20] A bipartite graph $G = (X, Y, E)$ is convex if there is an ordering $<$ of X or Y such that the neighbourhoods of the vertices in the other set are consecutive in the ordering $<$ (i.e. they are intervals).*

Theorem 7.9. *$\#VC$ is tractable for convex bipartite graphs.*

These theorems are based on the fact that these classes of bipartite graphs are captured by the lineage generated by tractable queries. The overall situation is visualized in Figure 7.2. Note that Theorem 7.9 already implies Theorem 7.7. We have stated them separately, because they are related to different queries.

Lemma 7.10. *Let $Q : -R(A), S(B), A < B$ without self-joins. Then*

$$\{G(\phi_{Q(D)})|D \textit{ t-i prob database}\} = \{G|G \textit{ bipartite chain graph}\} \quad (7.1)$$

Lemma 7.11. *Let $Q : -R(A, B), S(C), A < C, C < B$ without self-joins. Then*

$$\{G(\phi_{Q(D)})|D \textit{ t-i prob database}\} = \{G|G \textit{ convex bipartite graph}\} \quad (7.2)$$

Definition 7.12 (Chordal Bipartite Graph)**.** *[19] A bipartite graph is a chordal bipartite graph if it has no chordles cycle of length ≥ 6.*

Theorem 7.13. *Let $Q : -R(A, B), S(C, D), A < C, B < D$ without self-joins. Then*

$$\{G|G \textit{ convex bipartite graph}\} \subset \{G(\phi_{Q(D)})|D \textit{ t-i prob database}\} \quad (7.3)$$

and

$$\{G(\phi_{Q(D)})|D \textit{ t-i prob database}\} \subset \{G|G \textit{ chordal bipartite graph}\} \quad (7.4)$$

Here, "t-i prob" stands for "tuple-independent probabilistic". The lemmas are proven in the Appendix.

It follows from the proofs of Lemma 7.10 and 7.11 that for any bipartite chain or convex bipartite graph we can create a tuple-independent probabilistic database which generates an equivalent query lineage. Even simpler, we can find the necessary order for BDDLinIncl/BDDCofSep by sorting the variables according to the neighbourhoods. This concludes the proof of Theorems 7.7 and 7.9.

Chapter 8

Conclusion

8.1 Summary

This project was concerned with exploring the questions of tractability for Boolean conjunctive queries with inequalities ($<$) on tuple-independent probabilistic databases. We reformulated the query evaluation problem as a BDD compilation problem for Boolean formulas in disjunctive normal form and identified two important classes of such formulas - linear-inclusion k-DNFs and cofactor-separable 2-DNFs. Using their structural properties, we derived polynomial-time algorithms for these classes using in particular a new technique which keeps track of a polynomial number of parallel paths in a BDD.

By showing that the k-DNF classes include the query lineages of acyclic max-one queries and important cyclic and non max-one queries without self-joins, we proved tractability for these queries via the efficient BDD compilation algorithms. This considerably extends what is known about tractability of queries with inequalities and in particular subsumes all earlier results. The results also hold for smaller-than-or-equal inequalities ($\leq$). For the case of hard queries, we showed #P-completness for a new class of queries and relate our work to previous approaches.

Finally, we applied the BDD compilation algorithms to counting problems on graphs. We proved that counting vertex covers and counting independent sets is in PTIME for convex bipartite and bipartite chain graphs, which has to our best knowledge not been shown before. This also demonstrates how our results about BDD construction for k-DNFs can be used outside the field of database theory.

8.2 Open Problems and Future Work

Even though we were able to considerably extend previous tractability results as well as show hardness for new kinds of queries, there are still a lot of open questions. For example, there are many queries between acyclic max-one queries and currently known hard ones for which the complexity is unclear. A particular interesting starting point for further research would be the question of whether all max-one queries are in fact tractable (Conjecture 6.7). This could possible be investigated base on the ideas present in Section 5.4 about possible extensions of our BDD compilation algorithms.

Further interesting questions include the case of self-joins and combinations of equality and inequality queries. It would be particularly helpful to find a more unified framework, possibly based on extensions of the definition of hierarchical queries and equivalence classes of query variables. Connecting these ideas with the ordering and inclusion properties for our k-DNF classes could lead to new algorithms as well as a better understanding of what is structurally necessary to make a query hard. Ultimately, one should aim at a complete tractability classification of conjunctive queries with inequalities.

