EFFICIENT SCORING AND RANKING OF EXPLANATION FOR DATA EXCHANGE ERRORS IN VAGABOND

BY

ZHEN WANG

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science
in the Graduate College of the
Illinois Institute of Technology

Approved _____
Advisor

Chicago, Illinois
May 2014

ACKNOWLEDGMENT

I would like to express my special appreciation and thanks to my advisor Professor Dr. Glavic, who have supported me throughout entire process. You have been a tremendous mentor for me. Your advices have been priceless.

I would also like to thank Professor Francis Leung for serving as my committee member even during his most busy schedule. I want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

At the end I would like express appreciation to my friends and families who supported me in studying at IIT, and incentivized me to strive towards my goal.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Symbol	Definition
ϵ	error
λ	error explanation
Λ	error explanation set
$\mathcal{T}$	Error Types
$\rightsquigarrow$	explain(s)

ABSTRACT

Data exchange has been widely used in big data era. One challenge for data exchange is to identify the true cause of data errors during the schema translation. The huge amount of data and schemas make it nearly impossible to find "the" correct solution. Vagabond system is developed to address this problem and use best-effort methods to rank data exchange error explanations base on the likelihood that they are the correct solutions. Ranking done on scoring functions that model some aspects of explanation sets. Examples of these properties include complexity(size of explanation), and side effect size(number of correct data values that will be affected by the changes).

The thesis introduced three new scoring functions to increase the applicability of Vagabond under various data exchange scenarios. We prove that the monotonicity property required by Vagabond may not hold for some of the new scoring functions, so a new generic ranker is also introduced to efficiently rank error explanations for these new scoring functions as well as for future scoring functions that have boundary property. We can efficiently compute upper or lower bounds on the score of partial solutions. We also completed some performance experiments on the new scoring functions and the new ranker. The experiment result proves that the new scoring functions introduced in this thesis have a scalable performance.

CHAPTER 1

INTRODUCTION

Data exchange has been widely used in big data era. It takes source schema and target schema as input and generate mappings between the schemas. These mappings are then used to generate an instance of the target schema based on an existing instance of the source schema. During this process, data errors may occur due to any reason such as source data error, mapping error etc. Given the large size of data exchange scenarios, it may not be possible to identify the correct explanation for these data exchange errors.

The Vagabond system [6] addresses this problem. It can evaluate error explanations using scoring functions which compute explanation scores according to two existing scoring functions, e.g. size of explanation scoring function, and side effect size scoring function.

In this thesis, we introduce new scoring functions including ***weighted combined scoring function***, ***type weighted size of explanation scoring function***, and ***error explanation homogeneity scoring function***. We prove that the monotonicity property required by Vagabond for ranking explanations may not hold for some of the new scoring functions, so a new generic ranker is also introduced to efficiently rank error explanations for these new scoring functions as well as for future scoring functions that exhibit boundary properties. The new boundary ranker can exploit the upper and lower bound of the new scoring functions as well as that of future scoring functions to prune the solution space early on.

The thesis consists following chapters, Chapter 2 will review and evaluate how existing research addresses how to efficiently explain data exchange errors. Chapter 3 introduces necessary background knowledge that covers data exchange, explanation

ranking, Vagabond, monotonicity properties, and information theory. The chapter concludes by representing the essential problem of developing new scoring functions and ranking algorithms that exploit bound properties of scoring functions.

Chapter 4 will describe the new scoring functions and prove respective disprove their monotoncity properties as well as bounds. Chapter 5 will focus on the new generic ranker that use bounds on partial solutions to prune the solution space. The new ranker performance will be experimentally evaluated in chapter 6. The last chapter will summarize the results and discuss future works.

CHAPTER 2

RELATED WORKS

As data exchange error explanation becomes more common, it's not surprising that a large body of researches have addressed the problem. Each of these researches have its own solution. Some important works are stated here, and they all have their advantages and disadvantages.

The paper by Ronald, Phokion and Miller introduced and proved some properties of data exchange in [4], which proves the need of efficiently debugging in data exchange mapping scenarios. A systematic debugging and ranking system Vagabond was introduced to solve the debugging problem. [6]

The Vagabond system uses best-effort methods to solve the error explanation ranking issue for data exchanges. It currently supports two scoring functions and several rankers including the A-star ranker and the skyline ranker [2]. However, there are some cases where these two functions may not work well. Detailed examples will be given in background chapter.

The survey done in [8] evaluated techniques can efficiently rank SQL query results, however, ranking error explanations in data exchange can still be complicated due to the amount of candidates.

The pruning technique introduced in [11, 9] shows how pruning can increase efficiency in machine learning systems. The basic idea of pruning is to priority select partial solutions that have a higher possibility to get better results. The criteria of selection is based on bound values. Let's say a lower score is a better solution, than a partial solution with lower low bound is a better candidate for complete solution.

CHAPTER 3

BACKGROUND

To better understand the research topic, readers are expected to have knowledge including but not limited to database system, data exchange, data exchange errors, information theory. Some key terminologies will be explained in this chapter and examples will be provided. Then we prove the importance and uniqueness of the thesis topic.

Consider an online ticket booking system that allows users check available tickets and prices. This scenario covers all necessary materials to understand this thesis.

First off, we need to store flight and price information into database. Two database tables, 3.1 and 3.2 store required info, and these tables are called database schema. The formal definition of database schema is the following:

Definition 1 (Database Schema)**.** *: A database schema* $S = \{R_1, \ldots, R_n\}$ *is a set of relation schemas. Each relation schema* $R = \{A_1, \ldots, A_n\}$ *refers to a finite nonempty set of attributes. [12, 4, 3]*

Consider a database schema "BOOKINGS" including the following source and target schemas:

Source Schemas:

PlaneInfo(fno; carrier; dept; arrv; model; maxload, time)

TicketPool(fno; price; remain)

UserInfo(uid; fname; lname:string)

Booking(oid; uid; price)

Target Schemas:

FlightInfo(fno; carrier; dept; arrv; price; time)

MemberInfo(mid; uid; points)

Now consider that we need to map the data from source schemas to target schemas. We use source to target tuple generating dependency (st-tgd) to express mappings in logical formula. Let P denotes PlaneInfo, T denotes TicketPool, F denotes FlightInfo, a denotes fno, b denotes carrier, c denotes dept, d denotes arrv, e denotes model, f denotes model, g denotes maxload, h denotes time, i denotes price, j denotes remain, the mapping is:

$$\forall P(a,b,c,d,e,f,g) \wedge T(a,i,j) \rightarrow F(a,b,c,d,i,g)$$

Example instances of S are shown in tables 3.1, 3.2, 3.3, 3.4, 3.5, 3.6.

