Domain Ordering and Box Cover Problems for Beyond Worst-Case Join Processing

by

Kaleb Alway

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2019

Author's Declaration

Abstract

Join queries are a fundamental computational task in relational database management systems. For decades, complex joins were most often computed by decomposing the query into a query plan made of a sequence of binary joins. However, for cyclic queries, this type of query plan is sub-optimal. The worst-case run time of any such query plan exceeds the number of output tuples for any query instance.

Recent theoretical developments in join query processing have led to join algorithms which are worst-case optimal, meaning that they run in time proportional to the worst-case output size for any query with the same shape and the same number of input tuples. Building on these results are a class of algorithms providing bounds which go beyond this worst-case output size by exploiting the structure of the input instance rather than just the query shape.

One such algorithm, Tetris, is worst-case optimal and also provides an upper bound on its run time which depends on the minimum size of a geometric box certificate for the input query. A box certificate is a subset of a box cover whose union covers every tuple which is not present in the query output. A box cover is a set of n-dimensional boxes which cover all of the tuples not contained in the input relations. Many query instances admit different box certificates and box covers when the values in the attributes' domains are ordered differently. If we permute the input query according to a domain ordering which admits a smaller box certificate, use the permuted query as input to Tetris, then transform the result back with the inverse domain ordering, we can compute the query faster than was possible if the domain ordering was fixed. If we can efficiently compute an optimal domain ordering for a query, then we can state a beyond worst-case bound that is stronger than what is provided by Tetris [1].

This thesis defines several optimization problems over the space of domain orderings where the objective is to minimize the size of either the minimum box certificate or the minimum box cover for the given input query. We show that most of these problems are NP-hard. We also provide approximation algorithms for several of these problems.

The most general version of the box cover minimization problem we will study, $\text{BoxMinP}_{\text{DomF}}$, is shown to be NP-hard, but we can compute an approximation of size $\widetilde{O}((K_{\square}^{*})^{a \cdot r})$, where $K_{\square}^{*}$ is the minimum box cover size under any domain ordering, a is the maximum degree of an attribute in the query graph, and r is the maximum number of attributes in a relation. This result allows us to compute join queries in time $\widetilde{O}(N + (K_{\square}^{*})^{a \cdot r \cdot (w+1)} + Z)$, where N is the number of input tuples, w is the treewidth of the query, and Z is the number of output tuples. This is a new beyond worst-case bound. There are queries for which this bound is exponentially smaller than any bound provided by Tetris.

The most general version of the box certificate minimization problem we study, $\text{CertMinP}_{\text{DomF}}$, is also shown to be NP-hard. It can be computed exactly if the minimum box certificate size is at most 3, but no approximation algorithm for an arbitrary minimum size is known. Finding such an approximation algorithm is an important direction for future research.

Acknowledgements

First, I would like to thank my supervisors, Semih Salihoglu and Eric Blais, for their excellent guidance and weekly brainstorming sessions over the past 2 years.

I would like to thank my thesis readers, Anna Lubiw and Ian Munro. I would also like to thank Anna for her helpful suggestions which led to the proofs of Theorems 5.9 and 5.15.

I would like to thank my family for all the love and support over these 2 years and during all the years that brought me to this point. I would especially like to thank my parents Mike and Carla, my sister Mikayla, and my grandma Linda. You gave me everything I needed to accomplish this and more.

I would like to thank Sana for the constant support and encouragement throughout this process. I could not have done this without you.

There are several other groups of people I would like to thank for keeping me company during this time, including my Sarnia friends John, Dylan, and Franklin; my ISG alumni friends Max, Kevin, Ten, Leonard, Larry, Marie, Ed, Bryan, Travis, Chantelle, Abraham, Rob, Sean, and Akshaya; my other undergrad friends Harry, Adam, and Brandon; my undergrad roommates Luke, Ryan, Adam, Shane, Dan, and Grant; and my office mates Alex, Tiasa, and Harry.

Lastly, I would like to thank everyone else who touched my life in some way.

Dedication

This thesis is dedicated to the memory of my grandpa (Papa), Ted Alway.

Table of Contents

List of Tables

List of Figures

Abbreviations

BoxMinP$_{\mathbf{ColF}}$ Column Flexible Box Cover Minimization Problem 32–34, 37, 38, 63

BoxMinP$_{\mathbf{DomF}}$ Domain Flexible Box Cover Minimization Problem iii, 18, 29, 30, 45, 46, 49–52, 54–56, 58, 62, 63

BoxMinP$_{\mathbf{GBO}}$ Global Bit Order Box Cover Minimization Problem 63, 76, 78, 80, 81

BoxMinP$_{\mathbf{RowColF}}$ Row and Column Flexible Box Cover Minimization Problem 37, 38, 44, 45, 50, 63

CertMinP$_{\mathbf{BitF}}$ Bit Flexible Certificate Minimization Problem 63, 73–76, 80

CertMinP$_{\mathbf{DomF}}$ Domain Flexible Box Certificate Minimization Problem iv, 18, 29, 30, 56, 58–63

CertMinP$_{\mathbf{GBO}}$ Global Bit Order Certificate Minimization Problem 63, 75, 76, 80, 82

IntBlkMinP$_{\mathbf{ColF}}$ Column Flexible Interval Block Minimization Problem 35–37

2ConBlkMinP 2 Consecutive Block Minimization Problem x, 38, 39

BoxMinP Box Cover Minimization Problem 20, 34

ConBlkMinP Consecutive Block Minimization Problem 32, 33, 36–38

GAO global attribute order 70, 71, 74

GBO global bit order 74–81

GenOrdConMaxP Generalized Ordering Constraint Maximization Problem 80, 81

OrdConMaxP Ordering Constraint Maximization Problem 78–80

SAO splitting attribute order 12–14, 16, 55, 70, 71

List of Symbols

N the number of input tuples for a query

Z the number of output tuples for a query

n the number of attributes in a query

m the number of relations in a query

d each attribute's domain values are stored in d bits of memory

R a relation

A an attribute

$\mathcal{Q}$ a join query

w the treewidth of a query

$K_{\Box}(\mathcal{Q})$ the minimum box cover size for the query $\mathcal{Q}$

$C_{\Box}(\mathcal{Q})$ the minimum box certificate size for the query $\mathcal{Q}$ over all box covers of $\mathcal{Q}$

σ a domain ordering

$\sigma(\mathcal{Q})$ the query obtained from $\mathcal{Q}$ by permuting the attributes according to σ

Chapter 1

Introduction

Join query processing is an important part of any database management system [41]. In particular, natural joins over attributes with discrete, finite domains are useful for many applications such as for subgraph queries in social network data [49] and for joining fact and dimension relations in data warehouses [8]. This type of query is also general enough to express constraint satisfaction problems [26]. In this thesis we will exclusively study this type of join query. We take as input a query $\mathcal{Q}$ which consists of a set of relations $\mathcal{R}$ over a set of attributes $\mathcal{A}$. Each relation in $\mathcal{R}$ is a set of tuples over a subset of the attributes in $\mathcal{A}$.

As the demand grows for applications to process and analyze larger quantities of data, the need for efficient join algorithms increases. Many different algorithms have been designed and implemented in database management systems. Run time upper bounds for these algorithms are dependent on several different parameters, including the number of input tuples N, and the number of output tuples Z. Many of these upper bounds also incorporate parameters related to the shape of the query graph, such as the number of relations m, the number of attributes n, the treewidth w, the fractional edge cover number ρ, and the fractional hypertree width $fhtw$. These quantities will be defined precisely in Chapter 2.

Yannakakis' algorithm is a well known join algorithm for acyclic queries which runs in time $O(N + Z)$ [52]. This algorithm was later generalized to a version which runs in time $O(N^w + Z)$ [12] for queries with arbitrary treewidth w. Atserias, Grohe, and Marx [3] proved a bound, now known as the AGM bound, on the worst-case output size for a query based on the number of tuples in each relation and the fractional edge cover number of the query graph. Several algorithms have been shown to be worst-case optimal in the sense that their run time is bounded by the AGM bound [37, 36, 50]. A worst-case optimal join algorithm can be combined with Yannakakis' algorithm to compute any join query in time $O(N^{fhtw} + Z)$ [18].

Treewidth, fractional hypertree width, and other related notions depend only on the query graph, and say nothing about the specific tuples in the input relations. However, not all queries with the same query graph are equally difficult to compute. Tighter run time bounds are possible for query instances which are simpler or more nicely structured than the instances which give output sizes close to the AGM bound.

In recent years, a class of join algorithms which provide "beyond worst-case" run time bounds have been introduced [35, 1, 39, 22]. One such algorithm, from Abo Khamis et al., is called Tetris [1]. We will first introduce three important terms required to understand Tetris and the results of this thesis. These terms will be defined formally in Section 2.2.

- A *gap box* is an n-dimensional box which covers an area of an input relation where there are no tuples.
- A *box cover* is a set of gap boxes B such that for each relation R in the query and each tuple t *not* in R, there is some gap box in B which contains t. The minimum box cover size for a query $\mathcal{Q}$ is denoted $K_\square(\mathcal{Q})$.
- A *box certificate* for a box cover B is a subset of B whose union is equal to the union of all the gap boxes in B. The minimum box certificate size for a box cover B is denoted $C_\square(B)$. The minimum box certificate size for *any* box cover of the query $\mathcal{Q}$ is dentoed $C_\square(\mathcal{Q})$.

An example of a box cover and box certificate is depicted in Figure 2.2 of the following chapter. Tetris takes as input a box cover B for the query instead of the usual input tuples. Tetris is worst-case optimal because it meets the AGM bound, but it also provides beyond worst-case bounds which are dependent on $C_\square(B)$. Two of the run time bounds provided by Tetris are $\widetilde{O}\Big(\big(C_\square(B)\big)^{w+1} + Z\Big)$ and $\widetilde{O}\Big(\big(C_\square(B)\big)^{n/2} + Z\Big)$. Throughout this thesis, the $\widetilde{O}$-notation will hide any poly-logarithmic factors in N and Z, as well as the query graph dependent factors n and m. These parameters are constant when the query shape is fixed, so we consider them to be constants in our analysis. $C_\square(B)$ is at most N and can be much smaller than N, giving bounds which can be much tighter than the AGM bound.

Tetris takes a pre-constructed box cover as input, so the cost of computing a box cover is not included in the upper bounds it provides. Throughout this thesis we will focus on the setting where we begin with the tuples as input, construct a corresponding box cover, and then call Tetris. Better results can be obtained if box covers are indexed and reused so that they are not recomputed for each query, but this thesis will not discuss that case. We will show in Chapter 4 that it is possible to construct a box cover B for $\mathcal{Q}$ in $\widetilde{O}(N)$ time such that the minimum box certificate size for B is in $\widetilde{O}\big(C_{\square}(\mathcal{Q})\big)$. Starting from the input tuples, we can use this process to construct a box cover to use as input to Tetris, which yields total runtime bounds of $\widetilde{O}\Big(N + \big(C_{\square}(\mathcal{Q})\big)^{w+1} + Z\Big)$ and $\widetilde{O}\Big(N + \big(C_{\square}(\mathcal{Q})\big)^{n/2} + Z\Big)$. When $C_{\square}(\mathcal{Q}) = o(N)$, these bounds can be asymptotically much better than the AGM bound and fractional hypertree width bound.

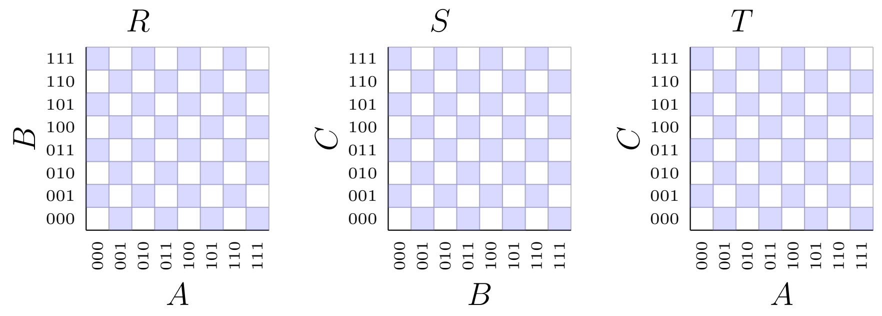
However, there are simple queries which can be geometrically complex and require a large box certificate. In many cases, these simple queries can be modified by reordering each attributes' domain so that a smaller box certificate is possible. Figure 1.1 shows an example of this. The queries $\mathcal{Q}$ and $\mathcal{Q}'$ are both triangle queries joining three relations with two attributes each, and the output of both queries is empty. These queries are equivalent up to reordering the domains of each attribute. That is, we reorder the rows and columns of the grid in Figure 1.1a to obtain Figure 1.1b. In this figure, σ is the set of three permutations on the domains of A, B, and C which transforms $\mathcal{Q}$ into $\mathcal{Q}'$. Despite this similarity between $\mathcal{Q}$ and $\mathcal{Q}'$, $\mathcal{Q}$ requires a larger box cover than $\mathcal{Q}'$, because each white grid cell in Figure 1.1a must have a unit gap box covering it, while the white cells in Figure 1.1b can be covered by a total of 6 gap boxes. For these queries, every gap box in the box cover must also be part of the box certificate, although this is not true in general. This means $C_{\square}(\mathcal{Q}) = 96$ and $C_{\square}(\mathcal{Q}') = 6$, so Tetris will run faster on $\mathcal{Q}'$ than on $\mathcal{Q}$.

By extending the domains of the attributes in this example and repeating the same pattern, the difference in box certificate sizes can be made arbitrarily large. If the input query looks like $\mathcal{Q}$, and we can efficiently find a domain ordering which transforms $\mathcal{Q}$ into $\mathcal{Q}'$, then we can compute the join faster by running Tetris on $\mathcal{Q}'$ instead.

Examples such as this motivate the problems studied in this thesis. We would like to find a permutation, or domain ordering, σ^* on the domains of the attributes in $\mathcal{A}$ such that the permuted relation, $\sigma^*(\mathcal{Q})$, admits a box certificate of minimum size. Unfortunately, little is known about solving this problem efficiently. In this thesis, we simply show that the problem is NP-hard, and that it can be solved efficiently in the very restricted case where the minimum box certificate size over all domain orderings is at most 3.

Instead, the majority of this thesis will focus on the related problem of finding a domain ordering for $\mathcal{Q}$ that minimizes the size of the minimum box cover for the permuted query $\sigma^*(\mathcal{Q})$. We will see in Section 5.6 that there are queries for which the ordering that minimizes the box cover size can be very different from the ordering that minimizes the certificate size. However, since the box certificate size is less than or equal to the box

(a) $\mathcal{Q} = R \bowtie S \bowtie T$



(b) $\mathcal{Q}' = R' \bowtie S' \bowtie T' = \sigma(R) \bowtie \sigma(S) \bowtie \sigma(T)$

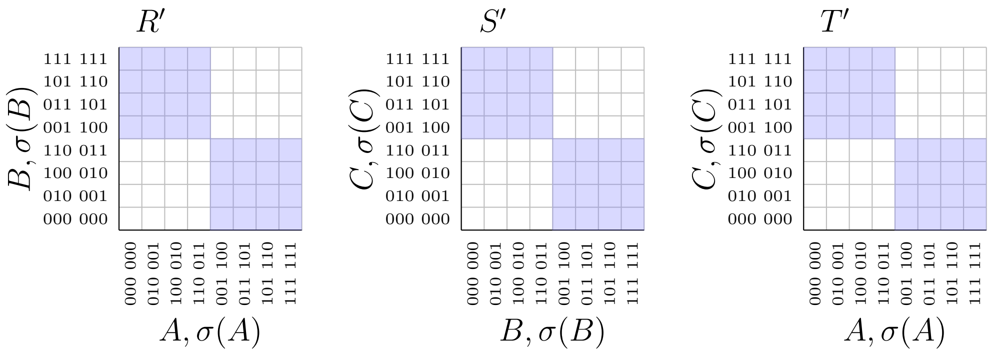


Figure 1.1: Queries with different box certificate sizes which are equivalent up to reordering the attributes' domains

cover size, there are still queries for which minimizing the box cover size by reordering the domain yields a better run time bound than any provided by Tetris without domain reordering.

We will show that this problem is also NP-hard. However, Corollary 5.21 shows that we can efficiently compute a domain ordering which yields a box cover of size $\widetilde{O}\left(\left(K_{\square}(\sigma^*(\mathcal{Q}))\right)^{a \cdot r}\right)$ where $K_{\square}(\sigma^*(\mathcal{Q}))$ is the minimum box cover size under the optimal domain ordering σ^*, a is the maximum number of relations any single attribute appears in, and r is the maximum number of attributes in a relation. Since a and r are constant when the query shape is fixed, this is a polynomial-factor approximation to the optimal solution. Theorems 5.22 and 5.23 state the following new bounds on join run time by combining Corollary 5.21 with

Tetris' bounds.

$$\widetilde{O}\Big(N + \big(K_{\square}(\sigma^*(\mathcal{Q}))\big)^{a\cdot r\cdot(w+1)} + Z\Big)$$
$$\widetilde{O}\Big(N + \big(K_{\square}(\sigma^*(\mathcal{Q}))\big)^{a\cdot r\cdot n/2} + Z\Big)$$

The thesis is organized as follows. Chapter 2 reviews prerequisites about join queries, join algorithms, box covers, and Tetris. Chapter 3 reviews several related results from the literature about box cover problems and join algorithms. In Chapter 4, we present an algorithm which computes a box cover B for any query $\mathcal{Q}$ such that the box certificate for B is of minimum size. In Chapter 5, we study several general domain ordering problems. We start by aiming to minimizing the box cover size of a single 2-dimensional relation in Sections 5.2 and 5.3. Even this case is shown to be NP-hard. Section 5.4 generalizes this to n-dimensional relations and presents an approximation algorithm which the aforementioned upper bounds are based upon. This algorithm is extended to multiple relations in Section 5.5, and Section 5.5.1 states Theorems 5.22 and 5.23. Finally, Chapter 6 consolidates all the findings in this thesis and identifies several open questions.

Chapter 2

Preliminaries and Research Questions

This chapter will provide several definitions and fundamental results which are necessary to explain the results of this thesis. Then, we will identify the research questions that this thesis seeks to solve. To begin, we will define the database model we will be working with, as well as our notion of a join query.

Definition 2.1 (Database, Attribute, Relation). A database $\mathcal{D} = (\mathcal{R}, \mathcal{A})$ is a collection of relations $\mathcal{R}$ over a set of attributes $\mathcal{A}$. Each attribute $A \in \mathcal{A}$ is a variable over a discrete, finite domain denoted $\mathrm{dom}(A)$. For simplicity, in this thesis we will assume all attributes are stored in d bits, so $\mathrm{dom}(A)$ is always equivalent to the set of binary strings $\{0,1\}^d$ and the set of non-negative integers $\{0, 1, \ldots, 2^d - 1\}$. Each relation $R \in \mathcal{R}$ is a set of tuples over a subset of attributes $\mathrm{attr}(R) \subseteq \mathcal{A}$. If $\mathrm{attr}(R) = \{A, B, C\}$, for example, then we often denote R by $R(A, B, C)$ and each tuple $t \in R$ is of the form $t = \langle a, b, c \rangle$ where $a \in \mathrm{dom}(A), b \in \mathrm{dom}(B)$, and $c \in \mathrm{dom}(C)$. If $t \in R$ and $A \in \mathrm{attr}(R)$, the value for A in t is denoted $t.A$.

Throughout this thesis, we will often discuss the geometric representation of a relation. A relation R over n attributes can be thought of as a subset of an n-dimensional hypercube H with one vertex at 0 and side lengths all equal to 2^d, extending in the positive direction along each dimension. Each axis is labelled with the domain values, in order, of some attribute $A \in \mathrm{attr}(R)$. Each individual value $a \in \mathrm{dom}(A)$ corresponds to a unit interval on the A-axis. Then every possible tuple $t \in \times_{A \in \mathrm{attr}(R)} \mathrm{dom}(A)$ naturally corresponds to a unit hypercube within H. The geometric representation of R is the union of the unit

R

A	B
000	010
001	001
010	000
010	100
010	101
010	110
010	111
011	000
100	000
101	000
101	100
101	101
101	110
101	111
110	001
111	010

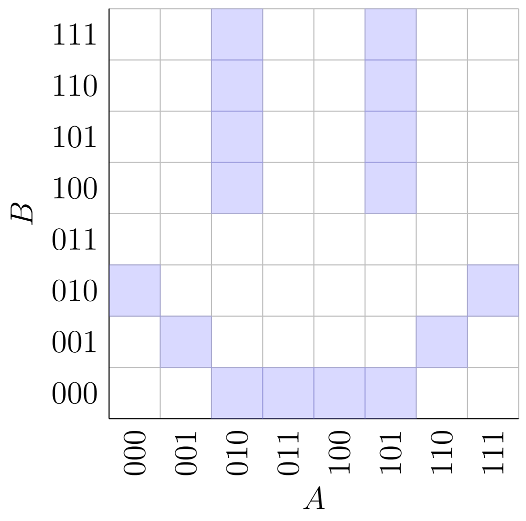


Figure 2.1: A relation represented as a list of tuples (left) and its geometric representation where each shaded square is a tuple (right)

hypercubes corresponding to each tuple in R. An example of this in the 2-dimensional case is shown in Figure 2.1, where the blue shaded grid cells are part of the geometric representation of R.

There are two relational algebra operators we will use in this thesis. The selection operator, denoted $\sigma_P(R)$ for some relation R and some predicate P, is defined as

$$\sigma_P(R) = \{t \in R : P\}$$

The projection operator, denoted $\pi_{\mathcal{A}'}(R)$ for some relation R and a set of attributes $\mathcal{A}' \subseteq \mathcal{A}$, is defined as

$$\pi_{\mathcal{A}'}(R) = \{t' \in \times_{A \in \mathcal{A}'} \mathrm{dom}(A) : \exists t \in \times_{A \in \mathcal{A}} \text{ such that } t.A = t'.A, \forall A \in \mathcal{A}'\}$$

We can now give the formal definition of a join query.

Definition 2.2 (Join query). A *join query* is a pair $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ where $\mathcal{R}$ is a set of relations over the set of attributes $\mathcal{A}$. The relations in $\mathcal{R}$ are the inputs of $\mathcal{Q}$. The output of $\mathcal{Q}$ is a relation $J = \bowtie_{R \in \mathcal{R}} R$ with $\text{attr}(J) = \mathcal{A}$. A tuple $t \in \times_{A \in \mathcal{A}} \text{dom}(A)$ is in J if and only if $\pi_{\text{attr}(R)} t \in R$ for each $R \in \mathcal{R}$. For the join of two relations $R(A_1, \ldots, A_{n_R})$ and $S(B_1, \ldots, B_{n_S})$, formally $\mathcal{Q} = (\{R, S\}, \{A_1, \ldots, A_{n_R}, B_1, \ldots, B_{n_S}\})$, we will also use the notation $R(A_1, \ldots, A_{n_R}) \bowtie S(B_1, \ldots, B_{n_S})$, or $R \bowtie S$ when the attributes of the relations are clear.

In all of the asymptotic analysis in this thesis, the $\widetilde{O}$-notation hides any poly-logarthmic factors in N and Z, where N is the total number of input tuples summed over all relations in $\mathcal{Q}$, and Z is the number of output tuples for $\mathcal{Q}$. This notation will also hide any polynomial factors in d, since d is assumed to be of size $O(\log N)$. It will also hide any factors of n, the number of attributes in $\mathcal{Q}$, and m, the number of relations in $\mathcal{Q}$, because these parameters are fixed by the query shape. In general, the notation $\widetilde{O}(f(N))$ represents the bound

$$O\Big(f(N) \cdot p\big(\log(N + Z)\big) \cdot q(d) \cdot g(m, n)\Big)$$

where p and q are polynomials of degree at most $m + n$, and g is an arbitrary function.

The next section will build upon these definitions by defining several quantities that describe the shape of a join query.

2.1 Query Graphs and Notions of Width

Many join algorithms have worst-case run times which depend on features of the *query graph* for the input query. The query graph is defined as follows.

Definition 2.3 (Query graph). The *query graph* for $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ is the hypergraph $\mathcal{H}(\mathcal{Q}) = (V, \mathcal{E})$ with vertex set $V = \mathcal{A}$ and the hyperedge set $\mathcal{E} = \{\text{attr}(R) : R \in \mathcal{R}\}$.

One common metric measuring the complexity of a query's shape is its *treewidth.* Any join can be computed in $O(N^w + Z)$ time, where w is its treewidth, N is the number of input tuples, and Z is the number of output tuples, using a generalization of Yannakakis' algorithm [52, 12]. To understand exactly what this bound says, we need to define tree decomposition and treewidth.

Definition 2.4 (Tree decomposition). For a hypergraph $\mathcal{H} = (V, \mathcal{E})$, a *tree decomposition* of $\mathcal{H}$ is a pair $\mathcal{D} = (\mathcal{T}, \phi)$ where $\mathcal{T}$ is a tree and $\phi : V(\mathcal{T}) \to 2^V$ maps each vertex in $\mathcal{T}$ to a subset of V. $\mathcal{D}$ must satisfy the following properties.