Chapter 9

Appendix

Lemma 3.16. *The graph B_ϕ returned by BDDCofSep(ϕ, π_X) is a BDD.*

Proof. Due to the way B_ϕ is constructed, each total truth assignment $\sigma : Vars(\phi) \to \{0,1\}$ belongs to exactly one path in B_ϕ and there are no loose edges and circles, which implies that every path ends in either the 0-terminal or the 1-terminal node. The x_1-node is the only node with no in-going edges, i.e. B_ϕ is rooted in x_1.

We now need to show that along any path in B_ϕ every Boolean variable is set at most once. This is clearly true for the variables in X. For the Y-variables, assume the contrary, i.e. a y_k variable is set along a path for an assignment σ at a level i, even though it was set before at a level $i' < i$. Mathematically, $\exists 1 \leq k \leq |Y|, 1 \leq i' < i \leq |X|, 1 \leq t, t'$:

$$y_k \in f(x_i') \setminus s_{t'}, f(x_i) \setminus s_t \tag{3.1}$$

and $\exists \sigma : Vars(\phi[1,i]) \to \{0,1\}$:

$$\sigma(x_{i'}) = \sigma(x_i) = 1, \tag{3.2}$$

$$\sigma(\phi[1,i-1]) = 0, \tag{3.3}$$

$$f(x_{i'}) \setminus f(\sigma, i'-1) \neq \emptyset, f(x_i) \setminus f(\sigma, i-1) \neq \emptyset \tag{3.4}$$

where $s_t, s_{t'}$ are the subsets of already falsified variables along σ at level i, i'. Since the subset $s_{i'}$ created at level i' contains y_k, but $y_k \notin s_t$ it holds $s_{i'} \neq s_t$ and there has to exist at least one level i'' between i' and i at which y_k is dropped from the subset of already falsified variables. Mathematically, $\exists i' < i'' < i$ with

$$\sigma(x_{i''}) = 1, f(x_{i''}) \setminus f(\sigma, i''-1) \neq \emptyset \tag{3.5}$$

and

$$y_k \notin f(x_{i''}). \tag{3.6}$$

Let now $\overline{\sigma} = \sigma|_{Vars(\phi[1,i'])}$ and $\sigma' : Vars(\phi[1, i'']) \rightarrow \{0, 1\}$ with $\sigma'(x_1) = \ldots = \sigma'(x_{i''-1}) = 0$, $\sigma'(x_i'') = 1$. Then $\overline{\sigma}, \sigma'$ fulfill the requirements in 3.12 at level i'', but

$$x_i \cdot y_j \notin \overline{\sigma}(\phi[i'' + 1, |X|]), x_i \cdot y_j \in \sigma(\phi[i'' + 1, |X|) \tag{3.7}$$
$$\Rightarrow \overline{\sigma}(\phi[i'' + 1, |X|]) \neq \sigma(\phi[i'' + 1, |X|]). \tag{3.8}$$

This contradicts the condition in equation (3.16). Thus, no variable is set twice and B_ϕ is a valid BDD. □

Lemma 5.2. *Let Q be an acyclic max-one query without self-joins. Then ϕ_Q is a linear-inclusion k-DNF.*

Proof. W.l.o.g. Q can be written as

$$Q : -R_1\left(X_1^1, ..., X_{N_1}^1, \overline{X}^1\right), ..., R_k\left(X_1^k, ..., X_{N_k}^k, \overline{X}^k\right), \Lambda, \Lambda' \tag{5.9}$$

where $\overline{X}^1$, ..., $\overline{X}^k$ are the lists of query variables not participating in inequalities between relations and Λ and Λ' are conjunctions of inequalities over the query variables between and within relations, resp. Only the first column of each relation participates in out-going inequalities.
The relations R_i, $1 \leq i \leq N$, can be written as

$$R_i = \left\{(v_{1,1}^i, ..., v_{1,N_i}^i, \overline{v}_1, x_1^i), ..., (v_{n_i,1}^i, ..., v_{n_i,N_i}^i, \overline{v}_{n_i}, x_{n_i}^i)\right\} \tag{5.10}$$

where the $v_{j,k}^i$ and $\overline{v}_j^i$ are the data entries of column X_k^i and columns $\overline{X}^i$, resp., and the x_j^i are the corresponding random variables in column $R_i.\mathbf{E}$. We will subsequently drop the $\overline{X}^i$ and $\overline{v}_j^i$ variables as well as Λ', because they are trivially dealt with during pre-processing. (Figure 4, step 1)