Definition 2 (Database Instance)**.** *: A database instance* ***I*** *is a set of relation instances, each relation instance represents a table of data that comes from a subset of crossproduct of attributes. [12, 4]*

One set of instances for the schema BOOKINGS is introduced to help illustrate future definitions.

Definition 3 (Data Exchange)**.** *: Data exchange takes a source schema and a target schema as input, then the system generates possible mappings between source schema and target schema[5, 10].*

Data lost and errors may occur during the data exchange process due to either incorrectness in data, or in queries or mapping. This is especially true for large schemata, where the multi-step process of generating a schema mapping is error-prone.

Table 3.1. FlightInfo (Target)

Fno	Carrier	Dept	Arrv	Price	Time
3368	United	ORD	IAD	240	4:40
3368	United	ORD	IAD	440	11:40
1566	United	IAD	ORD	280	7:40
1566	United	IAD	ORD	380	16:40
1259	AA	ORD	IAD	390	9:10
3405	AA	IAD	ORD	230	14:20
9321	JetBlue	ORD	IAD	340	17:00

For example, tables 3.3 and 3.2 are transformed to table 3.1 by the following SQL query that implements the mapping shown above.:

```
SELECT fno, carrier, dept, arrv, price, time
FROM   TicketInfo as t, PlaneInfo as p
WHERE  t.fno = p.fno
```

This SQL query generates a target instance that contains data shown in table 3.1. We may notice that this table contains some data error marked with grey background color, these prices are inconsistent with the value in source instance. Now we introduce some terminologies that will be used later.

Definition 4 (Error Types)**.** *: During the data exchange processing, many reasons may result in various errors in the output data. Based on the cause of errors, e.g. source data error, incorrect SQL query, we define different error categories. Currently we have defined defined Source Copy Error, Source Join Error, Correspondence Error, Superfluous Mapping Error, Source Skeleton Error and Target Skeleton Error. The*

Table 3.2. PlaneInfo(Source)

Fno	Carrier	dept	arrv	Model	MaxLoad
3368	United	ORD	IAD	B747	240
1566	United	IAD	ORD	B747	240
1259	AA	ORD	IAD	B737	320
3405	AA	IAD	ORD	B737	320
9321	JetBlue	ORD	IAD	MD90	130

understanding of each error type is not required. Detailed error type definitions will be introduced for reading purpose.

Definition 5 (Data Exchange Scenario)**.** *A data exchange scenario is a tuple* $\mathbb{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{ST}, \Sigma_S, \Sigma_T, \mathcal{C}, I, J, \mathcal{T}, \textsc{Map})$ *where* Σ_S *and* Σ_T *are source and target constraints, and* Σ_{ST} *is a set of st-tgds.* $\mathcal{T}$ *are the transformations implementing the mapping and* Map *represents the relationships between scenario elements. We require that* $(I, J) \vDash \Sigma_{ST}$*,* $I \vDash \Sigma_S$*, and* $J \vDash \Sigma_T$*.*

Definition 6 (Errors and Error Sets)**.** *Given an instance* I*, an error is a triple* $R.t.A$ *where* R *is a relation from* I*,* t *is a tuple from* R*, and* A *is an attribute from the schema of* R*. We use* $AttrPos(I)$ *to denote the set of all potential errors for an instance* I*. An error* e *for a data exchange scenario* $\mathbb{M}$ *is an element of* $AttrPos(J)$*. An error set* E *for a data exchange scenario* $\mathbb{M}$ *is a set of errors.*

Definition 7 (Error Groundings)**.** *Let* $e = R.t.A$ *be an error for a data exchange scenario* $\mathbb{M}$*. We define the groundings* $\Theta(e)$ *for* e *as the set of grounded tgds in* (I, J) *that have a grounded atom corresponding to* $R.t$ *in their implication:*

$$\Theta(e) = \{\sigma[\theta] \mid (I, J) \vDash \sigma[\theta] \wedge R(t) \triangleleft \sigma[\theta]\}$$

Table 3.3. TicketPool(Source)

Fno	Price	MaxLoad
3368	240	240
1566	380	240
1259	390	320
3405	230	320
9321	340	130

Table 3.4. UserInfo(Source)

	Uid	Fname	Lname
u_1	1	Zhen	Wang
u_2	2	Santa	Uno
u_3	3	Rui	Yao

For an error set E, we use $\Theta(E)$ to denote the union of the sets of groundings for all errors in the set:

$$\Theta(E) = \bigcup_{e \in E} \Theta(e)$$

Definition 8 (Explanation)**.** *Given a grounding $\sigma[\theta]$ for an error $e = R.t.A$ and data exchange scenario $\mathbb{M}$, a explanation λ for $\sigma[\theta]$ according to e is a tuple $\lambda = (T, O)$ where T is the type of explanation (one of a fixed set of explanation types* Type *to be defined below) and O is a set of elements from the data exchange scenario $\mathbb{M}$. We write $\lambda \rightsquigarrow (\sigma[\theta], e)$ to denote that λ is an explanation for a grounded tgd $\sigma[\theta] \in \Theta(e)$ according to e.*

Table 3.5. Booking(Source)

	Oid	Uid	$Price
o_1	1	1	240
o_2	2	1	334
o_3	3	3	630
o_4	4	3	330
o_5	5	1	340
o_6	6	3	190
o_7	7	3	270

For an explanation λ we define the coverage $\textsc{Cover}(\lambda)$ as the subset of $AttrPos(J)$ for which there exists a grounding that is explained by the explanation. The side-effect $\mathrm{SE}(\lambda)$ of an explanation is the target attribute values it covers in addition to e.

Definition 9 (Coverage and Side-effects)**.** *Let λ be an explanation for an error e. The coverageand side-effects of λ are defined as:*

$$\textsc{Cover}(\lambda) = \{e' \mid \exists \sigma[\theta] \in \Theta(e) : \lambda \rightsquigarrow (\sigma[\theta], e')\}$$

$$\mathrm{SE}(\lambda) = \textsc{Cover}(\lambda) - \{e\}$$

Definition 10 (Covering Explanation Set (CES))**.** *A set Λ of explanations is called a covering explanation set (CES) for an error set E iff:*

$$\forall e \in E : \forall \sigma[\theta] \in \Theta(e) : \exists \lambda \in \Lambda : \lambda \rightsquigarrow (e, \sigma[\theta])$$

The coverage and side-effects of an explanation set are defined as the union of the

Table 3.6. MemberInfo(Target)

	Mid	Uid	Fname	Lname	points
m_1	1	1	Zhen	Wang	24
m_2	2	3	Rui	Yao	63
m_3	3	3	Rui	Yao	33
m_4	4	3	Rui	Yao	19
m_5	5	3	Rui	Yao	27

coverage respective side-effects of the explanations in the set:

$$\textsc{Cover}(\Lambda) = \bigcup_{\lambda \in \Lambda} \textsc{Cover}(\lambda)$$

$$\mathrm{SE}(\Lambda) = \bigcup_{\lambda \in \Lambda} \mathrm{SE}(\lambda) - E$$

$$\textsc{FCover}(\Lambda) = \{e \mid \forall \sigma[\theta] \in \Theta(e) : \exists \lambda \in \Lambda : \lambda \rightsquigarrow (e, \sigma[\theta])\}$$

The number of CES can be exponential in the size of error set. Intuitively, we want to check more important, or more possible errors to fix most data problems in the output schema. By quantifying scores of error explanations, we may rank these explanations and have a chance to present more likely explanations first. Thus we come with the following definitions.