1. For each $E \in \mathcal{E}$, there must be some vertex t of $\mathcal{T}$ such that $E \subseteq \phi(t)$
2. For each $v \in V$, the set $t(v) = \{t \in V(T) : v \in \phi(t)\}$ must form a non-empty connected subtree of $\mathcal{T}$

Definition 2.5 (Treewidth). The *treewidth* of a tree decomposition $\mathcal{D} = (\mathcal{T}, \phi)$ of $\mathcal{H}$ is $\max_{t \in V(T)} |\phi(t)| - 1$. The treewidth of $\mathcal{H}$ is the minimum treewidth over all tree decompositions of $\mathcal{H}$. The treewidth of a query $\mathcal{Q}$ is the treewidth of $\mathcal{H}(\mathcal{Q})$.

In 2008, Atserias, Grohe, and Marx published an important result that would come to be known as the AGM bound [3]. The AGM bound is a worst-case upper bound on the number of output tuples for a join query based on the shape of the query graph and the number of tuples in each relation. The AGM bound is defined in terms of a *fractional edge cover* for the query graph.

Definition 2.6 (Fractional edge cover). For a hypergraph $\mathcal{H} = (V, \mathcal{E})$, a *fractional edge cover* for $\mathcal{H}$ is a collection of real numbers $\rho_E \in [0, 1]$ for each $E \in \mathcal{E}$ such that $\sum_{\{E \in \mathcal{E} : v \in E\}} \rho_E \geq 1$ for each vertex $v \in V$. The *minimum* fractional edge cover is the fractional edge cover ρ^* which minimizes the objective function $\sum_{E \in \mathcal{E}} \rho^*_E$.

For the query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$, if F is the set of all fractional edge covers for $\mathcal{H}(\mathcal{Q})$, then the AGM bound is

$$\text{AGM}(\mathcal{Q}) = \min_{\rho \in F} \prod_{R \in \mathcal{R}} (N_R)^{\rho_R}$$

Example 2.7. Consider the triangle query, $\mathcal{Q} = R(A, B) \bowtie S(B, C) \bowtie T(A, C)$. Suppose the relations R, S, and T have N tuples each. The query graph for $\mathcal{Q}$ is a 3-cycle (a triangle), whose minimum fractional edge cover is $\rho^*_R = \rho^*_S = \rho^*_T = \frac{1}{2}$. The AGM bound for this query is

$$\text{AGM}(\mathcal{Q}) = N^{\rho^*_R} \cdot N^{\rho^*_S} \cdot N^{\rho^*_T} = N^{3/2}$$

Therefore, the output of $\mathcal{Q}$ has at most $N^{3/2}$ tuples.

A join algorithm which runs in time $\widetilde{O}(\text{AGM}(\mathcal{Q}))$ is said to be worst-case optimal. Another quantity which is dependent on fractional edge covers is the *fractional hypertree width.* Any join query can be computed in time $\widetilde{O}(N^{fhtw} + Z)$, where $fhtw$ is the fractional hypertree width of $\mathcal{Q}$, by combining Yannakakis' algorithm with a worst-case optimal join algorithm [18].

Definition 2.8 (Fractional hypertree width). The *fractional hypertree width* of a tree decomposition $\mathcal{D} = (\mathcal{T}, \phi)$ of $\mathcal{H}$ is $\max_{t \in V(\mathcal{T})} \rho^*(\phi(t))$, where $\rho^*(\phi(t))$ denotes the minimum fractional edge cover of the subgraph of $\mathcal{H}(\mathcal{Q})$ induced by the vertex set $\phi(t)$. The fractional hypertree width of $\mathcal{H}$ is the minimum fractional hypertree width over all tree decompositions of $\mathcal{H}$. The fractional hypertree width of a query $\mathcal{Q}$ is the fractional hypertree width of $\mathcal{H}(\mathcal{Q})$.

2.2 Gap Boxes, Box Covers, and Box Certificates

The work of Abo Khamis, Ngo, Ré, and Rudra [1] introduced a new geometric framework for processing join queries. The join algorithm Tetris takes as input a geometric *box cover* of the query instead of the usual input tuples and has a run time that depends on the minimum size of a *box certificate* for the input box cover.

In order to understand the join processing upper bounds provided by Tetris, we need to define gap boxes, box covers, and box certificates.

Definition 2.9 (Box, Gap box). Let $R \in \mathcal{R}$ and let $\text{attr}(R) = \{A_1, A_2, \ldots, A_{n_R}\}$. A *box* for R is an n_R-tuple of intervals $b = \langle I_1, I_2, \ldots, I_{n_R} \rangle$ where each I_j is a contiguous interval $I_j = [a_1, a_2]$ with $a_1, a_2 \in \text{dom}(A_j)$ and $a_1 \leq a_2$. A tuple $t \in \times_{A \in \text{attr}(R)} \text{dom}(A)$ is covered by b, denoted $t \in b$, if $t.A_j \in I_j$ for each $j \in [n_R]$. The box b is a *gap box* for R if there is no $t \in R$ such that $t \in b$. When a gap box b is given as input to Tetris, it is "extended" to an n-tuple with one interval for each of the n attributes in $\mathcal{A}$. If $A_i \notin \text{attr}(R)$, then $b.A_i = [0, 2^d - 1]$. We will refer to b and the extended version of b interchangeably.

Definition 2.10 (Box cover). A *box cover for a relation* $R \in \mathcal{R}$ is a set of gap boxes B such that for each tuple $t \in R$, $t \notin \cup_{b \in B} b$ and for each tuple $t \notin R$, there exists $b \in B$ such that $t \in B$. The set of box covers for R is denoted $\mathcal{B}(R)$. The minimum box cover size for R is denoted $K_\square(R)$. A *box cover for a query* $\mathcal{Q}$ is a set of box covers $\mathcal{B} = \{B_1, \ldots, B_m\}$ such that each B_i is a box cover for $R_i \in \mathcal{R} = \{R_1, \ldots, R_m\}$. The minimum total box cover size for $\mathcal{Q}$ is denoted $K_\square(\mathcal{Q})$.

Definition 2.11 (Box certificate). A *box certificate* for a box cover $\mathcal{B}$ of $\mathcal{Q}$ is a subset $C \subseteq \cup_{B \in \mathcal{B}} B$ such that $\cup_{b \in C} b = \cup_{B \in \mathcal{B}} \cup_{b \in B} b$. The set of box certificates for a box cover $\mathcal{B}$ is denoted $\mathcal{C}(\mathcal{B})$. The minimum box certificate size for a box cover $\mathcal{B}$ is denoted $C_\square(\mathcal{B})$. The minimum box certificate size for $\mathcal{Q}$ over all possible box covers is denoted $C_\square(\mathcal{Q})$.

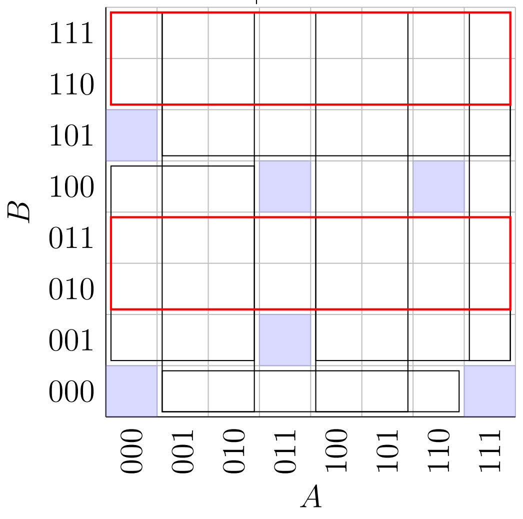
Example 2.12. Consider the query $\mathcal{Q} = R(A, B) \bowtie S(B, C)$ where each attribute has a 3-bit domain. Figure 2.2 shows an instance of this query. The blue shaded cells are tuples

R

A	B
000	000
000	101
011	001
011	100
110	100
111	000

S

B	C
010	000
010	001
010	110
011	000
110	000
110	110



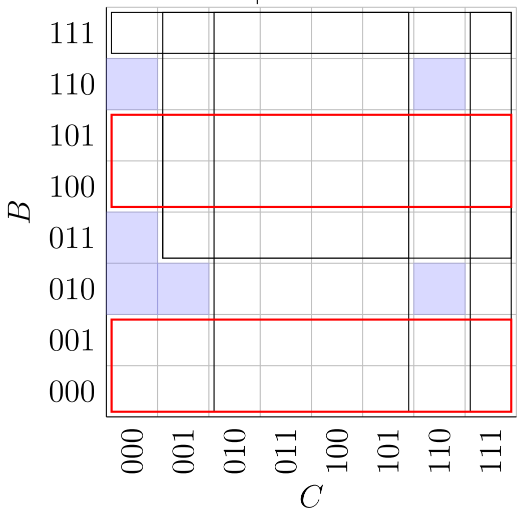


Figure 2.2: An example of a box cover and box certificate

in the relations, (the extensions of) the black and red outlined gap boxes are part of the box cover, and (the extensions of) the red outlined gap boxes are also part of the box certificate C. The 4 boxes in the certificate in their non-extended form are $\langle [000, 111], [010, 011] \rangle$ and $\langle [000, 111], [110, 111] \rangle$ from $R(A, B)$, and $\langle [000, 001], [000, 111] \rangle$ and $\langle [100, 101], [000, 111] \rangle$ from $S(B, C)$. By extending these boxes in attribute order (A, B, C), this certificate is written as

$$C = \Big\{ \big\langle [000, 111], [110, 111], [000, 111] \big\rangle, \big\langle [000, 111], [110, 111], [000, 111] \big\rangle,$$
$$\big\langle [000, 111], [000, 001], [000, 111] \big\rangle, \big\langle [000, 111], [100, 101], [000, 111] \big\rangle \Big\}$$

Note that these 4 gap boxes cover the entire output space, so the output of the query is empty.

This definition of gap boxes generalizes 2-dimensional axis-aligned rectangles to an arbitrary number of dimensions. For the majority of this thesis, this is the type of gap boxes we will be working with. However, the gap boxes which Tetris takes as input are not of this form. Instead, Tetris takes a specific type of gap boxes called *dyadic gap boxes*.

Definition 2.13 (Dyadic box, Dyadic gap box). A *dyadic box* for relation R over n_R attributes is an n_R-tuple $b = \langle s_1, s_2, \dots, s_{n_R} \rangle$ where each s_i is a binary string of length at most d. We will use $*$ to denote the empty string. Semantically, b is equivalent to the general box b' obtained from b by replacing each s_i with the interval $[s_i 0^{d-|s_i|}, s_i 1^{d-|s_i|}]$. A *dyadic gap box* is defined analogously to a general gap box.

Similarly, we define *dyadic box covers for R*, *dyadic box covers for $\mathcal{Q}$*, and *dyadic box certificates* for a dyadic box cover.

Dyadic boxes are a particularly useful analytical tool for join query processing because there are far fewer dyadic boxes than there are general boxes. This allows us to perform certain operations on dyadic boxes in $\widetilde{O}(1)$ time which would not be possible with general boxes. In particular, Lemma 2.14, states that there are not many dyadic gap boxes which contain any given tuple. Even better, Lemma 2.15 informs us that by restricting ourselves to only dyadic box certificates, we do not increase the size of the minimum box certificate by more than a poly-logarithmic factor. These facts are key ingredients to the upper bounds provided by Tetris.

Lemma 2.14 (Proposition B.12 in [1]). *Let R be a relation and $n_R = |attr(R)|$. For any tuple $t \notin R$, the number of dyadic gap boxes from R which contain t is $O(d^{n_R}) = \widetilde{O}(1)$.*

Lemma 2.15 (Proposition B.15 in [1]). *For every box certificate C, there is a dyadic box certificate of size $O((2d)^n|C|) = \widetilde{O}(|C|)$.*

In the next section, we will review the Tetris algorithm and state two of the upper bounds it provides.

2.3 Tetris

In order to understand the bounds provided by Tetris, we will review the algorithm itself. At a high level, Tetris begins with the input box cover, and performs a series of *ordered geometric resolutions* (defined momentarily) according to the *splitting attribute order (SAO)* to combine the dyadic gap boxes and determine whether they cover the entire output space. Every time an output tuple is encountered, it is reported as output and then inserted as a unit gap box, and the process continues.

Definition 2.16 (Splitting attribute order). A *splitting attribute order (SAO)* is a permutation $\sigma = (A_1, \dots, A_n)$ of the n attributes in $\mathcal{Q}$ which is passed as input to Tetris. Tetris splits dyadic boxes on the attributes according to σ.

Definition 2.17 (Ordered geometric resolution). An *ordered geometric resolution* of two dyadic boxes w_1 and w_2 produces a new dyadic box $w \subseteq w_1 \cup w_2$ and must take the following form for some $1 \leq i \leq n$ and some binary strings $s_1, \dots, s_{i-1}$, $t_1, \dots, t_{i-1}$, x, where the attributes are ordered according to the SAO $\sigma = (A_1, \dots, A_n)$.

$$\begin{aligned} w_1 =& \langle s_1, s_2, \dots, s_{i-1}, x0, *, \dots, * \rangle \\ w_2 =& \langle t_1, t_2, \dots, t_{i-1}, x1, *, \dots, * \rangle \\ w =& \langle s_1 \cap t_1, s_2 \cap t_2, \dots, s_{i-1} \cap t_{i-1}, x, *, \dots, * \rangle \end{aligned}$$

For each $j < i$, either s_j must be a prefix of t_j or t_j must be a prefix of s_j. Then $s_j \cap t_j$ denotes the shorter of the two strings s_j and t_j. w_1 and w_2 must differ by only the last bit in attribute A_i. For all $j > i$, we must have $w_1.A_j = w_2.A_j = *$. If all these conditions hold, then the ordered geometric resolution w of w_1 and w_2 is well-defined.

Algorithm 1 is the recursive subroutine which does most of the work in Tetris. The global set of dyadic boxes $\mathcal{K}$ can be thought of as a knowledge base which contains all of the gap boxes the algorithm has learned so far. The knowledge base is initialized as empty, and it grows as Algorithm 1 is called repeatedly. Algorithm 1 takes as input a dyadic box b, and returns a pair (v, w) where v is TRUE if b is a subset of $\cup_{a \in \mathcal{K}} a$. In this case, w will be some box in $\mathcal{K}$ which contains b. The box w may be either a box from the original box cover, or a box which was added to $\mathcal{K}$ after some previous resolution on line 22. Otherwise, v is FALSE, and w is a tuple (a unit box) not covered by any box in $\mathcal{K}$.

The RESOLVE call on line 21 performs ordered geometric resolution on w_1 and w_2, and it is indeed well defined. This is shown by induction in [1]. If that line is reached, then the if-conditions on lines 10 and 16 ensure that $b_1 \subseteq w_1$ and $b_2 \subseteq w_2$, so the resolvent w satisfies $b \subseteq w$.

The SPLITFIRSTTHICKDIMENSION call on line 8 simply splits b into two halves based on the next available bit in the SAO. That is, if

$$b = \langle s_1, s_2, \dots, s_{i-1}, s_i, *, \dots, * \rangle$$

where the length of s_i is less than d, then

$$\begin{aligned} b_1 =& \langle s_1, s_2, \dots, s_{i-1}, s_i0, *, \dots, * \rangle \\ b_2 =& \langle s_1, s_2, \dots, s_{i-1}, s_i1, *, \dots, * \rangle \end{aligned}$$

Algorithm 1 Recursive subroutine of Tetris which determines whether b is covered by the boxes in $\mathcal{K}$ (Algorithm 1 in [1])

```
1: TetrisSkeleton($b$):
2: Global parameters: global set of dyadic gap boxes $\mathcal{K}$, SAO $\sigma = (A_1, \dots, A_n)$
3: if there is a box $a \in \mathcal{K}$ such that $b \subseteq a$ then
4:     return (True, $a$)
5: else if $b$ is a unit box then
6:     return (False, $b$)
7: else
8:     $(b_1, b_2) :=$ SplitFirstThickDimension$(b, \sigma)$
9:     $(v_1, w_1) :=$ TetrisSkeleton$(b_1)$
10:    if $v_1$ is False then
11:        return (False, $w_1$)
12:    else if $b \subseteq w_1$ then
13:        return (True, $w_1$)
14:    end if
15:    $(v_2, w_2) :=$ TetrisSkeleton$(b_2)$
16:    if $v_2$ is False then
17:        return (False, $w_2$)
18:    else if $b \subseteq w_2$ then
19:        return (True, $w_2$)
20:    end if
21:    $w :=$ Resolve$(w_1, w_2)$
22:    $\mathcal{K} := \mathcal{K} \cup \{w\}$
23:    return (True, $w$)
24: end if
```

From this definition, there is a straightforward inductive proof which shows that b always takes the required form, so this step is well-defined [1].

Example 2.18. Let $\mathcal{Q}$ be as defined in Example 2.12 and Figure 2.2. Suppose that $\mathcal{K}$ contains the 4 boxes in the box certificate C. Figure 2.3 shows how these boxes can be written as dyadic boxes, and it shows the recursion tree formed when TetrisSkeleton (Algorithm 1) is called on $b = \langle *, *, * \rangle$ with this knowledge base $\mathcal{K}$. The nodes are labelled with the value of b for the respective recursive call. The nodes numbered 1, 2, and 3 correspond to the recursive calls where ordered geometric resolutions are performed on line 21. At node 1, the boxes $\langle *, 00, * \rangle$ and $\langle *, 01, * \rangle$ are resolved to obtain $\langle *, 0, * \rangle$. At

$$\mathcal{K} = \big\{\langle *, 01, * \rangle, \langle *, 11, * \rangle, \langle *, 10, * \rangle, \langle *, 00, * \rangle\big\}$$

$$\begin{tikzcd}
 & & & \langle *, *, * \rangle \arrow[d] & & & \\
 & & & \langle 0, *, * \rangle \arrow[d] & & & \\
 & & & 3\text{: } \langle 00, *, *, \rangle \arrow[dll] \arrow[drr] & & & \\
 & 1\text{: } \langle 00, 0, * \rangle \arrow[dl] \arrow[dr] & & & & 2\text{: } \langle 00, 1, * \rangle \arrow[dl] \arrow[dr] & \\
\langle 00, 00, * \rangle & & \langle 00, 01, * \rangle & & \langle 00, 10, * \rangle & & \langle 00, 11, * \rangle
\end{tikzcd}$$

Figure 2.3: The recursion tree formed by TetrisSkeleton on a simple example

node 2, $\langle *, 10, * \rangle$ and $\langle *, 11, * \rangle$ are resolved to obtain $\langle *, 1, * \rangle$. At node 3, $\langle *, 0, * \rangle$ and $\langle *, 1, * \rangle$ are resolved to obtain $\langle *, *, * \rangle$ which covers the entire output space, so no further resolutions are needed.

Algorithm 2 presents the pseudocode for the main algorithm, Tetris. The dyadic box $b = \langle *, \dots, * \rangle$ is the box which covers the entire output space. Tetris repeatedly calls TetrisSkeleton on b until the entire space is covered by boxes in $\mathcal{K}$. Tetris takes the box cover $\mathcal{B}$ of the query $\mathcal{Q}$ as input, but it initializes the knowledge base $\mathcal{K}$ to the empty set. This ensures boxes from $\mathcal{B}$ are added only when necessary, which is important for proving the certificate-based upper bounds.

Theorem 2.19 states one run time upper bound that Tetris (Algorithm 2) provides. By adding an additional load balancing preprocessing step before calling Tetris, Theorem 2.20 can be proven as well. Both of these bounds depend on the box certificate size of the input box cover, $C_{\square}(\mathcal{B})$. These results motivate finding box covers for $\mathcal{Q}$ of minimum size in order to improve these upper bounds on Tetris' run time. In Chapter 4, we show that it is possible to efficiently compute a box cover for $\mathcal{Q}$ which minimizes the certificate size when the domain ordering is fixed.

Algorithm 2 The Tetris algorithm (Algorithm 2 in [1])

```
1:  TETRIS($\mathcal{B}$):
2:  $\mathcal{K} := \emptyset$
3:  $J := \emptyset$
4:  $(v, w) :=$ TETRISSKELETON($\langle *, \ldots, * \rangle$)
5:  while $v =$ FALSE do
6:      $\mathcal{B}' := \{a \in \mathcal{B} : w \subseteq a\}$
7:      if $\mathcal{B}' = \emptyset$ then
8:          $J := J \cup \{w\}$
9:          $\mathcal{B}' := \{w\}$
10:     end if
11:     $\mathcal{K} := \mathcal{K} \cup \mathcal{B}'$
12:     $(v, w) :=$ TETRISSKELETON($\langle *, \ldots, * \rangle$)
13: end while
14: return $J$
```

Theorem 2.19 (Theorem 4.9 in [1]). *If $\mathcal{Q}$ has treewidth w, there exists an SAO σ such that, on box cover $\mathcal{B}$ of $\mathcal{Q}$ and σ, Tetris runs in time $\widetilde{O}(C_{\Box}(\mathcal{B})^{w+1} + Z)$, where Z is the number of output tuples for $\mathcal{Q}$.*

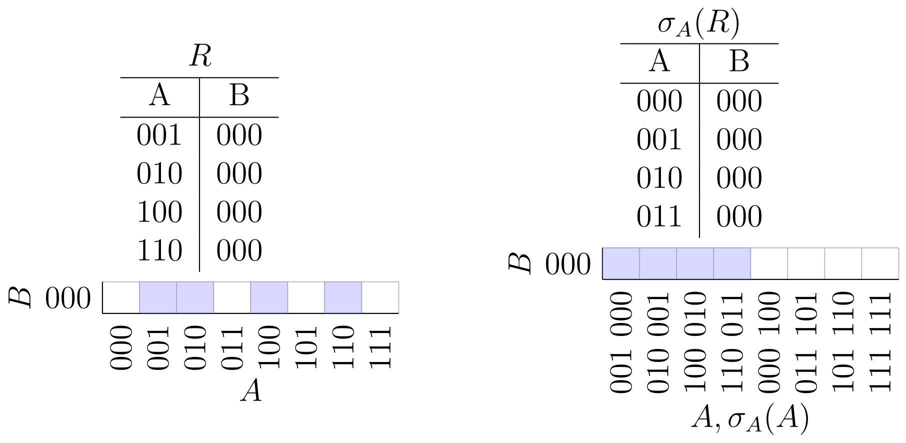
Theorem 2.20 (Theorem 4.11 in [1]). *If $\mathcal{Q}$ has n attributes, on input box cover $\mathcal{B}$, Tetris computes the result of $\mathcal{Q}$ in time $\widetilde{O}(C_{\Box}(\mathcal{B})^{n/2} + Z)$, where Z is the number of output tuples for $\mathcal{Q}$.*

The following section will introduce the research questions we intend to study in this thesis, and lay the groundwork for how we intend to reduce a query's box certificate size.

2.4 Research Questions

Since Tetris takes a box cover as input, and we are working in the setting where the tuples of the relations are input, it is necessary to construct a box cover from the input tuples before invoking Tetris. An algorithm to generate a box cover B for $\mathcal{Q}$ in $\widetilde{O}(N)$ time is known [1], however, it provides no upper bounds on $C_{\Box}(B)$ other than $C_{\Box}(B) \leq \widetilde{O}(N)$. This motivates our first research question.

Domain ordering $\sigma_A = (001, 010, 100, 110, 000, 011, 101, 111)$



R

A	B
001	000
010	000
100	000
110	000

$\sigma_A(R)$

A	B
000	000
001	000
010	000
011	000

Figure 2.4: A relation whose box cover size can be decreased by changing the domain ordering

Research Question 1. *Given a query $\mathcal{Q}$ under a fixed domain ordering where the relations are stored as arrays of input tuples, how can we efficiently construct a box cover B for $\mathcal{Q}$ such that $C_\square(B) = \widetilde{O}\big(C_\square(\mathcal{Q})\big)$?*

The first contribution of this thesis is presented in Chapter 4, where Theorem 4.2 shows that we can construct such a box cover in $\widetilde{O}(N)$ time.

Consider the relation $R(A, B)$ depicted in Figure 2.4. Attributes A and B both have domains $\{0, 1\}^3$, although only the value $B = 000$ is depicted in the image. The following set of gap boxes forms a box cover for R.

$$B = \Big\{\big\langle[000, 111], [001, 111]\big\rangle, \big\langle[000, 000], [000, 000]\big\rangle, \big\langle[011, 011], [000, 000]\big\rangle,$$
$$\big\langle[101, 101], [000, 000]\big\rangle, \big\langle[111, 111], [000, 000]\big\rangle\Big\}$$

It is easy to verify that B is a minimum size box cover for R. This is unsatisfactory, because R is structurally a very simple relation. It can be written as a Cartesian product $R = R_A \times R_B = \{001, 010, 100, 110\} \times \{000\}$. If the values of R_A formed a consecutive interval in $\mathrm{dom}(A)$, we could cover the gaps of R with only 2 gap boxes. If we were free to reorder the columns of R, we could do so in a way that makes all of these 4 A-values one consecutive interval. Assuming all boxes from R are part of the minimum box certificate, if we use this reordered relation and the corresponding box cover as input to Tetris, we obtain a better upper bound for the run time of Tetris (from Theorems 2.19 and 2.20) than otherwise possible.