Since ϕ_Q is a query lineage from a query without self-joins, it is a multipartite k-DNF. (Proposition 2.20) To show that ϕ_Q is a linear-inclusion k-DNF, we first choose a *topological* order of the subgoals in the query graph, i.e. a order where a path from a subgoals R_i to a subgoals R_j implies that R_i comes before R_j. This also determines the literal order in the conjunctions in ϕ_Q.

Such an order is always possible and can easily be found, because Q is acyclic. We will write $R_i \rightarrow R_j$ to denote that a subgoals R_i occurs before an subgoals R_j. W.l.o.g. we can rename the subgoals such that the order is $R_1 \rightarrow \ldots \rightarrow R_k$.

For each of the relations R_i, we order its random variables based on a ascending order of the data values in the first column. W.l.o.g. we rename the tuples such that the resulting order is $\pi_i = 1, 2, \ldots, n_i$.

Let now $1 \leq l \leq k$ and $1 \leq i_1 \leq n_1, ..., 1 \leq i_{l-1} \leq n_{l-1}$ be arbitrary. We can partition Λ into two conjunctions Λ_1 and Λ_2 of inequalities which contain and do not contain query variables from R_l, resp.

Let $1 \leq i_l \leq n_l, ..., 1 \leq i_k \leq n_k$ be arbitrary. If the product $x^1_{i_1} \cdot ... \cdot x^l_i \cdot ... \cdot x^k_{i_k}$ is in the lineage - i.e. $(i_1, ..., i_l, ..., i_k) \in I$ - then the corresponding column values fullfill all inequalities. Thus, both Λ_1 and Λ_2 are satisfied over the values $(v^1_{i_1,1}, \dots, v^1_{i_1,N_1}), \dots, (v^l_{i_l,1}, \dots, v^l_{i_l,N_l}), \dots, (v^k_{i_k,1}, \dots, v^k_{i_k,N_k})$.

Consider now an arbitrary $1 \leq i'_l < i$ and $f(x^1_{i_1} \dots x^{l-1}_{i_{l-1}} x^l_{i'_l}) \neq \emptyset$. Since the values for all columns other than X^l are not changed, Λ_2 is fulfilled over $(v^1_{i_1,1}, \dots, v^1_{i_1,N_1}), \dots, (v^l_{i'_l,1}, \dots, v^l_{i'_l,N_l}), \dots, (v^k_{i_k}, \dots, v^k_{i_k,N_k})$.

Because of the topological order and the max-one property, the out-going inequalities from relation R_l in Λ_1 are all from column 1. Therefore, all inequalities in Λ_1 are out-going inequalities from X^l_1 or in-going inequalities from relations with index $1 \leq l' < l$, i.e. of the form $X^{l'}_{m'} < X^l_m$ $(1 \leq m \leq n_l, 1 \leq m' \leq n_{l'})$.

If $f(x^1_{i_1} \dots x^{K-1}_{i_{K-1}} x^K_{i'_l}) \neq 0$ then all in-going inequalities of Λ_1 are fulfilled over the the values $(v^1_{i_1,1}, \dots, v^1_{i_1,N_1}), \dots, (v^l_{i'_l,1}, \dots, v^l_{i'_l,N_l})$. Furthermore,

$$v^l_{i'_l,1} < v^l_{i_l,1} < v^{l'}_{i_{l'},m'} \forall l' \geq l, 1 \leq m' \leq N_{l'}. \tag{5.11}$$

Thus, all out-going inequalities from Λ_1 are also satisfied. Since i_l and i'_l were arbitrary it follows

$$f(x^1_{i_1} \dots x^{K-1}_{i_{K-1}} x^K_{i_l}) \subseteq f(x^1_{i_1} \dots x^{K-1}_{i_{K-1}} x^K_{i'_l}) \tag{5.12}$$

for arbitrary $1 \leq l \leq k$, $1 \leq i'_l < i_l \leq n_l$ and $1 \leq i_1 \leq n_1, ..., 1 \leq i_{l-1} \leq n_{l-1}$ for which $f(x^1_{i_1} \dots x^{l-1}_{i_{l-1}} x^l_{i'_l}) \neq 0$. Thus, ϕ_Q is a linear-inclusion k-DNF. □