Each of these error types has quantifiable properties, thus we may match the errors with a mathematical equation, which gives a positive score for each error source. These scores should reflect the likelihood of the error cause to be correct. As a convention in Vagabond, lower scores means higher importance.

By ranking explanations, we can upfront determine what would be the optimal error explanation candidates, then we may apply pruning technique to avoid pre-

generating the complete solution space, which costs massive computation resources.

Vagabond is built to address the above issues. The system uses several functions to compute a score for each error explanation set and rank these explanation sets according to scores. These functions are called scoring functions. The monotonicity property is required by Vagabond ranker so we can incrementally add explanations into partial explanation set by ensuring the score will not decrease. During ranking, we can prune the solution space to avoid pre-generating too many CES.

Definition 11 (Monotonicity Properties for Scoring Functions)**.** *A scoring function* $f : \Lambda \to \mathbb{N}^0$ *is a function that maps explanation sets to natural number scores with* $f(\varnothing) = 0$*. A scoring function is called weakly monotone iff for all explanation sets* Λ *and single explanation* λ *for a error set* E*:*

$$f(\Lambda) \leq f(\Lambda \cup \{\lambda\}) \leq f(\Lambda) + f(\{\lambda\})$$

A scoring function is called monotone iff it is weakly monotone and for all explanation sets Λ_a*,* Λ_b*,* Λ_c*, and* Λ_d*:*

$$f(\Lambda_a) \leq f(\Lambda_b) \wedge f(\Lambda_c) \leq f(\Lambda_d)$$
$$\Rightarrow f(\Lambda_a \cup \Lambda_c) \leq f(\Lambda_b \cup \Lambda_d)$$

A scoring function is called strongly monotone iff for all explanation sets Λ_a *and* Λ_b *with* $\Lambda_a \cap \Lambda_b = \varnothing$*:*

$$f(\Lambda_a \cup \Lambda_b) = f(\Lambda_a) + f(\Lambda_b)$$

Obviously, every strong montone scoring functions is also monotone and weakly monotone.

Definition 12 (Scoring Function)**.** *: A scoring function matches each explanation with an appropriate mathematical equation to compute the error explanation score.*

In the Vagabond system lower score means better solution, or preferred ranking. Consider an data exchange error scenario. In table 3.6, the last name of m_2 is marked as incorrect. If we use the size of explanation as scoring function, the score will be 1, because the only explanation for this error is the instance of source data error. We need to modify the last name in u_3 to fix the error in m_2. However, consider the size of side effect scoring function, because tuples affected by the data fix are m_3, m_4, and m_5, the size of side effect is 3.

As the above example shows, different scoring functions have different numerical scores for the same scenario and explanations. To improve ranking adaptiveness as well as efficiency, we need to explore the property of scoring functions so we can select better error explanations correctly and wisely.

CHAPTER 4
NEW SCORING FUNCTIONS

4.1 Weighted Combined Scoring Function

As the example of last chapter shows, data exchange errors may be explained by different causes and same error explanation can have different numerical scores. Thus for the same data exchange scenario, different scoring functions may have completely different ranking results. Additionally, based on the complexity and amount of data exchange scenarios, it is impossible to determine "the" correct explanation for any data error and scenarios. There is a demand of a scoring function that can cover different data exchange scenarios and also works well for different types of error causes.

It has been proven that multiple reasoning will increase the confidence level of decision making. [7, 13] We propose the weighted combined scoring function $f_{WCS[g,w]}$ to solve the above concerns. The rational behind $f_{WCS[g,w]}$ is the same as in multiple reasoning. As one simple scoring function models only one numerical property of explanation set, we can combine several scoring functions to model multiple properties of a explanation set at the same time.

Let g denotes a finite non-empty set of weakly monotonic scoring functions, w is the corresponding weight vector such that $0 < w \leq 1, \sum(w) = 1$.

The weighted combined scoring function $f_{WCS[g,w]}$ combines a finite number of weakly monotonic scoring functions and assign different weights w to these functions g. These weights can be either inputted by experienced data exchange users who can utilize their experience to make error explanation bias by giving larger weights to more likely explanation types, or from historical statistics that store possibility/explanation power of different scoring functions. This allows data exchange users to build different

instances of $f_{WCS[g,w]}$ from different weights of error explanation categories. Users may leverage their knowledge to create a better suitable function for their specific mapping scenario.

The example 4.1 shows an an instance of the weighted combined scoring function and how to compute the score. We will prove later that the new scoring function, as the linear combination of weakly monotonic scoring functions, is also weak monotonic. This property makes sure that we can apply existing ranking algorithms in Vagabond to $f_{WCS[g,w]}$.

Definition 13 (Weighted Combined Scoring Function $f_{WCS[g,w]}$)**.** *: Let g_1 until g_n denotes weakly monotonic scoring functions, $w_i, (w_i \in [0,1], \sum_{i=1}^{n} w_i = 1)$ denotes the weight of scoring function g_i.*

The Weighted Combined Scoring Function $f_{WCS[g,w]}$ is defined as:

$$f_{WCS[g,w]}(\Lambda) = \sum_{i=1}^{n} w_i * g_i(\Lambda) \tag{4.1}$$

Theorem 1 (Weak Monotonicity of $f_{WCS[g,w]}$)**.** *The linear combination of weakly monotonic scoring functions is also a weakly monotonic scoring function.*

Proof. Let $f_{WCS[g,w]} = \sum_{i=1}^{n} w_i * g_i(\Lambda)$ denote an instance of the weighted combined scoring function, We need to prove that

$$f_{WCS[g,w]}(\Lambda) \leq f_{WCS[g,w]}(\Lambda \wedge \{\lambda\}) \leq f_{WCS[g,w]}(\Lambda) + f_{WCS[g,w]}(\{\lambda\})$$

