Synthetic Data Generator

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFIMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Technology

IN

Computer Science and Engineering

BY

Dharmendra Kumar Singh

Computer Science and Automation

Indian Institute of Science

Bangalore – 560 012 (INDIA)

June, 2018

Declaration of Originality

I, **Dharmendra Kumar Singh**, with SR No. **04-04-00-10-42-16-1-13343** hereby declare that the material presented in the thesis titled

Synthetic Data Generator

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2017-2018**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Prof. Jayant R. Haritsa Advisor Signature

DEDICATED TO

My Family

Acknowledgments

I would like to express my sincere gratitude to my project advisor, Prof. Jayant R. Haritsa for giving me an opportunity to work on this project. I am thankful to him for his valuable guidance and moral support. I feel lucky to be able to work under his supervision. I also sincerely thank Mr. Anupam Sanghi, IISc Bengaluru and Prof. Srikanta Tirthapura, Iowa State University. Without their collaboration this work would not have been possible. I would also like to thank the Department of Computer Science and Automation for providing excellent study environment. The learning experience has been really wonderful here. Finally I would like to thank all IISc staff, my family and friends for helping me at critical junctures and making this project possible.

Abstract

A common requirement of database vendors is to test their database engine on client data. Getting access to client data may not always be possible due to privacy concerns and high transfer cost. Therefore, vendors rely on generating synthetic version of customers database that can preserve the data characteristics that are relevant for testing purposes. **Hydra** is a data generation tool that tries to achieve this by using the concept of *cardinality constraints*. It formulates these constraints as a *linear program (LP)* and processes its solution to generate the data. As part of this work, firstly, we have optimized the algorithms used for formulating the LP. As a result, we have an improved version, **Hydra++**, which helps us in getting better efficiency. We have also extended the existing prototype solution of Hydra to create an end to end application software. Finally, we have widened the scope of cardinality constraints to include support for *projection constraints* (under certain assumptions), which are cardinality constraints containing projection relational algebraic operator.

Publications based on this Thesis

- A. Sanghi, R. Sood, D. K. Singh, J. R. Haritsa, and S. Tirthapura.
  HYDRA: A Dynamic Big Data Regenerator.
  In *PVLDB*, 2018. (In press)

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Database engine testing is a common requirement by the database vendors when upgrading their database engine or when a client is facing some problem with the current engine. For this, the database vendors may need the client's data at their site, which may not always be possible because of privacy of client's data or high data transfer cost. In these cases, vendors rely on taking minimal information from client site and generating data locally such that it closely mimics the properties of client's data.

Hydra [7] is a tool which allows generating synthetic database at vendor site using minimal information obtained from client's database such that the database generated at vendor's site is *volumetrically similar* to the client's database. That is, for a query from the predefined workload of the client and with a common query execution plan at the client and vendor sites for that query, the output row cardinalities of individual operators in these plans are very similar in the original and synthetic databases. This similarity helps to achieve similar performance on the clients workload at the client and vendor site.

In Hydra's pipeline of creating synthetic database, it uses *linear programming* as a tool. The techniques used by Hydra to formulate the LPs is a lot dependent on brute-force techniques because of which they are very inefficient and have a high time complexity.

We propose **Hydra++** which uses efficient algorithms to formulate the LPs which is also reflected in our experiments. Note that the LPs formulated by Hydra++ are the same as Hydra and only the algorithms for their formulation are different.

A prototype of Hydra was built by its authors. We have extended that prototype into an end to end application software [8] which can be downloaded from [1].

Hydra uses the concept of *Cardinality Constraints* (CCs) to achieve the property of volumetric similarity (CCs are explained in Section 1.1). *Projection Constraints* (PCs) are CCs which have projection operator in them. These type of constraints are not handled by Hydra.

We give an algorithm to handle these type of constraints and empirically show its correctness.

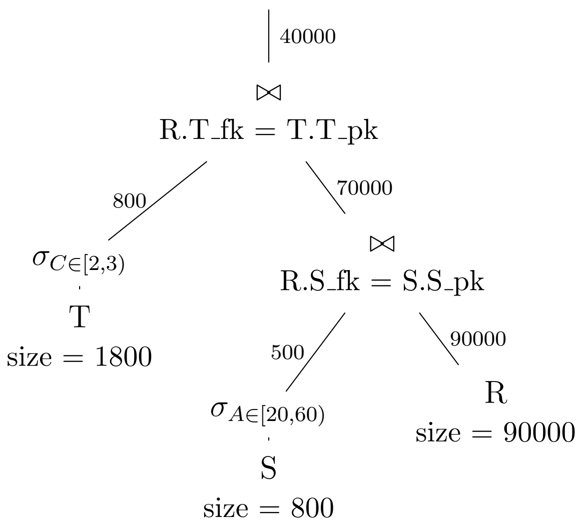
1.1 Cardinality Constraints

R (R_pk, S_fk, T_fk)
S (S_pk, A, B)
T (T_pk, C)

(a) Schema

```
SELECT *
FROM   R
JOIN   S ON (R.S_fk = S.S_pk)
JOIN   Region ON (R.T_fk = T.T_pk)
WHERE  S.A >= 20 AND S.A < 60
AND    T.C >= 2 AND T.C < 3
```

(b) Query



(c) AQP

$| R |= 90000$ $| R |= 800$ $| T |= 1800$
$| \sigma_{S.A\in[20,60)}(S) |= 500$ $| \sigma_{T.C\in[2,3)}(T) |= 800$
$| \sigma_{S.A\in[20,60)}(R \bowtie S) |= 70000$ $| \sigma_{S.A\in[20,60) \wedge T.C\in[2,3)}(R \bowtie S \bowtie T) |= 40000$

(d) CC

Figure 1.1: CC example

The CCs are extracted from *annotated query plans* (AQPs) [5] which are query execution plans with the output edge of each operator annotated with the associated row cardinality (as evaluated during the clients execution).

Consider a set of Relations $\mathcal{R}_1, \mathcal{R}_2, ..\mathcal{R}_m$. A Cardinality Constraint (CC) is of the form

$$| \pi_{\mathcal{A}} \sigma_P(\mathcal{R}_1 \bowtie \mathcal{R}_2 \bowtie .. \bowtie \mathcal{R}_m) |= k$$

where P is the selection predicate on the attributes of $\mathcal{R}_1, \mathcal{R}_2, ..\mathcal{R}_m$; $\mathcal{A}$ is the set of attributes over which the set of resultant tuples are projected while removing duplicates, and k is a non-negative integer. For example: Considering the schema given in Figure 1.1a, the AQP corresponding to query in Figure 1.1b, is given in Figure 1.1c and the CCs extracted are given in Figure 1.1d. A database instance is said to satisfy a CC if evaluating the left side expression on the database produces k tuples in the output.

Hydra can handle CCs corresponding to queries having multiple relations in its FROM clause. Since our work is independent of whether queries have single table or multiple in their FROM clause, hence we define everything in this report in context of single table.

We begin by considering CCs with selection predicates only. The general constraints with projections is discussed in Section 6.

Organization: Chapter 2 briefly explains the working of Hydra and introduce the problem associated with its technique of LP formulation. Hydra++ which tackle those problems is explained in Chapter 3. Comparison of Hydra and Hydra++ is presented in Chapter 4. Chapter 5 gives a brief overview of the functionalities of the *Hydra Software*. Algorithm for handling projection constraints is discussed in Chapter 6 and its performance evaluation is given in Chapter 7. Conclusions and future work are presented in Chapter 8.

Chapter 2

Hydra

2.1 Architecture

The CCs corresponding to the client's database and workload are created at client's site which are then transferred to the vendor site. Using these CCs, Hydra creates an LP for every relation and the solution of this LP is used to generate data for that relation.

2.2 Creating LP for Relation

The complexity of an LP depends on the number of variables in the LP which, as we'll see later, depends on the number of attributes in the relation. Hence, to optimize the LP, [4] proposed the following optimization which Hydra uses to decompose the relations into *sub-views*:

1. For every relation, a "relation-graph" is created by creating a node for each attribute in that relation and inserting an edge between a pair of nodes if they appear together in some CC.

2. The maximal cliques of this relation-graph are the sub-views for that particular relation.

A sub-view $\mathcal{V}$ with set of attributes $\{A_1, A_2, .., A_N\}$ can be visualized as an N dimensional space where every dimension corresponds to some A_i and every point in the space is a possible tuple from $\mathcal{V}$. A CC $\mathcal{C}$ is *applicable* on a sub-view $\mathcal{V}$ if the set of all attributes appearing in literals of $\mathcal{C}$ is a subset of the attributes of $\mathcal{V}$. For a particular relation, the set of sub-views obtained from the above steps along with the set of CCs applicable on them are given to the *partitioning algorithm* which divides the space corresponding to sub-views into *regions* as follows:

Consider a sub-view $\mathcal{V}$ with set of attributes $\{A_1, A_2, .., A_N\}$ and consider a set of CCs $\mathbb{C}$ applicable on $\mathcal{V}$. The partitioning algorithm divides the space corresponding to $\mathcal{V}$ into regions such that for any two tuples t_i and t_j, they are in the same region if and only if they *satisfy* the same subset of constraints from $\mathbb{C}$. The set of regions thus obtained corresponding to sub-view $\mathcal{V}$ is called the *Partition* of $\mathcal{V}$.

After getting partitions from the above partitioning algorithm, LP is created. Each region of the partitions acts as a variable for the LP. For each CC applicable on a sub-view, an LP condition is created as follows: Let us denote the regions of j^{th} sub-view, $\mathcal{V}_j$, by $Reg_1^j, Reg_2^j, .., Reg_m^j$, and variables corresponding to i^{th} region of $\mathcal{V}_j$ by x_i^j. Then the LP condition created corresponding to $\mathcal{V}_j$ and some CC $|\ \sigma_P(\mathcal{R})\ | = k$ will be

$$\sum_{Reg_i^j\ satisfies\ P} x_i^j = k$$

The solution of the LP assigns a non-negative integer to every such variable. The value of x_i^j determines how many tuples from Reg_i^j must be added into the solution for $\mathcal{V}_j$. The solutions of the different sub-views are then merged to obtain the synthesized database.

The above LP conditions ensure that the cardinality of tuples which satisfy the selection predicate of a certain CC in original database is same in the synthesized database also. But since the relation was divided into sub-views which may share common attributes, hence extra constraints known as *Consistency Constraints* are added into the LP to make sure that the distribution of values of common attributes in solutions of the different sub-views is same. The need of consistency constraints can be understood using the following example.

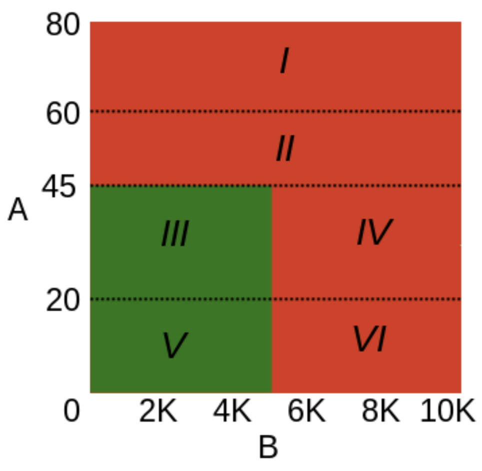
	Alias
Employee	**E**
Age	A
Bonus	B
Salary	S

Table 2.1: Schema

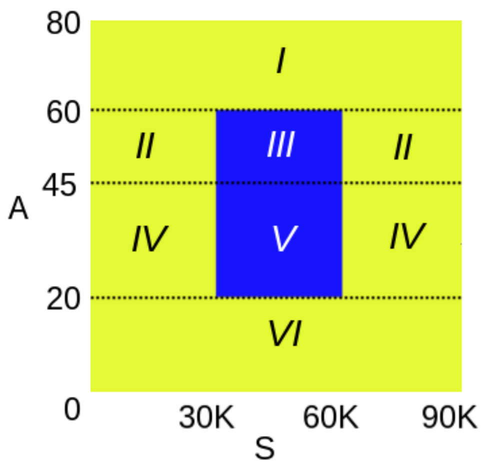
Consider the schema given in Table 2.1 and let the CCs be:

$$\begin{aligned}
&\mathcal{C}_1 :|\ \sigma_{A<45 \wedge B<5000}(E)\ | = 150\\
&\mathcal{C}_2 :|\ \sigma_{A\in[20,60) \wedge S\in[30000,60000)}(E)\ | = 100\\
&\mathcal{C}_3 :|\ (E)\ | = 400
\end{aligned}$$

Since B and S never appear together in any CC, hence two sub-views (A, B) and (A, S) will be created. Note that attribute A is common in the two sub-views. The partitions of the two sub-views are shown in Figure 2.1.



(a) Sub-view (A, B)



(b) Sub-view (A, S)

Figure 2.1: Partitions of sub-views

The green region in Figure 2.1a corresponds to $\mathcal{C}_1$ and the blue region in Figure 2.1b corresponds to $\mathcal{C}_2$. The LP solver will assign the following values to the regions: Green = 150, Red = 250, Blue = 100 and Yellow = 300. (Note that the solution in general may not be unique)

This implies that for the regions green, red, blue and yellow, 150, 250, 100 and 300 tuples have to be generated from them respectively. If all the 150 and 250 tuples of green and red region respectively are generated from the area where $A \in [0, 20)$, then, it's not possible to generate any tuples in sub-view (A, S) because the region of sub-view (A, S) which intersects with $A \in [0, 20)$ is yellow region and it is assigned only 300 tuples by LP solver.