Lemma 5.5. *Let Q be the Single-Guard Query without self-joins. Then ϕ_Q is a cofactor-separable 2-DNF.*

Proof. As explained above we first sort relations R ascendingly and rename the variables accordingly. This allows to use the trivial orders $\pi_R = \pi_{\mathbb{1}}$. For simplicity of notation we also assume that S is sorted ascendingly.

Let now σ, σ' be partial truth assignment which fulfill the properties (3.13) to (3.15) in Definition 3.12 for a level $1 \leq i \leq |X|$. We have to show that

$$\sigma(\phi[i+1, n]) = \sigma'(\phi[i+1, n]). \tag{5.13}$$

Assume the contrary, i.e.

$$\sigma(\phi[i+1, n]) = \phi[i+1, n] \backslash f(\sigma, i) \neq \phi[i+1, n] \backslash f(\sigma', i) = \sigma'(\phi[i+1, n]) \tag{5.14}$$

which is equivalent to

$$\phi[i+1, n] \cap f(\sigma, i) \neq \phi[i+1, n] \cap f(\sigma', i). \tag{5.15}$$

Then w.l.o.g. $\exists 1 \leq j \leq |Y|$ such that $y_j \in \phi[i+1, n], f(\sigma, i)$, $y_j \notin f(\sigma', i)$. We will soon show that

$$\nexists y_k \in Vars(f(\sigma, i) : c_k > b_i. \tag{5.16}$$

Also $c_k \notin (a_i, b_i)$, since $y_j \notin f(\sigma', i) \supseteq f(x_i)$. Thus, $c_j < a_i$ and $c_j \notin f(x_k) \forall k \geq i$. This is a contradiction to $c_j \in \phi[i+1, n]$.

To prove (5.16), assume that such a y_k existed. Then $\exists k' < i$ such that $\sigma(x_{k'}) = 1, y_k \in f(x_{k'})$, which implies $a_{k'} < c_k < b_{k'}$. The order on relation R assures $a_{k'} \leq a_i$. Also, $a_i < b_i$, because otherwise x_i would have an empty cofactor and thus be irrelevant. This gives

$$\begin{aligned}
&a_{k'} \leq a_i < b_i < c_k < b_{k'} \\
&\Rightarrow f(x_{k'}) \supseteq f(x_i) \\
&\Rightarrow f(\sigma, i-1) \supseteq f(x_{k'}) \supseteq f(x_i) \\
&\Rightarrow f(x_i) \setminus f(\sigma, i-1) = \emptyset,
\end{aligned}$$

which contradicts equation (3.15). □

Lemma 5.8. *Let Q be the 2-Edge Query without self-joins. Then ϕ_Q is a cofactor-separable 2-DNF.*

Proof. As in the proof of Lemma 5.5 we assume that R is sorted ascendingly first on A and then on B. Let $1 \leq i \leq |X|$ and $\sigma, \sigma' : Vars(\phi[1, i]) \to \{0, 1\}$ be partial truth assignments that fulfill properties (3.13) to (3.15) in Definition 3.12 for level i. Assume that

$$\phi[i+1, |X|] \cap f(\sigma, i) \neq \phi[i+1, |X|] \cap f(\sigma', i). \tag{5.17}$$

W.l.o.g. this again implies $\exists 1 \leq j \leq |Y|$ such that $y_j \in \phi[i+1, |X|], f(\sigma, i)$, $y_j \notin f(\sigma', i), f(x_i)$. Then $\exists j' \geq i+1 : y_j \in f(x_{j'})$. Because of the order on R we have either $a_i < a_{j'}$ or $a_i = a_{j'} \wedge b_i < b_{j'}$. The latter would imply $f(x_i) \supseteq f(x'_j)$ which contradicts $y_j \notin f(x_i)$. Thus $a_i < a_{j'} < c_j$ and $b_{j'} < d_j < b_i$ has to hold.