Given the definition of $f_{WCS[g,w]}$, we have:

$$f_{WCS[g,w]}(\Lambda) = w_1 * g_1(\Lambda)) + \cdots + w_k * g_k(\Lambda))$$

where all g_i functions are weakly monotone scoring functions. Since each g_i is weakly

monotonic, and weights are non-negative values, we have:

$$g_i(\Lambda) \leq g_i(\Lambda \wedge \{\Lambda\}) \leq g_i(\Lambda) + g_i(\{\lambda\})$$
$$\equiv w_i * g_i(\Lambda) \leq w_i * g_i(\Lambda \wedge \{\Lambda\}) \leq w_i * g_i(\Lambda) + w_i * g_i(\{\lambda\})$$

If we sum up all inequalities of g_i, by substituting the above equition with the f_{WCS} definition, since weights are non-negative. The above equation is equivalent to

$$f_{WCS[g,w]}(\Lambda) \leq f_{WCS[g,w]}(\Lambda \wedge \{\lambda\}) \leq f_{WCS[g,w]}(\Lambda) + f_{WCS[g,w]}(\{\lambda\})$$

□

Example - Computing f_{WCS}: Given the database instances in table 3.4, table 3.5 and table 3.6, the error is in tuple m_2, and it's caused by the source data error in u_3.

Let size of explanation scoring function $f_{SE}(\{m_2\})$ denotes the score from size of explanation scoring function. Since there is only one error explanation, the size of explanation is 1. Let side effect size scoring function $f_{SES(\{m_2\})}$ denotes the score from side effect size scoring function. Since there are three target attribute values will be affected if we modify the source data, the side effect size is 3.

Consider an instance of the weighted combined scoring function such that $g_1 = f_{SE}, g_2 = f_{SES}$, let w_1, w_2 denote the weight of these two scoring function components. If $w_1 = 0.33, w_2 = 0.67$, then the final score of this $f_{WCS[g,w]}$ instance is:

$$\begin{aligned} f_{WCS[g,w]} &= 0.67 f_{SES} + 0.33 f_{SE} \\ &= 0.67 * 3 + 0.33 * 1 \\ &= 2.34 \end{aligned}$$

Let $f[]$ denotes a set of weakly monotonic scoring functions, $w[]$ denotes the corresponding weights of $f[]$, $f_{WCS[g,w]}$ denotes the instance of the weighted combined

scoring function, $ExplColl$ denotes the explanation collection that needs to be ranked. s denotes the score of Λ given by the instance $f_{WCS[g,w]}$. The algorithm to compute the weighted combined score is described in algorithm 1.

Algorithm 1 Weighted Combined Scoring Function

1: **Initalization:**
2: $SizeofExplCollection \leftarrow ExplColl.size$
3: $f[] \leftarrow ScoringFunctions$
4: $w[] \leftarrow WeightsofScoringFunctions$
5: $s \leftarrow 0$
6: **Computation:**
7: **for** $i \in \{1, 2, \ldots, f[].size\}$ **do**
8: $\quad s \leftarrow s + w_j * g_j(ExplColl)$
9: **end for**
10: **Return** s

Bounds of f_{WCS}:

As we will see later in Chapter 5, the new ranker requires finite boundary property for scoring functions. Let f_{WCS} denotes an instance of weighted combined scoring function, w_i and g_i denotes component weights and scoring functions. Let $LOW_f(\Lambda)$ and $UP_f(\Lambda)$ denotes low bound and up bound of scoring function f given explanation set Λ, the bounds of $f_{WCS}[g, w]$ are defined as:

$$LOW_{f_{WCS}[g,w]}(\Lambda) = \sum_{i=1}^{n} w_i * LOW_{g_i}(\Lambda)$$

$$UP_{f_{WCS}[g,w]}(\Lambda) = \sum_{i=1}^{n} w_i * UP_{g_i}(\Lambda)$$

The trick here is to distribute the bound computation to component scoring

functions. Since the f_{WCS} is simply a linear combination of the scores, the distributed computation will keep the correct numerical results for f_{WCS} bounds.

Now we prove the up and low bounds of the scoring functions originally described in the Vagabond. Consider a data exchange scenario with candidate explanation collection Λ. Let E denotes the complete set of errors that Λ explains, $E_{partial}$ denotes the error set explained by partial solution $\Lambda_{partial}$, the bounds of f_{SE} are:

Up Bound: $|E| - |E_{partial}| + |\Lambda_{partial}|$

Low Bound: : $|\Lambda_{partial}| + 1$

Proof. The low bound is expecting the next additional explanation will explain all remaining unexplained errors, so it's simply the current size of partial explanation set plus 1.

The up bound expects every single unexplained error requires a unique explanation, thus the up bound is current size of explanation plus the size of remaining unexplained error set. □

For the same data exchange scenario, let $SE_{partial}$ denotes the current set of side effects for current partial solution, let $SE(\lambda)$ denotes the set of side effects for λ, let Λ_{remain} denotes the explanation set that explains uncovered errors in existing partial solution, let λ_i denotes every single explanation in Λ_{remain}, the bounds of f_{SES} are:

Up Bound: $|SE_{partial}| + sum(|SE(\lambda_i)|)$

Low Bound: : $max(|SE_{partial}|, max(min(|SE(\lambda_i)|)))$

Proof. The up bound is the worst case that all remaining explanations have side effects without overlap, that increases the size of side effect to its possible maximum value.

The low bound is the larger value between the current size of side effect, and the largest minimum size of side effect in remaining explanations. □

4.2 Type-Weighted Size of Explanation Scoring Function

As the example 3 shows, existing scoring functions such as side effect size scoring function and the size of explanation scoring function can only reflect naive numerical results. However, it's common that data exchange users want the score to reflect the category of error explanations.

We introduce the type-weighted explanation size scoring function f_{TWSE} to solves the above issue by calculating the score as type weighted explanation size.

Let TYPE denotes the domain of all explanation types, $w : \text{TYPE} \rightarrow [0,1]$ denotes a function that assigns a weight between 0 and 1 to each explanation type, E denotes the complete set of errors e_i, Λ_e denotes the set of all alternative explanations that explains e. Λ denotes the complete explanation sets. $\mathcal{T}(\lambda)$ denotes the type of explanation λ. The Type-Weighted Explanation Size Scoring Function f_{TWES} is defined as:

4.2.1 Definition.

Definition 14 (Type-Weighted Size of Explanation Scoring Function)**.**

$$f_{TWSE}[w](\Lambda) = \sum_{e \in E, \Lambda \rightsquigarrow E} \frac{\sum_{\lambda \in \Lambda_e} w * \mathcal{T}(\lambda)}{\mid \Lambda_e \mid} \tag{4.2}$$

Theorem 2 (TWSE Non-Monotonicity)**.** *The weak monotonic property does not hold for* f_{TWSE}*.*

Proof. New λ can potentially has overlap with every existing λ, bringing new or existing error types. If the weight of the new error type is lower than current average weight, then it will lower the total average, which means the weak monotonic property does not hold for this scoring function. □