In general, if any two sub-views $\mathcal{V}_i$ and $\mathcal{V}_j$ share common attributes $\mathcal{A}^{i,j}_{common}$, then the distribution of values of $\mathcal{A}^{i,j}_{common}$ must be consistent in $\mathcal{V}_i$ and $\mathcal{V}_j$. For our above example, the LP solution can be made consistent along attribute A by splitting the regions along dashed lines as shown in Figure 2.1, and adding the following (consistency) constraints into the LP

$$\begin{aligned}
x^{(A,B)}_{I} &= x^{(A,S)}_{I} \\
x^{(A,B)}_{II} &= x^{(A,S)}_{II} + x^{(A,S)}_{III} \\
x^{(A,B)}_{III} + x^{(A,B)}_{IV} &= x^{(A,S)}_{IV} + x^{(A,S)}_{V} \\
x^{(A,B)}_{V} + x^{(A,B)}_{VI} &= x^{(A,S)}_{VI}
\end{aligned}$$

2.3 Hydra's approach to make LP consistent

The process of creation of consistency constraints in Hydra can be divided into two parts. In the first part, the partitions obtained from partitioning algorithm (as discussed in Section 2.2) are given to Algorithm 1 which further splits those partitions such that for any two sub-views $\mathcal{V}_i$ and $\mathcal{V}_j$ sharing some set of common attributes $\mathcal{A}^{i,j}_{common}$, $\mathcal{V}_i$ and $\mathcal{V}_j$ have *consistent region boundaries* along $\mathcal{A}^{i,j}_{common}$, i.e. when the regions of $\mathcal{V}_i$ and $\mathcal{V}_j$ are projected along $\mathcal{A}^{i,j}_{common}$ then for any two regions r and s chosen from the projected space, either $r \cap s = \emptyset$ or boundary of r = boundary of s. For the example in Figure 2.1, this was obtained by splitting the sub-views along the dashed lines i.e. along $A = 20$, $A = 45$ and $A = 60$.

Algorithm 1: Split partitions for consistency

```
   Input: Set of all sub-views $\mathbb{V}$, Partition vector $\mathbb{P}$
   Output: Partition vector after splitting
1  SplittingForConsistency ($\mathbb{V}, \mathbb{P}$)
2      foreach distinct sub-views $\mathcal{V}_i, \mathcal{V}_j \in \mathbb{V}$ do
3          $\mathcal{A}^{i,j}_{common} = \texttt{Attribs}(\mathcal{V}_i) \cap \texttt{Attribs}(\mathcal{V}_j)$
4          if $\mathcal{A}^{i,j}_{common} = \emptyset$ then continue ;
5          $\mathcal{P}_{both} \leftarrow \texttt{SplitAll}(\mathbb{P}(\mathcal{V}_i) \cup \mathbb{P}(\mathcal{V}_j), \mathcal{A}^{i,j}_{common})$
6          foreach $r \in \mathbb{P}(\mathcal{V}_i)$ and $s \in \mathcal{P}_{both}$ do
7              Refine $r$ into $(r - s)$ and $(r \cap s)$
8          foreach $r \in \mathbb{P}(\mathcal{V}_j)$ and $s \in \mathcal{P}_{both}$ do
9              Refine $r$ into $(r - s)$ and $(r \cap s)$
10     return $\mathbb{P}$
1  SplitAll ($\mathcal{P}, \mathcal{A}_{common}$)
2      foreach $r, s \in \mathcal{P}$ do
3          if $\texttt{Proj}(r, \mathcal{A}_{common}) \cap \texttt{Proj}(s, \mathcal{A}_{common}) \neq \emptyset$ then
4              Refine $r$ and $s$ into $(r - s)$, $(s - r)$ and $(r \cap s)$
5      return $\mathcal{P}$
1  Proj ($r, \mathcal{A}_{common}$)
2      return Projection of $r$ along $\mathcal{A}_{common}$
```

Algorithm 1 takes as input the set of all the sub-views and the partition vector which has the partitions corresponding to every sub-view. The call to subroutine *SplitAll* splits all the regions in its input such that no two regions have inconsistent region boundaries along common attributes and return these newly formed regions. It can be seen that the for-each loop in subroutine *SplitAll* is a brute-force technique which tries to refine regions until their boundaries are consistent.

Algorithm 2: Create LP conditions for consistency constraints

```
Input: Set of all sub-views $\mathbb{V}$, Partition vector $\mathbb{P}$
CreateLPeqsForConsistency ($\mathbb{V}, \mathbb{P}$)
    foreach distinct sub-views $\mathcal{V}_i, \mathcal{V}_j \in \mathbb{V}$ do
        $\mathcal{A}^{i,j}_{common} = \texttt{Attribs}(\mathcal{V}_i) \cap \texttt{Attribs}(\mathcal{V}_j)$
        if $\mathcal{A}^{i,j}_{common} = \emptyset$ then continue ;
        $\mathcal{P}_{both} \leftarrow \texttt{SplitAll}(\mathbb{P}(\mathcal{V}_i) \cup \mathbb{P}(\mathcal{V}_j), \mathcal{A}^{i,j}_{common})$
        foreach $s \in \mathcal{P}_{both}$ do
            $LHS \leftarrow \emptyset$
            foreach $r \in \mathbb{P}(\mathcal{V}_i)$ do
                if $r \cap s \neq \emptyset$ then
                    $LHS \leftarrow LHS + x_r$
            $RHS \leftarrow \emptyset$
            foreach $r \in \mathbb{P}(\mathcal{V}_j)$ do
                if $r \cap s \neq \emptyset$ then
                    $RHS \leftarrow RHS + x_r$
            Add condition $LHS = RHS$ in solver
```

In the second part, the partitions obtained from Algorithm 1 are given to Algorithm 2 which creates LP equations for consistency constrains.

In Algorithm 2 also *SplitAll* is called further adding inefficiency to the brute-force nature of the complete process.

2.4 Adversarial Workload

To stress test the consistency constraint creation part of Hydra, we created a workload of 10 adversarial queries as follows:
Consider a relation $\mathcal{R}$ with 15 attributes $\{U_1, U_2, U_3, U_4, U_5, Z_1, Z_2, .., Z_{10}\}$. The i^{th} query of our workload is of the form

SELECT * **FROM** $\mathcal{R}$
WHERE U_1 >= i **AND** U_1 < (i+10)
AND U_2 >= i **AND** U_2 < (i+10)
AND U_3 >= i **AND** U_3 < (i+10)
AND U_4 >= i **AND** U_4 < (i+10)
AND U_5 >= i **AND** U_5 < (i+10)

AND Z_i >= i **AND** Z_i < (i+10)
There are 6 attributes in selection predicate of every query. The Z_i attribute is different in every query so that a new sub-view has to be created per query. For i^{th} query, the sub-view corresponding to it will have the attribute set $\{U_1, U_2, U_3, U_4, U_5, Z_i\}$. Every sub-view will be divided into 2 regions by the partitioning algorithm, one which satisfies the selection predicate of the query corresponding to that sub-view and the other which does not. The range in selection predicate of the 5 U attributes is slid by 1 in every consecutive query so that the 2 regions in all the 10 sub-views have inconsistent region boundaries along all 5 U attributes hence requiring a lot of splitting to make the LP consistent. When the above workload was tested on Hydra, it was observed that the time taken to formulate consistency constraints was orders of magnitude more than the time taken for other workloads of larger sizes. Specifically, Hydra took 8 hours to formulate LP for adversarial workload while it took only 13 seconds to formulate LP for a standard benchmark workload with close to 200 queries. This motivated us to create a better algorithm for creating consistency constraints.

Chapter 3

Hydra++

To tackle the problems specified in above sections, we created Hydra++. For a particular relation, the input to partitioning algorithm in Hydra was the set of sub-views and the set of CCs applicable on those sub-views. In Hydra++, we have modified the pipeline for creating consistency constraints. Instead of just applicable CCs, we now pass the union of applicable CCs and applicable *Consistency Filters* (CFs) to the partitioning algorithm alongside sub-views. The CFs are formulated in such a way that the partitions obtained after partitioning algorithm are same as the partitions obtained after Algorithm 1 of Hydra.
For a particular relation $\mathcal{R}$ a CF is of the form:

$$\sigma_P(\mathcal{R})$$

where P is the selection predicate on the attributes of $\mathcal{R}$. For creating the set of CFs we first define $\mathbb{A}_{common}$ as

$$\mathbb{A}_{common} = \{\mathcal{A}^{i,j}_{common} \mid \mathcal{A}^{i,j}_{common} = Attribs(\mathcal{V}_i) \cap Attribs(\mathcal{V}_j) \wedge \mathcal{A}^{i,j}_{common} \neq \emptyset\} \qquad \forall i, j \text{ such that } i \neq j$$

The algorithm to create CFs is given in Algorithm 4. Before explaining Algorithm 4, we first present Algorithm 3 which is an intuitive but incomplete algorithm (it fails when disjunctions are present in CCs) to create CFs, but helps in understanding Algorithm 4.

The intuition behind Algorithm 3 is that, for any CC $\mathcal{C}$ and two sub-views $\mathcal{V}_i$ and $\mathcal{V}_j$ having common attributes $\mathcal{A}^{i,j}_{common}$, when the filter made by the literals of $\mathcal{C}$ whose attributes are present in $\mathcal{A}^{i,j}_{common}$ is given to the partitioning algorithm, then, the partitioning algorithm separates those tuples of $\mathcal{V}_i$ and $\mathcal{V}_j$ into different regions which do not lie in same range along

Algorithm 3: Intuitive algorithm for creating CFs

```
    Input: All cardinality constraints $\mathbb{C}$, $\mathbb{A}_{common}$
    Output: List of CFs
1   IntuitiveCreateCF ($\mathbb{C}, \mathbb{A}_{common}$)
2       $\mathcal{L}_{CF} \leftarrow \emptyset$
3       foreach $\mathcal{A}_{common} \in \mathbb{A}_{common}$ do
4           foreach $\mathcal{C} \in \mathbb{C}$ do
5               Create a filter of all literals in $\mathcal{C}$ whose attribute is present in $\mathcal{A}_{common}$, and if
                  this filter is non empty then add it to $\mathcal{L}_{CF}$
6       return $\mathcal{L}_{CF}$
```

$\mathcal{A}^{i,j}_{common}$. This makes the resultant regions of $\mathcal{V}_i$ and $\mathcal{V}_j$ to have consistent region boundaries. For example, consider the setup corresponding to Figure 2.1. The CF corresponding to $\mathcal{C}_1$ and $\mathcal{C}_2$ will be $\mathcal{CF}_1 : \sigma_{A\leq 45}(E)$ and $\mathcal{CF}_2 : \sigma_{A\in[20,60)}(E)$ respectively. Note that $\mathcal{C}_1$ was applicable only on sub-view (A, B) and $\mathcal{C}_2$ was applicable only on sub-view (A, S) while both $\mathcal{CF}_1$ and $\mathcal{CF}_2$ are applicable on both the sub-views. When these CFs are given to partitioning algorithm alongside CCs, apart from normal splitting which was done because of CCs, the partitioning algorithm will also split both the sub-views along $A = 20$, $A = 45$ and $A = 60$ as was done by Algorithm 1.

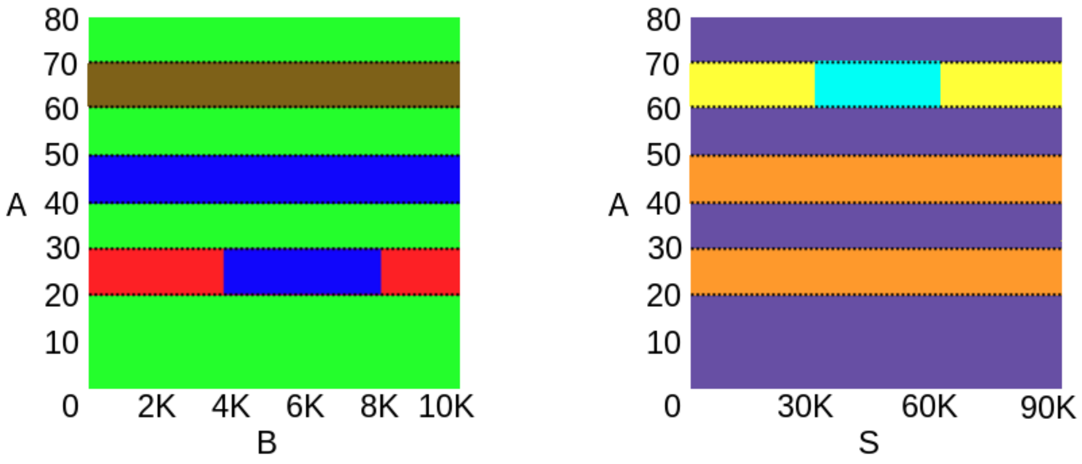
To understand why Algorithm 3 will fail in presence of disjunctions, let's consider the following example: Consider the schema from Table 2.1 and let the CCs be:

$$\begin{aligned}
&\mathcal{C}_1 :\mid \sigma_{(A\in[20,30))\vee(A\in[40,50)\wedge B\in[4000,8000))}(E) \mid= 90\\
&\mathcal{C}_2 :\mid \sigma_{(A\in[60,70)\wedge S\in[30000,60000))}(E) \mid= 160\\
&\mathcal{C}_3 :\mid (E) \mid= 400
\end{aligned}$$

Since B and S never appear together in any CC, hence two sub-views (A, B) and (A, S) will be created. $\mathbb{A}_{common}$ for this setup will be $\{\{A\}\}$. CFs created by Algorithm 3 will be:

$$\begin{aligned}
&\mathcal{CF}_1 : \sigma_{(A\in[20,30))\vee(A\in[40,50))}(E)\\
&\mathcal{CF}_2 : \sigma_{A\in[60,70)}(E)
\end{aligned}$$

The partitions of the two sub-views when the union of CCs and CFs are given to the partitioning algorithm are shown in Figure 3.1 To check if the two sub-views have consistent region boundaries, we project the two sub-views along common attribute (A). It can be observed that after projection, blue region and red region of sub-view (A, B) will intersect but their boundaries will not be the same. Same will be the case with red and orange regions of sub-view



(a) Sub-view (A, B) (b) Sub-view (A, S)

Figure 3.1: Partitions of sub-views

(A, B) and sub-view (A, S) respectively. One important point is that, this type of scenario will happen only when a region have disjoint components, which will happen only when the selection predicate of a constraint contains clauses separated by disjunction. The two clauses of C_1 $(A \in [20, 30)$ and $(A \in [40, 50) \wedge B \in [4000, 8000)))$, which are separated by disjunction, have different attribute set in their literals, allowing the part of blue region where $A \in [40, 50)$ to span completely along attribute B but restricting the part where $A \in [20, 30)$ to only span across attribute B where $B \in [4000, 8000)$. CF_1, which was based on C_1, created the red region but did not create a corresponding red region where $A \in [40, 50)$ resulting in regions being inconsistent. Hence, if a CC C_i have clauses separated by disjunction, then the clauses of the CF created from C_i must be separated into different CFs if the attribute set of the literals that were dropped from the clauses of C_i are different. For our running example, for C_1, the set of attributes appearing in the literals which are dropped from the clauses $A \in [20, 30)$ and $(A \in [40, 50) \wedge B \in [4000, 8000))$ are $\emptyset$ and $\{B\}$ respectively, hence the clauses of CF_1 must be separated into two different CFs.