This example motivates our second research question. We will allow ourselves the freedom to reorder the values of $\mathrm{dom}(A)$ according to an arbitrary *domain ordering*. A domain ordering σ is a set of permutations $\sigma_A \in S_{\mathrm{dom}(A)}$ for each attribute $A \in \mathcal{A}$. Any value in $\mathrm{dom}(A)$ can be mapped to any position in the permuted domain $\mathrm{dom}(\sigma_A(A))$. Figure 2.4 shows the "best" domain ordering for the relation R, in the sense that it minimizes the box certificate size. σ_A can be stored as an array of length at most N, the number of input tuples, because there is no need to store domain values which do not occur anywhere in the query. These values are implicitly placed consecutively at the end of the ordering where they can all be covered by a single gap box. This technical detail ensures that we can transform relations in $\widetilde{O}(N)$ time.

Research Question 2. *Given a query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$, can we efficiently compute a domain ordering σ for the attributes in $\mathcal{A}$ such that the certificate size of the permuted query, $C_\Box(\sigma(\mathcal{Q}))$, is minimized?*

If so, we can compute the domain ordering as a preprocessing step, transform the input according to the ordering, generate box covers of the transformed query to use as input to Tetris, and then transform Tetris' output back to the original ordering afterwards. If P is the run time taken to compute the optimal ordering on $\mathcal{Q}$ and N is the number of input tuples, then by Theorem 4.2 the total preprocessing time will be $\widetilde{O}(P + N)$. All the algorithms to find orderings we will discuss in this thesis will have run time $\Omega(N)$, so the $\widetilde{O}(N)$ cost of generating all maximal dyadic gap boxes for a query will not be a bottleneck.

In Chapter 5 we will define two minimization problems, $\mathrm{CertMinP}_{\mathrm{DomF}}$ and $\mathrm{BoxMinP}_{\mathrm{DomF}}$, which aim to find a domain ordering which minimizes the box certificate size of $\mathcal{Q}$ and the box cover size of $\mathcal{Q}$, respectively. We will show that both of these problems are NP-hard. Little more is known about $\mathrm{CertMinP}_{\mathrm{DomF}}$; we show that it can be computed exactly in the very limited case where the minimum certificate size under any domain ordering is at most 3. For $\mathrm{BoxMinP}_{\mathrm{DomF}}$ we obtain a more positive result. We show that $\mathrm{BoxMinP}_{\mathrm{DomF}}$ can be approximated in $\widetilde{O}(N)$ time to a polynomial factor. This yields the new join processing upper bounds given by Theorems 5.22 and 5.23.

Chapter 3

Related Work

This thesis draws from results in two disparate areas of research. Box cover problems, where the goal is to cover a polygon with as few rectangles as possible, are reviewed in Section 3.1. Join algorithms in relational database management systems are reviewed in Section 3.2.

3.1 Box Cover Problems

Much of this thesis focuses on finding a relation R over n attributes which has a small box cover. The notion of a box cover for a relation was defined in Chapter 2. Informally, a box cover is a set of n-dimensional rectangles which cover all the tuples which are *not* in R. The tuples not in the relation R form a set of n-dimensional, axis-aligned, rectilinear polytopes. These polytopes are not necessarily convex, and they may have holes. The problem of covering a polytope with a minimum number of boxes has been previously studied, primarily in the 2-dimensional case.

3.1.1 2 Dimensions

Most of the existing literature on box cover problems considers the input to be a polygon, represented as a list of vertices, and the objective is to cover the interior of the polygon using as few rectangles as possible. These results are largely still applicable to covering the gaps of a 2-dimensional relation, provided that the polygons are allowed to have holes. The gaps of a 2-dimensional relation will form a set of one or more rectilinear polygons, each of which may have holes. The number of vertices in these polygons is bounded (up to a constant factor) by the number of tuples in the relation, since each vertex is formed where one or more tuples in R are adjacent to some tuple not in R.

The specific problem of finding a minimum rectangle covering for a rectilinear polygon with holes has been well studied. In this thesis, we will refer to this problem as the Box Cover Minimization Problem (BoxMinP). BoxMinP is known to be NP-complete, even when the polygon is hole-free [11]. Furthermore, the problem is known to be MaxSNP-hard for polygons with holes, so there is no polynomial-time approximation scheme for it unless P=NP [5].

Franzblau [15] designed a simple algorithm for BoxMinP and showed that it approximates the optimal solution to a factor of $O(\log n)$, where n is the number of vertices in the polygon. If the polygon is hole-free, the approximation factor improves to 2. Anil Kumar and Ramesh [29] showed a tighter approximation ratio of $O(\sqrt{\log n})$ for the same algorithm on polygons with holes. This is the best known approximation factor for this problem, and it is not known whether a better ratio is possible. This algorithm and its approximation results will serve as motivation for the approach we use in Section 5.2. BoxMinP can also be cast as an instance of the Boolean basis problem, which provides a characterization of a class of rectilinear polygons for which the box cover problem is tractable [32]. We also discuss these connections further in Appendix C.

There is a special case of BoxMinP which is not NP-hard. A polygon P is vertically convex if any vertical line between two points in P is contained entirely within P. When P is rectilinear and vertically convex, the minimum number of rectangles required to cover P can be computed in polynomial time [16].

There is also a more general version of the problem for which a similar approximation ratio is possible. For the set P of polygons with only obtuse interior angles, of which rectilinear polygons are a subset, an algorithm exists to approximate the minimum number of rectangles needed to cover any $p \in P$ to an $O(\log n)$ factor [19]. The same paper also shows an exponential-time algorithm which yields a constant approximation factor.

Other research in this area considers a variant of BoxMinP where the input is a matrix of Boolean values, often representing a black and white image [33, 43]. This version of the problem is less useful for finding box covers of relations because the input size is no longer bounded by the number of tuples in the relation – the tuples which are not in the relation are counted as well.

Another variant of this problem considers covering the input polygon with squares instead of rectangles. This is a more restricted problem which happens to be decidedly easier. Levcopoulos and Gudmundsson [31] presented a polynomial-time algorithm for the square cover problem on any polygon with only obtuse interior angles with an approximation factor of 14. Zwoźniak [53] presented a polynomial-time algorithm for the same problem which produces a square cover of size at most $10.5n + \mu$, where μ is the size of the minimum square cover.

Chan and Grant [7] showed that many types of geometric set cover problems, including covering a set of points with axis-aligned rectangles, are APX-hard, so these problems have no polynomial-time approximation scheme unless P=NP.

Whether there exists a polynomial-time, constant-factor approximation algorithm for covering a rectilinear polygon with rectangles remains an open question. This problem is solved for hole-free polygons, but not for any more general case. This question is of independent interest, but is not vitally important to this thesis because our asymptotic analysis hides all constants and polylogarithmic factors in the $\widetilde{O}$ notation – so an improvement from an $O(\sqrt{\log n})$ approximation to a constant-factor approximation does not affect our upper bounds. It *is* important for this thesis to understand if any of these results generalize to more than 2 dimensions.

3.1.2 3 or More Dimensions

There is very little written about the more general case of covering an n-dimensional polytope with n-dimensional rectangles. In the 3-dimensional case, a polynomial-time algorithm has been published which determines whether a polyhedron P can be covered by the union of polyhedra $P_1 \cup P_2$ under any translation and rotation of P_1 and P_2 [51].

Similarly, the sphere covering problem involves covering an n-dimensional space with spheres, and lower bounds on the density of the sphere cover are known [4, 14]. The sphere covering problem is dual to the sphere packing problem. A survey of results on geometric packing and covering problems can be found in [48]. A book on sphere packings is [10].

Despite the lack of closely related prior work, we will show in Chapter 4 that constructing a box cover for a relation over any number of attributes can be done efficiently enough for our purposes.

3.2 Join Algorithms

Join queries have always been an important part of data processing tasks. There are several classical join algorithms, such as nested loop join, hash join, and sort-merge join [41, p.

454], which have been widely used in database management systems for decades. These algorithms are simple and used to join only two relations. They are often combined using a variety of heuristics to form a query plan for an arbitrarily large join query [28, 45, 46]. Finding the join plan of this type which minimizes the size of the intermediate results is known to be NP-hard [42]. Even though asymptotically faster alternatives exist, these algorithms are still widely used because of their simplicity and the large body of research focused on low-level optimizations and parallelization for these algorithms [30, 44, 13].

Yannakakis' algorithm [52] is an important early result which computes acyclic queries in time $O(N + Z)$.[1] This result was later generalized to an algorithm for arbitrary queries which runs in time $\widetilde{O}(N^w + Z)$, where w is the treewidth of the query [12].

3.2.1 Worst-Case Optimal Join Algorithms

In 2008, Atserias, Grohe, and Marx published an important result that would come to be known as the AGM bound [3]. The AGM bound is a worst-case upper bound on the number of output tuples for a join query based on the shape of the query graph and the number of input tuples. We defined the AGM bound in Section 2.1. A join algorithm is said to be worst-case optimal if it runs in time $\widetilde{O}(\text{AGM}(\mathcal{Q}))$. This result was surprising when it was published, because it showed that the classical join plans were *not* worst-case optimal, even for simple cyclic queries such as the triangle query.

In the following years, several worst-case optimal join algorithms were developed, such as NPRR [36] and Generic Join [37]. One algorithm which predated all of these algorithms, called Leapfrog Triejoin, was later shown to be worst-case optimal too [50]. An implementation of Leapfrog Triejoin has been experimentally shown to outperform classical join algorithms on some common benchmark data [9]. Grohe and Marx also introduced the notion of fractional hypertree width and presented an algorithm that runs in time $\widetilde{O}(N^{fhtw} + Z)$, which also matches the AGM bound. A recent survey on worst-case optimal join algorithms can be found in [34].

3.2.2 Beyond Worst-Case Optimal Join Algorithms

Worst-case optimal join algorithms are an asymptotic improvement over classical join algorithms, but there are query instances where the worst-case AGM bound is still unsatisfactory. For example, if the output of a join query $\mathcal{Q}$ is empty, and there exists a short

[1]A query with treewidth 1 is called acyclic.

proof that the output is empty, we would like to be able to compute the query quicker than $\Omega(\mathrm{AGM}(\mathcal{Q}))$. There are several algorithms which provide upper bounds that are beyond worst-case optimal in this sense.

Other algorithms work for queries of any shape, but seek to exploit highly structured or skewed parts of the input relations. Olteanu and Závodný [39] developed two *factorized* representations of relations, and designed an algorithm with a run time dependent on the size of the factorized input representations. This represents an improvement on the AGM bound for inputs with compact factorized representations.

Joglekar and Ré [22] developed an algorithm which takes advantage of degree information to place a tighter bound on the output size, and therefore computation time. The algorithm is aware of how the input data is skewed, and uses that to split the input into subqueries that can be bounded more tightly than the AGM bound. In a similar vein, taking into account other constraints on the input has allowed information theoretical bounds tighter than the AGM bound, such as functional dependencies [17] and more general degree constraints [2].

Another measure of the simplicity of a query instance's structure is its certificate size. A *comparison certificate* C is a set of comparisons between tuples in the sorted input instance which suffices to verify that the output of the query is correct. Ngo, Nguyen, Ré, and Rudra [35] developed the Minesweeper join algorithm, which runs in time $\widetilde{O}(|C|^{w+1} + Z)$, where Z is the number of output tuples and w is the treewidth of the query. Abo Khamis, Ngo, Ré, and Rudra built upon this result by developing a new, geometric notion of a certificate, the box certificates which we reviewed in Section 2.2. They also designed Tetris, a join algorithm with run time bounds dependent on the box certificate size, as we reviewed in Section 2.3. For every comparison certificate C, there is a corresponding box certificate of size at most $|C|$. In this sense, box certificates are stronger than comparison certificates, and Tetris subsumes the certificate-based results of Minesweeper. Tetris is also worst-case optimal, and we show in Appendix A that its run time is bounded by the run time of Generic Join on any input.

While the factorized bounds, degree-based bounds, and certificate-based bounds all improve on the AGM bound for certain classes of input queries, the relationships between these bounds are not known. The results of this thesis improve on the box certificate-based bounds provided by Tetris for a large class of queries, but we do not make a connection between our bounds and factorized or degree-based bounds. It is left to future work to make connections between the different types of beyond worst-case bounds, and to determine whether domain orderings can also be used to obtain tighter factorized and degree-based bounds.

As discussed in the previous chapter, this thesis explores choosing the domain ordering of the attributes in order to minimize the size of the box certificate. Since box certificates are powerful enough for Tetris to subsume and improve on the AGM bound and comparison certificate-based upper bounds, any tighter bounds represent a further improvement. The following two chapters present this thesis' contributions in this direction.

Chapter 4

Generating a Box Cover

Since Tetris takes a box cover of the input query $\mathcal{Q}$ as input, and our relations are stored as arrays of tuples, it is necessary to compute a box cover for $\mathcal{Q}$ as a preprocessing step. This preprocessing could be performed whenever a relation is updated and the box cover for each relation could be stored as an index to prevent computing the same box cover multiple times. Generally, we assume the worst case, where we must compute the box cover from scratch for every query. As the first contribution of this thesis, we show that one can generate a box cover which is guaranteed to contain the minimum size dyadic box certificate in $\widetilde{O}(N)$ time.

For any relation R with N tuples, there exists a box cover (and a dyadic box cover) for R of size $\widetilde{O}(N)$. This follows from Lemmas 2.14 and 2.15. Furthermore, the total number of *maximal* dyadic gap boxes for R is in $\widetilde{O}(N)$.

Definition 4.1 (Maximal dyadic gap box). A dyadic gap box b for R is *maximal* if for all $A \in \mathcal{A}$ and for all strict prefixes $b'.A$ of $b.A$, there exists a tuple $t \in R$ such that the dyadic box b' obtained from b by replacing the entry $b.A$ with $b'.A$ satisfies $t' \in A$.

Algorithm 3 generates all these maximal boxes in $\widetilde{O}(N)$ time, as proven in Theorem 4.2.

Theorem 4.2. *For any relation R with $N \geq 1$ tuples, Algorithm 3 generates all maximal dyadic gap boxes of R in $\widetilde{O}(N)$ time.*

Proof. Claim 1: Algorithm 3 generates all maximal dyadic gap boxes for the relation R.

Algorithm 3 Generate all maximal dyadic gap boxes for R

```
1:  GenAllMaxBoxes($R$):
2:  $B := \emptyset$
3:  $\overline{B} := \emptyset$
4:  for $t \in R$ do
5:      for every dyadic box $b$ such that $t \in b$ do
6:          for $A \in \text{attr}(R)$ such that $b.A \neq *$ do
7:              Let $b'$ be obtained from $b$ by flipping the last bit of $b.A$
8:              $B := B \cup \{b'\}$
9:              $\overline{B} := \overline{B} \cup \{b\}$
10:         end for
11:     end for
12: end for
13: return $B \setminus \overline{B}$
```

Let b' be a maximal dyadic gap box for R. Let A be an attribute of R for which b' specifies at least one bit. Let b be the dyadic box obtained from b' by flipping the last bit of $b'.A$. Since b' is maximal, b contains at least one tuple $t \in R$. Since b is a dyadic box containing t, some iteration of the for-loop on line 5 will reach the same box b. In the for-loop on line 6, A will be reached at some iteration. In that iteration, the dyadic box b' that is constructed on line 7 will be exactly the box b' we started with. Thus b' is added to the set B. Since b' is a gap box and therefore does not contain any tuples in R, b' will not be added to $\overline{B}$ at any point, and so b' will be returned by Algorithm 3. Note that the returned set does not contain any non-gap boxes of R, since every box which contains any tuple of R is added to $\overline{B}$.

Claim 2: Algorithm 3 has run time $\widetilde{O}(N)$.

The for-loop on line 4 has N iterations. The for-loop on line 5 has $\widetilde{O}(1)$ iterations by Lemma 2.14. The for-loop on line 6 has n iterations. The interior of the loops takes constant time. The total for these three nested for-loops is therefore $O(n \cdot N) = \widetilde{O}(N)$ time. The set difference on line 13 can be done by sorting both B and $\overline{B}$, then iterating through both in lockstep to compute the difference. This takes $\widetilde{O}(N \log N) = \widetilde{O}(N)$ time. $\square$

Theorem 4.2 implies that running Algorithm 3 as a preprocessing step is sufficient to generate a box cover which contains the minimum size box certificate. We make this explicit as follows.

Corollary 4.3. *Given a query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ stored as arrays of input tuples, by using Algorithm 3 as a preprocessing step and then calling Tetris on the resulting box cover, we obtain the following two upper bounds for computing $\mathcal{Q}$.*

$$\widetilde{O}\Big(N + \big(C_\square(\mathcal{Q})\big)^{w+1} + Z\Big)$$
$$\widetilde{O}\Big(N + \big(C_\square(\mathcal{Q})\big)^{n/2} + Z\Big)$$

Proof. By definition, there is some box cover $\mathcal{B} = (B_1, \ldots, B_m)$ and a corresponding box certificate $C \in \mathcal{C}(\mathcal{B})$ such that $|C| = C_\square(\mathcal{Q})$. By Lemma 2.15, we may assume $\mathcal{B}$ contains only dyadic gap boxes and still satisfies $|C| = \widetilde{O}(C_\square(\mathcal{Q}))$. Without loss of generality, we may assume that all dyadic gap boxes in $\mathcal{B}$ are maximal. Otherwise, we could replace any non-maximal box b with some maximal box b' such that $b \subseteq b'$ without increasing the size of the box cover. By running Algorithm 3 on each relation in $\mathcal{R}$, Theorem 4.2 implies that we generate a set of box covers $\mathcal{B}' = (B'_1, \ldots, B'_m)$ such that $B_i \subseteq B'_i$ for each $i \in [m]$. This means that C is also a box certificate for $\mathcal{B}'$. By Theorems 2.19 and 2.20, the run time of Tetris with $\mathcal{B}'$ as input is bounded by $\widetilde{O}\Big(\big(C_\square(\mathcal{Q})\big)^{w+1} + Z\Big)$ and $\widetilde{O}\Big(\big(C_\square(\mathcal{Q})\big)^{n/2} + Z\Big)$, where w is the treewidth of $\mathcal{Q}$, n is the number of attributes, and Z is the number of output tuples. In total, this yields the two bounds in the corollary statement. $\square$

These are the best bounds we can get from Theorems 2.19 and 2.20 when the query instance $\mathcal{Q}$ is fixed, since $C_\square(\mathcal{Q})$ is the minimum certificate size for $\mathcal{Q}$. In order to improve on this bound, we must modify $\mathcal{Q}$ in some way which reduces the box certificate size. The next chapter explores one such way to modify $\mathcal{Q}$: by changing the domain ordering.

Chapter 5

Domain Ordering Problems

In this chapter we will begin studying several optimization problems over the space of *domain orderings*. We aim to find the minimum box certificate size or box cover size which is possible under any domain ordering for the query. To begin, we will define a domain ordering.

Definition 5.1 (Domain ordering). A *domain ordering* for a query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ is a tuple of $|\mathcal{A}|$ permutations $\sigma = (\sigma_A)_{A \in \mathcal{A}}$ where each $\sigma_A \in S_{\mathrm{dom}(A)}$ is a permutation of $\mathrm{dom}(A)$.

Example 5.2. Let A and B be relations over 2-bit domains. Define a relation R as follows. R is presented under the default domain ordering $[00, 01, 10, 11]$ for both attributes.

$$R(A, B) = \big\{\langle 00, 00\rangle, \langle 01, 11\rangle, \langle 10, 00\rangle, \langle 11, 11\rangle\big\}$$

Consider the domain ordering σ where

$$\sigma_A = \sigma_B = \begin{cases} 00 & \mapsto & 00 \\ 01 & \mapsto & 10 \\ 10 & \mapsto & 11 \\ 11 & \mapsto & 01 \end{cases}$$

For notational convenience, we write this as $\sigma_A = \sigma_B = (00, 11, 01, 10)$. Then $\sigma(R)$ is the following relation.

$$\sigma(R)(A, B) = \big\{\langle 00, 00\rangle, \langle 10, 01\rangle, \langle 11, 00\rangle, \langle 01, 01\rangle\big\}$$

The choice of domain ordering can have a massive effect on the box certificate size for the query. Section 5.1 defines a class of queries which have box certificates of size $\Omega(N)$ under the default domain ordering, but have box certificates of size $\widetilde{O}(1)$ under another domain ordering. This illustrates that the choice of domain ordering is important.

There are two distinct optimization problems we will study in this chapter. The primary goal is to find the domain ordering which induces a box certificate of minimum size. The certificate size is the quantity which directly influences the worst-case run time of Tetris. However, the minimum box cover size is an upper bound on the minimum certificate size, so we will also study algorithms which aim to find a domain ordering which induces a box cover of minimum size. We have defined previously two quantities related to these problems. $C_\square(\mathcal{Q})$ denotes the minimum box certificate size of $\mathcal{Q}$, and $K_\square(\mathcal{Q})$ denotes the minimum box cover size of $\mathcal{Q}$. More precisely,

$$\begin{aligned} C_\square(\mathcal{Q}) &= \min_{B\in\mathcal{B}(\mathcal{Q})} \min_{C\in\mathcal{C}(B)} |C| \\ K_\square(\mathcal{Q}) &= \min_{B\in\mathcal{B}(\mathcal{Q})} |B| \end{aligned}$$

where $\mathcal{B}(\mathcal{Q})$ is the set of all box covers for $\mathcal{Q}$ and $\mathcal{C}(B)$ is the set of all box certificates for the box cover B. The problems we will study are defined as follows.

Definition 5.3 (CertMinP$_{\mathbf{DomF}}$). The Domain Flexible Box Certificate Minimization Problem (CertMinP$_{\text{DomF}}$) takes as input a query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$, where each relation is stored as a list of its tuples. It produces a domain ordering σ^* for $\mathcal{Q}$ such that

$$C_\square(\sigma^*(\mathcal{Q})) = \min_\sigma C_\square(\sigma(\mathcal{Q}))$$

Definition 5.4 (BoxMinP$_{\mathbf{DomF}}$). The Domain Flexible Box Cover Minimization Problem (BoxMinP$_{\text{DomF}}$) takes as input a join query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ where $\mathcal{A}$ is a set of attributes and $\mathcal{R}$ is a set of relations over $\mathcal{A}$. The output of BoxMinP$_{\text{DomF}}$ is a domain ordering σ^* such that

$$K_\square(\sigma^*(\mathcal{Q})) = \min_\sigma K_\square(\sigma(\mathcal{Q}))$$

In order to build up to solving these general problems, we will start with the special case when the input is just a single relation. It is not very meaningful to think of $\mathcal{Q}$ as a join query in this case. We are working with a single relation R over some set of attributes $\mathcal{A}$, and we would like to find a domain ordering σ^* which minimizes the size of the box cover necessary to cover $\sigma^*(R)$. In this case, there is no difference between the total number of boxes in the cover and the certificate size, since every box in a minimal cover is necessarily part of the certificate. We will focus only on minimizing the number of boxes needed to cover R.

The simplest possible version of this problem is when we also have only one attribute A. In this case we can easily compute the optimal domain ordering for A. Start with an empty ordering σ. For every $a \in \text{dom}(A)$, if the tuple $\langle a \rangle \in R$, then place a at the beginning of σ, otherwise place a at the end of σ. The result of this iterative process is a domain ordering σ for A which admits a box cover for R of size 1, since every tuple not in R lies in a consecutive block in $\sigma(R)$. So the one-dimensional case can be solved exactly in polynomial time. Adding a second attribute makes the problem considerably harder.

In Section 5.2, we take the next small step forward and try to find the optimal domain ordering of only one attribute in a single 2D relation. Section 5.3 takes the next step by seeking the optimal domain ordering for both attributes in a single 2D relation. Section 5.4 generalizes the problem further to a single relation over an arbitrary number of attributes n. In Section 5.5, we finally study $\text{BoxMinP}_{\text{DomF}}$ on any number of relations and attributes, and present a loose approximation algorithm. In Section 5.6 we present the limited results we obtained for $\text{CertMinP}_{\text{DomF}}$ in general.

In Appendix B, we study a subclass of domain ordering problems that we call bit ordering problems. These problems restrict our choice of orderings so that we can only change the ordering of the bits of each attribute's domain instead of ordering the entire domain arbtirarily. The bit ordering results we obtain are weaker than our results for general domain ordering, but they are presented in the appendix as a starting point for future research.

5.1 A Poorly Ordered Query

The choice of domain ordering can have a significant effect on the certificate size for $\mathcal{Q}$, and therefore on the run time of Tetris on $\mathcal{Q}$. Example 5.5 demonstrates the difference a domain ordering can make. This example is a generalization of Figure 1.1 and is adapted from the proof of Lemma G.5 in [1].

