Department of Informatics, University of Zürich

MSc Project Technical Report

Structuring and Processing Temporal Probabilistic Data in Migros

Oliver Leumann

Email: oliver.leumann@uzh.ch

July 16, 2015

supervised by Prof. Dr. M. Böhlen and A. Papaioannou

University of Zurich^{UZH}

Department of Informatics

Contents

1 Introduction

Migros is one of the biggest companies in Switzerland and the shopping habits of its customers is collected through loyalty cards. Migros uses the Teradata[1] database system to maintain and process the huge amount of data that is collected about the purchases of their customers. In this project we investigate and apply some temporal probabilistic concepts in the Migros Teradata database landscape. As part of that, we apply the parsimonious temporal aggregation, termed PTA [1], on some huge real life data sets. Since the project has been restricted to an almost pure SQL environment, we had to find some alternative algorithmic concepts for the PTA in order to successfully process these large data sets.

The paper is organized as follows. In Chapter 2 we investigate the relations in Migros' Teradata database system that are needed for the rest of the paper. In Chapter 3 we describe some approaches on how to calculate probabilities for an important relation in the database to get a time point timestamped probabilistic relation. In Chapter 4 we then show in which way the PTA is used to transform the time point stamped relation to a temporal probabilistic relation with interval timestamps. We introduce a new algorithmic approach which mirrors some PTA aspects in such a way that it can be executed in an almost pure SQL environment and afterwards, we present experimental results regarding its performance. Chapter 5 presents a reference query to outline a possible application of temporal probabilistic relations in the Migros domain. Finally, in Chapter 6 we draw some conclusions.

[1] http://www.teradata.com

2 Migros Dataset

For the scope of this project we use two very central relations of Migros' database. Both relations contain data that is collected when customers buy articles at a location of Migros. The data comprises general information about purchases as well as detailed insight of every article that is bought in the course of a purchase.

2.1 First Source Relation: Transaction

The first relation is called the **Transaction** relation and each of its tuples stores summarized information about a customer's purchase in a Migros branch. All the information stored applies to the whole purchase but without going into the details of the articles that are part of the purchase. Figure 2.1 shows the scheme of the **Transaction** relation with some assorted attributes and a sample tuple.

Transaction

	LocationID	*DateID*	*TimeID*	*CustomerID*	*SalesTotal*	*DiscountTotal*	...
tr_1	8502	2014-03-05	160135	312379620	16.70	1.30	...

Figure 2.1: The **Transaction** Relation

The example tuple tr_1 states that at the location with $\mathit{LocationID} = 8502$, on the 5th March in 2014 at 16:05:35, the customer with the $\mathit{CustomerID} = 312379620$ has made a purchase. The customer spent a total of $\mathit{SalesTotal} = 16.70$ CHF and saved $\mathit{DiscountTotal} = 1.30$ CHF due to special offers. As we will see in the following Chapter 3, we need to know the atttributes $\mathit{LocationID}$, DateID, TimeID and $\mathit{CustomerID}$. Important to note is that some of the attributes are displayed in a simplified manner since they're composed of multiple attributes of the implemented relation in the real life. But with the abstraction it is easier to understand and talk about the relation's attributes. E.g. $\mathit{LocationID}$ is actually a compound attribute, but for our purpose, it is enough to know that this attribute uniquely identifies at which branch the purchase has happened. It is quite evident that DateID is storing the date in the format 'yyyy-mm-dd' and that the TimeID attribute is an integer of the form 'hhmmss'. The $\mathit{CustomerID}$, e.g. 312379620, actually represents the number of a loyalty card. It is important to know that the number is not unique to a certain person. One household can request for multiple loyalty cards having all the same number. For example the mother of a family has a loyalty card with a specific number and her child can have a second card with the very same number. Therefore we can have entries in the **Transaction** relation with the same $\mathit{CustomerID}$ on very similar time-points, but very distinct locations.

2.2 Second Source Relation: TransactionArticle

The second relation is called **TransactionArticle** and as the name is indicating, a tuple of this relation stores detailed information regarding an article that is bought in a purchase. Figure 2.2 shows, again in a simplified manner, an example tuple. Besides the attributes *LocationID*, *DateID*, *TimeID* and *CustomerID* that we already know from the $Transaction$ relation, the most popular attribute is the *ArticleID* that uniquely identifies which article has been bought. The example tuple tra_1 states that at the location with $LocationID = 8502$, on the 5th March in 2014 at 16:05:35 in the afternoon, one article, $Quantity = 1$, with $ArticleID = 29857103$ has been bought by the customer with the loyalty card number $CustomerID = 312379620$. The prices for that article has been $Sales = 1.45$ CHF and the article price has been reduced by $Discount = 0.45$ CHF.