Since $y_j \in f(\sigma, i)$ there exists an $m < i$ such that $y_j \in f(x_m)$ and $\sigma(x_m) = 1$. This leads to

$$a_m < a_i < a_{j'} < c_j,\ b_m < d_j \text{ and } b_{j'} < d_j < b_i \tag{5.18}$$

which implies

$$a_m < a_i, b_m < b_i \Rightarrow f(x_m) \supseteq f(x_i). \tag{5.19}$$

This contradicts $f(x_i) \setminus f(\sigma, i) \neq \emptyset$, because $\sigma(x_m) = 1$ gives $f(\sigma, i) \supseteq f(x_m)$ and thus $f(x_i) \setminus f(\sigma, i) \subseteq f(x_i) \setminus f(x_m) = \emptyset$. □

Lemma 7.10. *Let $Q : -R(A), S(B), A < B$ without self-joins. Then*

$$\{G(\phi_{Q(D)})|D \text{ t-i prob database}\} = \{G|G \text{ bipartite chain graph}\} \quad (7.20)$$

Proof.
"$\subseteq$": Enumerating the variables in R and S ascendingly implies that $N(x_1) \supseteq N(x_2) \supseteq \dots$ and $N(y_1) \subseteq N(y_2) \subseteq \dots$. Thus, $G(\phi_{Q(D)})$ is a bipartite chain graph.
"$\supseteq$": Let $G = (X, Y, E)$ be a bipartite chain graph. Then there exists an enumeration for X and Y such that $N(x_1) \supseteq N(x_2) \supseteq \dots$ and $N(y_1) \supseteq N(y_2) \supseteq \dots$. Now create a database D with relations $R(A, \mathbf{E})$ and $S(B, \mathbf{E})$ according to the following scheme: For each $x_i \in X$ add a tuple (i, x_i) to R. For each $y_j \in Y$ add a tuple $(k + 0.1, y_j)$ to S, where k is the maximum i such that $y_j \in N(x_i)$.

For arbitrary i, j, the product x_iy_j is in $\phi_Q(D)$ if and only if $i < k+0-1 = max_i(y_j \in N(x_i)) + 0.1$. Thus, $x_iy_j \in \phi_{Q(D)}$ if and only if $\exists k \geq i$ such that $y_j \in N(x_k)$. Because of $N(x_1) \supseteq N(x_2) \supseteq \dots$ this implies that $x_iy_j \in \phi_Q$ if and only if $y_j \in N(x_i)$. Thus, an edge (i, j) is in $G(\phi_{Q(D)})$ if and only if (i, j) is in E. □

Lemma 7.11. *Let $Q : -R(A, B), S(C), A < C, C < B$ without self-joins. Then*

$$\{G(\phi_{Q(D)})|D \text{ t-i prob database}\} = \{G|G \text{ convex bipartite graph}\} \quad (7.21)$$

Proof.
"$\subseteq$": A variable x_i from a tuple (a_i, b_i, x_i) is paired with a variable y_j from a tuple (c_j, y_j) if and only if $a_i < c_j < b_i$. Thus, if we enumerate the Y-variables ascendingly, the X-neighborhoods are sets of consecutive y_j's ("intervals").
"$\supseteq$": W.l.o.g. let Y be the variable set for which an order exists such that the $N(x_i)$ are sets of consecutive y_j's. Then for each x_i there exist $1 \leq s_i \leq t_i \leq |Y|$ such that $N(x_i) = \{y_{s_i}, \dots, y_{t_i}\}$. Now create a database D with relations $R(A, B, \mathbf{E})$ and $S(C, \mathbf{E})$ according to the following scheme: For each $x_i \in X$ add a tuple $(s_i - 0.1, t_i + 0.1, x_i)$ to R. For each $y_j \in Y$ add a tuple (j, y_j) to S. Then $x_iy_j \in \phi_{Q(D)}$ if and only if $y_j \in N(x_i)$. □

Theorem 7.13. *Let $Q : -R(A, B), S(C, D), A < C, B < D$ without self-joins. Then*

$$\{G|G \text{ convex bipartite graph}\} \subset \{G(\phi_{Q(D)})|D \text{ t-i prob database}\} \quad (7.22)$$

and

$$\{G(\phi_{Q(D)})|D \text{ t-i prob database}\} \subset \{G|G \text{ chordal bipartite graph}\} \quad (7.23)$$