In Algorithm 4, for every CC $\mathcal{C}$ (which is in DNF) we iterate over its clauses as shown in line 6. A clause of $\mathcal{C}$ is denoted by $\mathcal{SC}$ (*Sub Constraint*). We split each $\mathcal{SC}$ into two sets which we call *Kept Part* and *Dropped Part* using subroutine *BreakSC*. Then we find the set $\mathcal{A}_{dropped}$ which is the set of attributes appearing in literals of *Dropped Part*. We keep all the *Kept Parts* which correspond to same $\mathcal{A}_{dropped}$ in a single list using map $\mathcal{M}$. Each list is then converted to a CF by subroutine *CreateCF* which essentially creates disjunction of *Kept Parts* in that list.

After getting CFs from Algorithm 4, the union of CFs and CCs is given to the partitioning algorithm which returns the partitions of sub-views along with a map $\mathcal{M}_i$ for every sub-view

Algorithm 4: Create Consistency Filters

```
Input: All cardinality constraints $\mathbb{C}$, $\mathbb{A}_{common}$
Output: List of CFs
1  CreateConsistencyFilters ($\mathbb{C}, \mathbb{A}_{common}$)
2      $\mathcal{L}_{CF} \leftarrow \emptyset$
3      foreach $\mathcal{A}_{common} \in \mathbb{A}_{common}$ do
4          foreach $\mathcal{C} \in \mathbb{C}$ do
5              $\mathcal{M} \leftarrow \emptyset$
6              foreach $\mathcal{SC} \in \mathcal{C}$ do
7                  $(kp, dp) \leftarrow \texttt{BreakSC}(\mathcal{SC}, \mathcal{A}_{common})$
8                  $\mathcal{A}_{dropped} \leftarrow \{Attrib(l) \mid l \in dp\}$
9                  $\mathcal{M}[\mathcal{A}_{dropped}] \leftarrow \mathcal{M}[\mathcal{A}_{dropped}] \cup kept$
10             foreach $(\mathcal{A}_{dropped}, \mathcal{L}_{kp}) \in \mathcal{M}$ do
11                 $\mathcal{L}_{CF} \leftarrow \mathcal{L}_{CF} \cup \texttt{CreateCF}(\mathcal{L}_{kp})$
12     return $\mathcal{L}_{CF}$
1  BreakSC ($\mathcal{SC}$ , $\mathcal{A}_{common}$)
2      $kept \leftarrow \emptyset$
3      $dropped \leftarrow \emptyset$
4      foreach $literal \in \mathcal{SC}$ do
5          if $Attribute(literal) \in \mathcal{A}_{common}$ then
6              $kept \leftarrow kept \cup literal$
7          else
8              $dropped \leftarrow dropped \cup literal$
9      return $(kept, dropped)$
```

$\mathcal{V}_i$. $\mathcal{M}_i$ maps every region of $\mathcal{V}_i$ to the set of CFs it satisfies i.e. $\mathcal{M}_i : Regions\ of\ \mathcal{V}_i \mapsto \{CF\}$. For every map $\mathcal{M}_i$, it's inverted map $\mathcal{M}'_i$ is created which maps from the set of CFs to the set of regions which satisfy all CFs in key set i.e. $\mathcal{M}'_i : \{CF\} \mapsto \{Regions\}$. If T_i and T_j are the values of $\mathcal{M}'_i$ and $\mathcal{M}'_j$ where key is same, the following condition is added to LP:

$$\sum_{Reg^i_k \in T_i} x^i_k = \sum_{Reg^j_k \in T_j} x^j_k \tag{3.1}$$

The above steps essentially equates the sum of the variables corresponding to the regions of the two sub-views that satisfy the same set of CFs.

Chapter 4

Comparison of Hydra and Hydra++

4.1 On Adversarial Workload

We tested Hydra and Hydra++ on the adversarial workload described in Section 2.4. The execution time taken by different components of Hydra and Hydra++ are given in Table 4.1. For

	Partitioning	LP Formulation
Hydra	5 minutes	8 hours
Hydra++	7 seconds	Less than 1 second

Table 4.1: Execution time split-up - Adversarial workload

Hydra, the first column (Partitioning) shows the total time taken up by the partitioning algorithm and Algorithm 1. For Hydra++, it's the time taken up by Algorithm 4 and partitioning algorithm.

It can be seen that Hydra took 5 minutes for partitioning and 8 hours for LP formulation. Since there were a lot of regions with inconsistent boundaries and the subroutine *SplitAll* compares every pair of regions for inconsistency, creating new regions in the process, hence blow-up in time was observed during LP formulation. On the other hand, Hydra++ took only a total of less than 8 secs to complete partitioning and formulate the same LP as Hydra.

The adversarial workload is a comparatively very small workload when compared to industry workloads. Hydra took 8 hours for adversarial workload which does not look problematic since data generation is a one time process. But, for an industry workload as complex as our adversarial workload and with hundreds of relations and thousands of queries, it may take years to generate data, and hence Hydra++ is required.

4.2 On JOB Benchmark

The adversarial workload and the database corresponding to it was created synthetically because of which it may not have been able to capture the aspects of real-world databases. To compare Hydra and Hydra++ on a real-world database, we executed them on the Join Order Benchmark (JOB) [6] which is based on the IMDB dataset. The workload was modified to fit the assumptions of Hydra (or Hydra++).

The size of database was 6 GB with 21 relations where the maximum sized table was 2.3 GB.

Figure 4.1 shows the number of constraints per relation.

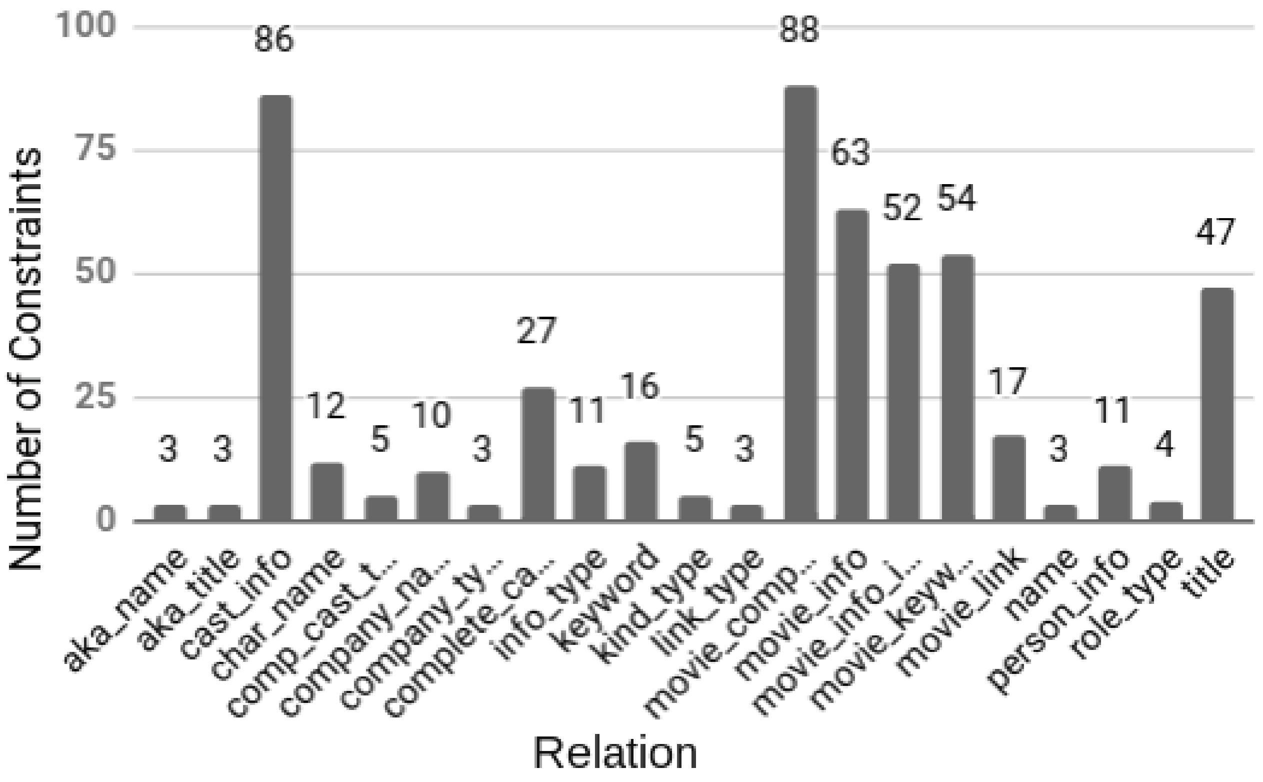


Figure 4.1: Count of constraints per relation in JOB benchmark

Figure 4.2 shows the number of variables in the LP corresponding to each relation. Note that the number of variables in LP formed by Hydra and Hydra++ will be same as Hydra++ produces the same LP as Hydra. In Figure 4.1 we saw that the number of CC applicable on “movie_companies” is almost equal to that of “cast_info” but Figure 4.2 shows that the number of LP variables for “cast_info” is 7 times that of “movie_companies”. This is because

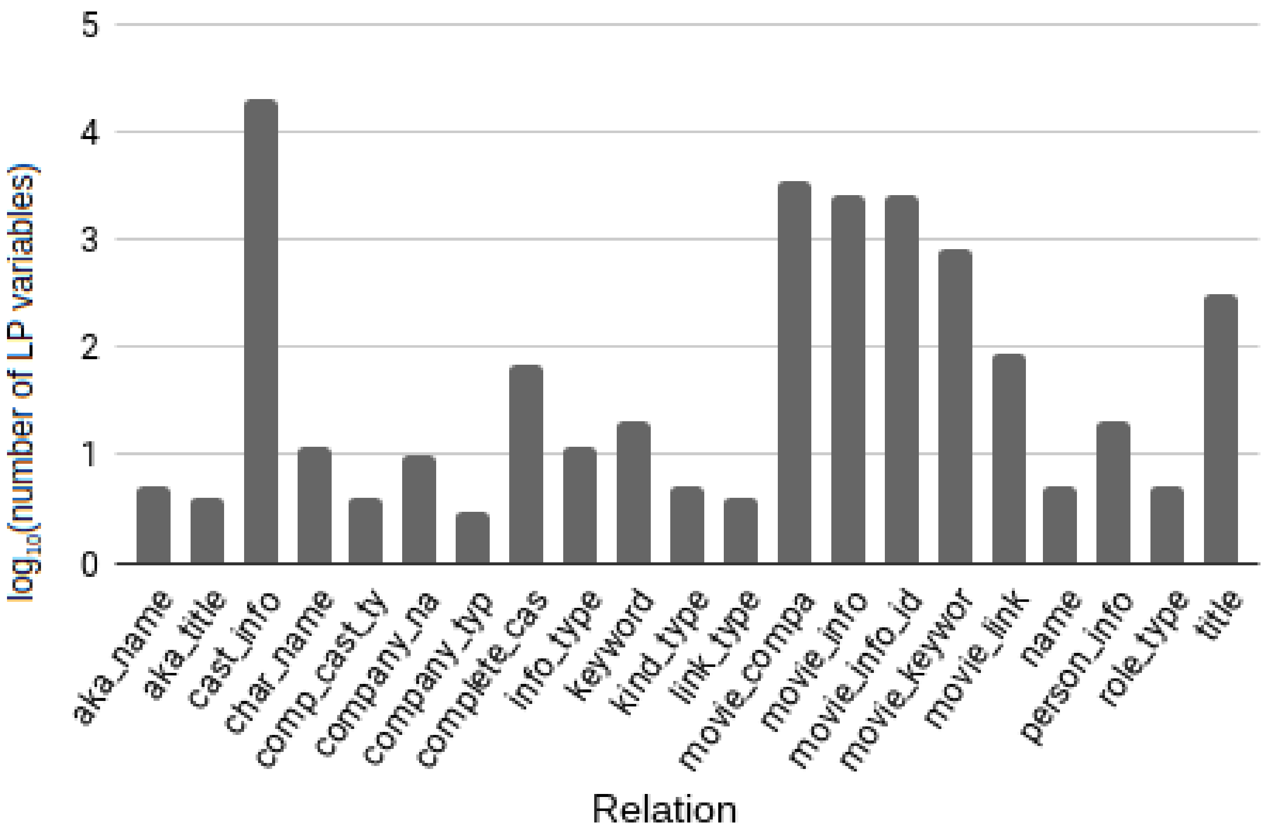


Figure 4.2: Number of variables in LP for JOB benchmark

"cast_info" has sub-views with a lot of regions with inconsistent region boundaries. This again is a motivation for us to have a better algorithm for creating consistency constraints because we believe that real-world workloads will usually show a similar type of behavior.