Algorithm 2 Type-Weighted Explanation Size Scoring Function

Initalization:
$E \leftarrow \varnothing$
$M \leftarrow \varnothing$
$score \leftarrow 0$
for $e \in \{e' | \exists \lambda \in \Lambda : \lambda \rightsquigarrow e'\}$ **do**
    $E \leftarrow E \cup \{e\}$
end for
for $\Lambda_i \in \{\Lambda\}$ **do**
    for $\lambda_j \in \Lambda_i$ **do**
        for $e \in \{e' | \lambda_j \rightsquigarrow e'\}$ **do**
            $M(e) \leftarrow M(e) \cup \{\lambda_j\}$
        end for
    end for
end for
Computation:
for $e \in E$ **do**
    for $\lambda \in \Lambda_e$ **do**
        $score \leftarrow score + w * \mathcal{T}(\lambda)$
    end for
end for
Return *score*

Consider a data exchange scenario with candidate explanation collection Λ. Let $LOW_{current}$ and $UP_{current}$ denotes the low bound and up bound that need to be updated. E denotes the complete set of errors that Λ explains, $E_{partial}$ denotes the error set explained by partial solution $\Lambda_{partial}$, $W : w_1, w_2, \ldots w_n$ denotes the weights

for corresponding error explanation types. the bounds of f_{SE} are:

Up Bound: $UP_{current} + max(w_i) * (|E| - |E_{partial}|)$

Low Bound: : $LOW_{current} + min(w_i)$

Proof. The low bound is expecting the next additional explanation will explain all remaining unexplained errors, so it's simply the current size of partial explanation set plus 1 * the minimum weight of explanation type.

The up bound expects every single unexplained error requires a unique explanation, thus the up bound is current up bound plus the size of remaining unexplained error set times the maximum weight of error types. □

4.3 Error Type Homogeneity Scoring Function

4.3.1 Introduction. Consider a common data exchange scenario that one target schemar attribute contains several errors. Intuitively, each e in target schema comes from identical transformation as well as identical source, thus the error type, if any, should be identical and here is where we introduce information theory to evaluate the error source homogeneity for single error.

Entropy function is characterized from Shannon's entropy[14]. It quantifies the homogeneity of system parameters, which is the error homogeneity in Vagabond. We introduce a new scoring function that will quantify the homogeneous property of ideal error explanation set. The new scoring function, error type homogeneity scoring function f_{EH}, uses entropy function to compute the homogeneous level of error explanation candidates.

Let $\rho_{(R,A)}(\mathcal{T})$ denotes the density of error type $\mathcal{T}$ among all the error explanations that explain attribute (R, A), the density for error type $\mathcal{T}$ that explains (R, A) is computed as:

$$\rho(\mathcal{T}) = \frac{|\lambda|\lambda \in \Lambda_{(R,A)} \wedge \mathcal{T}(\lambda) = \mathcal{T}|}{|\Lambda_{(R,A)}|}$$

such that $\Lambda_{(R,A)} \rightsquigarrow E_{(R,A)}$.

Given the fact that there could be several types of explanation all explain $E_{(R,A)}$, and the definition of entropy formula, the error source homogeneity score for attribute R.A, $f_{EH}(\Lambda_{(R,A)})$ is computed as:

$$f_{EH}(\Lambda_{(R,A)}) = -\sum_{\mathcal{T} \in \text{TYPE}} \rho(\mathcal{T}) * \log(\rho(\mathcal{T}))$$

Since a explanation set may contain several unrelated attributes, we sum up the above equation for all attributes in the error set as a final score for f_{EH} scoring function.

Definition 15 (Error Type Homogeneity Scoring Function f_{EH})**.**

$$f_{EH}(\Lambda) = \sum_{(R,A)\epsilon TA} \frac{|\Lambda_{(R,A)}|}{|\Lambda|} * H(\Lambda_{(R,A)})$$

To be consistent with all other scoring functions, the lower score indicates better solution.

Theorem 3 (f_{EH} Non-Monotonicity - Lemma)**.** *The best case of homogeneity score is 0, while only one type of error dominate the error set.*

Depending on the size of error type domain, if all error types are evenly distributed, the worst case of homogeneity score is infinite.

Theorem 4 (f_{EH} Non-Monotonicity)**.** *The explanation homogeneity scoring function is not weakly monotonic. Scores may either increase or decrease while explanation set size increases.*

Proof. **Counter Example**

Following tables represents the error type distribution for an abstract data exchange scenario. Each table shows the error type amount for that attribute. For example, for attribute A, it there are 98+2 = 100 errors in this attribute, and 98 out of these 100 errors are explained by T_2 type of error explanation, or can be categorized into T_2.

A	
2	98
T_1	T_2

B	
1	99
T_1	T_2

C	
2	98
T_1	T_2

D	
1	99
T_3	T_4

Figure 4.1. Counter-example of Monotonicity for f_{EH}

Each table represents a $E_(R', A')$. For above four $E_(R', A')$, we know that

$$f(a) \leq f(b), f(c) \leq f(d)$$

If the entropy scoring function has the weakly monotonic property, we should have:

$$f(a \cup c) \leq f(b \cup d)$$

However, according to our scoring function equation, clearly the above conclusion is incorrect. So the weakly monotonic property does not hold for the error explanation homogeneity scoring function. □

4.3.2 Computation Logic. We now present the scoring function algorithm. The f_{EH} scoring function has four stages, mapping, partitioning, evaluating, as well as merging.

Mapping

First stage is partitioning the target schema based on e, and create a mapping relation from single target marker to a set of error explanations.

Let E denotes all target errors, e.g.

$$E = \{e_1, e_2, \ldots, e_n\}, e_i = (R, t, a)$$

where R is relation, t is tuple, a is attribute.

An explanation set for all target errors is

$$\Lambda = \{\lambda_m | \forall \lambda_m, \exists e \in E : \lambda_m \rightsquigarrow e\}$$

Group errors by target attributes:

$$TA = \{(R, A) \mid \exists t : (R, t, A) \in E\}$$

Stage 1 mapping for errors and explanations is as follow:

$$P_{TA-\Lambda} : TA \to \mathcal{P}(\Lambda)$$

which is defined as

$$P_{TA-E} : TA \to \mathcal{P}(E)$$

Partitioning Second stage is merge the error explanations into partial explanation sets according to e by their combination of relation and attribute. Stage 2 will merge the partions in stage 1, formulating several partial explanation sets.