R	A	B	E
	1	3	x_1
	2	2	x_2
	3	1	x_3

S	C	D	E
	1.1	3.1	y_1
	2.1	3.1	y_2
	2.1	2.1	y_3
	3.1	2.1	y_4
	3.1	1.1	y_5
	3.1	3.1	y_6

Figure 7.1: Tuple-independent probabilistic database D to show "convex bipartite $\subset$ 2-Edge Query"

Proof.
"convex bipartite $\subseteq$ 2-Edge Query": We already now from Lemma 7.11 that for each convex bipartite graph there exists a tuple-independent database D with relations $R(A, B, \mathbf{E})$ and $S(C, \mathbf{E})$ such that the Single-Guard Query Q_{SG} on this database creates an equivalent query lineage. Create a database D' with relations $R'(A, B, \mathbf{E})$ and $S'(C, D, \mathbf{E})$ according to the following scheme: For each tuple (a_i, b_i, x_i) in R add a tuple $(a_i, -b_i, x_i)$ to R' and for each tuple (c_j, y_j) in S add a tuple $(c_j, -c_j, y_j)$ to S'. Now a product $x_i y_j$ is in $\phi_{Q(D')}$ if and only if

$$\begin{aligned} & a_i < c_j \wedge -b_i < -c_j \\ & a_i < c_j \wedge b_i > c_j \\ \Leftrightarrow \quad & a_i < c_j < b_i \\ \Leftrightarrow \quad & x_i y_j \in \phi_{Q_{SG}}(D). \end{aligned}$$

"convex bipartite $\subset$ 2-Edge Query": Consider the database D in Figure 7.1. The query lineage is

$$\phi_Q(D) = x_1(y_1 + y_2 + y_6) + x_2(y_2 + y_3 + y_4 + y_6) + x_3(y4 + y_5 + y_6).$$

We now show by contradiction that the y_j's cannot be ordered in a way such that all $N(x_i)$, $1 \leq i \leq$, are intervals: Assume the contrary. Since intersections of intervals are intervals, $N(x_1) \cap N(x_2) = \{y_2, y_6\}$, $N(x_1) \cap N(x_3) = \{y_6\}$ and $N(x_1) \cap N(x_2) = \{y_4, y_6\}$ have to be intervals. Thus, y_6 has to be adjacent to y_2 on the one side and y_4 on the other. W.l.o.g. $y_2 < y_6 < y_4$.

From $N(x_1)$ and $N(x_3)$ we get $y_1 < y_2 < y_6 < y_4 < y_5$. Now $N(x_2)$ implies that y_3 has to be adjacent to y_2, y_6 or y_4, which is not possible. Hence, $G(\phi_{Q(D)})$ is not a convex bipartite graph.
"2-Edge Query $\subseteq$ chordal bipartite": Chordal bipartite graphs are bipartite and weakly chordal. $G(\phi_{Q(D)})$ is by definition bipartite, which also implies that it has no cycle of odd length. Thus we only have to show that $G(\phi_{Q(D)})$ has no chordless cylce C_{2n} with $n \geq 3$.

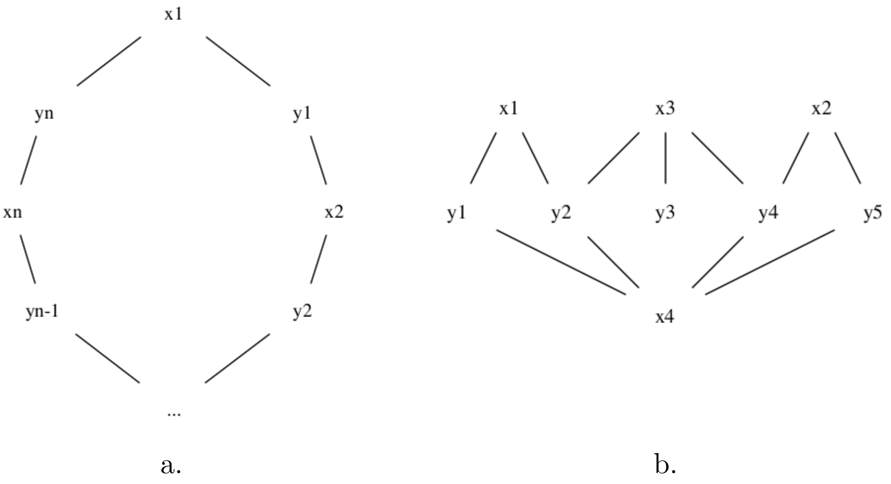


Figure 7.2: Chordless cycle C_{2n} (a.) and chordal bipartite graph (b.)