The execution time statistics for Hydra and Hydra++ on JOB benchmark are given in Table 4.2. The split up of time for the first column is same as was described for Table 4.1. In this

	Partitioning	LP Formulation
Hydra	3 seconds	10 seconds
Hydra++	3 seconds	1 second

Table 4.2: Execution time split-up - JOB benchmark

case also Hydra++ proved to be faster than Hydra, specifically, with a 3 times speedup.

Chapter 5

Hydra Software

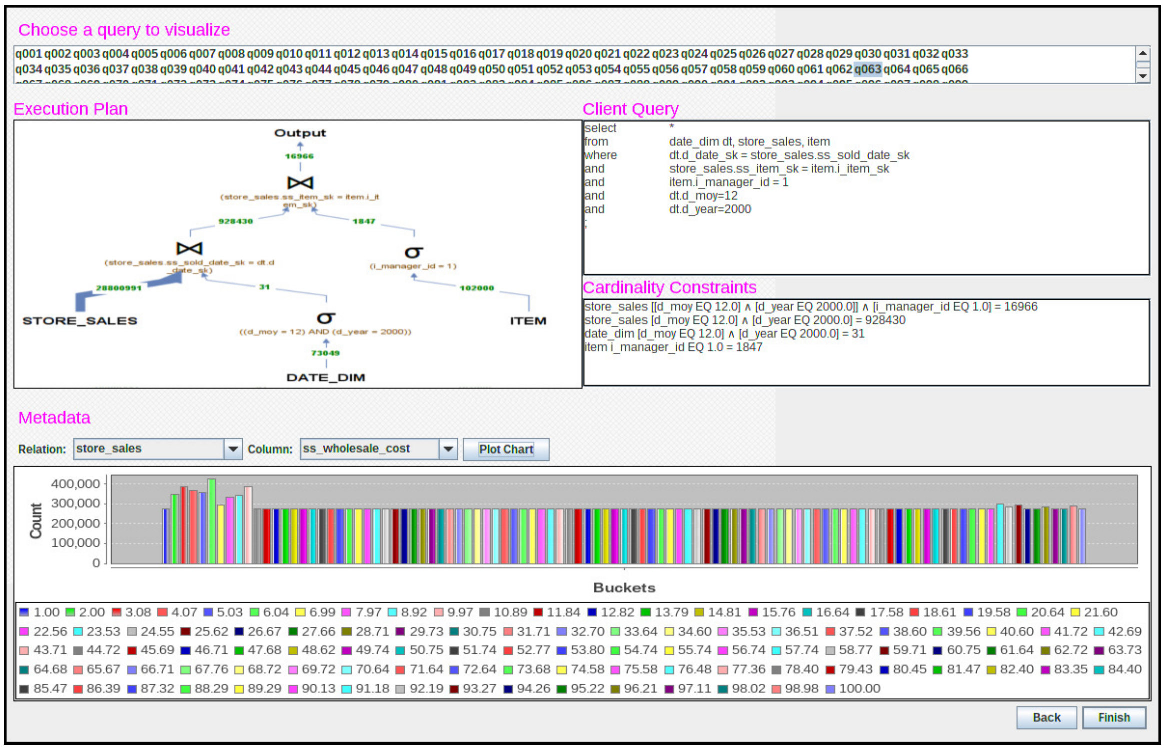
The Hydra prototype was a tool which generated the database with encoded values for attributes of tuples and also a changed schema than the original schema of clients workload because of which the original workload of client could not be used with the generated database. It also facilitated a very basic GUI with a lot of hard coded parameters.

We extended the prototype of Hydra into an end to end software which implements an intuitive interface that facilitates modeling of enterprise database environments, delivers feedback on generated data, and tabulates performance reports on the quality of regeneration.

Hydra is completely written in Java, running to over 15K lines of code. It is currently operational on PostgreSQL engine but requires some changes inside the engine for which the required files are provided with the Hydra package.

At client site, Hydra provides an input panel to acquire connection details of client database and its workload and using this information, generates the set of CCs. Figure 5.1 displays the panel shown after generating CCs. It allows visualizing the plans and CCs corresponding to the queries of client's workload. It also displays the metadata corresponding to the client's database. On clicking the *Finish* button, the CCs and other necessary data required for generating the synthetic database at vendor site is saved which can be then transferred to the vendor site.

At the vendor site, Hydra reads the data files provided by the client. It then formulates the data generation problem as an LP and solves it using Z3 Theorem Prover [3]. During the LP solving stage, a window is shown that tabulates the complexity of the various LPs in terms of their number of variables and their actual run times. After solving the LPs, Hydra creates a small summary of the database. This summary is used to either dump the synthetic database or dynamically generate database when a query is executed. Figure 5.2 shows the panel which allows visualizing the summary corresponding to the queries of the client workload and allows dumping static database.



Choose a query to visualize

q001 q002 q003 q004 q005 q006 q007 q008 q009 q010 q011 q012 q013 q014 q015 q016 q017 q018 q019 q020 q021 q022 q023 q024 q025 q026 q027 q028 q029 q030 q031 q032 q033
q034 q035 q036 q037 q038 q039 q040 q041 q042 q043 q044 q045 q046 q047 q048 q049 q050 q051 q052 q053 q054 q055 q056 q057 q058 q059 q060 q061 q062 q063 q064 q065 q066

Execution Plan

Output
16966
(store_sales.ss_item_sk = item.i_item_sk)
928430
1847
(store_sales.ss_sold_date_sk = dt.d_date_sk)
σ
(i_manager_id = 1)
28800991
31
102000
STORE_SALES
σ
((d_moy = 12) AND (d_year = 2000))
ITEM
73049
DATE_DIM

Client Query

```
select      *
from        date_dim dt, store_sales, item
where       dt.d_date_sk = store_sales.ss_sold_date_sk
and         store_sales.ss_item_sk = item.i_item_sk
and         item.i_manager_id = 1
and         dt.d_moy=12
and         dt.d_year=2000
;
```

Cardinality Constraints

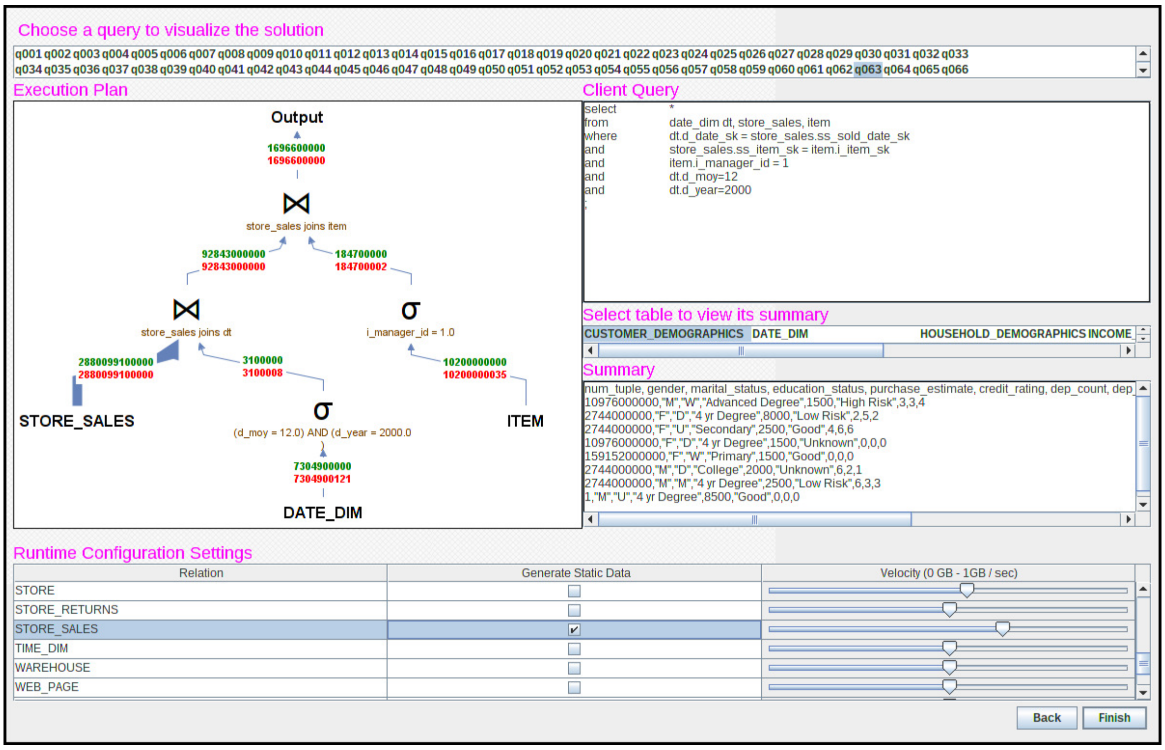
```
store_sales [[d_moy EQ 12.0] ∧ [d_year EQ 2000.0]] ∧ [i_manager_id EQ 1.0] = 16966
store_sales [d_moy EQ 12.0] ∧ [d_year EQ 2000.0] = 928430
date_dim [d_moy EQ 12.0] ∧ [d_year EQ 2000.0] = 31
item i_manager_id EQ 1.0 = 1847
```

Metadata

Relation: store_sales Column: ss_wholesale_cost Plot Chart

1.00 2.00 3.08 4.07 5.03 6.04 6.99 7.97 8.92 9.97 10.89 11.84 12.82 13.79 14.81 15.76 16.64 17.58 18.61 19.58 20.64 21.60
22.56 23.53 24.55 25.62 26.67 27.66 28.71 29.73 30.75 31.71 32.70 33.64 34.60 35.53 36.51 37.52 38.60 39.56 40.60 41.72 42.69
43.71 44.72 45.69 46.71 47.68 48.62 49.74 50.75 51.74 52.77 53.80 54.74 55.74 56.74 57.74 58.77 59.71 60.75 61.64 62.72 63.73
64.68 65.67 66.71 67.76 68.72 69.72 70.64 71.64 72.64 73.68 74.58 75.58 76.48 77.36 78.40 79.43 80.45 81.47 82.40 83.35 84.40
85.47 86.39 87.32 88.29 89.29 90.13 91.18 92.19 93.27 94.26 95.22 96.21 97.11 98.02 98.98 100.00

Figure 5.1: Hydra software client site



Choose a query to visualize the solution

q001 q002 q003 q004 q005 q006 q007 q008 q009 q010 q011 q012 q013 q014 q015 q016 q017 q018 q019 q020 q021 q022 q023 q024 q025 q026 q027 q028 q029 q030 q031 q032 q033
q034 q035 q036 q037 q038 q039 q040 q041 q042 q043 q044 q045 q046 q047 q048 q049 q050 q051 q052 q053 q054 q055 q056 q057 q058 q059 q060 q061 q062 q063 q064 q065 q066

Execution Plan

Client Query

```
select      *
from        date_dim dt, store_sales, item
where       dt.d_date_sk = store_sales.ss_sold_date_sk
and         store_sales.ss_item_sk = item.i_item_sk
and         item.i_manager_id = 1
and         dt.d_moy=12
and         dt.d_year=2000
;
```

Select table to view its summary

CUSTOMER_DEMOGRAPHICS DATE_DIM HOUSEHOLD_DEMOGRAPHICS INCOME

Summary

```
num_tuple, gender, marital_status, education_status, purchase_estimate, credit_rating, dep_count, dep
10976000000,"M","W","Advanced Degree",1500,"High Risk",3,3,4
2744000000,"F","D","4 yr Degree",8000,"Low Risk",2,5,2
2744000000,"F","U","Secondary",2500,"Good",4,6,6
10976000000,"F","D","4 yr Degree",1500,"Unknown",0,0,0
159152000000,"F","W","Primary",1500,"Good",0,0,0
2744000000,"M","D","College",2000,"Unknown",6,2,1
2744000000,"M","M","4 yr Degree",2500,"Low Risk",6,3,3
1,"M","U","4 yr Degree",8500,"Good",0,0,0
```

Runtime Configuration Settings

Relation	Generate Static Data	Velocity (0 GB - 1GB / sec)
STORE	☐	
STORE_RETURNS	☐	
STORE_SALES	☑	
TIME_DIM	☐	
WAREHOUSE	☐	
WEB_PAGE	☐	

Figure 5.2: Hydra software vendor site

Chapter 6

Handling 1-D Projection Constraints

$\mathcal{FC}_1$: $|\ \sigma_{B\in[2000,8000)}(E)\ | = 200$
$\mathcal{PC}_1$: $|\ \pi_A\sigma_{B\in[2000,8000)}(E)\ | = 40$
$\mathcal{FC}_2$: $|\ \sigma_{B\in[4000,8000)}(E)\ | = 130$
$\mathcal{PC}_2$: $|\ \pi_A\sigma_{B\in[4000,8000)}(E)\ | = 25$

Figure 6.1: Projection example 1 constraints

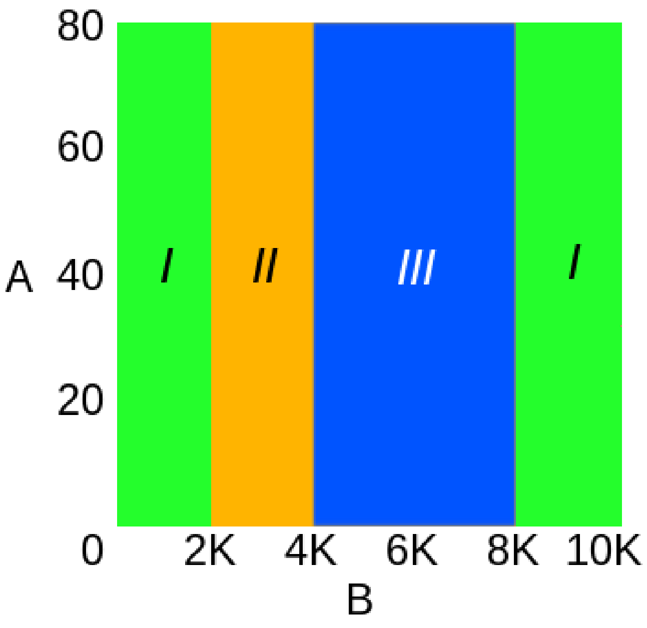


Figure 6.2: Partition for constraints of Figure 6.1

The CCs considered so far were formed out of selection operator only. We now consider CCs which include projection operator alongside selection operator as was introduced in Section 1.1.