TransactionArticle

	LocationID	*DateID*	*TimeID*	*ArticleID*	*CustomerID*	*Sales*	*Discount*	*Quantity*	...
tra_1	8502	2014-03-05	160135	29857103	312379620	1.45	0.45	1	...

Figure 2.2: The $Transaction$ Relation

2.3 Helper-Relations

Due to some data normalization measures, most attributes of the two source relations are quite expressionless id values, e.g. *LocationID* or *ArticleID*. For every kind of ID there exists a corresponding relation that stores names, translations and other attributes to the specific ID values. With the help of joins on the corresponding ID attributes we are able to visualize the data with more meaningful content. For example as shown in Figure 2.3, we can give *LocationID*s some real location names with the **Location** relation and with the *Article* relation we are able to replace *ArticleID*s with meaningful article names. In the interest of keeping things simple for the examples in the paper, we will directly use more meaningful content.

Location

	LocationID	*LocationName*	..
l_1	8502	MMM Waedenswil	...

Article

	ArticleID	*ArticleName*	..
a_1	160135	Premium Coke	...

Figure 2.3: The *Location* and *Article* Relation

The **Calendar** relation helps us to implicitly define some properties out of specific dates. Figure 2.4 shows some of its attribute. *DateID* is the actual identifying date and all other attributes refer in some extend to that date. *WeekdayID* represents what weekday, e.g. 3 = *Wednesday*, we have on a particular date and *YearWeekID* provides us the number of the week of the current year that the date is placed, e.g. *201409* represents that the date is in the 9th week of 2014. In Chapter 3 we will make actual use of the method to derive further attributes from a date.

Calendar

	DateID	*WeekdayID*	*YearWeekID*	*YearMonthID*	*YearID*	...
c_1	2014-03-05	3	201409	201403	2014	...

Figure 2.4: The **Calendar** Relation

2.4 Working Subset

There exist billions of rows in the two source relations $Transaction$ and $Transaction$-$Article$. Each day more than 1.6 million shoppings are recorded and more than 16 million articles are purchased in the Migros domain. In order to keep the focus of the project into a certain direction, we reduce the scope of the domain to a certain type of Migros Stores. Migros has thousands of different cost units that represent different markets and even services. Of course on the one hand we have the Migros supermarkets but additionally on the other hand there exist a lot of specific specialist warehouses like Micasa, Obi, SportXX, Valora etc. as well and furthermore Migros has also business lines like catering services and Migros restaurants. For this project we will focus the scope on the basic Migros markets of the detail bussines division which are categorized to the three different size types 'M', 'MM' and 'MMM' that most of the Swiss know. An additional reduction of the scope is that we only focus on collected data that have a $CustomerID$ associated with it. That means that we only consider data that is created of purchases where a customer actually scanned their loyalty card at the cash box.

3 Computing Probabilities per Time Point

In this chapter we will show some possibilities of how to transform a time point stamped relation into a temporal probabilistic relation. From the two main source relations **Transaction** and **TransactionArticle** we will use the **Transaction** relation to derive the temporal probabilistic relation **ShopsIn**. The main focus lies on finding appropriate probability values for the tuples of this targeted relation but we will also show some concerns that came up during the investigation.

Figure 3.1a shows the **Transaction** relation with a sample tuple tr_1. The tuple tr_1 states that Jane Doe has been shopping in the Migros market ZH MM Meilen on the *5th March in 2014* at *16:01:35* in the afternoon. As indicated, we will derive the **ShopsIn** relation from the **Transaction** relation. So first of all, we define the scheme of the target relation **ShopsIn**. **ShopsIn** is a time point timestamped probabilistic relation which will store events in the form of who (Cst) goes shopping at which location (Loc) in which point in time (T) and what the probability (P) has been for this event anyway. Figure 3.1b shows a sample tuple of how the data should look like for this relation. The tuple s_1 is representing the fact that Jane Doe has been shopping at ZH MM Meilen on the *5th March in 2014* and the probability for this event has been *0.33*. Comparing it to the **Transaction** relation from Figure 3.1a we see that we can directly adopt the $CustomerID$ and $LocationID$ attributes from the **Transaction** relation for the Cst and Loc attributes of the **ShopsIn** relation. The Timestamp T is equal to the $DateID$ attribute, but we do not reuse the $TimeID$ attribute in the **ShopsIn** relation. This means that we theoretically could have multiple tuples with the same values, representing customers that went shopping at the same location multiple times a day. But as we see in later sections, we take care of that in the probability calculations by a proper grouping of such events and therefore there won't be value equivalent tuples in the final **ShopsIn** relation.