Assume that there exists such a chordless cycle C_{2n} as shown in Figure 7.2.a. We show that along each direction one of the values A and B has to increase monotonically and the other has to decrease monotonically. This leads to a contradiction when the circle is close.

Since x_1 not connected to y_2 but x_2 is, either $a_1 > a_2$ or $b_1 > b_2$. Due to symmetry we can w.l.o.g. assume that the latter is true. Then $a_1 < a_2$, because otherwise $N(x_1) \supseteq N(x_2)$, which is not true.

With similar reasoning, either $a_2 > a_3 \wedge b_2 < b_3$ (2a) or $a_2 < a_3 \wedge b_2 > b_3$ (2b). Assume that (2a) is true. Then $c_1 > a_2 > a_1, a_3$ and $b_3 > d_1 > b_2 > b_1$. Also, x_2 is connected to y_2 and y_3, which gives $c_2 > a_2$ and $d_2 > b_3$. In conclusion, $c_2 > a_2 > a_1$ and $d_2 > b_3 > b_1$ and x_1 has to be connected to y_2, which is a contradiction. Thus, (2b) has to be true.

We have $a_1 < a_2 < a_3$ and $b_1 > b_2 > b_3$. By inductively repeating the argument we get

$$a_1 < a_2 < \ldots < a_n \text{ and } b_1 > b_2 > \ldots > b_n. \tag{7.24}$$

But x_1 and x_n are connected to y_n, which means $c_n > a_1, a_n$ and $d_n > b_1 b_n$. Together with (7.24) this implies

$$a_i < a_n < c_n, \, b_i < b_1 < d_n \;\; \forall 1 \leq i \leq n$$

$$\Rightarrow y_n \in N(x_i) \;\; \forall 1 \leq i \leq n,$$

which is a contradiction to C_{2n} being chordless.
"2-Edge Query $\subset$ chordal bipartite": Consider the chordal bipartite graph in Figure 7.2.b. The equivalent 2-DNF is

$$\phi = x_1(y_1+y_2)+x_2(y_4+y_5)+x_3(y_2+y_3+y_4)+x_4(y_1+y_2+y_4+y_5). \tag{7.25}$$

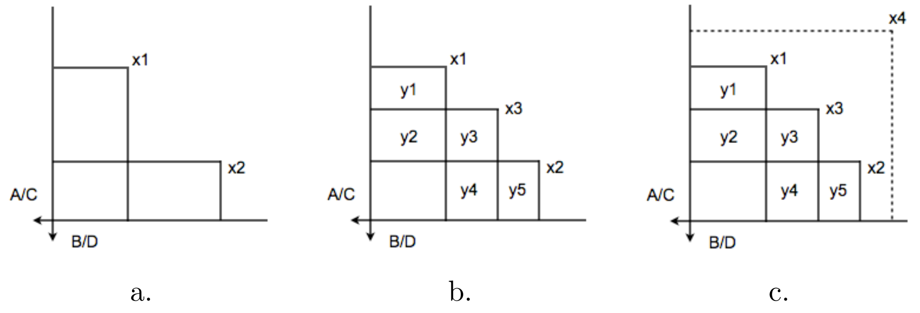


Figure 7.3: Counter-example to show ”2-Edge Query $\subset$ chordal bipartite”

We again use proof by contradiction to show that ϕ cannot be generated by a 2-Edge Query: Assume there exists a tuple-independent probabilistic database D such that $\phi_{Q(D)} = \phi$. Due to symmetry we can w.l.o.g. assume that $Vars(R) = X$ and $Vars(S) = Y$. Since the neighborhoods of x_1 and x_2 are no included in each other we have $a_1 < a_2 \wedge b_1 > b_2$ or $a_1 > a_2 \wedge b_1 < b_2$. Again due to symmetry we w.l.o.g. assume the the former is true. (Figure 7.3.a)