Projection operator is introduced in CCs when "GROUP BY" or "DISTINCT" clause is used in SQL queries. It also removes duplicate tuples from the output of a query.

We'll use the term Filter Constraints (FC) for CCs which don't have a projection operator. Handling of projection constraints (PCs) is different from FCs in the sense that in FCs we were only concerned about how many tuples must be created from a region while in case of PCs apart from how many tuples, care must be taken about which tuples to create.

To understand this, let's consider an example with the constraints given in Figure 6.1 whose partition is given in Figure 6.2. If only FCs are considered then a possible solution to the above problem is to generate 70 instances of tuple (0, 2000) and 130 instances of tuple (0, 4000) where the first value in the tuple is for A and the second for B (The same convention will be used for the rest of this section). When PCs are also considered along with FCs, then according to constraint $\mathcal{PC}_1$, projecting the union of all the tuples generated from the region II and the region III along dimension A must give 40 unique values. Similarly for $\mathcal{PC}_2$, projecting all the tuples generated from region III along dimension A must give 25 unique values. A possible solution is to generate 56 instances of tuple (0, 2000) and tuples {(1, 2000), (2, 2000), (.., 2000), (14, 2000)} from region II; and 106 instances of tuple (15, 4000) and 14 tuples {(16, 4000), (17, 4000), (.., 4000), (39, 4000)} from region III.

Our algorithm for handling PCs only handles 1-D projections on a single table i.e. in all the queries, there must be at most one attribute present in GROUP BY or DISTINCT clause (Although that single attribute may be different in different queries) and the queries having GROUP BY or DISTINCT clause must have a single table in their FROM clause.

The algorithm is an addition to Hydra (or Hydra++) and hence follows the same architecture as was described in Section 2.1. The only addition is that now PCs are also transferred to vendor site along with FCs.

6.1 Algorithm

The algorithm given in this section don't support sub-view optimization which was described in Section 2.2. So, in all the cases the relation itself is the one and only sub-view.

We incrementally build the algorithm by showing various cases. For each case, we show the algorithm and its shortcomings also.

6.1.1 Projection is performed on a common attribute across all queries and there is no filter on projected attribute

Let $\mathcal{A}$ denote the attribute on which projection is performed and $\mathbb{Q}_{\mathcal{A}}$ the set of all regions that satisfy the filter of any PC.

Recall that for handling FCs, Hydra used to create an LP variable (x_i) corresponding to every region Reg_i in the partition. Let's call these LP variables as *selection variables.* The value of those selection variables in the LP solution gave the number of tuples to be generated from their corresponding regions.

For handling PCs, alongside selection variables, *projection variables* are created as follows: For n regions $Reg_1, Reg_2, .., Reg_n$ in $\mathbb{Q}_{\mathcal{A}}$, the power set $\mathbb{B}$ of $\mathbb{Q}_{\mathcal{A}}$ is taken. For every set $\{Reg_i, Reg_j, .., Reg_k\}$ in $\mathbb{B}$ except $\emptyset$, a projection variable $y_{(i,j,..,k)}$ is created. The value of $y_{(i,j,..,k)}$ in LP solution gives the number of unique common values which must be used for $\mathcal{A}$ while generating tuples from every region in $\{Reg_i, Reg_j, .., Reg_k\}$.

The following 4 types of LP conditions are created:

1. **For handling FCs :** These are the ones which are created by Hydra.

2. **For handling PCs :** For every PC $\mathcal{PC}$ an LP condition is created: LHS of condition is the sum of all projection variables whose at least one of the corresponding regions satisfy filter of $\mathcal{PC}$. RHS of condition is the RHS value of $\mathcal{PC}$.

3. **Upper bounding count of unique values :** These make sure that the number of unique values in the tuples generated from a region does not exceed the total tuples generated from that region. For each region Reg_i in $\mathbb{Q}_{\mathcal{A}}$ the following condition is added to the LP: $x_i \geq$ Sum of all projection variables corresponding to Reg_i.

4. **Constructibility check :** These conditions ensure that for a region Reg_i in $\mathbb{Q}_{\mathcal{A}}$ if at least one tuple has to be generated from Reg_i i.e. $x_i > 0$ then at least one unique value must be generated for $\mathcal{A}$ from Reg_i. For each region Reg_i in $\mathbb{Q}_{\mathcal{A}}$ the following condition is added to the LP: Sum of all projection variables corresponding to $Reg_i \times N \geq x_i$

After solving the LP, a set of values for $\mathcal{A}$ are associated with every projection variable. The values associated to every projection variable are disjoint i.e. for two projection variables y_i and y_j with values 8 and 5 respectively in LP solution, if values (1, 2, .., 8) are associated to y_i, then, to y_j, values can be associated only from $(D_{\mathcal{A}} - [1, 8])$ where $D_{\mathcal{A}}$ is the domain of $\mathcal{A}$. It is because of this requirement of assigning disjoint values to projection variables that we have taken the power set of regions to create projection variables so that if two regions have to have values which are common, the projection variable which corresponds to both the regions will be assigned the common values. We associate the values sequentially i.e. to the first projection variable y_i, values 0 to $y_i - 1$ are associated, to the next projection variable y_j, values y_i to $y_i + y_j - 1$ are associated, and so on. The values associated to projection variables are used to generate tuples from their corresponding regions present in $\mathbb{Q}_{\mathcal{A}}$. For regions not present in $\mathbb{Q}_{\mathcal{A}}$, the usual method of Hydra is used to generate tuples from them.

Let's take the constraints given in Figure 6.1 to understand the above algorithm. $\mathcal{A}$ is A and $\mathbb{Q}_A$ is $\{Reg_{II}, Reg_{III}\}$. The set of selection and projection variables are $\{x_I, x_{II}, x_{III}\}$ and $\{y_{(II)}, y_{(III)}, y_{(II,III)}\}$ respectively. Following are the LP conditions:

1. $x_{II} + x_{III} = 200$
   $x_{III} = 130$

2. $y_{(II)} + y_{(III)} + y_{(II,III)} = 40$
   $y_{(III)} + y_{(II,III)} = 25$

3. $x_{II} \geq y_{(II)} + y_{(II,III)}$
   $x_{III} \geq y_{(III)} + y_{(II,III)}$

4. $(y_{(II)} + y_{(II,III)}) \times N \geq x_{II}$
   $(y_{(III)} + y_{(II,III)}) \times N \geq x_{III}$

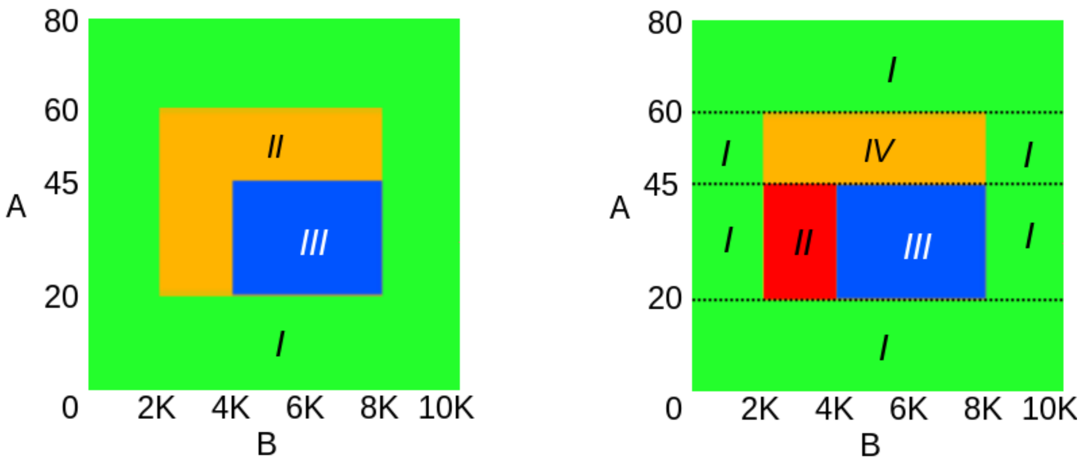
A solution of the LP is $x_{II} = 70$, $x_{III} = 130$, $y_{(II)} = 15$ and $y_{(II,III)} = 25$. Values 0 to 14 are associated to $y_{(II)}$ and values 15 to 39 are associated to $y_{(II,III)}$. Hence, tuples with values 0 to 14 for A must be generated from Reg_{II} and tuples with values 15 to 39 for A must be generated from both Reg_{II} and Reg_{III}. For generating tuples for a region, each value of A associated to that region is used at least once and then the first value is repeated for the leftover tuples (although any associated value can be repeated). This gives the following tuples for the different regions: 31 instances of tuple (0, 2000) and tuples {(1, 2000), (.., 2000), (39, 2000)} for Reg_{II}; and 106 instances of tuple (15, 4000) and tuples {(16, 4000), (17, 4000), (.., 4000), (39, 4000)} for Reg_{III}.

6.1.2 Projection is performed on a common attribute across all queries and there are filters on projected attribute

$\mathcal{FC}_1$:	$\lvert \sigma_{A\in[20,60)\wedge B\in[2000,8000)}(E) \rvert = 180$
$\mathcal{PC}_1$:	$\lvert \pi_A \sigma_{A\in[20,60)\wedge B\in[2000,8000)}(E) \rvert = 33$
$\mathcal{FC}_2$:	$\lvert \sigma_{A\in[20,45)\wedge B\in[4000,8000)}(E) \rvert = 110$
$\mathcal{PC}_2$:	$\lvert \pi_A \sigma_{A\in[20,45)\wedge B\in[4000,8000)}(E) \rvert = 21$

Figure 6.3: Projection example 2 constraints

Consider the constraints given in Figure 6.3 with their partition given in Figure 6.4a. Suppose after creating an LP using the algorithm from previous section, we get the following solution: $x_{II} = 70$, $x_{III} = 110$, $y_{(II)} = 8$, $y_{(III)} = 21$,and $y_{(II,III)} = 0$. Since the domain of A for regions Reg_{II} and Reg_{III} start from 20, hence values for A will be associated to Reg_{II} and Reg_{III} from 20. If values 20 to 27 are associated to $y_{(II)}$ and values 28 to 48 are associated to $y_{(III)}$, then, it creates a problem because values (45, 46, 47, 48) associated to $y_{(III)}$ are outside the domain of Reg_{III} which leads to generation of incorrect tuples for Reg_{III}.



(a) Original partition (b) After intervalization

Figure 6.4: Partition for constraints of Figure 6.3

This problem is solved using *intervalization*. In intervalization, the regions of $\mathbb{Q}_{\mathcal{A}}$ are further refined such that no two regions span across the *split points* of $\mathcal{A}$. The split points are the constants present in literals conditioned on $\mathcal{A}$. For example: For the constraints given in Figure 6.3, the split points of A are 20, 45 and 60, and the effect of intervalization is shown in Figure 6.4b where Reg_{II} of Figure 6.4a is refined into Reg_{II} and Reg_{IV}. After intervalization, $\mathbb{Q}_{\mathcal{A}}$ is updated by removing the old regions which were split and adding the new refined regions. The selection variables are also updated according to the newly formed regions.

The set of regions of $\mathbb{Q}_{\mathcal{A}}$ which fall between two consecutive split points make up a *Split Point Interval (SPI)*. Since the regions of two different SPIs can never share common values of $\mathcal{A}$ as their domains are separated because of intervalization, for creating projection variables, instead of taking the power set of all the regions in $\mathbb{Q}_{\mathcal{A}}$, the power set of regions in $\mathbb{Q}_{\mathcal{A}}$ intersection SPI is taken for every SPI. Also, the following 5^{th} type of LP condition is added to our original 4 LP conditions of the above algorithm:

5. **Domain size constraint :** For each SPI $\mathcal{J}$, the following condition is added to LP : Sum of set of all projection variables corresponding to all regions in $\mathcal{J} \leq D_{\mathcal{J}}$ where $D_{\mathcal{J}}$ is the domain size of any region in $\mathcal{J}$ along $\mathcal{A}$.