Example 5.5. Let $B^n = \{1^{i-1}0 : i \in [n-1]\} \cup \{1^{n-1}\}$. We will use this set to define our query. For any integers $n \geq 2$ and $d \geq 0$, define $\mathcal{Q}_{n,d} = (\mathcal{R}, \mathcal{A})$ by $\mathcal{A} = \{A_1, \ldots, A_n\}$ and $\mathcal{R} = \{R_{i,j}(A_i, A_j) : i, j \in [n]\}$ as follows. Each attribute A_i has a $(d+n-2)$-bit domain, so $\text{dom}(A_i) = \{0,1\}^{d+n-2}$. For each $i, j \in [n]$, the relation $R_{i,j}$ is defined as

$$R_{i,j}(A_i, A_j) = \left\{ \langle p_1 s_1, p_2 s_2 \rangle : \left(p_1, p_2 \in \{0,1\}^d \wedge s_1, s_2 \in \{0,1\}^{n-2} \wedge (s_1 \neq s_2 \vee s_1 \notin B^{n-1}) \right) \right\}$$

Informally, this definition ensures that only the last $n-2$ bits in each attribute matter, while the first d bits vary over all possible values for any fixed value of the last $n-2$ bits. When $n=3$, $\mathcal{Q}_{3,d}$ is a triangle query in which each relation contains the Cartesian products of all the even numbers with all the even numbers and all the odd numbers with all the odd numbers. $\mathcal{Q}_{3,2}$ is the specific instance of this query which was illustrated in Figure 1.1.

For each $i, j \in [n]$, the set of unit boxes

$$B_{i,j} = \{\langle p_1 s, p_2 s\rangle : p_1, p_2 \in \{0,1\}^d \wedge s \in B^{n-1}\}$$

is the minimum size box cover for $R_{i,j}$ under the default domain ordering. It is worth noting that the join result of $\mathcal{Q}_{n,d}$ is empty, and furthermore, every box $b \in B_{i,j}$ must be part of the certificate. If any one of these boxes is removed, the query is no longer empty. This means the optimal certificate size for this domain ordering is $\Omega(2^{2d})$. Given these gap boxes, Tetris must perform $\Omega(2^{nd})$ resolutions to compute this join, as shown in Lemma G.5 of [1].

However, under a different domain ordering, we can obtain a much better run time for Tetris. Consider a domain ordering σ^* where for each attribute A_i and each string $s \in B^{n-1}$, the domain values with their last $n-2$ bits equal to s are placed consecutively in $\sigma^*_{A_i}$. Under this ordering, for each $i, j \in [n]$ and each $s \in B^{n-1}$, the gap tuples

$$\left\{\langle p_1 s, p_2 s\rangle : p_1, p_2 \in \{0,1\}^d\right\}$$

can be covered by a single gap box. Then each relation requires only $n-2 = \widetilde{O}(1)$ gap boxes to cover all of its gaps. The query $\mathcal{Q}_{3,2}$ under such an ordering σ^* was also shown in Figure 1.1. Again, each of these boxes must be part of the certificate. The certificate size is $\widetilde{O}(1)$ since it depends only on n, which is a query-dependent constant. Under this domain ordering, and given this set of boxes, Tetris is able to compute $\mathcal{Q}_{n,d}$ in $\widetilde{O}(1)$ time.

5.2 Reordering One Attribute in a 2D Relation

In this section, we consider a setting in which there is only one relation R over two attributes. We will model this relation as an $m \times n$ Boolean matrix M where $M_{i,j} = 1$ if and only if the tuple $\langle i, j\rangle$ is *not* in R. Instead of taking an array of tuples as input, we are taking all $n \times m$ Boolean entries of M as input. In this model, a box cover for M is a set of rectangles which covers all the 1-cells of M and does not cover any 0-cells of M. This definition ensures that a box cover of M corresponds exactly to a box cover of the relation R. It is important to note that the number of entries in M is greater than or equal to the number of tuples in R, so the hardness results we present for problems which take M as an input also apply for the version of the problems which instead takes an array of tuples in R as an input. This is because an array of the tuples in R is the same size as an array of all the 0-cells in M, which is less than or equal to size of M.

For now, we are interested in finding the best domain ordering we can get when we are only allowed to reorder the domain of one of the two attributes. In terms of the matrix M, we are looking for a permutation σ_c on the columns of M such that the permuted matrix $\sigma_c(M)$ admits a box cover of minimum size. This problem is defined precisely as follows.

Definition 5.6 (BoxMinP$_{\mathbf{ColF}}$). The *Column Flexible Box Cover Minimization Problem (BoxMinP$_{ColF}$)* takes as input an $m \times n$ Boolean matrix M and produces an ordering σ_c^* on the columns of M such that

$$K_\square(\sigma_c^*(M)) = \min_{\sigma_c} K_\square(\sigma_c(M))$$

In Section 5.2.1, we will show that BoxMinP$_{\text{ColF}}$ is NP-hard. In Section 5.2.2, we show that in a restrictive special case, BoxMinP$_{\text{ColF}}$ is approximable to an $\widetilde{O}(1)$ factor.

5.2.1 BoxMinP$_{\mathbf{ColF}}$ is NP-hard

A similar problem to BoxMinP$_{\text{ColF}}$ is known as the *Consecutive Block Minimization Problem (ConBlkMinP)*. We can show that BoxMinP$_{\text{ColF}}$ is NP-hard with a simple reduction from ConBlkMinP. In order to define ConBlkMinP, we need to define the notion of a *consecutive block*.

Definition 5.7 (Consecutive block). In a Boolean matrix M, a *consecutive block* is a maximal consecutive run of 1-cells in a single row of M, where each is bounded on the left by either the beginning of the row or a 0-cell, and bounded on the right by either the end of the row or a 0-cell.

We will use $\text{cb}(M)$ to denote the total number of consecutive blocks in M, summed over all rows.

Definition 5.8 (ConBlkMinP). The *Consecutive Block Minimization Problem (ConBlkMinP)* takes as input an $m \times n$ Boolean matrix M and produces a permutation $\sigma_c^* \in S_n$ on the columns of M such that

$$\text{cb}(\sigma_c^*(M)) = \min_{\sigma_c} \text{cb}(\sigma_c(M))$$

ConBlkMinP has been previously studied and is known to be NP-complete [27]. It is a generalization of the *consecutive ones problem*, which returns true if there exists a column ordering σ_c such that each row of $\sigma_c(M)$ contains at most one consecutive block. The consecutive ones problem is more widely studied, and can be solved in polynomial time [6]. Theorem 5.9 shows, via a reduction from ConBlkMinP, that $\text{BoxMinP}_{\text{ColF}}$ is NP-hard.

Theorem 5.9. *$BoxMinP_{ColF}$ is NP-hard.*

Proof. We will prove this via a polynomial-time reduction from ConBlkMinP.

Let M be an $m \times n$ Boolean matrix input to ConBlkMinP. Define a transformed matrix M' as follows. Add all the rows of M to M'. Between each pair of consecutive rows, add one additional row containing all 0-cells. A small example of this transformation is seen below.

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \Rightarrow M' = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Note that M' is a $(2m-1) \times n$ matrix, which is polynomial size with respect to M. It suffices to prove that for any column ordering σ_c, $\text{cb}(\sigma_c(M)) = k$ if and only if $\sigma_c(M')$ has a minimal box cover of size at most k.

Suppose that $\text{cb}(\sigma_c(M)) = k$. Every consecutive block in $\sigma_c(M)$ also appears as a consecutive block in the corresponding row of $\sigma_c(M')$. This block c in $\sigma_c(M')$ can be covered by a single box b of height 1 and length equal to the length of c. Every 1-cell in $\sigma_c(M')$ corresponds to some consecutive block of $\sigma_c(M)$, so the set of all k of these boxes b forms a box cover of $\sigma_c(M')$ of size k.

Conversely, suppose that $\mathcal{B}$ is a minimal box cover for $\sigma_c(M')$ of size k. Since there is no 1-cell in M' that has another 1-cell immediately above or below it, each box $b \in \mathcal{B}$ has height 1. Furthermore, we may assume without loss of generality that the length of b is maximal, bounded on the left by the beginning of the matrix or a 0-cell, and bounded on the right by the end of the matrix or a 0-cell. In other words, the 1-cells covered by b are exactly a consecutive block in $\sigma_c(M')$. Additionally, no two boxes $b_1, b_2 \in \mathcal{B}$ can cover the same consecutive block, or $\mathcal{B}$ is not a minimal box cover. Thus, $\sigma_c(M')$ has exactly k consecutive blocks, and so $\sigma_c(M)$ also has exactly k consecutive blocks. □

5.2.2 Approximating $\textbf{BoxMinP}_{\textbf{ColF}}$

Although $\text{BoxMinP}_{\text{ColF}}$ is NP-hard, we can explore some approaches to approximate it. One natural approach is to generalize algorithms which approximate the Box Cover Minimization Problem (BoxMinP), the problem of finding the minimum size box cover for M when the ordering of the rows and columns is fixed. If we can find a column ordering which minimizes the number of boxes produced by some approximation algorithm for BoxMinP, then this ordering is also an approximation for $\text{BoxMinP}_{\text{ColF}}$.

Consider the BoxMinP approximation algorithm of Anil Kumar and Hamesh [29]. In this section, we refer to this algorithm as ApproxMinCover. ApproxMinCover yields a box cover which is an $O(\sqrt{\log n})$-factor approximation of the minimum box cover, where n is the vertical complexity of the rectilinear polygon formed by the 1-cells of the input matrix. For our purposes, it suffices to note that the vertical complexity is loosely bounded by the number of 1-cells (alternatively, the number of 0-cells) in the input matrix.

ApproxMinCover uses a simple heuristic. For each vertical consecutive block of 1-cells in the input matrix M, place one box of width 1 spanning the whole consecutive block. Then, extend each of these boxes horizontally in both directions as far as possible while still covering only 1-cells. Finally, remove any duplicate boxes created by this process.

Consider how the box cover generated by ApproxMinCover changes if we permute the columns of the input matrix. The initial width-1 boxes generated in the first step do not change, since they each span only one column. When these boxes are extended horizontally, how far they can be extended depends on the ordering of the columns. The box b is extended until it is bounded on either side by columns which "block" b in the sense that they contain a 0-cell somewhere in the vertical span of b. If the column where b originated is moved closer to a blocking column for b, the result is a smaller box being generated. These changes in the positions of blocking columns affect the number of duplicate boxes we end up removing in the last step of the algorithm, which determines the size of the resulting box cover.

Suppose that a column c_1 in the input matrix M contains a vertical consecutive block of 1-cells which starts at the i-th row of M and ends at the j-th row of M. We say that c_1 *contains* the interval $[i, j]$. ApproxMinCover must generate a box b which vertically spans from row i to row j, and no further. Suppose another column c_2 also contains a vertical consecutive block from row i to row j. If we can extend b horizontally from c_1 to c_2 without hitting a blocking column for b in between, then the box corresponding to the consecutive block in c_2 will be removed as a duplicate in the last step of ApproxMinCover, which decreases the size of the generated box cover by 1.

$$M: \quad \begin{array}{r} 1: \\ 2: \\ 3: \\ 4: \end{array} \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\text{BNR strings for } M: \quad \begin{array}{rc|ccccc|c} [2,2]: & R & B & N & R & N & R & R \\ [1,3]: & R & R & B & R & N & R & R \\ [3,3]: & R & R & N & B & N & B & R \\ [1,4]: & R & R & R & R & B & R & R \end{array}$$

$$\text{Interval matrix } M' \text{ for } M: \quad \begin{array}{r} [2,2]: \\ [1,3]: \\ [3,3]: \\ [1,4]: \end{array} \begin{bmatrix} 1 & * & 0 & * & 0 \\ 0 & 1 & 0 & * & 0 \\ 0 & * & 1 & * & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 5.1: A matrix, its BNR strings, and its corresponding interval matrix

For any particular vertical interval $[i, j]$, let us define three types of columns. There are *black* columns, which contain exactly the interval $[i, j]$. There are *neutral* columns, which contain some interval $[\ell, k] \supset [i, j]$. There are *red* columns, which contain at least one 0-cell between row i and row j (inclusive). In other words, for interval $I = [i, j]$, black columns exactly contain I, neutral columns contain an interval $I' \supset I$, and red columns block I.

For each unique interval I which is contained in any column of M, any box cover of M generated by ApproxMinCover will contain at least one box with a vertical span which is exactly equal to I. Looking only at the column ordering and the type of each column with respect to I, we can determine how many boxes I will contribute to the box cover generated by ApproxMinCover. Consider the sequence of black, neutral, and red columns as a string of the characters B, N and R (respectively) from left to right as they appear in M, where the left and right boundaries of the matrix are "red" for all intervals. An example of these strings can be found in Figure 5.1. Then, the number of boxes contributed by I is exactly the number of matches for the regular expression $B(N) * R$ in this string. Let $\text{BNR}(M, I)$ denote this number of matches. We can define a corresponding minimization problem as follows.

Definition 5.10 (IntBlkMinP$_{\textbf{ColF}}$). The Column Flexible Interval Block Minimization Problem (IntBlkMinP$_{\text{ColF}}$) takes as input an $m \times n$ Boolean matrix M and produces a column ordering $\sigma_c^* \in S_n$ such that

$$\sum_{I \in \mathcal{I}(\sigma_c^*(M))} \text{BNR}(\sigma_c^*(M), I) = \min_{\sigma_c} \sum_{I \in \mathcal{I}(\sigma_c(M))} \text{BNR}(\sigma_c(M), I)$$

where $\mathcal{I}(M)$ is the set of vertical intervals $[i, j]$ contained in any column of M.

As the above discussion illustrates, IntBlkMinP$_{\text{ColF}}$ produces an ordering which minimizes the size of the box cover produced by ApproxMinCover. This problem is closely related to ConBlkMinP, as shown by the following two results. Theorem 5.11 shows that $\text{IntBlkMinP}_{\text{ColF}}$ is NP-hard by a reduction from ConBlkMinP, and Theorem 5.12 shows that for a special case of the problem, when no interval has any neutral columns in M, $\text{IntBlkMinP}_{\text{ColF}}$ is approximable to a $\frac{3}{2}$ factor by a similar, converse reduction to ConBlk-MinP.

Theorem 5.11. *$\text{IntBlkMinP}_{\text{ColF}}$ is NP-hard.*

Proof. We will prove this by a polynomial-time reduction from ConBlkMinP.

Let M be an $m \times n$ Boolean matrix as input to ConBlkMinP. Construct the matrix M' by inserting a row of all 0-cells between each adjacent pair of rows in M. Then every vertical consecutive block in M' spans only one row, and therefore the intersection between any two distinct intervals $I_1, I_2 \in \mathcal{I}(M')$ is empty ($I_1 \cap I_2 = \emptyset$). This means there are no neutral columns for any interval in $\mathcal{I}(M')$, only black and red columns.

Each interval $I \in \mathcal{I}(M')$ corresponds to one row r from the original matrix M. Since there are no neutral columns for I, for any column ordering σ_c, the number of horizontal consecutive blocks of 1-cells in $\sigma_c(r)$ is exactly equal to the number of consecutive blocks of black columns for I in $\sigma_c(M')$, which is equal to $\text{BNR}(\sigma_c(M'), I)$. That is, $\text{cb}(M') = \sum_{I \in \mathcal{I}(M')} \text{BNR}(M', I)$. By a simple observation, we also have $\text{cb}(M') = \text{cb}(M)$. Thus, σ_c is optimal for $\text{IntBlkMinP}_{\text{ColF}}$ on M' if and only if σ_c is optimal for ConBlkMinP on M. □

The proof of Theorem 5.11 used the fact that when there are no neutral columns for any interval contained in any column of M, $\text{IntBlkMinP}_{\text{ColF}}$ is essentially the same problem as ConBlkMinP. We can characterize this property as being satisfied by M whenever there are no two intervals in any two columns of M such that one of the intervals is a strict subset of the other. Theorem 5.12 states this characterization and shows that in this case, we can approximate $\text{IntBlkMinP}_{\text{ColF}}$ to a constant factor.

Theorem 5.12. *Let M be an input matrix to $\text{IntBlkMinP}_{\text{ColF}}$. If there are no two intervals $I_1, I_2 \in \mathcal{I}(M)$ such that $I_1 \subset I_2$, then $\text{IntBlkMinP}_{\text{ColF}}$ on M can be approximated in polynomial time to a factor of $\frac{3}{2}$.*

Proof. Let M be an $m \times n$ input matrix to $\text{IntBlkMinP}_{\text{ColF}}$ such that no two intervals $I_1, I_2 \in \mathcal{I}(M)$ satisfy $I_1 \subset I_2$. This implies that for any $I \in \mathcal{I}(M)$, there are no neutral columns in M. Let $k = |\mathcal{I}(M)|$. Define the $k \times n$ Boolean matrix M' by adding one row for each interval $I \in \mathcal{I}(M)$ in arbitrary order as follows. If column j is a black column for interval I, then set $M'_{I,j} = 1$. Otherwise, column j is red for I, so set $M'_{I,j} = 0$.

Call M' an *interval matrix*. An example of such a matrix is shown in Figure 5.1, with neutral columns represented as *-cells. For the purposes of this reduction, however, there will be no *-entries.

For any column ordering σ_c, a consecutive block of 1-cells in $\sigma_c(M')$ corresponds exactly to a consecutive block of black columns for the corresponding interval in $\sigma_c(M)$ followed by either a red column or the end of the matrix. Thus the number of consecutive blocks of 1-cells in row I of $\sigma_c(M')$ is equal to $\mathrm{BNR}(\sigma_c(M), I)$, and so $\mathrm{cb}(\sigma_c(M')) = \sum_{I \in \mathcal{I}(M)} \mathrm{BNR}(\sigma(M), I)$.

It has been shown that ConBlkMinP can be approximated in polynomial time to a factor of $\frac{3}{2}$ via a reduction from the metric travelling salesman problem [21], which completes the proof. □

Since IntBlkMinP$_{\mathrm{ColF}}$ minimizes the number of boxes created by ApproxMinCover, Theorem 5.12 also implies that BoxMinP$_{\mathrm{ColF}}$ can be approximated to an $\widetilde{O}(1)$ factor under the same conditions. Furthermore, the number of intervals in $\mathcal{I}(M)$ is polynomial with respect to the number of 0-cells in M. If we ignore any columns of M which are all 1-cells, since these can be placed last in our column ordering and covered with a single box, then the interval matrix for M is also polynomial in size with respect to the number of 0-cells in M. This means that this $\widetilde{O}(1)$ approximation for BoxMinP$_{\mathrm{ColF}}$ still runs in polynomial time, even for the version of the problem which takes an array of tuples as input instead of the full matrix M.

5.3 Reordering Both Attributes in a 2D Relation

In this section we will take the next natural step to generalizing BoxMinP$_{\mathrm{ColF}}$. We are still working with a single relation R over exactly two attributes, but we are free to reorder the domains of both attributes. When we model R as an $m \times n$ Boolean matrix M as we did in the previous section, we are now looking for a pair of orderings $\sigma = (\sigma_r, \sigma_c)$ on the rows and columns of M respectively which induce a minimum size box cover of $\sigma(M)$. This new problem can be defined precisely as follows.

Definition 5.13 (BoxMinP$_{\mathrm{RowColF}}$). The Row and Column Flexible Box Cover Minimization Problem (BoxMinP$_{\mathrm{RowColF}}$) takes as input an $m \times n$ Boolean matrix M and produces a pair of orderings $\sigma^* = (\sigma_r^*, \sigma_c^*)$ of the rows and columns of M such that

$$K_{\square}(\sigma^*(M)) = \min_{\sigma} K_{\square}(\sigma(M))$$

Since Section 5.2 showed that $\text{BoxMinP}_{\text{ColF}}$ is NP-hard, it is reasonable to expect that $\text{BoxMinP}_{\text{RowColF}}$ is NP-hard is well. Theorem 5.15 provides the polynomial-time reduction necessary to prove this. The reduction is from 2ConBlkMinP, a variant of ConBlkMinP in which each row of the input matrix has at most two 1-cells. 2ConBlkMinP was shown to be NP-hard in a note by S. Haddadi [20].

Definition 5.14 (2ConBlkMinP). The 2ConBlkMinP takes as input an $m \times n$ Boolean matrix M such that each row of M contains at most 2 1-cells. The output of the problem is an ordering $\sigma_c^* \in S_n$ on the columns of M such that

$$\text{cb}(\sigma_c^*(M)) = \min_{\sigma_c} \text{cb}(\sigma(M))$$

Theorem 5.15. *$\text{BoxMinP}_{\text{RowColF}}$ is NP-hard.*

Proof. Let M be an $n \times m$ Boolean matrix input to 2ConBlkMinP. We will construct a $(4n) \times (m+2n)$ matrix M' to use as input to $\text{BoxMinP}_{\text{RowColF}}$. For each row r_i $(i \in [n])$ in M, we will insert four rows into M'. Let S_i be the set of columns which contain 1-cells in row r_i of M. Let e_S be the row vector of length $m + 2n$ with value 1 on all indices in $S \subseteq [m + 2n]$, and value 0 everywhere else.

$$\begin{aligned}
p_{i,1} &= e_{\{m+2i-1)\}} \\
r_{i,1} &= e_{S\cup\{m+2i-1\}} \\
r_{i,2} &= e_{S\cup\{m+2i\}} \\
p_{i,2} &= e_{\{m+2i\}}
\end{aligned}$$

An example of this transformation from M to M' is shown in Figure 5.2.

To prove this theorem, it suffices to prove that there exists an ordering σ_c on the columns of M such that $\text{cb}(\sigma_c(M)) = k$ if and only if there exists orderings $\sigma' = (\sigma'_r, \sigma'_c)$ on the rows and columns of M' such that M' admits a box cover of size $k + 2n$.

Proving one direction of this claim is simple. If there exists an ordering σ_c on the columns of M such that $\text{cb}(\sigma_c(M)) = k$, then set σ'_r equal to the default ordering on the rows of M' as defined above. Also, set the last $2n$ columns in σ'_c equal to the default ordering of the last $2n$ columns of M'. Then, set the first m columns in σ'_c equal to σ_c. Then, the 1-cells in the first m columns of $\sigma'(M')$ can be covered by k boxes, and the 1-cells in the last $2n$ columns can be covered by $2n$ boxes, for a total box cover size of $k + 2n$.

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$M' = \left[\begin{array}{cccc|cc|cc|cc}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

Figure 5.2: An example of the 2ConBlkMinP input matrix M and its corresponding M' matrix

Proving the converse is significantly more involved. Let $\sigma' = (\sigma'_r, \sigma'_c)$ be an ordering on the rows and columns of M' such that $\sigma'(M')$ admits a box cover B of size $k + 2n$. We will show through a sequence of steps that we can transform σ'_r to exactly match the default ordering of M' and we can transform σ'_c so that the last $2n$ columns exactly match the last $2n$ columns in the default ordering of M'. After each step, we will modify the box cover B to cover $\sigma'(M')$ under the modified σ', without increasing the total number of boxes in B.

Before we list the steps of this process, we need two definitions. Two rows $r_{i,j}$ and $r_{k,\ell}$ in M' ($i, k \in [n]$ and $j, \ell \in \{1, 2\}$) are *equivalent* if r_i and r_k are equal rows in M (ie. r_i and r_k have 1-cells in the same columns in M). A *run* of equivalent rows is a sequence R of one or more $r_{i,j}$ rows which are consecutive in σ'_r such that all rows in R are equivalent to one another.

Below are the steps we will take to reorder σ'. For each step, we will prove that we can reorder σ' such that the claim is true of $\sigma'(M')$ without increasing the number of boxes, assuming that all of the previous claims hold.

1. Every $r_{i,j}$ row can be made adjacent to some equivalent $r_{k,\ell}$ row.

2. Every run of equivalent $r_{i,j}$ rows can be made to have even length.

3. Every run of equivalent $r_{i,j}$ rows can be made to have length 2.
4. The padding rows $p_{i,j}$ can be made adjacent to their matching $r_{i,j}$ rows.
5. The row order σ'_r can be made to exactly match the default row order of M'.
6. The column order σ'_c can be made to exactly match the default column order of M' on the last $2n$ columns.

Step 1

Claim. *Every $r_{i,j}$ row can be made adjacent to some equivalent $r_{k,\ell}$ row.*

Let $r_1 := r_{i,j}$ be a row which is not adjacent to any equivalent row. Let $r_2 := r_{k,\ell}$ be any row equivalent to r_1. Since r_1 is not adjacent to any equivalent row, and there are an even number of rows equivalent to r_1, there must be some run R of rows equivalent to r_1 with odd length. If R has length 1, we assume r_2 is the one row in R, and therefore r_2 is not adjacent to any equivalent row. If R has length at least 3, we assume r_2 is the second row in R, and therefore r_2 is not adjacent to $p_{k,\ell}$. Let $p_1 := p_{i,j}$ and let $p_2 := p_{k,\ell}$. Let c_{p1} be the column where p_1 has a 1-cell, and let c_{p2} be the column where p_2 has a 1-cell. Let c_1 and c_2 be the columns where r_1 and r_2 both have 1-cells. Let $b_1 \in B$ be the box covering the padding column in r_1 with greatest width. Let $b_2 \in B$ be the box covering the padding column in r_2 with greatest width. Let $b_3 \in B$ be the box covering the padding column in p_1. Let $b_4 \in B$ be the box covering the padding column in p_2.