Let $E_{(R',A')}$ denotes the partial target error set having unique combination of relation and attribute. e.g.

$$P_{TA-E}((R', A')) = \{e \mid e \in E \wedge e.R = R' \wedge e.A = A'\}$$

We will create several partial explanation sets by the following rule:

$$P_{TA-\Lambda}((R', A')) = \{\lambda \mid \lambda \in \Lambda \wedge \exists e \in E_{(R',A')} : \lambda \rightsquigarrow e\}$$

For convenience we write $E_{(R,A)}$ for $P_{TA-E}((R, A))$

Definition 16 (Error Explanation Partition)**.** *: Let Λ denotes the explanation set of an explanation generator, $\lambda_i, i = 1, \ldots, n$ are explanations in Λ, The following constraint holds:* $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$

1. $E(\lambda_i) \nsubseteq E(\lambda_j), i \neq j$
2. $\forall \Lambda' \subseteq \Lambda, \forall \lambda \in \Lambda - \Lambda', E(\lambda) \nsubseteq \Lambda'$

Evaluating Third stage is evaluate the homogeneity of each partial explanation set by computing the entropy score of the error explanation set. Stage 3 we compute entropy score for each partial explanation set.

$$H(\Lambda_{(R,A)}) = -\sum_{\mathcal{T} \epsilon \text{TYPE}} \rho(\mathcal{T}) * \log(\rho(\mathcal{T}))$$
$$\rho(\mathcal{T}) = \frac{|\{\lambda | \lambda \in \Lambda_{(R,A)} \wedge \mathcal{T}(\lambda) = \mathcal{T}\}|}{|\Lambda_{(R,A)}|}$$

Merging The last stage is merge partial entropy scores by weighting each entropy score by their size of coverset percentage over the complete size of target error set. The last stage is compute weighted combined entropy scores as follow:

$$f_{EH}(\Lambda) = \sum_{(R,A) \epsilon TA} \frac{|\Lambda_{(R,A)}|}{|\Lambda|} * H(\Lambda_{(R,A)})$$

Algorithm 3 Error Explanation Homogeneity Scoring Function

Initalization:
$E \leftarrow \varnothing$
$M \leftarrow \varnothing$
$score \leftarrow 0$
for $e \in \{e' | \exists \lambda \in \Lambda : \lambda \rightsquigarrow e'\}$ **do**
    $E \leftarrow E \cup \{e\}$
end for
for $\Lambda_i \in \{\Lambda\}$ **do**
    for $\lambda_j \in \Lambda_i$ **do**
        for $e \in \{e' | \lambda_j \rightsquigarrow e'\}$ **do**
            $M(e) \leftarrow M(e) \cup \{\lambda_j\}$
        end for
    end for
end for
Computation:
for $e \in E$ **do**
    $\Lambda_e \leftarrow E(e)$
    for $\lambda \in \Lambda_e$ **do**
        $score \leftarrow score + w * \mathcal{T}(\lambda)$
    end for
end for
Return *score*

Given the fact that f_{EH} is not weakly monotonic, we still want to find boundary property for f_{EH} so we could apply pruning in partial solution base, thus allow us to increase efficiency of ranking algorithm.

Let $\mathcal{T}$ denotes the domain of all error types, $\mathcal{T}_{current}$ denotes the domain of partial explanation error types, let $|T_i|_\Lambda$ denotes the amount of explanations that belongs to type T_i in Λ, the low bound and up bound computation of f_{EH} are described as:

Algorithm 4 Low Bound Update For f_{EH}:

Initialization:
for $attr$ *in* E **do**
    if $|attr| > |attr_{update}|$ **then**
        $attr_{update} \leftarrow attr$
    end if
end for
for $\mathcal{T}$ *in* $\{\mathcal{T}\}$ **do**
    if $|\mathcal{T}| > |\mathcal{T}_{update}|$ **then**
        $\mathcal{T}_{update} \leftarrow \mathcal{T}$
    end if
end for
Update Low Bound:
$\rho_{new}(\mathcal{T}_{update}) \leftarrow (|\mathcal{T}_{update}| + 1)/(|attr_{update}| + 1)$
$H(\Lambda_{(R,A)\cup\{\lambda\}}) \leftarrow H(\Lambda_{(R,A)}) + \rho(\mathcal{T}) * \log(\rho(\mathcal{T})) - \rho_{new}(\mathcal{T}_{update}) * \log(\rho_{new}(\mathcal{T}_{update}))$
$tmp \leftarrow lowBound * |\Lambda| - H(\Lambda_{(R,A)}) * |\lambda_{(R,A)}|$
$tmp \leftarrow tmp + H(\Lambda_{(R,A)\cup\{\lambda\}}) * (|\lambda_{(R,A)}| + 1)$
$tmp \leftarrow tmp/(|\Lambda| + 1)$
$lowBound \leftarrow tmp$
Return $lowBound$

Algorithm 5 Up Bound Update For f_{EH}:

Initialization:
for $attr$ *in* E **do**
  if $|attr| < |attr_{update}|$ **then**
    $attr_{update} \leftarrow attr$
  end if
end for
for $\mathcal{T}$ *in* $\{\mathcal{T}\}$ **do**
  if $|\mathcal{T}| < |\mathcal{T}_{update}|$ **then**
    $\mathcal{T}_{update} \leftarrow \mathcal{T}$
  end if
end for
Update Up Bound:
$\rho_{new}(\mathcal{T}_{update}) \leftarrow (|\mathcal{T}_{update}| + 1)/(|attr_{update}| + 1)$
$H(\Lambda_{(R,A)\cup\{\lambda\}}) \leftarrow H(\Lambda_{(R,A)}) + \rho(\mathcal{T}) * \log(\rho(\mathcal{T})) - \rho_{new}(\mathcal{T}_{update}) * \log(\rho_{new}(\mathcal{T}_{update}))$
$tmp \leftarrow upBound * |\Lambda| - H(\Lambda_{(R,A)}) * |\lambda_{(R,A)}|$
$tmp \leftarrow tmp + H(\Lambda_{(R,A)\cup\{\lambda\}}) * (|\lambda_{(R,A)}| + 1)$
$tmp \leftarrow tmp/(|\Lambda| + 1)$
$upBound \leftarrow tmp$
Return $upBound$

CHAPTER 5

BOUNDARY RANKER

5.1 Introduction

The existing rankers in Vagabond require monotonicity property. However, this property may not hold for new scoring functions. To match the requirement of ranking partial explanation solutions, we need to introduce a new generic ranker that prunes partial solutions based on score bounds.

To keep consistency of scoring functions, this boundary ranker error explanations set with lower score to be better solutions.

5.2 Boundaries of Scoring Functions

Definition 17 (Finite Boundary)**.** *For any given scoring function f and error explanation collection Λ, let q denote the score for any partial explanation at it's incremental step i.*

The up bound $UP(f(\Lambda))$ is a value α such that $\forall q, \alpha \geq q$.

The low bound $LOW(f(\Lambda))$ is a value β such that $\forall q, \beta \leq q$.