We have $x_2y_4, x_3y_4 \in \phi, x_1y_4 \notin \phi$. This implies $b_3 < d_4 \leq b_1$. Similarly, $x1y_2, x_3y_2 \in \phi, x_2y_2 \notin \phi$ implies $a_3 < c_2 \leq a_2$. Combined with this, $x_3y_3 \in \phi, x_1y_3, x_2y_3 \notin \phi$ gives $a_1 < a_3 < a_2$ and $b_2 < b_3 < b_1$. (Figure 7.3.b)

Finally, $x_3y_3, x_4y_1, x_4y_5 \in \phi, x_3y_1, x_3y_5 \notin \phi$. This implies

$$a_4 < c_1 \leq a_4 < c_3 \text{ and } b_4 < d_5 \leq b_1 < b_3 \tag{7.26}$$

and thus $x_4y_3 \in \phi$, which is a contradiction. (Figure 7.3.c) □

Bibliography

[1] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *Proc. ICDE*, 2008.

[2] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *Proc. ICDE*, pages 983–992, 2008.

[3] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *Proc. VLDB*, 2006.

[4] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. Mystiq: A system for finding more answers by using probabilities. In *SIGMOD Conference*, 2005.

[5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[6] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[7] N. Dalvi and D. Suciu. Efficient query evaluationon probabilistic databases. *VLDB Journal*, 16(4):523–544, 2007.

[8] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *In Proc. PODS*, 2007.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[10] C. P. Gomes, A. Sabharwal, and B. Selman. *Handbook of Satisfiability, Chapter 20.* IOS Press, 2008.

[11] J. Huang, L. Antova, C. Koch, and D. Olteanu. Maybms: A probabilistic database management system. In *Proc. SIGMOD*, 2009.

[12] R. Jampani, M. Wu F. Xu, L. L. Perez, C. M. Jermaine, and P. J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *Proc. SIGMOD*, pages 687–700, 2008.

[13] M.-S. Lin. Fast and simple algorithms to count the number of vertex covers in an interval graph. *Information Processing Letters*, 102:143–146, 2007.

[14] Y. Okamoto, T. Uno, and R. Uehara. Linear-time counting algorithms for independent sets in chordal graphs. In *Proc. of 31st Int. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 433–444, 2005.

[15] D. Olteanu and J. Huang. Using obdds for efficient query evaluation on probabilistic databases. In *Proc. of 2nd Int. Conf. on Scalable Uncertainty Management (SUM),*, Oct 2008.

[16] D. Olteanu and J. Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *ACM SIGMOD 2009*, 2009.

[17] D. Olteanu, J. Huang, and C. Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *Proc. ICDE*, pages 640–651, 2009.

[18] University Rostock Information System on Graph Class Inclusions Project. Graphclass: bipartite chain. http://wwwteo.informatik.uni-rostock.de/isgci/classes/gc_442.html.

[19] University Rostock Information System on Graph Class Inclusions Project. Graphclass: chordal bipartite. http://wwwteo.informatik.uni-rostock.de/isgci/classes/gc_79.html.

[20] University Rostock Information System on Graph Class Inclusions Project. Graphclass: convex. http://wwwteo.informatik.uni-rostock.de/isgci/classes/gc_67.html.

[21] S. J. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[22] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. ICDE*, 2007.

[23] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Proc. ICDE*, 2007.

[24] D. Sieling and I. Wegener. Reduction of obdds in linear time. *Information Processing Letters*, 48(3):139–144, 1993.

[25] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: Native support for uncertain data (demo). In *Proc. SIGMOD*, 2008.

[26] M. Soliman, I. Ilyas, and K. Chang. Probabilistic top-k and ranking-aggregate queries. *ACM TODS*, 33, 2008.

[27] S. Toda. On the computational power of pp and p. In *Proceedings of IEEE FOCS'89*, pages 514–419, 1989.

[28] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[29] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):419–421, 1979.

[30] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. M. Hellerstein. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. In *Proc. VLDB*, 2008.

[31] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. Second Biennial Conference on Innovative Data Systems Research (CIDR '05)*, 2005.

[32] M. Xia and W. Zhao. #3-regular bipartite planar vertex cover is #p-complete. In *TAMC*, pages 356–364, 2006.