Our approach in this step will be to remove the rows r_1, r_2, p_1, and p_2 from M', then insert them in the order (p_1, r_1, r_2, p_2) at the bottom of M'. In this order, the 1-cells of these 4 rows can be covered by at most 4 boxes, regardless of the column ordering. A box of width 1 and height 2 can be used to cover the two 1-cells in each of the columns in $\{c_1, c_2, c_{p1}, c_{p2}\}$. To show that this action does not increase the number of boxes in B, it suffices to show that there are at least 4 boxes which can be removed from B when we remove these 4 rows from M'. We will split our analysis into four cases.

1. $b_1 \neq b_3$ and $b_2 \neq b_4$. In this case, all of $\{b_1, b_2, b_3, b_4\}$ are distinct and all 4 of these boxes are removed when we remove the rows r_1, r_2, p_1, p_2.
2. $b_1 \neq b_3$ and $b_2 = b_4$. Since $b_2 = b_4$, r_2 is adjacent to p_2. By our previous assumptions about r_2, this means r_2 is not adjacent to any equivalent row. Without loss of generality, assume that p_2 is directly below r_2. Let r_3 be the row directly above r_2. r_3 is not equivalent to r_2, so there exists a box b_5 covering at least one of c_1 or c_2 in

r_2 which has height 1, since it cannot extend vertically to either p_2 or r_3. b_5 is not equal to b_2, because b_2 has height 2. Now, the set of boxes $\{b_1, b_2, b_3, b_5\}$ is a set of 4 distinct boxes which are removed when we remove the rows $\{r_1, r_2, p_1, p_2\}$.

3. $b_1 = b_3$ and $b_2 \neq b_4$. Since $b_1 = b_3$, r_1 is adjacent to p_1. Suppose without loss of generality that p_1 is directly above r_1. Let r_3 be the row directly below r_1. Since r_1 is not adjacent to any equivalent rows, r_3 is not equivalent to r_1. Therefore, there is a box $b_6 \in B$ covering at least one of c_1 or c_2 in r_1 which has height 1, since it cannot extend vertically to either p_1 or r_3. b_6 is not equal to b_1, since b_1 has height 2. Now the set of boxes $\{b_1, b_2, b_4, b_6\}$ is a set of 4 distinct boxes which are removed when we remove the rows $\{r_1, r_2, p_1, p_2\}$.

4. $b_1 = b_3$ and $b_2 = b_4$. This case can be proven by combining the arguments from the previous two cases. Since $b_2 = b_4$, we can define the box b_5 exactly as in case 2. Since $b_1 = b_3$, we can define the box b_6 exactly as in case 3. Then, $\{b_1, b_2, b_5, b_6\}$ is a set of 4 distinct boxes which are removed from B when we remove the rows $\{r_1, r_2, p_1, p_2\}$.

Step 2

Claim. *Every run of equivalent $r_{i,j}$ rows can be made to have even length.*

Let R_1 be a run of equivalent $r_{i,j}$ rows of odd length. By the claim of step 1, R_1 has length at least 3. Let r_1 be the second row in R_1. Since R_1 has odd length and there are an even number of total rows equivalent to r_1, there exists another run R_2 of rows equivalent to r_1 with odd length. R_2 also has length at least 3.

Let c_{p1} be the column which has a 1-cell only in r_1 and its corresponding padding row. Let $b \in B$ be the box which covers c_{p1} in r_1. Since r_1 is not adjacent to its padding row, b has height 1. If we remove r_1 from M', b can be removed. By inserting r_1 directly below the first row in R_2, a unit box can be used to cover c_{p1} in r_1.

Let r_2 be the first row in R_2. Let c_1 and c_2 be the two columns of M' where r_1 and r_2 share 1-cells. To cover these other two 1-cells in r_1, we can extend vertically the boxes covering c_1 and c_2 in r_2. We may assume these boxes can be extended vertically, because at most two of the rows in R_2 have their c_1 (or c_2) cell covered by a box which streches horizontally from a padding column. That is, there is *some* row in R_2 where the box covering the c_1 (or c_2) cell can be extended vertically to cover the c_1 (or c_2) cell of r_1. This ensures that this transformation can be made without increasing the number of boxes in B. After this, both R_1 and R_2 have even length. Continue this process until step 2 is complete.

Step 3

Claim. *Every run of equivalent $r_{i,j}$ rows can be made to have length 2.*

Let R be a run of equivalent $r_{i,j}$ rows of length greater than 2. Since R has even length, R has length at least 4. Let r_1 be the second row in R and let r_2 be the third row in R. Since R has length at least 4, neither r_1 nor r_2 are adjacent to their respective padding rows, p_1 and p_2. Furthermore, we would like to claim the boxes covering the padding columns in r_1 and r_2 have width 1. We will split into two cases. Below, c_1 and c_2 are the two columns where r_1 and r_2 both have 1-cells.

1. c_1 and c_2 are adjacent. At most 2 of the rows in R have their padding columns adjacent to (c_1, c_2) on either side. This means there is some row r_3 in R for which the box b covering c_1 and c_2 does not also cover its padding column. b can be extended vertically to cover c_1 and c_2 in all rows of R. Then, any boxes covering padding columns for rows in R can be replaced with boxes of width 1, and all of the 1-cells in the rows of R remain covered.

2. c_1 and c_2 are not adjacent. At most 2 rows in R have their padding columns adjacent to c_1 on either side. This means there is some row r_3 in R for which the box b covering c_1 does not also cover its padding column. b can be extended vertically to cover c_1 in all rows of R. The same argument can be applied for c_2. Then, any boxes covering padding columns for rows in R can be replaced with boxes of width 1, and all 1-cells in the rows of R remain covered.

Now, removing p_1 and p_2 removes two boxes from B, since unit boxes must be covering the single 1-cells in p_1 and p_2. Inserting (p_1, p_2) in order in between r_1 and r_2, we can cover the 1-cells in (c_{p1}, p_1) and (c_{p2}, p_2) by extending vertically the width 1 boxes covering (c_{p1}, r_1) and (c_{p2}, r_2). This splits any boxes which vertically streched from r_1 to r_2 into two. There were at most two such boxes, so the total number of boxes in B does not increase. Now R is split into two distinct runs of equivalent r-rows, one of length 2 and one of length $|R| - 2$. This process can be repeated until all runs have length exactly 2.

Step 4

Claim. *The padding rows $p_{i,j}$ can be made adjacent to their matching $r_{i,j}$ rows.*

Let $r_1 := r_{i,j}$ be a row which is not adjacent to its padding row $p_1 := p_{i,j}$. By the claim of step 3, we know r_1 is adjacent to exactly one equivalent r-row, r_2. Let p_2 be the padding row matching r_2. Let c_1 and c_2 be the columns where r_1 and r_2 share 1-cells. Let c_{p1} be the column which has 1-cells only in r_1 and p_1. Let c_{p2} be the column which has 1-cells only in r_2 and p_2. Let b_1 be the box which covers the 1-cell in row r_1 and column c_{p1} of greatest width. Let b_2 be the box which covers the 1-cell in row r_2 and column c_{p2} of greatest width. Let b_3 be the box which covers the 1-cell in p_1. Let b_4 be the box which covers the 1-cell in p_2. We will split into two cases.

1. r_2 is adjacent to p_2. In this case, similar to our argument in step 1, there exists a box $b_5 \in B$ with height 1 which covers c_1 or c_2 (or both) in r_2. By removing the rows $\{r_1, r_2, p_1, p_2\}$, the 4 distinct boxes $\{b_1, b_2, b_3, b_5\}$ are all removed from B. By inserting the rows (p_1, r_1, r_2, p_2) in order at the bottom of the matrix, we can cover their 1-cells with at most 4 boxes, so the total number of boxes in B does not increase.

2. r_2 is not adjacent to p_2. In this case, r_1 is not adjacent to p_1 and r_2 is not adjacent to p_2, so $\{b_1, b_2, b_3, b_4\}$ is a set of 4 distinct boxes in B which are removed if we remove rows $\{r_1, r_2, p_1, p_2\}$. By inserting the rows (p_1, r_1, r_2, p_2) in order at the bottom of the matrix, we can cover their 1-cells with at most 4 boxes, so the total number of boxes in B does not increase.

We can repeat this process until all $r_{i,j}$ rows are adjacent to their matching $p_{i,j}$ rows.

Step 5

Claim. *The row order σ'_r can be made to exactly match the default row order of M'.*

By the claims of steps 3 and 4, all of the rows are now divided into separate 4-row units containing a run of two equivalent $r_{i,j}$ rows surrounded by their two matching padding rows. There are no boxes in B which can stretch vertically across two or more of these separate units, because there are no two $p_{i,j}$ rows which share a 1-cell. Thus, we are free to reorder these units arbitrarily. Order the units so that for all i, the i-th unit contains two $r_{i,j}$ rows which correspond to the i-th row of the original matrix M. The resulting row order σ'_r is then equal to the default ordering of the rows in M', since there exists a column ordering which transforms $\sigma'_r(M')$ back to the original M'. This is sufficient for our purposes.

Step 6

Claim. *The column order σ'_c can be made to exactly match the default column order of M' on the last $2n$ columns.*

For each padding row $p_{i,j}$, the box b covering the single 1-cell in $p_{i,j}$ has width 1. By step 4, each padding row is adjacent to its corresponding $r_{i,j}$ row. This means b extends vertically to also cover the only other 1-cell in its column. Therefore, by moving this column to the right side of the matrix, we do not increase the total number of boxes in B.

Once all of these padding columns have been moved to the right, the boxes covering all of their 1-cells all have width 1. Thus, we can reorder them to exactly match the last $2n$ columns in the default ordering of M' without modifying any boxes in B.

After these 6 steps, the only difference between M' and $\sigma'(M')$ is the ordering of the first m columns. In $\sigma'(M')$, the last $2n$ columns contain an indepenent set of 1-cells of size $2n$, by taking the single 1-cell from each of the $p_{i,j}$ rows. All of these $2n$ 1-cells are independent from all of the 1-cells in the first m columns of σ'_c.

Let σ_c be the ordering of the first m columns in σ'_c. We claim that the first m columns contain an independent size of size $\text{cb}(\sigma_c(M))$. First, any two 1-cells in separate 4-row units are independent from one another, because the padding rows between them contain only 0-cells on the first m columns. If a row of $\sigma_c(M)$ has only one consecutive block, then add a 1-cell from the corresponding 4-row unit to the indepenent set. If a row of $\sigma_c(M)$ contains two consecutive blocks, then there are two 1-cells in the first m columns of the corresponding 4-row unit which are independent from one another. Add both of these to the independent set. Combining the independent sets from the first m columns and the last $2n$ columns, we obtain an indepenent set of size $\text{cb}(\sigma_c(M)) + 2n$.

Furthermore, there exists a box cover for $\sigma'(M')$ of size $\text{cb}(\sigma_c(M)) + 2n$. All of the 1-cells in the last $2n$ columns can be covered by $2n$ boxes. For row r_i in M, if $\sigma_c(r_i)$ contains 1 consecutive block, then the 1-cells in the first m columns of the corresponding 4-row unit in $\sigma'(M')$ can be covered by a single 2×2 box. If $\sigma_c(r_i)$ contains 2 consecutive blocks, then the 1-cells in the first m columns of the corresponding 4-row unit in $\sigma'(M')$ can be covered by two 2×1 boxes. In total, this yields a box cover of size $\text{cb}(\sigma_c(M)) + 2n$.

Since our initial assumption was that $\sigma'(M')$ has a box cover of size $k + 2n$, this implies that $\text{cb}(\sigma_c(M)) \leq k$, which completes the reduction. □

This hardness result is important for all the more general versions of $\text{BoxMinP}_{\text{RowColF}}$ we will study in later sections, because it implies that they are NP-hard as well. As of this writing, there are no known algorithms to approximate $\text{BoxMinP}_{\text{RowColF}}$ with a tighter approximation ratio than the one presented in Section 5.4 which works for a single relation over any number of attributes.

5.4 Reordering All Attributes in an n-ary Relation

In this section we will remove the restriction that our single relation R has exactly two attributes. Instead, suppose R is defined over n attributes $A_1, A_2, \ldots, A_n$. We will no longer model R as a Boolean matrix. Instead, we assume R is stored as an array of its tuples in arbitrary order. This problem is simply $\text{BoxMinP}_{\text{DomF}}$ on a single input relation. The first observation we can make about $\text{BoxMinP}_{\text{DomF}}$ on a single relation is that it is NP-hard. This follows from Theorem 5.15, which implies that it is NP-hard when $n = 2$, and therefore NP-hard in general. For any number of attributes greater than 1, we can invoke $\text{BoxMinP}_{\text{DomF}}$ on any $\text{BoxMinP}_{\text{RowColF}}$ problem with all attributes except two set to a constant value. In the rest of this section, we will work towards establishing an efficient approximation algorithm for $\text{BoxMinP}_{\text{DomF}}$ on a single relation. Section 5.4.1 develops some machinery necessary to prove our approximation upper bound, and Section 5.4.2 presents the algorithm.

5.4.1 Dividing Relations into Hyperplanes

In the simplest case, suppose that the domain ordering σ^* for R which minimizes the box cover size satisfies $K_\square(\sigma^*(R)) = 1$. Fix an arbitrary attribute A_i and let b be the single box in the minimum box cover of $\sigma^*(R)$. We can partition the domain of A_i into two sets: $A_{i,b} = \{a \in \text{dom}(A_i) : a \in \pi_{A_i}(b)\}$ and $A_{i,\bar{b}} = \{a \in \text{dom}(A_i) : a \notin \pi_{A_i}(b)\}$. Here, the notation $\pi_{A_i}(b)$ denotes the set of A_i domain values spanned by the box b. That is, each element $a \in \text{dom}(A_i)$ is either spanned by the box b or it is not. Since there is only one box in the cover, this partition is the only meaningful way to differentiate between two values of $\text{dom}(A_i)$ in R. Consider the domain ordering σ_i obtained by placing all the domain elements in $A_{i,b}$ first in any order, followed by all the elements in $A_{i,\bar{b}}$. Do this for each $i \in [n]$ to obtain the ordering $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ which recovers the box b and therefore attains the minimum box cover size of 1.

This approach yields an exact optimal solution when the minimum box cover size is 1. Intuitively, any domain values which intersect the same set of boxes in the minimum box cover should be placed adjacent to one another. This idea can be generalized to an approximation algorithm which works for any minimum box cover size. In order to make this generalization, the following definition is necessary.

Definition 5.16 (A_i-hyperplane). Let $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$. Let $i \in [n]$ and let $a \in \text{dom}(A_i)$. The *A_i-hyperplane defined by a* is the relation $H(R, A_i, a) = \pi_{\mathcal{A}\setminus\{A_i\}}(\sigma_{A_i=a}(R))$. The *set of A_i-hyperplanes in R* is the set of unique relations in the collection $\{H(R, A_i, a) :$

$a \in \mathrm{dom}(A_i)\}$ and is denoted $\mathcal{H}(R, A_i)$. The *number of A_i-hyperplanes in R* is the size of $\mathcal{H}(R, A_i)$.

The A_i-hyperplane defined by a can be thought of as a "slice" of the n-dimensional space occupied by R containing only the $(n-1)$-dimensional subspace where the A_i attribute is fixed to the value a. This is a natural generalization of "rows" and "columns" which were useful for discussing 2-dimensional relations in Sections 5.2 and 5.3. For each attribute A_i and domain ordering σ, we would like to relate the number of A_i-hyperplanes to the minimum box cover size for $\sigma(R)$. Lemma 5.17 establishes this useful relationship.

Lemma 5.17. *Let σ be a domain ordering for R. Then for each $i \in [n]$, the number of A_i-hyperplanes in R is at most $2 \cdot K_\Box(\sigma(R)) + 1$.*

Proof. Let $i \in [n]$. Let $H_1 = H(R, A_i, a_1)$ and $H_2 = H(R, A_i, a_2)$ be two distinct A_i-hyperplanes such that a_1 and a_2 are adjacent in σ_i. Then there is some tuple t which is in one of H_1 or H_2 but not the other. Without loss of generality, assume $t \in H_1$ and $t \notin H_2$. This means that the tuple $t_1 = \langle a_1, t\rangle \in R$ and $t_2 = \langle a_2, t\rangle \notin R$. Let B be the minimum size box cover for $\sigma(R)$. Let $b \in B$ be a box containing t_2. Since $t_1 \in R$, b does not also contain t_1. Since t_1 and t_2 are adjacent in σ_i, H_1 and H_2 form one of the two boundaries of the box b which are perpendicular to the A_i-axis. The faces of the box b which bound it in the A_i dimension form bounded hyperplanes which are geometrically perpendicular to any line parallel to the A_i-axis. These faces are therefore parallel to every A_i-hyperplane. Every box $b \in B$ has exactly two such boundaries, since it is defined over attribute A_i by the contiguous range in $\mathrm{dom}(A_i)$ which it spans. Therefore, the number of pairs of adjacent, distinct A_i-hyperplanes in σ_i is at most $2 \cdot K_\Box(\sigma(R))$. Figure 5.3 shows an example of this in the 2-dimensional case.

If $h = |\mathcal{H}(R, A_i)|$ is the number of A_i-hyperplanes in R, then the number of pairs of adjacent, distinct A_i-hyperplanes in σ_i is at least $h - 1$, with this minimum being attained if all identical A_i-hyperplanes are placed in consecutive blocks. Thus, $h \leq 2 \cdot K_\Box(\sigma(R)) + 1$. ☐

Lemma 5.17 suggests an idea for an approximation algorithm for $\mathrm{BoxMinP_{DomF}}$ on a single relation. This algorithm is presented in the next section.

5.4.2 Approximating $\mathbf{BoxMinP_{DomF}}$ on a Single Relation

Let σ^* be the optimal domain ordering for R. In this section, we will let $k = K_\Box(\sigma^*(R))$ be the minimum box cover size for R over all domain orderings. If for each $i \in [n]$ we construct

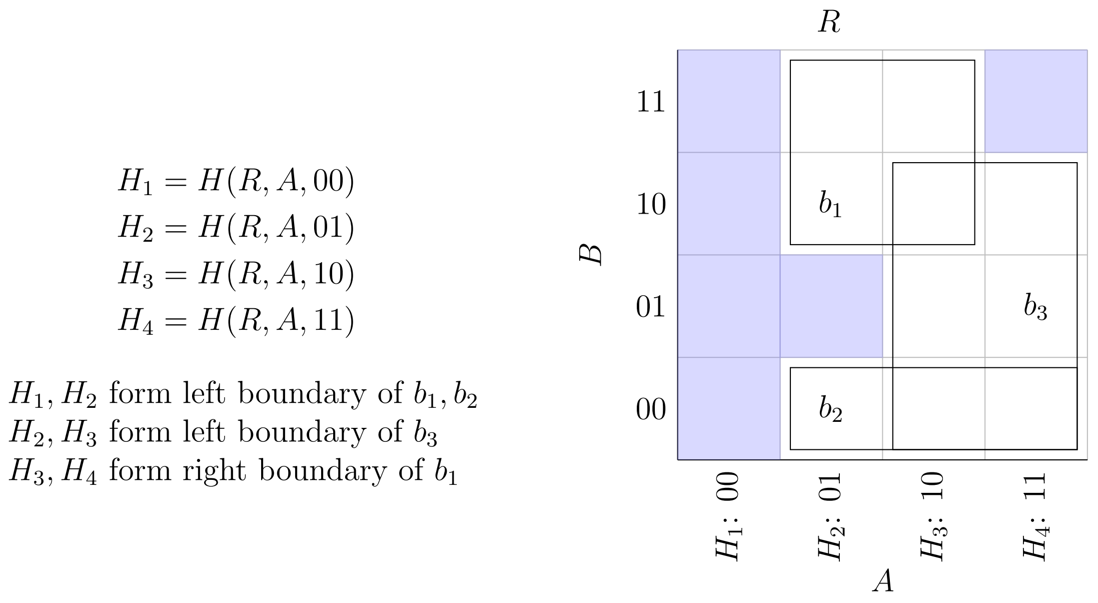


Figure 5.3: An illustration of how adjacent, distinct A-hyperplanes form the boundaries of the gap boxes of R

an ordering σ_i by placing identical A_i-hyperplanes in consecutive blocks, this effectively divides $\sigma(R)$ geometrically into at most $(2k)^n$ grid boxes of identical hyperplanes in each direction, by Lemma 5.17. Each grid box is either filled entirely with tuples from R or does not contain any tuples from R. Each grid box that does not contain any tuples from R can then be covered by a single gap box, yielding a box cover for $\sigma(R)$ of size at most $(2k)^n = \widetilde{O}(k^n)$.

Algorithm 5 presents the pseudocode for this approach. Algorithm 4 is a subroutine which takes a relation R and an attribute A, and splits $\text{dom}(A)$ into a set of buckets such that elements in the same bucket all have identical A-hyperplanes in R. Lemma 5.18 proves the correctness and verifies the $\widetilde{O}(N)$ run time of this subroutine. Algorithm 5 iterates over each attribute and calls Algorithm 4 to compute a corresponding set of buckets. Then, it iterates over the boxes defined by the Cartesian product of these buckets, and generates a gap box for any empty grid cell, resulting in an ordering σ and corresponding box cover for $\sigma(R)$. Theorem 5.19 proves Algorithm 5's correctness and verifies its run time.

Lemma 5.18. *Let R be a relation with $A \in attr(R)$ and $N = |R|$. Then Algorithm 4 runs in time $\widetilde{O}(N)$ on input R and A. Furthermore, Algorithm 4 partitions $dom(A)$ into $\widetilde{O}(k)$*

Algorithm 4 Partition dom(A) into buckets, grouping by equivalent A-hyperplanes

```
SplitBucket($R, A$):
$\phi$ := any attribute ordering of $\{A_1, \ldots, A_n\}$ which places $A$ first
Sort $R$ lexicographically according to $\phi$
$\mathcal{T} := \emptyset$
for $a \in \pi_A(R)$ do
    $\mathcal{T}$.add($[a, \sigma_{A=a}(R)]$)
end for
Sort $\mathcal{T}$ in ascending order of $|\sigma_{A=a}(R)|$
$\mathcal{W} := \emptyset$
$j := 0$
while $j < |\mathcal{T}|$ do
    $i := \mathcal{T}[j].|\sigma_{A=a}(R)|$
    $\mathcal{T}_i := \{[a, \sigma_{A=a}(R)] \in \mathcal{T} : |\sigma_{A=a}(R)|\}$
    Sort $\mathcal{T}_i$ lexicographically according to $\sigma_{A=a}(R)$
    $k := 0$
    while $k < |\mathcal{T}_i|$ do
        $P := \mathcal{T}_i[k][2]$
        $W := \{a \in \mathrm{dom}(A) : [a, \sigma_{A=a}(R)] \in \mathcal{T}_i\}$
        $\mathcal{W}$.add($W$)
        $k := k + |W|$
    end while
    $j := j + |\mathcal{T}_i|$
end while
return $\mathcal{W}$
```

buckets where $k = K_\square(\sigma(R))$ for any domain ordering σ.

Proof. The first sort of R on line 3 requires $O(N \log N)$ comparisons, and each comparison takes $\widetilde{O}(1)$ time.

The for-loop beginning on line 5 simply iterates over the sorted relation R and generates triplets representing groups of tuples which share the same A-value. This takes $O(N)$ time.

The sort of $\mathcal{T}$ on line 8 similarly takes $O(N \log N)$ comparisons with each comparison taking $\widetilde{O}(1)$ time.

The main while-loop beginning on line 11 iterates over the sorted $\mathcal{T}$ in ascending order of the size, i, of the sub-relations that each element in $\mathcal{T}$ corresponds to. For each i, suppose there are k_i elements of $\mathcal{T}$ with size i. Note then that $\sum_i i \cdot k_i = N$. For each i, we sort the elements with size i on line 14 to ensure that all A-values with identical A-hyperplanes in R are adjacent. Then, the interior while-loop beginning on line 16 splits these A-values into buckets such that each bucket contains only A-values with identical A-hyperplanes in R. Constructing the buckets W on line 18 can be done in a total of $O(k_i \cdot i)$ time by advancing in the array $\mathcal{T}_i$ and comparing the corresponding A-hyperplanes until a difference is found. Comparing two A-hyperplanes in $\mathcal{T}_i$ takes $\widetilde{O}(i)$ time. The run time of this is therefore dominated by the sorting, which requires $O(k_i \log k_i)$ comparisons, each of which has cost $\widetilde{O}(i)$. Thus, the total run time of the iteration of the line 11 while-loop is $\widetilde{O}(i \cdot k_i \log k_i)$. Summing over all i, we get

$$\sum_{1 \le i \le N} i \cdot k_i \log k_i \le \log N \sum_{1 \le i \le N} i \cdot k_i = N \log N = \widetilde{O}(N)$$

Since SplitBucket places all elements in $\mathrm{dom}(A)$ with identical A-hyperplanes in R together, Lemma 5.17 implies that the number of buckets returned is at most $2k = \widetilde{O}(k)$. □

Algorithm 5 Compute an ordering and box cover which approximates BoxMinP$_{\text{DomF}}$ on a single relation

```
1:  ApproxBoxMinP1($R(A_1, \ldots, A_n)$):
2:  for $A \in \{A_1, \ldots, A_n\}$ do
3:      $\mathcal{W}[A]$ := SplitBucket($R, A$)
4:      $\sigma[A]$ := Flatten($\mathcal{W}[A]$)
5:  end for
6:  $B := \emptyset$
7:  for $(W_1, \ldots, W_n) \in (\mathcal{W}[A_1], \ldots, \mathcal{W}[A_n])$ do
8:      if $(b = (W_1 \times \cdots \times W_n)) \cap R = \emptyset$ then
9:          $B := B \cup \{b\}$
10:     end if
11: end for
12: return $(\sigma, B)$
```

Theorem 5.19. *Let $R(A_1, \ldots, A_n)$ be a single relation input to BoxMinP$_{DomF}$. Let σ^* be an optimal domain ordering for BoxMinP$_{DomF}$ on R. On input R, Algorithm 5 will produce a domain ordering σ and corresponding box cover B of size $\widetilde{O}(k^n)$ in time $\widetilde{O}(N+k^n)$, where N is the number of tuples in R and $k = K_\square(\sigma^*(R))$.*

Proof. First we will analyze the run time of the for-loop beginning on line 2. It iterates over each attribute $A \in \mathcal{A}$ and calls SPLITBUCKETS to construct $\mathcal{W}[A]$, a partition of the domain of A into some number of buckets. By Lemma 5.18, this takes $\widetilde{O}(N)$ time and partitions $\text{dom}(A)$ into buckets of elements with identical A-hyperplanes. The FLATTEN function simply takes an array of arrays and flattens it into a single array, maintaining that elements of $\text{dom}(A)$ from the same bucket will remain as a consecutive block in the resulting array. This can be done in $O(N)$ time. The run time of this for-loop is therefore $\widetilde{O}(n \cdot N) = \widetilde{O}(N)$.