Also, because of intervalization, projection variables can be associated values for $\mathcal{A}$ SPI-wise. For each SPI $\mathfrak{I}$ with first value in it's domain $D^1_{\mathfrak{I}}$, associate to the first projection variable $y_{(T1)}$, the values $D^1_{\mathfrak{I}}$ to $(D^1_{\mathfrak{I}} + y_{(T1)} - 1)$, to the second projection variable $y_{(T2)}$, the values $(D^1_{\mathfrak{I}} + y_{(T1)})$ to $(D^1_{\mathfrak{I}} + y_{(T1)} + y_{(T2)} - 1)$, and so on.

To understand the modifications in the algorithm, let's consider the constraints given in Figure 6.3 along with their intervalized partition given in Figure 6.4b. $\mathbb{Q}_A$ is $\{Reg_{II}$, Reg_{III}, $Reg_{IV}\}$. There are two SPIs $\{Reg_{II}$, $Reg_{III}\}$ and $\{Reg_{IV}\}$. The set of selection variables is $\{x_I, x_{II}, x_{III}, x_{IV}\}$. The set of projection variables for regions in first SPI is $\{y_{(II)}$, $y_{(III)}$, $y_{(II,III)}\}$ and for region in second SPI is $\{y_{(IV)}\}$. Following are the LP conditions:

1. $x_{II} + x_{III} + x_{IV} = 180$
   $x_{III} = 110$

2. $y_{(II)} + y_{(III)} + y_{(II,III)} + y_{(IV)} = 33$
   $y_{III} + y_{(II,III)} = 21$

3. $x_{II} \geq y_{(II)} + y_{(II,III)}$
   $x_{III} \geq y_{(III)} + y_{(II,III)}$
   $x_{IV} \geq y_{(IV)}$

4. $(y_{(II)} + y_{(II,III)}) \times N \geq x_{II}$
   $(y_{(III)} + y_{(II,III)}) \times N \geq x_{III}$
   $y_{(IV)} \times N \geq x_{IV}$

5. $y_{(II)} + y_{(III)} + y_{(II,III)} \leq 45 - 20$
   $y_{(IV)} \leq 60 - 45$

A possible solution to the above LP is $x_{II} = 40$, $x_{III} = 110$, $x_{IV} = 30$, $y_{(II)} = 2$, $y_{(III)} = 21$, $y_{(II,III)} = 0$ and $y_{(IV)} = 10$. Values 20 and 21 are associated to $y_{(II)}$, values 22 to 42 are associated to $y_{(III)}$ and values 45 to 54 are associated to $y_{(IV)}$. The tuples generated are: 39 instances of tuple (20, 2000) and tuple (21, 2000) from Reg_{II}; 90 instances of tuple (22, 4000) and tuples {(23, 4000), (24, 4000), (.., 4000), , (42, 4000)} from Reg_{III}; and 21 instances of tuple (45, 2000) and tuples {(46, 2000), (47, 2000), (.., 2000), (54, 2000)} from Reg_{IV}.

6.1.3 Projection is performed on different attributes across queries

Let the different attributes on which projection is performed be $\mathbb{A} = \{\mathcal{A}_1, \mathcal{A}_2, .., \mathcal{A}_m\}$ and let $\mathbb{Q}_{\mathcal{A}_i}$ denote the set of all regions that satisfy filter of any PC having projection on $\mathcal{A}_i$.

The following steps are performed for each $\mathcal{A}_i \in \mathbb{A}$:

1. Intervalization is performed i.e. $\mathbb{Q}_{\mathcal{A}_i}$ is refined along the split points of $\mathcal{A}_i$.
2. Set of SPIs $\mathbb{I}_{\mathcal{A}_i}$ is created where each SPI is a set of regions of $\mathbb{Q}_{\mathcal{A}_i}$ which fall between two consecutive split points of $\mathcal{A}_i$.
3. For each SPI $\mathfrak{I} \in \mathbb{I}_{\mathcal{A}_i}$, the power set of the regions in $\mathfrak{I}$ is taken and a projection variable $y_T^{\mathcal{A}_i}$ is created corresponding to each set T in the power set.

The selection variables are created as usual i.e. a selection variable x_i per region Reg_i of partition.

Let's denote the set of all projection variables corresponding to $\mathcal{A}_i$ by $\mathbb{V}_{\mathcal{A}_i}$ The changes in the creation of the 5 types of LP conditions are as follows:

1. Will remain same
2. For every PC $\mathcal{PC}$ with projection on $\mathcal{A}_i$ an LP condition is created: LHS of condition is the sum of all projection variables of $\mathbb{V}_{\mathcal{A}_i}$ whose at least one of the corresponding regions satisfy filter of $\mathcal{PC}$. RHS of condition is the RHS value of $\mathcal{PC}$.

   For each $\mathcal{A}_i \in \mathbb{A}$: For each region Reg_j in $\mathbb{Q}_{\mathcal{A}_i}$ the following two conditions are added to LP:
3. $x_j \geq$ Sum of all projection variables in $\mathbb{V}_{\mathcal{A}_i}$ corresponding to Reg_j
4. Sum of all projection variables in $\mathbb{V}_{\mathcal{A}_i}$ corresponding to $Reg_j \times N \geq x_j$
5. For each $\mathcal{A}_i \in \mathbb{A}$: For each SPI $\mathfrak{I} \in \mathbb{I}_{\mathcal{A}_i}$, the following condition is added: Sum of set of all projection variables corresponding to all regions in $\mathfrak{I} \leq D_{\mathfrak{I}}$ where $D_{\mathfrak{I}}$ is the domain size of any region in $\mathfrak{I}$ along $\mathcal{A}_i$.

$$\boxed{\begin{array}{l} \mathcal{FC}_1\text{: } \mid \sigma_{A<45 \wedge B\in[2000,4000)}(E) \mid = 60 \\ \mathcal{PC}_1\text{: } \mid \pi_B \sigma_{A<45 \wedge B\in[2000,4000)}(E) \mid = 18 \end{array}}$$

Figure 6.5: Projection example 3 constraints

Let's consider the constraints given in Figure 6.3 and Figure 6.5 together to understand the algorithm. The partition after intervalization for this case is given in Figure 6.6. It can be noted that since Reg_{IV} does not satisfy filter of any PC which have projection on B, hence region Reg_{IV} is not split along split point 4000 of B.
$\mathbb{A} = \{A, B\}$

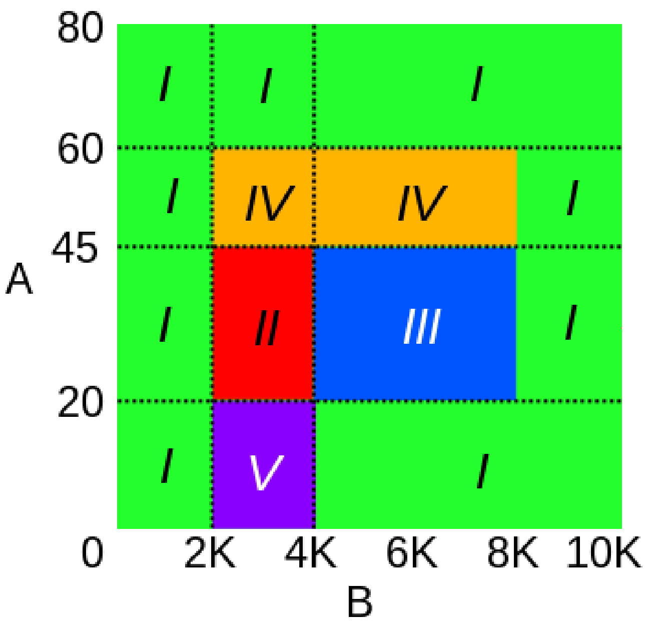


Figure 6.6: Partition for constraints from Figure 6.3 and 6.5 taken together

$\mathbb{Q}_A = \{Reg_{II}, Reg_{III}, Reg_{IV}\}$
$\mathbb{Q}_B = \{Reg_{II}, Reg_V\}$
$\mathbb{I}_A = \{\{Reg_{II}, Reg_{III}\}, \{Reg_{IV}\}\}$
$\mathbb{I}_B = \{\{Reg_{II}, Reg_V\}\}$
The projection variables corresponding to A, $\mathbb{V}_A$, are $\{y^A_{(II)}, y^A_{(III)}, y^A_{(II,III)}, y^A_{(IV)}\}$
The projection variables corresponding to B, $\mathbb{V}_B$, are $\{y^B_{(II)}, y^B_{(V)}, y^B_{(II,V)}\}$

Following are the LP conditions:

1. $x_{II} + x_{III} + x_{IV} = 180$
   $x_{III} = 110$
   $x_{II} + x_V = 60$

2. $y^A_{(II)} + y^A_{(III)} + y^A_{(II,III)} + y^A_{(IV)} = 33$
   $y^A_{III} + y^A_{(II,III)} = 21$
   $y^B_{II} + y^B_V + y^B_{(II,V)} = 18$

3. $x_{II} \geq y^A_{(II)} + y^A_{(II,III)}$
   $x_{III} \geq y^A_{(III)} + y^A_{(II,III)}$
   $x_{IV} \geq y^A_{(IV)}$
   $x_{II} \geq y^B_{(II)} + y^B_{(II,V)}$
   $x_V \geq y^B_{(V)} + y^B_{(II,V)}$

4. $(y^A_{(II)} + y^A_{(II,III)}) \times N \geq x_{II}$
   $(y^A_{(III)} + y^A_{(II,III)}) \times N \geq x_{III}$
   $y^A_{(IV)} \times N \geq x_{IV}$
   $(y^B_{(II)} + y^B_{(II,V)}) \times N \geq x_{II}$
   $(y^B_{(V)} + y^B_{(II,V)}) \times N \geq x_V$

5. $y^A_{(II)} + y^A_{(III)} + y^A_{(II,III)} \leq 45 - 20$
   $y^A_{(IV)} \leq 60 - 45$
   $y^B_{(II)} + y^B_{(V)} + y^B_{(II,V)} \leq 4000 - 2000$

A possible solution to the above LP is $x_{II} = 21$, $x_{III} = 110$, $x_{IV} = 49$, $x_V = 39$, $y^A_{(II)} = 0$, $y^A_{(III)} = 0$, $y^A_{(II,III)} = 21$, $y^A_{(IV)} = 12$, $y^B_{(II)} = 0$, $y^B_{(V)} = 0$ and $y^B_{(II,V)} = 18$. Values 20 and 40 are associated to $y^A_{(II,III)}$, values 45 to 56 are associated to $y^A_{(IV)}$ and values 2000 to 2017 are associated to $y^B_{(II,V)}$. The tuples generated are: Tuples {(20, 2000), (21, 2001), (.., ..), (37, 2017)} and tuples {(38, 2000), (39, 2000), (40, 2000)} from Reg_{II}; 90 instances of tuple (20, 4000) and tuples {(21, 4000), (22, 4000), (.., 4000), (40, 4000)} from Reg_{III}; 38 instances of tuple (45, 2000) and tuples (46, 2000), (47, 2000), (.., 2000), , (56, 2000) from Reg_{IV}; and 22 instances of tuple (0, 2000) and tuples {(0, 2001), (0, 2002), (0, ..), (0, 2017)} from Reg_V.

6.2 Sub-view optimization with Projection Constraints

The algorithm in Section 6.1 didn't used sub-view optimization which lead to the following two problems:

1. The projection variables were created by taking the power set of the regions of partition. Since the number of regions in a partition depends on the number of attributes in that partition, it became infeasible in our experiments to compute the power set of the regions if sub-view optimization was not used.

2. The complexity of partitioning algorithm depends on the number of attributes in a sub-view. A blow up in the time and memory required by the partitioning algorithm was observed when the sub-view optimization was not used.

The requirement for using sub-view optimization, as explained in Section 2.2, is that for any two sub-views $\mathcal{V}_i$ and $\mathcal{V}_j$ the distribution of values corresponding to the common attributes must be same. In absence of PCs, this was achieved using consistency constraints which made sure that the sum of selection variables corresponding to regions of $\mathcal{V}_i$ and $\mathcal{V}_j$ which satisfy the same set of CFs is equal (Equation 3.1). Then, while creating tuples from those regions the least values of common attributes in those regions was chosen, which was actually same in all the regions. The least value cannot always be chosen in presence of PCs because there may be requirement of multiple unique values for common attributes.