5.3 Implementation of Boundary Ranker

Data Strucuture of Boundary Ranker

Definition 18 (IDMap Array)**.** *IDMap Array is an array with size equal to the size of error set. The index of the array i indicates corresponding marker e_i. Each element in the array is an IDMap that represents the mapping relationship between e_i and the Λ_i, where Λ_i is the set contains all λ_j such that $\lambda_j \rightsquigarrow e_i$.*

Definition 19 (Partial Explanation Queue)**.** *A partial explanation set queue $Queue_{partial}$ is a priority queue that ranks elements $ele_{PartialQ}$ by their low and up boundaries.*

Each element consists a vector represents the indexes of λ of this partial solution, and values of up boundary, low boundary, paritial score, explanation set, as well as expansion step.

Definition 20 (Complete Explanation Queue)**.** *A complete explanation set queue $Queue_{complete}$ is a priority queue that ranks elements $ele_{CompleteQ}$ by their score.*

Each element consists a vector represents the indexes of λ of this solution, and the Λ such that $\Lambda \rightsquigarrow \{e\}$.

Let $E.size$ denotes the size of error set, $ScoringFunction$ denotes the scoring function that uses the boundary ranker, $ExplanationCollection$ denotes the explanation collection that we need to rank, the algorithm of boundary ranker is described in algorithm 6.

Algorithm 6 Boundary Ranker

Initalization:
$SizeofMarkerSet \leftarrow E.size$
$IDMap[] \leftarrow IDMap < \epsilon, \Lambda > [SizeofErrorSet]$
$f \leftarrow ScoringFunction$
for $i \in \{1, 2, \ldots, SizeofExplanationCollection\}$ **do**
    for $j \in \{1, 2, \ldots, SizeofExplanationSet\}$ **do**
        $CurrentExpl \leftarrow \lambda_{i,j}$
        $IDMap[k] \leftarrow IDMap[k] \cup \{\lambda_{i,j}\}, (\lambda_{i,j} \rightsquigarrow \epsilon_k)$
    end for
end for
$\Lambda_0 = ID_Map[0]$
for $i \leftarrow 1$, $\Lambda_0.size$ **do**
    $\lambda_i = \Lambda_0[i], e \leftarrow$ **new** $PartialEle, e.\Lambda \leftarrow \{\lambda_i\}$
    $e.upBound \leftarrow UP(f(\Lambda_0))$
    $e.lowBound \leftarrow LOW(f(\Lambda_0))$
    $e.score \leftarrow f(\Lambda_0)$
    Insert(PartialQueue, e)
end for

19: **Generate Ranked Explanation Sets:**
20: **while** $CompleteQueue.size < RequiredRankingSize$ **do**
21: $PartialEle \leftarrow PartialQueue.PopTopElement$
22: **if** $ExpandedPartialEle.size == SizeofMarkerSet$ **then**
23: Insert(CompleteQueue, ExpandedPartialEle)
24: **end if**
25: **for** $i \leftarrow 1$, $SizeofNextExplset$ **do**
26: $ExpandedPartialEle.partialExplSet \leftarrow PartialEle.partialExplSet \cup \{\lambda_i\}$
27: $ExpandedPartialEle.upBound \leftarrow UP(f(PartialEle.partialExplSet))$
28: $ExpandedPartialEle.lowBound \leftarrow UP(f(PartialEle.partialExplSet))$
29: $ExpandedPartialEle.score \leftarrow f(PartialEle.partialExplSet)$
30: Insert(PartialQueue, ExpandedPartialEle)
31: **end for**
32: **end while**

5.4 Example

Consider an abstract data exchange scenario that uses size of explanation as scoring function. The IDMap after initialization is:

$$IDMap[0] = 0 : \lambda[1,1], 1 : \lambda[1,2]$$

$$IDMap[1] = 0 : \lambda[2,1], 1 : \lambda[2,2]$$

$$IDMap[2] = 0 : \lambda[1,2]$$

Recall that the length of this IDMap represents the size of error set, so we have an error set with size equals to 3. Each IDMap element indicates possible error explanation for the error having the element index. (e.g. for e_0, there are two λ can explain this error, and these two explanations are $\lambda_{1,1}$ and $\lambda_{1,2}$.)

The partial queue is initialized as empty, and we start inserting elements to the queue by expanding the partial error set, which is in an arbitrary order of error index. Here we start expanding with e_0.

We generate several partial solutions for e_0 and insert these partial solutions as element into partial queue. The queue will sort elements base on bound values and scores. Assume that after first expanding, the queue status is as follow:

$$PartialQueue[0] =< ExplIndexVe1 : [1, -1, -2], 2, 1, 1, \{\lambda_{1,2}\}, 1 >$$

$$PartialQueue[1] =< ExplIndexVe0 : [0, -1, -1], 3, 1, 1, \{\lambda_{1,1}\}, 1 >$$

Notice that the top element includes $\lambda_{1,2}$ because the explanation also explains e_2, this results in a lower up bound for the solution candidates, thus it's a preferred partial solution.

After the first expanding, we pop out the top element in the partial queue, examine it to check its completeness, now it's not complete, so we continue to expand it, the partial queue after inserting new expanded nodes is:

$$PartialQueue[0] =< ExplIndexVe1 : [1, 0, -2], 2, 2, 2, \{\lambda_{1,2}, \lambda_{2,1}\}, 2 >$$

$$PartialQueue[1] =< ExplIndexVe1 : [1, 1, -2], 2, 2, 2, \{\lambda_{1,2}, \lambda_{2,2}\}, 2 >$$

$$PartialQueue[2] =< ExplIndexVe0 : [0, -1, -1], 3, 1, 1, \{\lambda_{1,1}\}, 1 >$$

Now we can see we have two complete nodes in the partial queue, these two elements will be pop out and transfer to complete queue, which is also the ranked explanation set as final output. The partial queue after this operation is:

$$PartialQueue[0] =< ExplIndexVe0 : [0, -1, -1], 3, 1, 1, \{\lambda_{1,1}\}, 1 >$$

The algorithm then continue to run until the partial queue is empty or the required amount of ranked solutions are filled, whichever comes earlier.

CHAPTER 6

EXPERIMENTS

Experiments are conducted on a Ubuntu 14.04 LTS machine with 8GB RAM, Intel Core i7-4770k @ 3.50GHz x 4. The experiments are designed into two categories, one focusing on the impact of error ratio with fix amount of data, the other focusing on the processing time against data size increase.

All data are generated using STBenchmark [1], and error sets are generated by uniformly select target instance attributes.

Table 6.1 shows the first set of experiments, the fixed size of data is 1000. The experiment result shows that all scoring functions are nearly linear increasing proportional to the error ratio. However, the explanation homogeneity scoring function is most sensitive to the error ratio, while size of explanation scoring function is least sensitive to the error ratio.