Now we examine the for-loop beginning on line 7, which iterates over the grid cells formed by the intersections of the buckets created for each attribute and adds a box to our cover if the grid cell is empty. By Lemma 5.17, we have $|\mathcal{W}[A_i]| \leq 2k$ for each $i \in [n]$. So this loop iterates at most $(2k)^n = \widetilde{O}(k^n)$ times. The if-statement in the loop takes $\widetilde{O}(1)$ time, so this for-loop takes $\widetilde{O}(k^n)$ time, bringing the total run time for APPROXBOXMINP1 to $\widetilde{O}(N + k^n)$.

For each $(W_1, \ldots, W_n) \in \mathcal{W}[A_1] \times \cdots \times \mathcal{W}[A_n]$, any tuples $t_1, t_2 \in W_1 \times \cdots \times W_n$ must satisfy $t_1 \in R \wedge t_2 \in R$ or $t_1 \notin R \wedge t_2 \notin R$, since the A_i-hyperplanes for their A_i-values are identical for any $i \in [n]$. This implies the generated set of boxes B is indeed a box cover of $\sigma(R)$. $\square$

It is worth noting that Algorithms 4 and 5 completely ignore any domain values which do not appear in any tuple of the input relation R. These domain values are implicitly assumed to be placed at the end of the resulting domain ordering σ. Their corresponding hyperplanes can then be covered by a single box without affecting the asymptotic size of the resulting box cover.

Revisiting Section 5.3, we can also observe that Algorithm 5 can be easily adapted into an algorithm which approximates $\text{BoxMinP}_{\text{RowColF}}$. For an optimal domain ordering σ^* of the input matrix M, this algorithm would produce a box cover of size $\widetilde{O}((K_{\square}(\sigma^*(M)))^2)$.

The simple notion that values with identical A-hyperplanes in R should be adjacent to one another yielded the results of this section. Future work on this problem should look into other approaches which may lead to better approximation ratios. For example, $\text{BoxMinP}_{\text{DomF}}$ is connected to problems in Boolean algebra. Some notes on these connections are presented in Appendix C.

5.5 Minimizing the Box Cover for Multiple Relations

This section expands on the results of Section 5.4 by generalizing Algorithm 5 to work with more than one input relation. We are now focusing on $\text{BoxMinP}_{\text{DomF}}$ in general. We defined this problem at the beginning of Chapter 5. In this setting, we take a query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ as input, where each relation in $\mathcal{R}$ is represented as a list of its tuples. We are seeking an ordering on the domains of each of the attributes in $\mathcal{A}$ which yields a box cover for $\mathcal{Q}$ of minimum size.

We would like to generalize ApproxBoxMinP1 (Algorithm 5 from Section 5.4) so that it can be used to approximate $\text{BoxMinP}_{\text{DomF}}$. It would be convenient to run ApproxBoxMinP1 on each relation $R \in \mathcal{R}$ separately to obtain an approximately optimal domain ordering σ. However, if an attribute $A \in \mathcal{A}$ is in the attribute sets for two different relations $R_1, R_2 \in \mathcal{R}$, then the ordering σ_A must be the same for both R_1 and R_2. For any two values $a_1, a_2 \in \text{dom}(A)$ such that $H(R_1, A, a_1) = H(R_1, A, a_2)$, it may be the case that $H(R_2, A, a_1) \neq H(R_2, A, a_2)$. It is not always possible to construct an ordering σ_A on $\text{dom}(A)$ such that all identical A-hyperplanes are placed in consecutive blocks in *both* R_1 and R_2.

In order to avoid this problem while maintaining an efficient algorithm, we can use the following process to obtain a good ordering of $\text{dom}(A)$. First, partition $\text{dom}(A)$ into buckets of identical A-hyperplanes in R_1, just as we did in the single relation case. Note that within a single bucket, the ordering of the domain values does not affect any box cover of R_1, so long as all the elements remain in a single consecutive block. For each of the resulting buckets $W \subseteq \text{dom}(A)$, we can further partition W into buckets of identical A-hyperplanes in R_2. If the optimal ordering σ^* yields a box cover for $\sigma^*(R_1)$ of size k_1 and a box cover for $\sigma^*(R_2)$ of size k_2, then this process results in a set of buckets of size $\widetilde{O}(k_1 k_2)$ which partitions $\text{dom}(A)$ and where each bucket contains elements with identical A-hyperplanes in both R_1 and R_2.

This process easily generalizes to any number of relations. We can continue partitioning the resulting buckets until we have iterated over all $R \in \mathcal{R}$ such that $A \in \text{attr}(R)$. Algorithm 6 presents the pseudocode for this approach, which is a generalization of Algorithm 5 and also uses Algorithm 4 as a subroutine. Theorem 5.20 provides the approximation ratio and verifies the run time of this algorithm.

Theorem 5.20. *Let $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ be a query with a total of N input tuples. Let σ^* be an optimal domain ordering for $\text{BoxMinP}_{\text{DomF}}$ on $\mathcal{Q}$. Then Algorithm 6 produces a domain*

Algorithm 6 Compute an approximation to $\text{BoxMinP}_{\text{DomF}}$ and a corresponding box cover

```
1: $\textsc{ApproxBoxMinP}(\mathcal{Q} = (\mathcal{R}, \mathcal{A}))$:
2: for $A \in \mathcal{A}$ do
3:     $\mathcal{W}[A] := \{\cup_{R \in \mathcal{R}: A \in \text{attr}(R)} \pi_A(R)\}$
4:     for $R \in \mathcal{R} : A \in \text{attr}(R)$ do
5:         Sort $R$ by $A$-value
6:         $\mathcal{W}' := \emptyset$
7:         for $W \in \mathcal{W}[A]$ do
8:             $\mathcal{W}'.\text{add}(\textsc{SplitBucket}(\sigma_{A \in W}(R), A))$
9:             $\mathcal{W}'.\text{add}(W \setminus \pi_A(R))$
10:        end for
11:        $\mathcal{W}[A] := \mathcal{W}'$
12:    end for
13:    $\sigma[A] := \textsc{Flatten}(\mathcal{W}[A])$
14: end for
15: for $R \in \mathcal{R}$ do
16:     $\mathcal{B}[R] := \emptyset$
17:     $\{A_1, \ldots, A_n\} := \text{attr}(R)$
18:     for $(W_1, \ldots, W_n) \in (\mathcal{W}[A_1], \ldots, \mathcal{W}[A_n])$ do
19:         if $(b = (W_1 \times \cdots \times W_n)) \cap R = \emptyset$ then
20:             $\mathcal{B}[R] := \mathcal{B}[R] \cup \{b\}$
21:         end if
22:     end for
23: end for
24: return $(\sigma, \mathcal{B})$
```

ordering σ and a box cover for $\mathcal{Q}$ of size $\widetilde{O}(K)$ in time $\widetilde{O}(N + K)$, where

$$K = \sum_{R_1 \in \mathcal{R}} \prod_{A \in attr(R_1)} \prod_{R_2 : A \in attr(R_2)} K_{\square}(\sigma^*(R_2))$$

Proof. First we will analyze the run time of the for-loop beginning on line 2. It iterates over each attribute $A \in \mathcal{A}$ and constructs $\mathcal{W}[A]$, a partition of the domain of A into buckets. $\mathcal{W}[A]$ is initialized to have all the domain values of A which occur anywhere in $\mathcal{Q}$ in a single bucket. Note that after each iteration of the second for-loop beginning on line 4, $\mathcal{W}[A]$ remains a valid partition in the sense that any specific domain element is contained in exactly one bucket. Thus, the total number of elements in the buckets of $\mathcal{W}[A]$ is at most N.

The for-loop beginning on line 4 iterates over each relation R such that $A \in \mathrm{attr}(R)$, and the innermost for-loop beginning on line 7 iterates over all the buckets currently in the partition $\mathcal{W}[A]$ and further partitions each of these into buckets which contain elements with identical A-hyperplanes in R by invoking SplitBucket (Algorithm 4). On line 8, $\sigma_{A\in W}(R)$ denotes the subrelation of tuples in R whose A-value is in the bucket W. This relation can be computed in $\widetilde{O}(|W| + |\sigma_{A\in W}(R)|)$ time by iterating over W and doing a binary search in R for each A-value. By Lemma 5.18, the SplitBucket subroutine call takes $\widetilde{O}(|\sigma_{A\in W}(R)|)$ time. On line 9, we add one additional bucket containing the values from W which do not appear in R. This can also be done in $\widetilde{O}(|W|)$ time.

Since $\sum_{W\in\mathcal{W}[A]}(|W|+|\sigma_{A\in W}(R)|) \leq 2N$, the total amount of work done in one iteration of the for-loop on line 4 is $\widetilde{O}(N)$. There are at most $|\mathcal{A}|$ iterations of this for-loop, and there are $|\mathcal{R}|$ iterations of the for-loop on line 2. Both of these quantities are constant when the query shape is fixed, so the total run time of the for-loop on line 2 is $\widetilde{O}(N)$.

Now, we analyze the run time of the for-loop beginning on line 15. It iterates over each relation $R \in \mathcal{R}$ and then over all possible grid cells formed by the corresponding buckets of the attributes of R. An upper bound on the number of such grid cells is an upper bound on the run time of this for-loop.

Lemma 5.18 implies that the final number of buckets in $\mathcal{W}[A]$ is at most

$$\widetilde{O}\left(\prod_{R:A\in\mathrm{attr}(R)} K_{\square}(\sigma^*(R))\right)$$

Thus, the number of grid cells formed in the relation R_1 by the buckets from $\mathrm{attr}(R_1)$ is bounded by

$$\widetilde{O}\left(\prod_{A\in\mathrm{attr}(R_1)}\ \prod_{R_2:A\in\mathrm{attr}(R_2)} K_{\square}(\sigma^*(R_2))\right)$$

Summing over all relations, we can place the following bound on the total number of iterations of the inner for-loop on line 18, and therefore also on the number of gap boxes in the box cover $\mathcal{B}$ produced by ApproxBoxMinP.

$$\sum_{R\in\mathcal{R}} |\mathcal{B}[R]| \leq \widetilde{O}\left(\sum_{R_1\in\mathcal{R}}\ \prod_{A\in\mathrm{attr}(R_1)}\ \prod_{R_2:A\in\mathrm{attr}(R_2)} K_{\square}(\sigma^*(R_2))\right) = \widetilde{O}(K)$$

□

Theorem 5.20 provides a reasonably tight and granular upper bound on the box cover size produced by Algorithm 6. If we have additional information about individual relations in $\mathcal{R}$, such as bounds on the number of boxes which come from each relation in the optimal solution, then it would be possible to place a tighter bound on the output size using the same algorithm and lemmas. On the other hand, we can also simplify the result of this theorem to obtain a looser bound that is easier to work with. Corollary 5.21 provides one such simplification of this theorem which depends only on the minimum box cover size, the maximum degree of any vertex in the query graph, and the maximum size of any hyperedge in the query graph.

Corollary 5.21. *Let $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ be a query with a total of N input tuples. Let σ^* be an optimal domain ordering for $\text{BoxMinP}_{\text{DomF}}$ on $\mathcal{Q}$, and let $k = K_\square(\sigma^*(\mathcal{Q}))$. Then on input $\mathcal{Q}$, Algorithm 6 computes a domain ordering σ and a corresponding box cover of size $\widetilde{O}(k^{a\cdot r})$ in time $\widetilde{O}(N + k^{a\cdot r})$, where $a = \max_{A\in\mathcal{A}} |\{R \in \mathcal{R} : A \in attr(R)\}|$ and $r = \max_{R\in\mathcal{R}} |attr(R)|$.*

Proof. Note that for each $R \in \mathcal{R}$ we have $k \geq K_\square(\sigma^*(R))$. Let $|\mathcal{C}|$ be the total size of the box covers produced by Algorithm 6 on input $\mathcal{Q}$. We can apply Theorem 5.20 here with ordering σ^* and then use the inequality $K_\square(\sigma^*(R)) \leq k$ to obtain the following.

$$\begin{aligned}
|\mathcal{C}| &\leq \widetilde{O}\left(\sum_{R_1\in\mathcal{R}} \prod_{A\in\text{attr}(R_1)} \prod_{R_2:A\in\text{attr}(R_2)} K_\square(\sigma^*(R_2))\right) \\
&\leq \widetilde{O}\left(\sum_{R_1\in\mathcal{R}} \prod_{A\in\text{attr}(R_1)} \prod_{R_2:A\in\text{attr}(R_2)} k\right) \\
&\leq \widetilde{O}\left(\sum_{R_1\in\mathcal{R}} \prod_{A\in\text{attr}(R_1)} k^a\right) \\
&\leq \widetilde{O}\left(\sum_{R_1\in\mathcal{R}} k^{a\cdot r}\right) \\
&= \widetilde{O}(k^{a\cdot r})
\end{aligned}$$

□

Other similar variants and simplifications of Theorem 5.20 are possible. As of this writing, these loose approximation ratios are the best known polynomial-time results for $\text{BoxMinP}_{\text{DomF}}$. This is the last box cover problem we will study in this thesis. In Section

5.6, we study the analogue of $\text{BoxMinP}_{\text{DomF}}$ which minimizes the box certificate size. First, we will examine what the results of this section mean for computing join queries.

5.5.1 $\text{BoxMinP}_{\text{DomF}}$ and Join Processing

Theorem 5.20 and Corollary 5.21 are the first results in this thesis which are applicable to a join query with any number of attributes, relations, and input tuples. Using Algorithm 6 as a preprocessing step before calling Tetris allows us to express a new upper bound on the run time of join queries. Algorithm 7 presents the simple pseudocode for this approach.

Algorithm 7 Preprocess the Tetris input to obtain a better domain ordering

```
1: TetrisReordered($\mathcal{Q}, \phi$):
2: $(\sigma, \mathcal{B})$ := ApproxBoxMinP($\mathcal{Q}$) (Algorithm 6)
3: $J$ := Tetris($\mathcal{B}, \phi$)
4: return $\sigma^{-1}(J)$
```

Theorems 5.22 and 5.23 state two of the bounds we obtain from Algorithm 7. Recall that $\mathcal{H}(\mathcal{Q})$ denotes the query graph for $\mathcal{Q}$.

Theorem 5.22. *Let $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ be a join query. Let σ^* be an optimal solution to $\text{BoxMinP}_{\text{DomF}}$ on $\mathcal{Q}$ and let $k = K_{\Box}(\sigma^*(\mathcal{Q}))$. Then there exists an SAO ϕ such that Algorithm 7 computes $\mathcal{Q}$ in time*

$$\widetilde{O}(N + k^{a \cdot r \cdot (w+1)} + Z)$$

where N is the number of input tuples, Z is the number of output tuples, a is the maximum degree of a vertex in $\mathcal{H}(Q)$, r is the maximum size of a hyperedge in $\mathcal{H}(\mathcal{Q})$, and w is the treewidth of $\mathcal{Q}$.

Proof. This is a straightforward combination of Theorem 2.19 and Corollary 5.21. The call to ApproxBoxMinP takes $\widetilde{O}(N + k^{a \cdot r})$ time and returns a box cover $\mathcal{B}$ of size $\widetilde{O}(k^{a \cdot r})$. The box certificate size is bounded by the box cover size, so the Tetris call runs in time $\widetilde{O}(|\mathcal{B}|^{w+1} + Z) = \widetilde{O}(k^{a \cdot r \cdot (w+1)} + Z)$. The resulting output relation J has size Z. Converting J back to the original ordering can be done in $\widetilde{O}(Z)$ time if σ is stored in ordered arrays, where the reverse lookup of each domain value can be done in $\widetilde{O}(1)$ time. $\Box$

Theorem 5.23. *Let $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ be a join query. Let σ^* be an optimal solution to $BoxMinP_{DomF}$ on $\mathcal{Q}$ and let $k = K_\Box(\sigma^*(\mathcal{Q}))$. Then Algorithm 7 computes $\mathcal{Q}$ in time*

$$\widetilde{O}(N + k^{a\cdot r\cdot n/2} + Z)$$

where N is the number of input tuples, Z is the number of output tuples, a is the maximum degree of a vertex in $\mathcal{H}(\mathcal{Q})$, r is the maximum size of a hyperedge in $\mathcal{H}(\mathcal{Q})$, and $n = |\mathcal{A}|$.

Proof. Similarly, this theorem is a straightforward combination of Theorem 2.20 and Corollary 5.21. $\Box$

In the next section, we will see that these bounds are not always better than the bounds of Theorems 2.19 and 2.20. However, the example in Section 5.1 demonstrates that there are queries for which these bounds are exponentially smaller than any bound provided by Tetris with a fixed domain ordering.

5.6 Minimizing the Box Certificate for Multiple Relations

In this section we will finally turn our attention to the domain ordering problem which most directly affects the run time of Tetris. Instead of minimizing the box cover size, we now seek to minimize the box certificate size for a query $\mathcal{Q} = (\mathcal{R}, \mathcal{Q})$ with any number of relations over any number of attributes. This problem, $\text{CertMinP}_{\text{DomF}}$, was defined at the beginning of Chapter 5.

As mentioned in previous sections, when the input is a single relation, minimizing the box cover size and minimizing the certificate size are the same problem. Theorem 5.15 proved that the single relation, two attribute case of $\text{CertMinP}_{\text{DomF}}$ is NP-hard, thus the problem is NP-hard in general. Example 5.24 demonstrates that there are queries for which the ordering which minimizes the box cover size can be very different from the ordering which minimizes the certificate size. This serves to illustrate that $\text{BoxMinP}_{\text{DomF}}$ and $\text{CertMinP}_{\text{DomF}}$ are indeed distinct problems.

Example 5.24. Let k be an even positive integer and consider the query

$$\mathcal{Q}_k = R_1(A, B) \bowtie R_2(A, B) \bowtie R_3(A, B) \bowtie R_4(A, B)$$

where the relations are defined as follows. C_1 and C_2 are intermediate relations we will use to define $R_1, \ldots, R_4$.

$$\begin{aligned}
C_1(A,B) =& \bigcup_{i=0}^{k/2} \big\{ \langle 2i, i\rangle, \langle 2i+1, i\rangle, \langle 2i, k-i-1\rangle, \langle 2i+1, k-i-1\rangle, \\
& \langle k+2i, i\rangle, \langle k+2i+1, i\rangle, \langle k+2i, k-i-1\rangle, \langle k+2i+1, k-i-1\rangle \big\} \\
C_2(A,B) =& \bigcup_{i=0}^{k/2} \big\{ \langle 2i, k+i\rangle, \langle 2i+1, k+i\rangle, \langle 2i, 2k-i-1\rangle, \langle 2i+1, 2k-i-1\rangle, \\
& \langle k+2i, k+i, \rangle, \langle k+2i+1, k+i\rangle, \langle k+2i, 2k-i-1\rangle, \langle k+2i+1, 2k-i-1\rangle \big\}
\end{aligned}$$

$$\begin{aligned}
R_1(A,B) =& \big(([0,k-1]\times[0,k-1]) \cup ([k,2k-1]\times[0,k-1]) \\
& \cup ([0,k-1]\times[k,2k-1])\big) \setminus C_1 \\
R_2(A,B) =& \big(([0,k-1]\times[0,k-1]) \cup ([k,2k-1]\times[0,k-1]) \\
& \cup ([k,2k-1]\times[k,2k-1])\big) \setminus C_1 \\
R_3(A,B) =& \big(([0,k-1]\times[k,2k-1]) \cup ([k,2k-1]\times[k,2k-1]) \\
& \cup ([0,k-1]\times[0,k-1])\big) \setminus C_2 \\
R_4(A,B) =& \big(([0,k-1]\times[k,2k-1]) \cup ([k,2k-1]\times[k,2k-1]) \\
& \cup ([k,2k-1]\times[0,k-1])\big) \setminus C_2
\end{aligned}$$

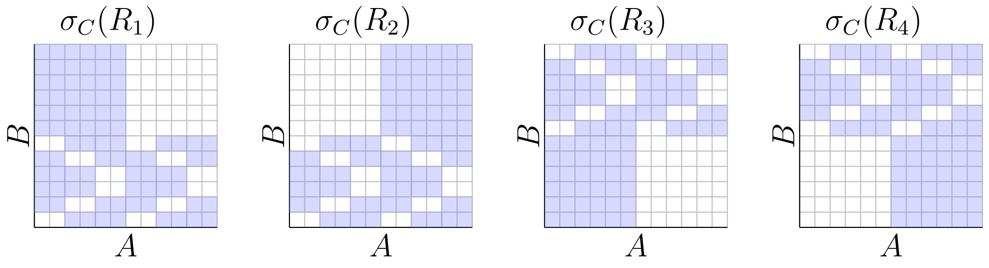
Since all four relations are defined over the same two attributes, this query is simply an intersection of the four relations. The instance of this query when $k = 6$ is shown in Figure 5.4. The tuples of the relations are shaded in blue.

Let $\sigma_1 = (0, 1, \ldots, 2k-1)$ and $\sigma_2 = (0, 1, k, k+1, 3, 4, k+2, k+3, \ldots, k-2, k-1, 2k-2, 2k-1)$. Consider the two domain orderings $\sigma_C = (\sigma_1, \sigma_1)$ and $\sigma_K = (\sigma_2, \sigma_1)$. σ_C is the "natural" ordering for the query (the ordering in which the query was defined), and the optimal box cover size for $\mathcal{Q}_k$ under this ordering is $K_\Box(\sigma_C(\mathcal{Q}_k)) = 8k-4$, since each relation R_i can be covered by $2k-1$ boxes. However, the optimal certificate size under this ordering is $C_\Box(\sigma_C(\mathcal{Q}_k)) = 4$, by taking the single largest gap box from each relation. This illustrates that the certificate size can be much smaller than the box cover size.

Under the ordering σ_K, the size of the optimal box cover decreases, but the size of the optimal certificate increases. The transformed relation is also presented in Figure 5.4. Here, each relation can be covered by $\frac{3}{2}k - 1$ boxes, so $K_\Box(\sigma_K(\mathcal{Q})) = 6k - 4$. But the large boxes which were used as the certificate under the previous ordering have been split into $\frac{k}{2}$ pieces under this ordering, so $C_\Box(\sigma_K(\mathcal{Q}_k)) = 2k$. This illustrates that an ordering with a better box cover size does not necessarily have a better certificate size.

$$
\begin{aligned}
\mathcal{Q}_6 &= R_1(A,B) \bowtie R_2(A,B) \bowtie R_3(A,B) \bowtie R_4(A,B)\\
\sigma_1 &= (0,1,2,3,4,5,6,7,8,9,10,11)\\
\sigma_2 &= (0,1,6,7,2,3,8,9,4,5,10,11)
\end{aligned}
$$

(a) $\mathcal{Q}_6$ under ordering $\sigma_C = (\sigma_1, \sigma_1)$



(b) $\mathcal{Q}_6$ under ordering $\sigma_K = (\sigma_2, \sigma_1)$

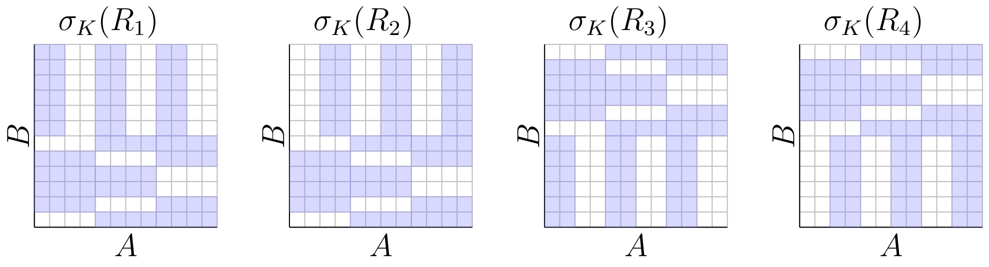


Figure 5.4: A query for which the optimal certificate size and optimal box cover size occur under different domain orderings

Example 5.24 confirms that $\text{CertMinP}_{\text{DomF}}$ is a distinct problem from $\text{BoxMinP}_{\text{DomF}}$. Algorithm 6, which approximates $\text{BoxMinP}_{\text{DomF}}$, does not provide any bound directly related to the minimum box certificate size.