Following are the changes in pipeline of hydra for handling PCs alongside sub-view optimization:

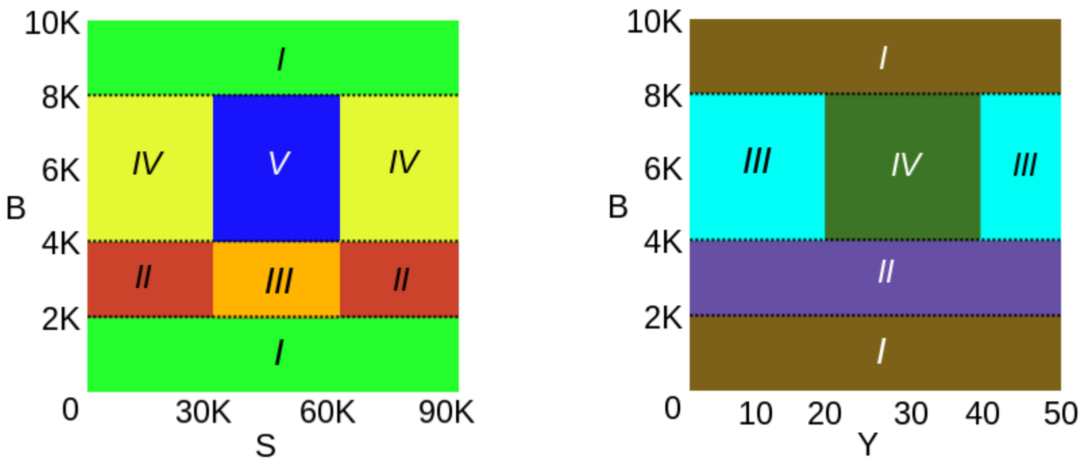
1. All the operations done in Hydra up to the formation of sub-views is same.
2. After forming sub-views, an LP is formed per sub-view using the algorithm of Section 6.1.3.
3. Consistency constraints are created using algorithms of Section 3 and added to the LP.
4. **Phase 1 of Projection Consistency**
5. **Phase 2 of Projection Consistency**

6.2.1 Phase 1 of Projection Consistency

Let $\mathbb{A}^{i,j}_{p-common}$ denote the set of all common attributes of the two sub-views $\mathcal{V}_i$ and $\mathcal{V}_j$, on which projection is performed in some PC applicable on any or both of the sub-views. Also, let $\mathbb{Q}^{j}_{\mathcal{A}_i}$ denote the set of all regions of sub-view $\mathcal{V}_j$ which satisfy filter of some PC applicable on $\mathcal{V}_j$ having projection on attribute $\mathcal{A}_i$. Define *Consistent Pair* (CP) as a pair $\langle\{Reg\},\{Reg\}\rangle$ of set of regions of two sub-views which satisfy the same set of CFs. Let the list of all CPs corresponding to two sub-views $\mathcal{V}_i$ and $\mathcal{V}_j$ be denoted by $\mathbb{G}_{i,j}$.

In this phase, LP conditions are created which make sure that for any two sub-views $\mathcal{V}_i$ and $\mathcal{V}_j$, the number of unique values which can be used to generate tuples from the two sets of regions in pairs of $\mathbb{G}_{i,j}$, is same for every attribute in $\mathbb{A}^{i,j}_{p-common}$.

Consider the schema of Table 2.1 along with an extra attribute *Years_Remaining* with alias Y, and the constraints given in Figure 6.8. We'll use this setup as a running example to explain the concepts presented in this section. There are two sub-views (A, B, S) and (A, B, Y). The partitions of the two sub-views are given in Figure 6.7. Since there are no constraints on regions where $A < 25$ or $A \geq 60$, hence we have not shown the axis corresponding to attribute A and only shown the slice where $A \in [25, 60)$. We'll use $\mathcal{V}_1$ and $\mathcal{V}_2$ to denote sub-views (A, B, S) and (A, B, Y) respectively. The values of different variables explained till now will be:
$\mathbb{A}^{1,2}_{p-common} = \{A\}$
$\mathbb{Q}^{1}_{A} = \{Reg^1_{III}, Reg^1_{V}\};$ $\qquad\mathbb{Q}^{2}_{A} = \{Reg^2_{IV}\}$
$\mathbb{G}_{1,2} = \{\langle\{Reg^1_{IV}, Reg^1_{V}\}, \{Reg^2_{III}, Reg^2_{IV}\}\rangle,$
$\langle\{Reg^1_{II}, Reg^1_{III}\}, \{Reg^2_{II}\}\rangle\}$



(a) Sub-view (A, B, S) (b) Sub-view (A, B, Y)

Figure 6.7: Partitions for constraints of Figure 6.8

$\mathcal{FC}_1$:	$\mid \sigma_{A\in[25,60)\wedge B\in[4000,8000)\wedge S\in[30000,60000)}(E) \mid= 60$
$\mathcal{PC}_1$:	$\mid \pi_A\sigma_{A\in[25,60)\wedge B\in[4000,8000)\wedge S\in[30000,60000)}(E) \mid= 20$
$\mathcal{FC}_2$:	$\mid \sigma_{A\in[25,60)\wedge B\in[2000,8000)\wedge S\in[30000,60000)}(E) \mid= 90$
$\mathcal{PC}_2$:	$\mid \pi_A\sigma_{A\in[25,60)\wedge B\in[2000,8000)\wedge S\in[30000,60000)}(E) \mid= 27$
$\mathcal{FC}_3$:	$\mid \sigma_{A\in[25,60)\wedge B\in[4000,8000)\wedge Y\in[20,40)}(E) \mid= 50$
$\mathcal{PC}_3$:	$\mid \pi_A\sigma_{A\in[25,60)\wedge B\in[4000,8000)\wedge Y\in[20,40)}(E) \mid= 15$

Figure 6.8: Projection example 4 constraints

For an attribute $\mathcal{A}$ and some CP $\langle\mathbb{R}_i, \mathbb{R}_j\rangle$ in $\mathbb{G}_{i,j}$ corresponding to sub-views $\mathcal{V}_i$ and $\mathcal{V}_j$, it might be possible that not all the regions of $\mathbb{R}_i$ are contained in $\mathbb{Q}^i_{\mathcal{A}}$ and not all the regions of $\mathbb{R}_j$ are contained in $\mathbb{Q}^j_{\mathcal{A}}$ i.e. there might be regions in $\mathbb{R}_i$ and $\mathbb{R}_j$ which do not satisfy filter of any PC having projection on $\mathcal{A}$. Any value of $\mathcal{A}$ can be used for generating tuples from those regions.

Let's denote the set of regions of $\mathbb{R}_i$ which are also present in $\mathbb{Q}^i_{\mathcal{A}}$ by $\mathbb{R}^{\mathcal{A}}_i$ and those which are not present in $\mathbb{Q}^i_{\mathcal{A}}$ by $\mathbb{R}^{\bar{\mathcal{A}}}_i$. The number of unique values which are to be used for $\mathcal{A}$ for generating tuples from regions of $\mathbb{R}^{\mathcal{A}}_i$ are given by the projection variables corresponding to those regions. Let the sum of projection variables corresponding to regions of $\mathbb{R}^{\mathcal{A}}_i$ and $\mathbb{R}^{\mathcal{A}}_j$ be d_i and d_j respectively, and let $d_i > d_j$. Then, $d_i - d_j$ unique values for $\mathcal{A}$, which are different from those used in creating tuples from regions of $\mathbb{R}^{\mathcal{A}}_i$, must be mandatorily used for generating tuples from regions of $\mathbb{R}^{\bar{\mathcal{A}}}_j$. This is possible only if the sum of selection variables of $\mathbb{R}^{\bar{\mathcal{A}}}_j$ is greater than $d_i - d_j$. To ensure that this requirement is met, *slack variables* are used that are created using Algorithm 5. The partitioning of Step 4 of Algorithm 5 is required because it makes the elements of $\mathcal{L}'$ to be disjoint because of which the slack variables created are independent of each other. The requirement for having independent slack variables is explained later when the tuples are generated. Note that, the values which had been used for generating tuples from region of $\mathbb{R}_j^{\mathcal{A}}$ can be reused to generate tuples from regions of $\mathbb{R}_j^{\overline{\mathcal{A}}}$ if required.

For our running example, for first pair in $\mathbb{G}_{1,2}$:
$\mathbb{R}_1^{A} = \{Reg_V^1\}$; $\mathbb{R}_1^{\overline{A}} = \{Reg_{IV}^1\}$
$\mathbb{R}_2^{A} = \{Reg_{IV}^2\}$; $\mathbb{R}_2^{\overline{A}} = \{Reg_{III}^2\}$
For second pair:
$\mathbb{R}_1^{A} = \{Reg_{III}^1\}$; $\mathbb{R}_1^{\overline{A}} = \{Reg_{II}^1\}$
$\mathbb{R}_2^{A} = \emptyset$; $\mathbb{R}_2^{\overline{A}} = \{Reg_{II}^2\}$
For $\mathcal{V}_1$, $\mathcal{L} = [\{Reg_{IV}^1\}, \{Reg_{II}^1\}]$. $\mathcal{L}'$ will be same as $\mathcal{L}$ and a slack variable will be created for each set $\{Reg_{IV}^1\}$ and $\{Reg_{II}^1\}$. If suppose $\mathcal{L}$ was $[\{Reg_{IV}^1, Reg_{II}^1\}, \{Reg_{II}^1\}]$, then, $\mathcal{L}'$ would have been $[\{Reg_{IV}^1\}, \{Reg_{II}^1\}]$ because Reg_{IV}^1 is present only in first element of $\mathcal{L}$ while Reg_{II} is present in both the elements of $\mathcal{L}$. For $\mathcal{V}_2$, $\mathcal{L} = [\{Reg_{III}^2\}, \{Reg_{II}^2\}]$ and $\mathcal{L}'$ will be same as $\mathcal{L}$.

Algorithm 5: Create Slack Variables

```
1 foreach Sub-view $\mathcal{V}_i$ do
2     foreach Attribute $\mathcal{A}$ of $\mathcal{V}_i$ do
3         Let $\mathcal{L}$ be the list of all possible $\mathbb{R}_i^{\overline{\mathcal{A}}}$ across all the $\mathbb{G}_{i,j}$s where $j$ corresponds to all
            the other views except $\mathcal{V}_i$.
4         The regions contained in the elements of $\mathcal{L}$ are partitioned into sets of regions $\mathcal{L}'$
            such that all the regions of each set $L' \in \mathcal{L}'$ are present in exactly the same
            elements of $\mathcal{L}$.
5         A slack variable $z_k$ which corresponds to attribute $\mathcal{A}$ is created for every $L'_k \in \mathcal{L}'$
            and the following condition is added to LP:
6         $z_k \leq$ Sum of selection variables corresponding to regions in $L'_k$
```

The LP conditions for phase 1 are created as follows: For every pair $\langle \mathbb{R}_i, \mathbb{R}_j \rangle \in \mathbb{G}_{i,j}$ and attribute $\mathcal{A} \in \mathbb{A}_{p-common}^{i,j}$ corresponding to every pair $\mathcal{V}_i$ and $\mathcal{V}_j$, the following condition is added to LP: Sum of projection variables corresponding to regions of $\mathbb{R}_i^{\mathcal{A}}$ + sum of slack variables corresponding to regions of $\mathbb{R}_i^{\overline{\mathcal{A}}}$ = Sum of projection variables corresponding to regions of $\mathbb{R}_j^{\mathcal{A}}$ + sum of slack variables corresponding to regions of $\mathbb{R}_j^{\overline{\mathcal{A}}}$.

The objective function of the LP is set to minimize the sum of all slack variables. The reason for the choice of the objective function is explained in the next section. We now solve the LP.

6.2.2 Phase 2 of Projection Consistency

In this phase, first we assign values to the projection variables, and then determine frequency of every unique value such that the sub-view solutions can be merged.

For a pair $\langle \mathbb{R}_i, \mathbb{R}_j \rangle$ in some $\mathbb{G}_{i,j}$ and an attribute $\mathcal{A}$, if the sum of projection variables corresponding to regions of $\mathbb{R}_i^{\mathcal{A}}$ and $\mathbb{R}_j^{\mathcal{A}}$ is not same, then Phase 1 ensures that we are able to generate tuples with unique values equal to the difference of their sum from $\mathbb{R}_i^{\overline{\mathcal{A}}}$ or $\mathbb{R}_j^{\overline{\mathcal{A}}}$ appropriately. Let $\langle \mathbb{S}_i, \mathbb{S}_j \rangle$ be another pair from $\mathbb{G}_{i,j}$, such that some region $\mathcal{R}_i$ of $\mathbb{R}_i^{\mathcal{A}}$ and some region $\mathcal{S}_i$ of $\mathbb{S}_i^{\mathcal{A}}$ share a common projection variable y_{i1} having value 5 in LP solution and all the other shared projection variables have value 0 in LP solution. Similarly, suppose same is the case with $\mathbb{R}_j^{\mathcal{A}}$ and $\mathbb{S}_j^{\mathcal{A}}$ where the shared common projection variable having value 5 is y_{j1} and the regions are $\mathcal{R}_j$ of $\mathbb{R}_j^{\mathcal{A}}$ and $\mathcal{S}_j$ of $\mathbb{S}_j^{\mathcal{A}}$. Also, suppose there are projection variables y_{i2} and y_{j2} corresponding to regions $\mathcal{R}_i$ and $\mathcal{R}_j$ each having value 5 in LP solution, and all the other projection variables have value 0. Let the values assigned to y_{i1} are 1 to 5 and to y_{i2} are 6 to 10, but in $\mathbb{S}_i^{\mathcal{A}}$, values 1 to 5 are assigned to y_{j2} and values 6 to 10 are assigned to y_{j1}. This makes pair $\langle \mathbb{R}_i, \mathbb{R}_j \rangle$ to have same unique values, but creates inconsistency for the pair $\langle \mathbb{S}_i, \mathbb{S}_j \rangle$ as the values which regions $\mathcal{S}_i$ and $\mathcal{S}_j$ get are 1 to 5 and 6 to 10, respectively, because of y_{i1} and y_{j1}. Because of this inconsistency the values 6 to 10 have to be used in tuples generated from $\mathbb{S}_i^{\overline{\mathcal{A}}}$ and the values 1 to 5 have to be used in tuples generated from $\mathbb{S}_j^{\overline{\mathcal{A}}}$. This might not always be possible as the size of slack ($\mathbb{S}_i^{\overline{\mathcal{A}}}$ and $\mathbb{S}_j^{\overline{\mathcal{A}}}$) might not be enough. In that case, we we have to backtrack and assign values 1 to 5 to y_{j1} and values 6 to 10 to y_{j2}.