Table 6.1. Explanation Generation Time(Sec), Data Size = 1000, Default Ranker

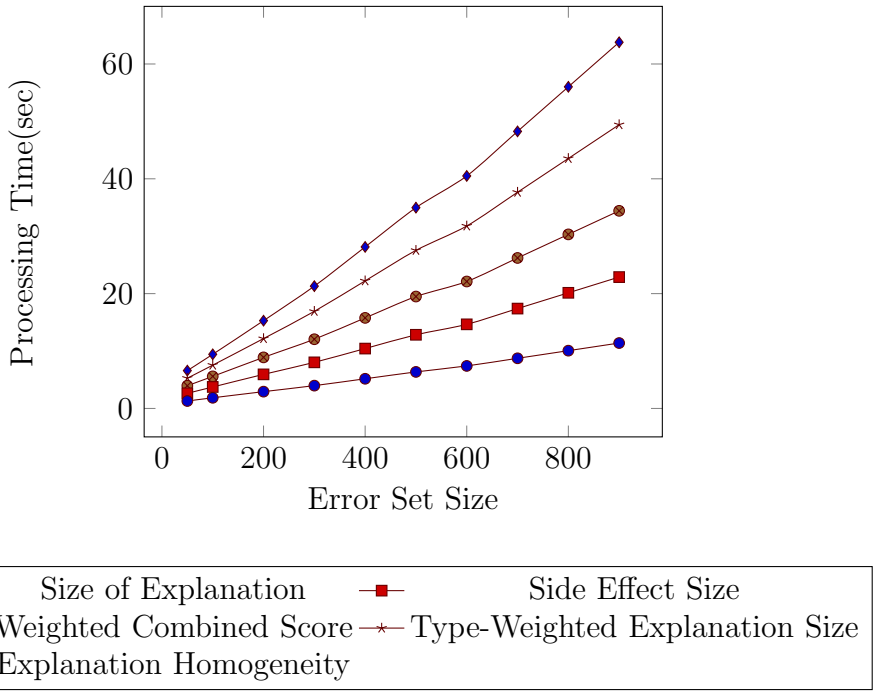
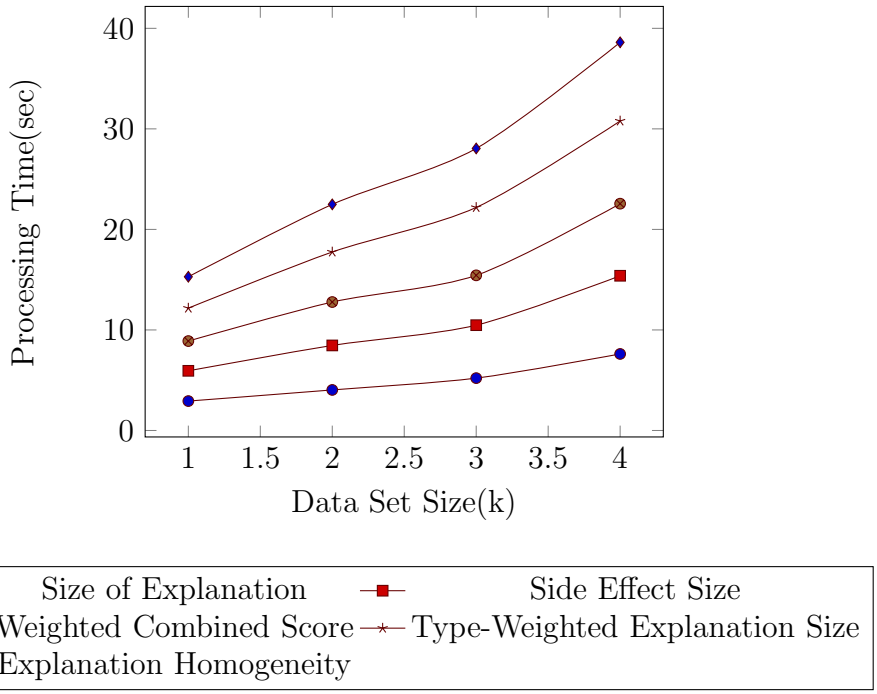


Table 6.2 shows the second set of experiments, the fixed size of error set is 2000. The experiment result shows that all scoring functions are nearly linear increasing proportional to the data size. However, the explanation homogeneity scoring function is most sensitive to the data size, while size of explanation scoring function is least sensitive to the data size.

Table 6.2. Explanation Generation Time(Sec), Error Size = 200, Default Ranker



CHAPTER 7

CONCLUSIONS

As experiment 6.1 and experiment 6.2 show, given fix amount of data, the error explanation generation time is linear increasing with the error ratio. Also, given fix amount of error, the error explanation generation time of scoring functions is linear increasing with the size of data.

For both cases, the explanation homogeneity scoring function is most sensitive function, while size of explanation scoring function is least sensitive one.

The linear increasing time corresponding to the linear increasing size serves as a preliminary proof that the new scoring functions introduced in this thesis have a scalable performance. Additionally, given the limited experiment platform resources, further large scale experiments are necessary to reveal the the impact of boundary ranker on the solution base pruning.

BIBLIOGRAPHY

[1] Bogdan Alexe, Wang-Chiew Tan, and Yannis Velegrakis. Stbenchmark: Towards a benchmark for mapping systems. 2008.

[2] S Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 421–430. IEEE, 2001.

[3] Rosine Cicchetti, A Hameurlain, and R Traunmuller. Database and expert systems applications, 2002.

[4] Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Database TheoryICDT 2003*, pages 207–224. Springer, 2003.

[5] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[6] Boris Glavic, Jiang Du, Renée J Miller, Gustavo Alonso, and Laura M Haas. Debugging data exchange with vagabond. *Proceedings of the VLDB Endowment*, 4(12), 2011.

[7] Van-Nam Huynh, Yoshiteru Nakamori, Tu-Bao Ho, and Tetsuya Murai. Multiple-attribute decision making under uncertainty: the evidential reasoning approach revisited. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 36(4):804–822, 2006.

[8] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, October 2008.

[9] Michael J Kearns and Yishay Mansour. A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. In *ICML*, volume 98, pages 269–277. Citeseer, 1998.

[10] Phokion G Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 61–75. ACM, 2005.

[11] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.

[12] Henryk Rybiński. On first-order-logic databases. *ACM Trans. Database Syst.*, 12(3):325–349, September 1987.

[13] Glenn Shafer. *A mathematical theory of evidence*, volume 1. Princeton university press Princeton, 1976.

[14] Bhu Dev Sharma and RP Singh. A generalization of entropy equation: homogeneous entropies. *Kybernetika*, 21(2):157–163, 1985.