In fact, very little is known about approximating $\text{CertMinP}_{\text{DomF}}$. We are, however, able to solve the problem exactly in the very restricted case where the minimum box certificate size is at most 3, and the query result is empty. This follows from the fact that if the entire output space can be covered by 3 gap boxes from some 3 input relations, one of these 3 gap boxes must span the entire domains of all the attributes except one. Proposition 5.25 states the result formally.

Proposition 5.25. *If $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ is a query for which the output is empty and there exists a domain ordering σ^* for $\mathcal{Q}$ such that $C_\square(\sigma^*(\mathcal{Q})) \leq 3$, then $CertMinP_{DomF}$ on $\mathcal{Q}$ can be solved in polynomial time.*

Proof. If $C_\square(\sigma^*(\mathcal{Q})) \leq 3$, then there is a box cover for $\sigma^*(\mathcal{Q})$ which admits a box certificate C of size at most 3.

Claim: There is a box $b \in C$ and an attribute $A \in \mathcal{A}$ with values $a_1, a_2 \in \text{dom}(A)$ such that $b = [a_1, a_2] \times \big(\times_{A' \in \mathcal{A} \setminus \{A\}} \text{dom}(A') \big)$. In other words, b is a gap box which spans some interval from a_1 to a_2 over attribute A and spans the entire domain of every other attribute in the query.

Proof of claim: This claim can be proven by counting how many "corners" of the output space are covered by the 3 boxes in the certificate. A corner is a tuple in the output space where the value for each attribute A is either the first value in σ^*_A or the last value in σ^*_A. Suppose that all 3 boxes in C have at least 2 attributes for which they do not span the entire domain. Suppose $b \in C$ does not span the entire domain of A or B. Then b either does not span the last value in σ^*_A, or does not span the first value in σ^*_A. The same can be said of b and σ^*_B. The total number of corners in the output space is 2^n, where $n = |\mathcal{A}|$. The restrictions on b mean that it can cover at most 2^{n-2} corners. $3 \cdot 2^{n-2} < 2^n$, so the boxes in C do not cover all the corners of the output space, a contradiction. Therefore the claim holds.

Without loss of generality, we may assume that the values in the interval $[a_1, a_2]$ are the last values in σ^*_A. If they were not, we could modify σ^*_A by moving the interval $[a_1, a_2]$ to the end of σ^*_A without increasing the number of boxes required for the certificate. For some relation $R \in \mathcal{R}$ such that $A \in \text{attr}(R)$, b corresponds to a set of elements in $\text{dom}(A)$ which do not appear in R. For each relation R and each attribute $A \in \text{attr}(R)$, we obtain a partial ordering for σ_A by placing all elements of $\text{dom}(A)$ which do not appear in R at the beginning of σ_A. This ensures that some box which is a "candidate" for b will be recoverable under the domain ordering we are constructing.

Algorithm 8 Compute a domain ordering matching the optimal certificate size when $C_\square(\sigma^*(\mathcal{Q})) \leq 3$

```
 1: ComputeCertMinPLeq3($\mathcal{Q} = (\mathcal{R}, \mathcal{A}), i$):
 2: if $i = 1$ then
 3:     if $\exists R \in \mathcal{R}$ such that $R = \emptyset$ then
 4:         return arbitrary domain ordering $\sigma$
 5:     else
 6:         return False
 7:     end if
 8: end if
 9: for $R \in \mathcal{R}$ do
10:     for $A \in \text{attr}(R)$ do
11:         $D := \{a \in \text{dom}(A) : a \notin \pi_A(R)\}$
12:         $\sigma :=$ ComputeCertMinPLeq3($\sigma_{A \notin D}(\mathcal{Q}), i - 1$)
13:         if $\sigma \neq$ False then
14:             $\sigma[A] := \text{append}(\sigma[A], D)$
15:             return $\sigma$
16:         else
17:             return False
18:         end if
19:     end for
20: end for
```

Algorithm 8 iterates over all these candidates b with the for-loops on lines 9 and 10. Here, b is the box which spans the elements in D on attribute A, but spans the entire domains of each other attribute. Fixing this b, Algorithm 8 recursively solves this problem (line 12) on the query $\mathcal{Q}'$ obtained from $\mathcal{Q}$ by restricting each relation $R \in \mathcal{R}$ to $R' = \{t \in R : t.A \notin D\}$ and restricting the domain of A to $\text{dom}(A) \setminus D$.

Since $\mathcal{Q}$ has an optimal certificate size of at most 3, for some R and A, $\mathcal{Q}'$ has an optimal certificate size of at most 2. Inductively, this recursion occurs at most twice before we reach the base case, where the optimal certificate is of size 1, and $\mathcal{Q}'$ is a query containing at least one empty relation. If ComputeCertMinPLeq3 is initially called with parameter $i = 3$, any recursive paths with length longer than 3 will be terminated within the if-block on line 1, ensuring that this algorithm runs in polynomial time. $\square$

Proposition 5.25 shows that $\text{CertMinP}_{\text{DomF}}$ can be solved exactly in polynomial time if we add two strong assumptions about $\mathcal{Q}$. The proof of this proposition does not apply when the minimum certificate size is 4 or greater, because the claim that begins the proof does not hold in those cases. Beyond this result and NP-hardness, nothing more is known about $\text{CertMinP}_{\text{DomF}}$. An important direction for future research is finding an approximation algorithm for $\text{CertMinP}_{\text{DomF}}$ in general, since such an algorithm would directly decrease the upper bound on Tetris' run time.

Chapter 6

Conclusions

The previous chapters defined several domain ordering problems, and presented various hardness and approximation results for them. In Appendix B, we present preliminary results for several bit ordering problems, a subclass of domain ordering problems. Our results for all of these problems are summarized in Table 6.1.

The results for the domain ordering problems in Chapter 5 are more substantial and promising. We established a polynomial-time, polynomial-factor approximation algorithm (Algorithm 6) for $\text{BoxMinP}_{\text{DomF}}$ using only the simple observation that identical A-hyperplanes should be made adjacent to one another whenever possible. Since this algorithm provides a bound for arbitrary join queries, it allowed us to prove Theorems 5.22 and 5.23, new beyond worst-case join processing bounds. These were obtained by adding an efficient preprocessing step to Tetris (Algorithm 7), improving on any previously known bound in cases where the input has a bad domain ordering.

The approximation factor to $\text{BoxMinP}_{\text{DomF}}$ used in these bounds is a high-degree polynomial when the query shape is complex. The question remains open whether this approximation factor can be improved to a smaller polynomial, or better yet, to an $\widetilde{O}(1)$ subpolynomial factor. Despite the loose worst-case approximation factor, there exist queries such as the example in Section 5.1 for which Algorithm 6 produces a box cover of size exponentially smaller than what would be possible without domain reordering. On these queries, Algorithm 7 runs faster than any previously known join algorithm.

Unfortunately, these positive results for $\text{BoxMinP}_{\text{DomF}}$ did not generalize to provide similar results for $\text{CertMinP}_{\text{DomF}}$, the analogous box certificate minimization problem. Since the box certificate size directly influences the run time of Tetris, approximating

Problem	Type	NP-hard	Approximable
$\text{BoxMinP}_{\text{ColF}}$	Domain ordering Cover & cert.	Yes Theorem 5.9	$\widetilde{O}(1)$ if no sub-intervals Theorem 5.12
$\text{BoxMinP}_{\text{RowColF}}$	Domain ordering Cover & cert.	Yes Theorem 5.15	$\widetilde{O}\big(K_{\square}(\sigma^*(\mathcal{Q}))^2\big)$ Theorem 5.19
$\text{BoxMinP}_{\text{DomF}}$ single-relation	Domain ordering Cover & cert.	Yes Theorem 5.15	$\widetilde{O}\big(K_{\square}(\sigma^*(\mathcal{Q}))^n\big)$ Theorem 5.19
$\text{BoxMinP}_{\text{DomF}}$	Domain ordering Cover	Yes Theorem 5.15	$\widetilde{O}\big(K_{\square}(\sigma^*(\mathcal{Q}))^{a\cdot r}\big)$ Corollary 5.21
$\text{CertMinP}_{\text{DomF}}$	Domain ordering Certificate	Yes Theorem 5.15	Exact if $C_{\square}(\sigma^*(\mathcal{Q})) \leq 3$ Proposition 5.25
$\text{CertMinP}_{\text{BitF}}$	Bit ordering Certificate	–	$\widetilde{O}(1)$ for single rel. & attr. Corollary B.12
$\text{CertMinP}_{\text{GBO}}$	Bit ordering Certificate	–	$\widetilde{O}(1)$ for single relation Corollary B.12
$\text{BoxMinP}_{\text{GBO}}$	Bit ordering Cover	Yes when $\mathcal{Q}$ not fixed Corollary B.16	$\widetilde{O}(1)$ for single relation Corollary B.12

Table 6.1: A summary of the hardness and approximation results in this thesis

$\text{CertMinP}_{\text{DomF}}$ (or proving an inapproximability result) is the most important open problem identified in this thesis.

The bit ordering problems studied in Appendix B remain largely unsolved. It remains an open question whether $\text{CertMinP}_{\text{BitF}}$ or $\text{CertMinP}_{\text{GBO}}$ can be solved in $\widetilde{O}(\text{poly}(N))$ time when the query shape is constant. One direction for future research is to either find such an algorithm, or prove that these problems are NP-hard even when the query shape is constant. Finding an $\widetilde{O}(1)$-factor approximation algorithm for the general case of any of these problems would also be valuable, since it would yield a new beyond-worst case join processing bound. Another direction for future research is discussed briefly in Appendix C. Box cover problems are related to problems in Boolean algebra. Exploring these connections further may lead to better results for $\text{CertMinP}_{\text{DomF}}$ and $\text{BoxMinP}_{\text{DomF}}$.

References

[1] Mahmoud Abo Khamis, Hung Q Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems (TODS)*, 41(4):22, 2016.

[2] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 327–342, New York, NY, USA, 2016. ACM.

[3] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.

[4] RP Bambah and H Davenport. The covering of n-dimensional space by spheres. *Journal of the London Mathematical Society*, 1(2):224–229, 1952.

[5] Piotr Berman and Bhaskar DasGupta. Complexities of efficient solutions of rectilinear polygon cover problems. *Algorithmica*, 17(4):331–356, 1997.

[6] Kellogg S Booth and George S Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[7] Timothy M Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric set cover. In *Proc. 23rd Canadian Conference on Computational Geometry*, pages 431–436, 2011.

[8] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *ACM Sigmod Record*, 26(1):65–74, 1997.

[9] Shumo Chu, Magdalena Balazinska, and Dan Suciu. From theory to practice: Efficient join query evaluation in a parallel database system. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 63–78, New York, NY, USA, 2015. ACM.

[10] John Horton Conway and Neil James Alexander Sloane. *Sphere Packings, Lattices and Groups*, volume 290. Springer Science & Business Media, 2013.

[11] Joseph C Culberson and Robert A Reckhow. Covering polygons is hard. *J. Algorithms*, 17(1):2–44, 1994.

[12] Rina Dechter and Judea Pearl. Tree-clustering schemes for constraint-processing. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*, AAAI'88, pages 150–154. AAAI Press, 1988.

[13] David J DeWitt, Randy H Katz, Frank Olken, Leonard D Shapiro, Michael R Stonebraker, and David A. Wood. Implementation techniques for main memory database systems. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 1–8, New York, NY, USA, 1984. ACM.

[14] Paul Erdős and CA Rogers. The covering of n-dimensional space by spheres. *Journal of the London Mathematical Society*, 1(3):287–293, 1953.

[15] Deborah S. Franzblau. Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles. *SIAM Journal on Discrete Mathematics*, 2(3):307–321, 1989.

[16] Deborah S Franzblau and Daniel J Kleitman. An algorithm for covering polygons with rectangles. *Information and Control*, 63(3):164–189, 1984.

[17] Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16:1–16:35, June 2012.

[18] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG)*, 11(1):4, 2014.

[19] Joachim Gudmundsson and Christos Levcopoulos. Close approximations of minimum rectangular coverings. *Journal of combinatorial optimization*, 3(4):437–452, 1999.

[20] Salim Haddadi. A note on the NP-hardness of the consecutive block minimization problem. *International Transactions in Operational Research*, 9:775–777, 11 2002.

[21] Salim Haddadi and Zoubir Layouni. Consecutive block minimization is 1.5-approximable. *Information Processing Letters*, 108(3):132 – 135, 2008.

[22] Manas R Joglekar and Christopher M Ré. It's all a matter of degree: Using degree information to optimize multiway joins. In *19th International Conference on Database Theory (ICDT 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[23] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[24] KH Kim and FW Roush. Inclines and incline matrices: A survey. *Linear Algebra and its Applications*, 379:457–473, 2004.

[25] Ki Hang Kim. *Boolean Matrix Theory and Applications*, volume 70. Dekker, 1982.

[26] Phokion G Kolaitis and Moshe Y Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000.

[27] L. Kou. Polynomial complete consecutive information retrieval problems. *SIAM Journal on Computing*, 6(1):67–75, 1977.

[28] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of nonrecursive queries. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 128–137. Morgan Kaufmann Publishers Inc., 1986.

[29] VS Anil Kumar and H Ramesh. Covering rectilinear polygons with axis-parallel rectangles. *SIAM Journal on Computing*, 32(6):1509–1541, 2003.

[30] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: A survey. *SIGMOD Rec.*, 40(4):11–20, January 2012.

[31] Christos Levcopoulos and Joachim Gudmundsson. Approximation algorithms for covering polygons with squares and similar problems. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 27–41. Springer, 1997.

[32] Anna Lubiw. The boolean basis problem and how to cover some polygons by rectangles. *SIAM Journal on Discrete Mathematics*, 3(1):98–115, 1990.

[33] Dipen Moitra. Finding a minimal cover for binary images: An optimal parallel algorithm. *Algorithmica*, 6(1-6):624–657, 1991.

[34] Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, SIGMOD/PODS '18, pages 111–124, New York, NY, USA, 2018. ACM.

[35] Hung Q Ngo, Dung T Nguyen, Christopher Ré, and Atri Rudra. Beyond worst-case analysis for joins with Minesweeper. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 234–245. ACM, 2014.

[36] Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):16, 2018.

[37] Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: New developments in the theory of join algorithms. *CoRR*, abs/1310.3314, 2013.

[38] Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

[39] Dan Olteanu and Jakub Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):2, 2015.

[40] Phillip L Poplin and Robert E Hartwig. Determinantal identities over commutative semirings. *Linear Algebra and its Applications*, 387:99–132, 2004.

[41] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw Hill, 2000.

[42] Wolfgang Scheufele and Guido Moerkotte. On the complexity of generating optimal plans with cross products. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 238–248. ACM, 1997.

[43] David S Scott and S Sitharama Iyengar. TID – a translation invariant data structure for storing images. *Communications of the ACM*, 29(5):418–429, 1986.

[44] Eugene J Shekita, Honesty C Young, and Kian-Lee Tan. Multi-join optimization for symmetric multiprocessors. In *Proceedings of the 19th International Conference on Very Large Data Bases*, pages 479–492. Morgan Kaufmann Publishers Inc., 1993.

[45] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *The VLDB Journal*, 6(3):191–208, August 1997.

[46] Arun Swami and Anoop Gupta. Optimization of large join queries. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, pages 8–17, New York, NY, USA, 1988. ACM.

[47] Yijia Tan. On invertible matrices over antirings. *Linear Algebra and its Applications*, 423(2-3):428–444, 2007.

[48] Gábor Fejes Tóth and Wlodzimierz Kuperberg. A survey of recent results in the theory of packing and covering. In *New Trends in Discrete and Computational Geometry*, pages 251–279. Springer, 1993.

[49] Charalampos E Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *Proceedings of the 2008 8th IEEE International Conference on Data Mining*, pages 608–617. IEEE Computer Society, 2008.

[50] Todd L. Veldhuizen. Leapfrog triejoin: A simple, worst-case optimal join algorithm. In *Proc. 17th International Conference on Database Theory (ICDT)*, pages 96–106, 2014.

[51] Cao An Wang, Bo-Ting Yang, and Binhai Zhu. On some polyhedra covering problems. *Journal of Combinatorial Optimization*, 4(4):437–447, 2000.

[52] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases*, pages 82–94. VLDB Endowment, 1981.

[53] Grazyna Zwoźniak. A better approximation algorithm for covering polygons with squares. Technical report, Institute of Computer Science, University of Wroclaw, 1998.

APPENDICES

Appendix A

Tetris Subsumes Generic Join

The join algorithms Tetris and Generic Join are both known to run in time $\widetilde{O}(\text{AGM}(\mathcal{Q}))$ for some global attribute order (GAO) [1, 37]. It is also known that the certificate-based bounds for Tetris are much smaller than the AGM bound for many queries. Intuitively, these facts seem to suggest that the worst-case runtime of Tetris is bounded by the runtime of Generic Join on the same input query. Theorem A.2 confirms that this intuition is true.

The proof of this theorem is a generalization of the proof that Tetris meets the AGM bound given in [1]. The theorem statement uses the terms GAO and SAO. This terminology is also borrowed from [1]. In this context, the GAO is the ordering of the attributes of $\mathcal{Q}$ that Generic Join will process the query in. The SAO is the ordering of the attributes that Tetris will process the query in. In order to state the theorem we need one more definition.

Definition A.1. *SAO-consistent gap boxes* are dyadic gap boxes of the form

$$\langle s_1, s_2, \ldots, s_i, *, \ldots, * \rangle$$

for some $i \in \{1, \ldots, |\mathcal{A}|\}$, where the attributes are sorted according to the SAO and $|s_j| = d$ for all $j < i$.

Theorem A.2. *Let $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ be a join query. Let $T_{GJ}(\mathcal{Q})$ be the runtime of Generic Join on $\mathcal{Q}$ with fixed GAO σ. Then with SAO σ and σ-consistent gap boxes, Tetris computes $\mathcal{Q}$ in time $\widetilde{O}(T_{GJ}(\mathcal{Q}))$.*

Proof. We will proceed by induction on $|\mathcal{A}|$. When $|\mathcal{A}| = 1$, there is only one choice of SAO, and $\mathcal{Q}$ is an intersection of $|\mathcal{R}|$ single-attribute relations, which Generic Join solves in time $\widetilde{O}(\min_{R\in\mathcal{R}} |R|)$.

Let $R = \text{argmin}_{R\in\mathcal{R}}|R|$. Consider the tree T consisting of all the boxes b that Tetris visits in all its recursive calls while computing $\mathcal{Q}$. Each leaf of T is either covered by a gap box or is an output tuple. Consider any tuple $t \in R$. t is either an output tuple, in which case it is a leaf of T, or it is covered by a gap box from another relation. Any gap box leaf of T which does not contain any tuples $t \in R$ can be assumed to be a gap box from R. So the number of leaves of T is at most the sum of $|R|$ and the number of gap boxes from R, which is bounded by $\widetilde{O}(|R|)$. Therefore the number of nodes in T is in $\widetilde{O}(|R|) = \widetilde{O}(\min_{R\in\mathcal{R}}|R|)$.

Now suppose that $n = |\mathcal{A}| > 1$, and assume that the theorem statement holds whenever for any query with fewer than n attributes. Generic-Join chooses an arbitrary set $I \subset \mathcal{A}$ such that $I \neq \emptyset$ and $I \neq \mathcal{A}$ to split the query. Since the GAO is fixed to σ, we know the attributes in I come before the attributes in $\mathcal{A} \setminus I$ in σ.

Let B_1 be the set of boxes Tetris visits which have all wildcards in the attributes of $\mathcal{A} \setminus I$. Tetris visits these boxes exactly as if it were computing the join $Q_I = \Join_{R\in\mathcal{R}} \pi_I(R)$. That is, Tetris will visit the node with the first $d \cdot |I|$ bits specified (and will not find a box which covers it) if and only if the corresponding tuple $t \in Q_I$. Therefore, by induction, $|B_1| \in \widetilde{O}(T_{GJ}(Q_I))$.

Now for each $t \in Q_I$, let $B_2(t)$ be the set of boxes Tetris visits which have no wildcards in the I attributes, and the bits in I which are fully specified correspond to the tuple t. Tetris visits the boxes in $B_2(t)$ exactly as if it were computing $Q_t = \Join_{R\in\mathcal{R}_{\mathcal{A}\setminus I}} \pi_{\mathcal{A}\setminus I}(\sigma_{I=t}(R))$. Note that since the GAO is fixed, all the I sets chosen by the Q_t subqueries will indeed be compatible with the SAO of Tetris. By induction, $|B_2(t)| \in \widetilde{O}(T_{GJ}(Q_t))$. Generic Join recursively computes the queries Q_I and Q_t for each $t \in Q_I$, so $T_{GJ}(\mathcal{Q}) \leq T_{GJ}(Q_I) + \sum_{t\in Q_i} T_{GJ}(Q_t)$.

Combining the above facts, we have the following where $T_{Tet}(Q)$ is the runtime of Tetris on Q.

$$T_{Tet}(Q) = \widetilde{O}(|B_1| + \sum_{t\in Q_I} |B_2(t)|) \leq \widetilde{O}(T_{GJ}(Q_I) + \sum_{t\in Q_I} T_{GJ}(Q_t)) \leq \widetilde{O}(T_{GJ}(Q))$$

□

Appendix B

Finding an Optimal Bit Ordering

In this appendix we will explore the benefits of changing the *bit ordering* that Tetris uses to compute join queries.

Definition B.1 (Bit ordering). Given a query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ where each attribute $A \in \mathcal{A}$ has $\text{dom}(A) = \{0,1\}^d$, a *bit ordering* of $\mathcal{Q}$ is a tuple of $|\mathcal{A}|$ permutations $\phi = (\phi_A)_{A\in\mathcal{A}} \in \times_{A\in\mathcal{A}} S_d$. That is, each ϕ_A is a permutation of the d bits of $\text{dom}(A)$.

We will use $\phi(\mathcal{Q})$ to denote the query obtained by permuting all the attributes and relations of $\mathcal{Q}$ according to ϕ. When we run Tetris on $\mathcal{Q}$ with bit ordering ϕ, this means the dyadic gap boxes passed to Tetris must be consistent with ϕ.

Example B.2. Let A and B be attributes with 3-bit domains. Consider the relation $R(A,B)$ containing the tuples $\{\langle 001, 010\rangle, \langle 001, 101\rangle, \langle 100, 110\rangle\}$. By default, we assume the bit ordering is $\phi_A = \phi_B = (1,2,3)$. If we change the bit ordering so that $\phi_A = (3,2,1)$ and $\phi_B = (2,1,3)$, then $\phi(R) = \{\langle 100, 100\rangle, \langle 100, 011\rangle, \langle 001, 110\rangle\}$.

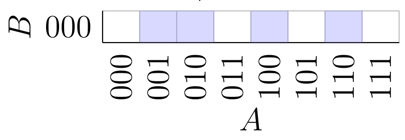
Bit orderings are less general than domain orderings, because for any query $\mathcal{Q}$ and any bit ordering ϕ of $\mathcal{Q}$, there is a corresponding domain ordering σ of $\mathcal{Q}$ such that $\phi(\mathcal{Q}) = \sigma(\mathcal{Q})$, but the converse is not true. Figure B.1 helps to illustrate this. Here, ϕ is the optimal bit ordering, and σ is the optimal domain ordering. We can obtain a smaller box cover for $\sigma(R)$ than we can for $\phi(R)$.

Bit ordering $\phi_A = (1, 3, 2)$

Domain ordering $\sigma_A = (001, 010, 100, 110, 000, 011, 101, 111)$

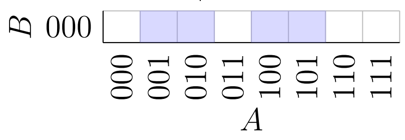
R

A	B
001	000
010	000
100	000
110	000



$\phi_A(R)$

A	B
001	000
010	000
100	000
101	000



$\sigma_A(R)$

A	B
000	000
001	000
010	000
011	000

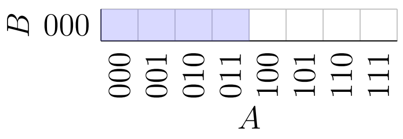


Figure B.1: A relation whose box cover size can be decreased by changing the bit ordering or domain ordering

B.1 Finding an Optimal Bit Ordering

The example query we defined in Section 5.1 has an $\Omega(N)$ size box certificate under the default domain ordering, but under a good domain ordering it has an $\widetilde{O}(1)$ size box certificate. A similarly small box certificate can be obtained by choosing a good bit ordering. In fact, any bit ordering which places the last $n-2$ bits first also yields a box certificate of size $\widetilde{O}(1)$ for this query. This motivates finding the bit ordering that minimizes the certificate size of the input query. The problem is defined as follows.