In general, this problem of assignment of values to the projection variables in the context of sub-view optimization is non-trivial and we suspect that it is NP hard. We use the following heuristic algorithm to assign values to projection variables.

Consistent Tuples (CTs) are created using CPs. Consider three sub-views $\mathcal{V}_i$, $\mathcal{V}_j$ and $\mathcal{V}_k$ such that attribute $\mathcal{A}$ and $\mathcal{B}$ are common in all of them and attribute $\mathcal{E}$ is common in $\mathcal{V}_i$ and $\mathcal{V}_j$. Consider a pair $\langle \mathbb{R}_i, \mathbb{R}_j \rangle$ from $\mathbb{G}_{i,j}$ and a pair $\langle \mathbb{R}'_j, \mathbb{R}_k \rangle$ from $\mathbb{G}_{j,k}$ where $\mathbb{R}_j$ and $\mathbb{R}'_j$ is same. Then the tuple $\langle \mathbb{R}_i, \mathbb{R}_j, \mathbb{R}_k \rangle_{\mathcal{A}}$ forms a CT which corresponds to attribute $\mathcal{A}$ i.e. the frequency distribution of values of attribute $\mathcal{A}$ must be same in regions of $\mathbb{R}_i$, $\mathbb{R}_j$ and $\mathbb{R}_k$. Similarly CTs $\langle \mathbb{R}_i, \mathbb{R}_j, \mathbb{R}_k \rangle_{\mathcal{B}}$ and $\langle \mathbb{R}_i, \mathbb{R}_j \rangle_{\mathcal{E}}$ are formed. When we talk about projection variables or slack variables in context of a CT, they correspond to the attribute that the CT corresponds to. In a CT there is at least one element (set of regions) such that sum of slack variables corresponding to regions of that element has a sum 0 in LP solution. This happens because of the LP's objective function. We create all the possible CTs using existing CPs.

For each CT which corresponds to attribute $\mathcal{A}$, one element (set of regions) $\mathbb{R}$ whose corresponding slack variables have sum 0 is selected and values for $\mathcal{A}$ are assigned to projection variables corresponding to regions of $\mathbb{R}^{\mathcal{A}}$. The values are assigned sequentially i.e. the next available values are assigned. Some projection variables which are shared with other regions may already have been assigned so they are not reassigned. Then, for every other element $\mathbb{S}$ of the CT, an attempt is made to assign those values to the projection variables of $\mathbb{S}^{\mathcal{A}}$ which already have been assigned to the projection variables of $\mathbb{R}^{\mathcal{A}}$. It may be possible that some value which had been assigned to some projection variable of $\mathbb{R}^{\mathcal{A}}$ can't be assigned to projection variables of $\mathbb{S}^{\mathcal{A}}$ because it has already been assigned to some other projection variable. In that case, we assume that the value which was not available will be used to generate tuples from $\mathbb{S}^{\bar{\mathcal{A}}}$ and assign the next available values to the leftover projection variables of $\mathbb{S}^{\mathcal{A}}$. It might also be possible that we need to assign values to projection variables of $\mathbb{S}^{\mathcal{A}}$ which were not assigned to any projection variable of $\mathbb{R}^{\mathcal{A}}$. In that case, we assume that these values will be used to generate tuples from regions of $\mathbb{R}^{\bar{\mathcal{A}}}$. The above assumptions might not always hold while generating tuples and we might not be able to make the frequency distribution of common attributes same across sub-views in which case the heuristic algorithm has failed. Our heuristic algorithm doesn't work for every case but our empirical study indicates that it works for most of the cases.

For our example from previous section, since there are only two sub-views, hence the CPs themselves will be the CTs and they will correspond to attribute A. Let, after solving the LP, the values of different projection variables be:
In $\mathcal{V}_1$, $y^{A}_{(III,V)} = 20$, $y^{A}_{(III)} = 7$
In $\mathcal{V}_2$, $y^{A}_{(IV)} = 15$
Then, for CT $\{\langle\{Reg^1_{IV}, Reg^1_V\}, \{Reg^2_{III}, Reg^2_{IV}\}\rangle$, the element whose corresponding slack variables have sum 0 is $\{Reg^1_{IV}, Reg^1_V\}$. So, Reg^1_{IV} will be assigned values 0 to 19. Now the algorithm attempts to assign the same values to Reg^2_{IV} and hence assigns it values 0 to 14 and assumes that values 15 to 19 will be used to generate tuples from Reg^2_{III}. Suppose in $\mathcal{V}_2$ the values 0 to 9 were already assigned to some other projection variable, then, our algorithm will assign values 10 to 24 to Reg^2_{IV} and assumes that values 0 to 9 will be used in generating tuples from Reg^2_{III}. Since values 20 to 24 were not assigned to Reg^1_V, the algorithm also assumes that these values will be used in generating tuples from Reg^1_{IV}.

Now, the frequency of every unique value is determined such that the sub-view solutions can be merged. For the regions which do not appear in any CT, at least one tuple is created corresponding to every unique value assigned to its projection variables and then any value can be repeated for creating remaining tuples. For the case of CTs, an LP is created for each CT and the solution of that LP is used to create tuples from the regions contained in the elements of CT. For a CT which corresponds to the attribute $\mathcal{A}$ the LP is created using Algorithm 6.

Algorithm 6: Create LP for CT

```
1  $\mathcal{H}$ = set of all values which are assigned to the projection variables corresponding to
     regions contained in the elements of the CT.
2  foreach $\mathbb{R} \in CT$ do
3      foreach $\mathcal{R} \in \mathbb{R}^{\mathcal{A}}$ do
4          An LP variable $n\_\mathcal{R}\_v$ is created for each value $v$ assigned to the projection
             variables corresponding to $\mathcal{R}$ which represents the frequency of value $v$ to be
             used while creating tuples from $\mathcal{R}$.
5          Since the frequency of every LP variable $n\_\mathcal{R}\_v$ must be at least 1, the condition
             $n\_\mathcal{R}\_v \geq 1$ is added to the LP.
6          Then, the following condition is added to the LP: Sum of all LP variables created
             in above step = Value of selection variable of $\mathcal{R}$
7      foreach Slack variable $z$ corresponding to $\mathbb{R}^{\overline{\mathcal{A}}}$ do
8          An LP variable $n\_z\_v$ is created for each value $v \in \mathcal{H}$ which represents the
             frequency of value $v$ to be used while creating tuples from regions corresponding
             to $z$.
9          The non-negativity condition $n\_z\_v \geq 0$ is added to the LP for each $v \in \mathcal{H}$.
10         Then, the following condition is added to the LP: Sum of all LP variables created
             in above step = Sum of values of selection variables of regions corresponding to
             $z$.
11 Let $\mathbb{R}$ be the first element of CT
12 foreach $\mathbb{S} \in CT, \mathbb{S} \neq \mathbb{R}$ do
13     foreach $v \in \mathcal{H}$ do
14         The following condition is added to the LP: The sum of all LP variables created
             corresponding to $v$ and $\mathbb{R}$ = The sum of all LP variables created corresponding
             to $v$ and $\mathbb{S}$
```

Once the LP is solved, the LP variables corresponding to the regions and the slack variables give the exact frequency for every value that must be used for generating tuples from that region and the regions corresponding to the slack variables respectively. The $n\_\mathcal{R}\_v$ tuples are generated from region $\mathcal{R}$ having the value v for the attribute the LP variable corresponds to. Since the regions corresponding to any two slack variables are always disjoint, for an LP variable n_z_v corresponding to slack variable z, the value v can be used to generate tuples from any region corresponding to z. So, we iterate over the regions corresponding to z and generate as many tuples as possible with value v till either n_z_v exhausts or the size of the region exceeds. If n_z_v exhausts then we move to the next variable, else if the size of the region exhausts then we move on to the next region.

Chapter 7

Evaluation of 1-D Projection Algorithm

The experiments for 1-D Projections were done on 10 GB database of TPC-DS benchmark [2]. Workloads for two fact tables, catalog_sales and store_sales, were created by modifying the original workload such that it fits into assumptions of Hydra and our algorithm. Since our algorithm handles only 1-D projections, hence, queries having projections on multiple columns were split into different queries having single column projection each.

Table 7.1 shows the statistics of different workloads generated for tables catalog_sales and store_sales. The columns **# FCs** and **# PCs** are the number of filter constraints and projection constraints extracted from the corresponding workload, and the column **# A** is the number of attributes on which projection has been performed.

Name of workload	**# FCs**	**# PCs**	**# A**
CS_1	15	9	7
CS_2	20	13	9
CS_3	37	24	19

(a) catalog_sales

Name of workload	**# FCs**	**# PCs**	**# A**
SS_1	8	4	3
SS_2	18	11	5
SS_3	48	32	20

(b) store_sales

Table 7.1: Workload statistics

Table 7.2 shows the execution statistics of our algorithm on different set of workloads when sub-view optimization is not used. The different columns of the table are as follows: **Partitioning** is the time taken in seconds by the partitioning algorithm, **# S Vars** is the number of selection variables required, **# P Vars** is the number of projection variables required, **F & S** is the sum of time taken in seconds to formulate and solve the LP.

Name	Partitioning	# S Vars	# P Vars	F & S
CS_1	1	26	303296	405
CS_2	1	38	$> 2^{30}$	-
CS_3	*OOM*	-	-	-

(a) catalog_sales

Name	Partitioning	# S Vars	# P Vars	F & S
SS_1	1	36	274428	308
SS_2	1	432	$>2^{144}$	-
SS_3	84	10992	$>2^{576}$	-

(b) store_sales

Table 7.2: Without sub-view optimization

It can be seen that the number of required projection variables blow up very rapidly with increase in number of constraints. For cases where the number of projection variables were more than 2^{20}, it was impossible to create it's power set. *OOM* at **Partitioning** column for workload CS_3 of catalog_sales statistics is *OutOfMemory* error. Our experiments were done on a machine with 32 GB RAM. Since the complexity of the partitioning algorithm depends on the number of attributes in a sub-view, An *OutOfMemory* error motivates that the tables need to be divided into sub-views.

Name	Part.	# S Vars	# P Vars	F & S	P1 & P2
CS_1	1	48	59	1	1
CS_2	1	74	191	1	1
CS_3	1	568	2678	3	2009

(a) catalog_sales

Name	Part.	# S Vars	# P Vars	F & S	P1 & P2
SS_1	1	13	4	1	1
SS_2	1	24	33	1	1
SS_3	1	358	29939	10	1

(b) store_sales

Table 7.3: With sub-view optimization

Table 7.3 shows the execution time statistics on the workloads when sub-view optimization is used. The last column, **P1 & P2**, is the sum of time in seconds taken by Phase 1 and Phase 2. It can be seen that the number of variables required for solving projections has drastically reduced. But since we are taking power set of regions to create projection variables, the number of projection variables still increase very rapidly. Time required by P1 & P2 for workload corresponding to CS_3 is significantly more than workload corresponding to SS_3. This is because the variables required for the largest LP corresponding to Phase 2 of CS_3 was 1.7 million while for SS_3 it was 1.5 thousand.

Although using sub-view optimization with our heuristic algorithm do not guarantee a solution, we hasten to add that it works in most of the cases and reduces time and space requirement significantly over the counterpart.

Our algorithm for 1-D projections achieve complete volumetric similarity with no loss of quality, i.e., all the constraints are satisfied by generated synthetic database and the AQPs at vendor site are exactly the same as the ones obtained at client site.

Chapter 8

Conclusions and Future Work

We provided an end to end application software of Hydra. We then introduced Hydra++, an enhanced version of Hydra. By empirical evaluations over our adversarial workload and the JOB benchmark we showed that Hydra++ take less time to create consistency constraints than Hydra. We also gave an algorithm for handling 1-D Projection Constraints. We saw that the algorithm requires computing power set of a set of variables for creating the projection variables. Also, the heuristic algorithm which we used for handling 1-D projections alongside sub-view optimization handles most of the cases but not all.

Hence our future work includes reducing the number of projection variables by devising efficient techniques for creating projection variables, and creating an algorithm which can replace heuristic algorithm and guarantee solution in every case.

Bibliography

[1] Hydra Tool. http://dsl.cds.iisc.ac.in/projects/HYDRA. 1

[2] TPC-DS. http://www.tpc.org/tpcds/. 34

[3] Z3 Theorem Prover. https://github.com/Z3Prover/z3. 17

[4] A. Arasu, R. Kaushik, and J. Li. Data generation using declarative constraints. In *ACM SIGMOD*, 2011. 4

[5] C. Binnig, D. Kossmann, E. Lo and M. Tamer zsu. QAGen: generating query-aware test databases. In *ACM SIGMOD*, 2007. 2

[6] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? In *PVLDB*, 9(3), 2015. 15

[7] A. Sanghi, R. Sood, J. R. Haritsa, and S. Tirthapura. Scalable and dynamic regeneration of big data volumes. In *EDBT*, 2018. 1

[8] A. Sanghi, R. Sood, D. K. Singh, J. R. Haritsa, and S. Tirthapura. HYDRA: A Dynamic Big Data Regenerator. In *PVLDB*, 2018. (In press) 1