Definition B.3 (CertMinP$_{\mathbf{BitF}}$). The *Bit Flexible Certificate Minimization Problem (CertMinP$_{BitF}$)* takes as input a query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ and produces a bit ordering ϕ^* such that

$$C_{\Box}(\phi^*(\mathcal{Q})) = \min_{\phi} C_{\Box}(\phi(\mathcal{R}))$$

In Example 5.5, each attribute has d bits which are irrelevant to the query in the sense that for each $k \in [d]$ and $t \in R_{i,j}$, whenever $(t.A)_k = 1$, the tuple t' obtained from t by flipping bit k of $t.A$ to 0 also satisfies $t' \in R_{i,j}$. If bit k is placed last in the bit ordering for A, then no maximal dyadic gap box will specify a prefix for attribute A of length d, because such a box could always be made larger by removing the d-th bit without covering any tuples in R. This means that any bits which satisfy this "irrelevant" property should be placed last in the ordering for their attribute if we aim to decrease number of gap boxes.

This notion of an irrelevant bit is actually a special case of bit *influence*, a well studied concept in the analysis of Boolean functions [38, Ch. 2]. Let R be a relation over n attributes, each with a d-bit domain. Each tuple can then be thought of as a binary string of length $n \cdot d$. Consider the Boolean function $f_R : \{0,1\}^{n \cdot d} \to \{0,1\}$ defined by $f_R(t) = 1$ if and only if $t \in R$. Then the bit i in attribute A is irrelevant to the query, as described above, if and only if the corresponding bit in f_R has *influence* equal to 0.

The heuristic which moves bits with 0 influence to the end is enough to solve $\text{CertMinP}_{\text{BitF}}$ on Example 5.5, but its usefulness is limited to the small subset of queries which have bits with 0 influence. Beyond this, nothing more is known about solving this version of the problem. The next section presents some more fruitful results about a restricted version of $\text{CertMinP}_{\text{BitF}}$.

B.2 Minimizing the GBO-Consistent Box Cover

A more restrictive notion of a dyadic gap box is a *GBO-consistent dyadic gap box.*

Definition B.4 (GBO). Suppose $\mathcal{Q}$ has n attributes, each over d bits. A *global bit order (GBO)* $\sigma \in S_{dn}$ is a permutation of all the bits in $\mathcal{Q}$.

A GBO is a generalization of a GAO where we set one global order for all of the bits in all of the attributes of the query instead of a global order just for the attributes themselves.

Definition B.5 (GBO-consistent dyadic gap box). Let $\phi \in S_{dn}$ be a fixed GBO. A *ϕ-consistent dyadic gap box* for the relation $R(\mathcal{A}_R)$ is a binary string $b \in \{0,1,*\}^k$ for some $k \in [dn]$ such that $b_i = *$ if and only if $\phi(i)$ is a bit from attribute A and $A \notin \mathcal{A}_R$. Furthermore, b must not contain any tuples in R.

Similar to an ordinary dyadic gap box, a tuple t is *contained* in b if $t(\phi(i)) = b(i)$ or $b(i) = *$ for all $i \leq k$. A ϕ-consistent box cover for $\mathcal{Q}$ is a set of ϕ-consistent boxes which cover the gaps in all relations of $\mathcal{Q}$. A ϕ-consistent box certificate is a subset of a ϕ-consistent box cover whose union is equal to the union of all the boxes in the cover.

Example B.6. Let A, B, and C be attributes with 3-bit domains with bit labels (a_1, a_2, a_3), (b_1, b_2, b_3), and (c_1, c_2, c_3), respectively. Let $\mathcal{Q} = (\{R(A, B), S(B, C)\}, \{A, B, C\})$. Suppose that $R = \{\langle 000, 000\rangle\}$ under the default bit ordering. Consider the GBO $\phi = (a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3)$. Then $b = 1$ and $b' = 00{*}01$ are examples of valid ϕ-consistent dyadic gap boxes for R. $b = 00$ is *not* a ϕ-consistent dyadic gap box for R because it contains a tuple of R. $b = {*}1$ is also *not* a ϕ-consistent dyadic gap box for R because bit b_1 is specified and a_1 is not, but R is defined over both A and B and a_1 comes before b_1 in ϕ.

We will use $C_{GBO}(\mathcal{Q})$ and $K_{GBO}(\mathcal{Q})$ to denote the minimum size GBO-consistent box certificate and GBO-consistent box cover respectively for a fixed GBO. More precisely,

$$C_{GBO}(\mathcal{Q}) = \min_{B \in \mathcal{B}_{GBO}(\mathcal{Q})} \min_{C \in \mathcal{C}(B)} |C|$$
$$K_{GBO}(\mathcal{Q}) = \min_{B \in \mathcal{B}_{GBO}(\mathcal{Q})} |B|$$

where $\mathcal{B}_{GBO}(\mathcal{Q})$ is the set of GBO-consistent box covers of $\mathcal{Q}$ and $\mathcal{C}(B)$ is the set of certificates for the box cover B.

GBO-consistent dyadic boxes are more restrictive than dyadic boxes because for any GBO ϕ and ϕ-consistent dyadic box b, there is a corresponding bit ordering for $\mathcal{Q}$ and dyadic box b' such that b and b' cover the same tuples. However, the converse is not necessarily true.

As we will see in Section B.2.1, any query $\mathcal{Q}$ under any GBO ϕ admits a ϕ-consistent box cover of size $\widetilde{O}(N)$ where N is the number of input tuples. In fact, the minimum size of a ϕ-consistent box cover depends on the number of pairs of tuples in the query which are lexicographically consecutive according to ϕ.

With these definitions in mind, we can define a more restricted version of $\text{CertMinP}_{\text{BitF}}$, and a similar problem where the objective is to minimize the size of the box cover instead of the size of the certificate.

Definition B.7 ($\mathbf{CertMinP_{GBO}}$). The *Global Bit Order Certificate Minimization Problem ($\mathit{CertMinP_{GBO}}$)* takes as input a query $\mathcal{Q} = (\mathcal{R}, \mathcal{A})$ and produces a GBO ϕ^* such that

$$C_{GBO}(\phi^*(\mathcal{Q})) = \min_{\phi} C_{GBO}(\phi(\mathcal{Q}))$$

Definition B.8 (BoxMinP$_{\mathbf{GBO}}$). The *Global Bit Order Box Cover Minimization Problem (BoxMinP$_{GBO}$)* takes as input a query $(Q) = (\mathcal{R}, \mathcal{A})$ and produces a GBO ϕ^* such that

$$K_{GBO}(\phi^*(\mathcal{Q})) = \min_{\phi} K_{GBO}(\phi(\mathcal{Q}))$$

As discussed in Chapter 5, the minimum box cover size is an upper bound on the minimum certificate size, and if there is only one relation in the query, then these two problems are the same. In the coming sections we will present two results about BoxMinP$_{\text{GBO}}$, and determine if these results say anything about CertMinP$_{\text{GBO}}$ and CertMinP$_{\text{BitF}}$.

B.2.1 Generating GBO-Consistent Gap Boxes

Before proving these results, let's deepen our understanding of the problem. Let ϕ be a GBO and let R be a relation with ϕ-consistent dyadic box cover B. Suppose $b_1, b_2 \in B$ and $b_1 \cap b_2 \neq \emptyset$. If $b_1 \not\subseteq b_2$ and $b_2 \not\subseteq b_1$, then there are some bits i and j such that $i \neq j$, $b_1(i) \neq *$, $b_1(j) = *$, $b_2(i) = *$, and $b_2(j) \neq *$. Since b_1 and b_2 are ϕ-consistent, both come from R, and b_1 specifies bit i but not j, we must have $\phi(i) \leq \phi(j)$. Similarly, we must have $\phi(j) \leq \phi(i)$, a contradiction. Thus, we must have $b_1 \subseteq b_2$ or vice-versa, and so any minimal ϕ-consistent dyadic box cover of R contains mutually disjoint boxes.

Consider the ordering $\pi \in S_R$ on the tuples in R obtained by sorting R lexicographically according to the GBO ϕ. Let B be a ϕ-consistent dyadic box cover for R. For any $b \in B$, there are two tuples $t_1, t_2 \in R$ such that all the tuples covered by b lie lexicographically between t_1 and t_2 in π. Conversely, for every two tuples $t_1, t_2 \in R$ which are adjacent in π, there must be a box $b \in B$ which lies lexicographically between t_1 and t_2 if and only if t_1 and t_2 are not direct neighbours in the lexicographical order of $\{0, 1\}^{dn}$.

Furthermore, every gap between two such tuples $t_1, t_2 \in R$ can be covered by $\widetilde{O}(1)$ ϕ-consistent dyadic boxes. This follows from Proposition B.14 in [1]. Algorithm 9 presents how this is done in pseudocode. It generates a ϕ-consistent dyadic box cover of R of size $\widetilde{O}(N_g)$, where N_g is the number of tuples t in R for which the tuple t' after t in π does not come immediately after t in the lexicographical ordering of $\{0, 1\}^{nd}$. Therefore, if we can efficiently compute a GBO which minimizes N_g, we can efficiently approximate BoxMinP$_{\text{GBO}}$ (and CertMinP$_{\text{GBO}}$) in the single-relation case. The following section shows exactly how we can do this.

Algorithm 9 Generate a ϕ-consistent box cover for R and GBO ϕ

```
1: GenGboBoxes($R$, $\phi$):
2: Sort $R$ according to $\phi$
3: $B := \emptyset$
4: for $0 \leq i < |R| - 1$ do
5:     $s :=$ longest common prefix of $R[i], R[i+1]$
6:     $(a, b) := (R[i] = sa \wedge R[i+1] = sb)$
7:     for $0 \leq j < |a|$ do
8:         if $a[j] = 0$ then
9:             $B := B \cup \{sa[0, \ldots, j-1]1\}$
10:        end if
11:    end for
12:    for $0 \leq j < |b|$ do
13:        if $b[j] = 1$ then
14:            $B := B \cup \{sb[0, \ldots, j-1]0\}$
15:        end if
16:    end for
17: end for
```

B.2.2 Finding the Optimal GBO for a Single Relation

Let $\mathcal{Q}$ contain a single relation R, and let $t_1, t_2 \in R$. t_1 and t_2 are directly consecutive according to a GBO ϕ if and only if there exists some integer i and binary string s such that $t_1 = s01^i$ and $t_2 = s10^i$. In this case, there must be exactly one bit i for which $t_1(i) = 0$ and $t_2(i) = 1$. In fact, this condition is sufficient to ensure that there exists a GBO such that t_1 directly precedes t_2. If so, the bits of relation R are partitioned into three different types: $P_1(t_1, t_2) = \{j : t_1(j) = t_2(j)\}$, $P_2(t_1, t_2) = \{j : t_1(j) = 1 \wedge t_2(j) = 0$, and the single bit i which satisfies $t_1(i) = 0 \wedge t_2(i) = 1$. t_1 directly precedes t_2 according to GBO ϕ if and only if all the bits in $P_1(t_1, t_2)$ come before i in ϕ and all the bits in $P_2(t_1, t_2)$ come after i in ϕ. These conditions form what we will call an *ordering constraint.*

Definition B.9 (Ordering constraint). An *ordering constraint* over a set of d bits is a triple $c = (P_1, i, P_2)$ where $P_1, P_2 \subseteq [d]$, $i \in [d]$, $P_1 \cup \{i\} \cup P_2 = [d]$, and P_1, $\{i\}$, and P_2 are all disjoint. A GBO ϕ is said to *satisfy* c if all the bits in P_1 come before i in ϕ and all the bits in P_2 come after i in ϕ.

For every pair of tuples $t_1, t_2 \in R$, we can quickly find the bit i and sets $P_1(t_1, t_2)$ and $P_2(t_1, t_2)$ which define the ordering constraint $C(t_1, t_2)$ on the GBO which must be satisfied in order for t_1 to directly precede t_2 according to the GBO. Let $C(R)$ be the set of ordering constraints generated by all pairs of tuples in R. For each pair of tuples $t_1, t_2 \in R$, we can determine the corresponding ordering constraint (if one exists) in $O(d) = \widetilde{O}(1)$ time. Iterating over all pairs of tuples, we can compute $C(R)$ in $\widetilde{O}(N^2)$ time. If we can find a GBO which satisfies the maximum number of constraints from $C(R)$ in polynomial time, then we can minimize N_g for R in polynomial time, giving us a polynomial-time $\widetilde{O}(1)$ approximation algorithm for $\text{BoxMinP}_{\text{GBO}}$ in the single-relation case. The problem we want to solve is defined as follows.

Definition B.10 (OrdConMaxP). The Ordering Constraint Maximization Problem (OrdConMaxP) takes as input a set of unique ordering constraints C and for each $c \in C$ a corresponding positive integer weight w_c and produces a GBO ϕ^* such that

$$\sum_{c \in C(\phi^*)} w_c = \min_{\phi} \sum_{c \in C(\phi)} w_c$$

where $C(\phi) \subseteq C$ is the set of constraints satisfied by ϕ.

OrdConMaxP can be solved with a simple dynamic programming algorithm presented in Algorithm 10. Theorem B.11 verifies that this algorithm is correct. As discussed above, this immediately yields Corollary B.12.

Theorem B.11. *Algorithm 10 solves OrdConMaxP in polynomial time.*

Proof. First, some clarifications about what Algorithm 10 is doing. The input C is an array of all the unique ordering constraints and W is an array of their corresponding weights.

The FLATTEN subroutine used on line 25 takes as input an array of sets and returns a corresponding one-dimensional array by appending each of the member sets together, in order. The array P defined on line 2 is an array of partial solutions corresponding to GBOs such that FLATTEN($P[i]$) is optimal if $C[i]$ is the constraint satisfied with the largest value of $|C[i].P_1|$. $P[i]$ is a partition of the d bits into any number of sets, which can be stored as an array of sets. $O[i]$ is the total weight of constraints guaranteed to be satisfied by the partial solution $P[i]$. Note that each input constraint c alone admits a partial solution $[c.P_1, \{c.i\}, c.P_2]$ which is only guaranteed to satisfy c. The COMBINE subroutine used on line 20 takes as input two partial solutions which can be simultaneously satisfied and produces a partial solution which satisfies both. The condition of the if-statement on line 10 ensures that the two partial solutions passed to COMBINE can indeed be simultaneously satisfied.

Algorithm 10 Compute a GBO which solves OrdConMaxP

```
OcmpDP($C, W$):
P := array of partial solutions of length $|C|$
O := array of integers of length $|C|$
for $i := 1$ to $dn$ do
    for $j := 1$ to $|C| - 1$ do
        if $|C[j].P_1| = i - 1$ then
            $m := 0$
            $u :=$ false
            for $k := 1$ to $|C| - 1$ do
                if $C[k].P_1 \cup C[k].i \subseteq C[j].P_1$ and $O[k] > m$ then
                    $m := O[k]$
                    $u := k$
                end if
            end for
            if $u =$ false then
                $O[j] := W[j]$
                $P[j] := C[j]$
            else
                $O[j] := W[j] + O[u]$
                $P[j] :=$ Combine($C[j], P[u]$)
            end if
        end if
    end for
end for
return Flatten($P[\text{argmax}_i O[i]]$)
```

Now, let's verify that Algorithm 10 produces the optimal solution. Let $S \subseteq C$ be the maximum weight set of constraints which can satisfied by a single GBO ϕ^*.

For a constraint $C[i]$, let $C[i](\phi)$ be a predicate which is true if and only if ϕ satisfies $C[i]$. For each $0 \leq i < |C|$, let ϕ_i be a GBO such that $OPT_i = W[i] + \sum_{j \in S_i} W[j]$ is maximized, where $S_i = \{j : |C[j].P_1| < |C[i].P_1| \land C[j](\phi_i)\}$.

Let $C[i]$ be an input constraint and let $k = |C[i].P_1|$. If $k = 0$, then $F[i]$ is the only term of the sum in OPT_i, and so $P[i]$ indeed produces a partial solution which satisfies at least OPT_i constraints.

Suppose that $k \geq 1$ and assume inductively that Algorithm 10 sets $P[j]$ to a partial solution which satisfies at least OPT_i constraints for every j in S_i. Note that $OPT_i = W[i] + \max_{j \in S_i} OPT_j$, because the constraint $C[i]$ leaves us free to rearrange the first k bits of ϕ_i to satisfy as many simultaneously satisfiable constraints as possible. By induction, Algorithm 10 does this on line 20 when it calls COMBINE on the maximum weight partial solution so far and the current constraint $C[i]$. Thus the inductive assumption holds for all i.

On line 25, we return a GBO corresponding to the partial solution which satisfies the maximum number of possible constraints over all i, which is an optimal solution to OrdConMaxP.

Algorithm 10 runs in time $O(dn|C|^2) = \widetilde{O}(|C|^2)$, which is polynomial in $|C|$. □

Corollary B.12. *If $\mathcal{Q}$ contains only one relation, BoxMinP$_{GBO}$ on $\mathcal{Q}$ can be approximated to an $\widetilde{O}(1)$-factor in polynomial time.*

Corollary B.12 also applies to CertMinP$_{\text{GBO}}$, since the problems are equivalent in the single-relation case. Furthermore, if this single relation has only one attribute, Corollary B.12 applies to CertMinP$_{\text{BitF}}$ as well.

B.2.3 Finding an Optimal GBO for Multiple Relations

We would like to generalize these results to any BoxMinP$_{\text{GBO}}$ instance with more than one relation. The general version is more complex because not every relation in the query is defined over all attributes of the query. If $A \notin \text{attr}(R)$, then the placement of the bits of A in the GBO has no effect on the GBO-consistent box cover of R. We can capture this difference by generalizing our definition of an ordering constraint to contain an extra set, P_3, that holds all the bits which the constraint is indifferent towards.

Definition B.13 (Generalized ordering constraint). A *generalized ordering constraint* over d bits is a 4-tuple $c = (P_1, i, P_2, P_3)$ such that $P_1, P_2, P_3 \subseteq [d]$ and $i \in [d]$ such that the sets $P_1, \{i\}, P_2$, and P_3 partition the set $[d]$. c is satisfied by ϕ if the bits in P_1 occur before bit i in ϕ and the bits of P_2 occur after i in ϕ.

For these generalized constraints, we can define a maximization problem analogous to OrdConMaxP.

Definition B.14 (GenOrdConMaxP). The Generalized Ordering Constraint Maximization Problem (GenOrdConMaxP) takes as input a collection of generalized ordering constraints C and produces a GBO ϕ^* which satisfies the maximum number of constraints in C.

Unfortunately, this problem is NP-hard, as shown in Theorem B.15. This means it is not a viable avenue to obtain an approximation algorithm for $\text{BoxMinP}_{\text{GBO}}$.

Theorem B.15. *GenOrdConMaxP is NP-hard.*

Proof. We will prove this by a reduction from the maximum independent set problem. The maximum independent set problem takes as input a graph G and returns an independent set $S \subseteq V(G)$ of maximum size. This problem is known to be NP-complete [23].

Let the graph G be an input graph to maximum independent set with n vertices. We will define a set of n generalized ordering constraints over $n+1$ bits corresponding to this graph, with each constraint corresponding to one vertex.

For each node $v \in V(G)$, define one bit labelled A_v. We will also define one additional "separator" bit labelled M. Our set of $n+1$ bits is $B = \{A_v : v \in V(G)\} \cup \{M\}$. For each node v, we will define a single constraint, $c_v = (A_{N(v)}, M, \{A_v\}, B \setminus (A_{N(v)} \cup \{M, A_v\}))$, where $A_{N(v)} = \{A_u : (u, v) \in E(G)\}$.

Now, if G has an independent set S of size k, then the constraints in the set $C_S = \{c_v : v \in S\}$ can all be satisfied by the GBO ϕ which places all bits in $A_S = \{A_v : v \in S\}$ last, then M before them, and then the remaining bits first. Since there are no edges between vertices in S, all the constraints in C_S are satisfied by ϕ. Therefore, ϕ satisfies $|C_S| = k$ constraints.

Conversely, suppose there exists a GBO ϕ which satisfies k of the constraints we have defined. Let C be this set of k constraints, and let $S = \{v : c_v \in C\} \subseteq V(G)$. Suppose that $u, v \in S$. Then, since ϕ satisfies c_u and c_v, both A_u and A_v must appear after M in ϕ. If $(u, v) \in E(G)$, then ϕ satisfying c_u implies that A_v appears before M in ϕ, a contradiction. So $(u, v) \notin E(G)$, and therefore S is an independent set in G of size k. $\square$

Theorem B.15 can be used to show that $\text{BoxMinP}_{\text{GBO}}$ is NP-hard, as shown by Corollary B.16. However, this reduction only applies when the query shape is not fixed, because the number of tuples created is only $2m$, where m is the number of relations in the query. When the query is fixed, and we consider m to be $\widetilde{O}(1)$, as we do for all of the analysis in this thesis, we see that this reduction does not preclude the possibility of an $\widetilde{O}(\text{poly}(N))$ algorithm for $\text{BoxMinP}_{\text{GBO}}$.

Corollary B.16. *$\text{BoxMinP}_{\text{GBO}}$ is NP-hard when the query shape is not fixed.*

Proof. We can reduce GenOrdConMaxP to $\text{BoxMinP}_{\text{GBO}}$ by creating a single-bit attribute A_b for each bit b which appears in any constraint, and a relation R_c for each constraint

$c = (P_1, i, P_2, P_3)$. R_c should be defined over the attributes corresponding to the bits in $P_1 \cup \{i\} \cup P_2$, and should contain exactly two tuples which are made adjacent to one another if and only if the constraint c is satisfied. Applying Theorem B.15, this means CertMinP$_{\text{GBO}}$ is NP-hard. $\square$

Appendix C

Box Covers and Boolean Algebra

This appendix will discuss some connections between box cover problems and Boolean algebra where the arithmetic operations are defined as follows.

$$\begin{aligned} 0+0&=0 \qquad & 0\cdot 0&=0\\ 0+1&=1 & 0\cdot 1&=0\\ 1+0&=1 & 1\cdot 0&=0\\ 1+1&=1 & 1\cdot 1&=1 \end{aligned}$$

The set of Boolean vectors under these arithmetic operations do not form a vector space, so we cannot apply results from linear algebra to problems in Boolean algebra. However, these operations do form a commutative semiring, which is a widely studied algebraic construct. Furthermore, there is an abundance of literature specifically studying problems in Boolean algebra. Boolean algebra provides more structure than a general semiring. A book covering many concepts in Boolean matrix theory is [25]. A slightly more general version of this theory is surveyed in [24]. Concepts from linear algebra have been extended to commutative semirings in many works, including [40, 47].

There are many definitions and results for Boolean algebra which share useful properties with their analogous linear algebra counterparts. In particular, we can define the Boolean rank of a matrix as follows.

Definition C.1. The *Boolean rank* of an $m \times n$ Boolean matrix M is the minimum integer r such that there exists an $m \times r$ matrix A and an $r \times n$ matrix B satisfying $M = AB$.

When studying the single-relation, 2-dimensional domain ordering problem in Sections 5.2 and 5.3, we considered the input to be an $m \times n$ Boolean matrix M, and we sought a domain ordering that induced the minimum size box cover for all the 1-cells in M. Under Boolean algebra, a box cover can serve to satisfy the definition of Boolean rank, as demonstrated with the following small matrix.

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

In this equation, the leftmost matrix has Boolean rank at most 2, a direct consequence of the fact that it has a box cover of size 2. Lemma C.2 proves this is true in general.

Lemma C.2. *If a Boolean matrix M has a box cover of size k, then the Boolean rank of M is at most k.*

Proof. Let $b_1, \ldots, b_k$ be a set of k rectangles which cover the 1-cells of M. For each b_i, let r_i and c_i be the indicator vectors for the set of rows and columns spanned by b_i, respectively. Then it is easy to see that $M = \sum_{i=1}^{k} r_i c_i^T$. Let $R = [r_1, \cdots, r_k]$ and $C = [c_1, \cdots, c_k]$. Under Boolean algebra, we can rewrite the aforementioned sum as $M = \sum_{i=1}^{k} r_i c_i^T = RC^T$, so M has Boolean rank at most k. □

Unfortunately, the converse of Lemma C.2 is not true. However, if we change our notion of a "box" from a contiguous rectangle to a generalized rectangle defined by arbitrary subsets of the rows and columns of M, then the converse is true. Finding a minimum size Boolean basis for a matrix is equivalent to finding a minimum size generalized box cover for the matrix.

This observation was made by Lubiw [32], where she notes that a generalized box cover for the *swath matrix* of M is equivalent to a (contiguous) box cover of the original matrix M. We will not define the swath matrix here, but it is a Boolean matrix of polynomial size with respect to M. A class of swath matrices for which minimizing the generalized box cover is tractable is given in [32]. This yields a class of Boolean matrices for which finding the minimal box cover is tractable when the order of the rows and columns is fixed. When we reorder the rows and columns of M, the swath matrix of M changes. Examining how the swath matrix changes when the original matrix is permuted is an interesting direction for future research.