Transaction

	LocationID	*DateID*	*TimeID*	*CustomerID*	...
tr_1	ZH MM Meilen	2014-03-05	160135	Jane Doe	...

(a) The **Transaction** Relation

ShopsIn^P

	Cst	*Loc*	*T*	*P*
s_1	Jane Doe	ZH MM Meilen	2014-03-05	0.33

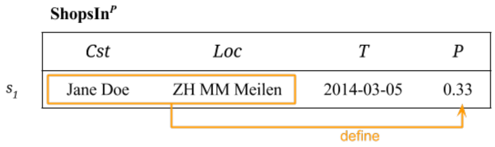
(b) The **ShopsIn** Relation

Figure 3.1: Deriving the **ShopsIn** Relation from the **Transaction** Relation

We describe 3 different approaches based on which we calculate probability values for the tuples of $ShopsIn$. Subsection 3.1 describes probabilities that depend on a customer's likeliness to shop at a specific location, subsection 3.3 looks at how weekdays might be used to define probability values and subsection 3.4 shows a combination of both as well as an explanation why we decided that this is the most reasonable approach.

3.1 Time-Independent Shopping Preferences

Our first approach to calculate probabilities relies on a customer's preference to shop at a specific location compared to the other locations the customer went shopping. Figure 3.2 shows again the sample tuple from Figure 3.1b of how the data of the $\mathbf{ShopsIn}^P$ relation should look like but additionally points out that we will focus on the Cst and the Loc attributes to calculate the probabilities.



ShopsIn^P

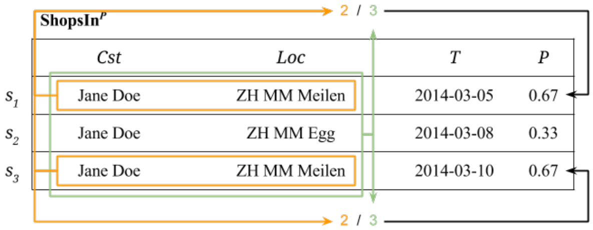
	Cst	Loc	T	P
s_1	Jane Doe	ZH MM Meilen	2014-03-05	0.33

Figure 3.2: The Attributes Cst and Loc are used to calculate P

In order to get the probability values of the customer's likeliness to shop at specific locations we need to calculate the portion of how many times a customer has been shopping at the location of the current tuple compared to the total amount of times the customer went shopping, no matter which location. Let $\boldsymbol{s}$ be an abbreviation $\mathbf{ShopsIn}$, then the calculation formula to get the probability P for every $s_i \in \boldsymbol{s}$ looks as follows:

$$s_i.P \;=\; \frac{|\{s \mid s \in \boldsymbol{s} \wedge s.Cst = s_i.Cst \wedge\, s.Loc = s_i.Loc\}|}{|\{s \mid s \in \boldsymbol{s} \wedge s.Cst = s_i.Cst\}|}$$

Figure 3.3 shows an example with two more rows of *Jane Doe* shopping at *ZH MM Egg* on *2014-03-08* and at *ZH MM Meilen* on *2014-03-10*. Since *Jane Doe* has been shopping at *ZH MM Meilen* two times of the overall three times, the two tuples s_1 and s_3 get each a probability of $2/3$, i.e 0.67, which is shown in the Figure as well.



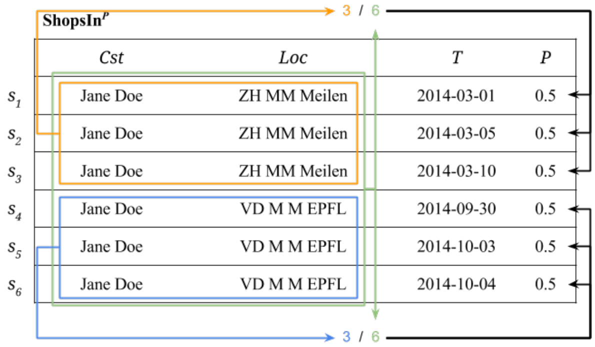
ShopsIn^P

	Cst	Loc	T	P
s_1	Jane Doe	ZH MM Meilen	2014-03-05	0.67
s_2	Jane Doe	ZH MM Egg	2014-03-08	0.33
s_3	Jane Doe	ZH MM Meilen	2014-03-10	0.67

Figure 3.3: Calculating the Probabilities

The probability values reflect our desire to give higher probabilities to tuples that contain locations that *Jane Doe* has visited more often. Nevertheless, if we consider that there exist much more tuples of *Jane Doe* in Migros' database, we see that this approach is rather premature. One rather undesirable fact is that all tuples where *Jane Doe* shops at *ZH MM Meilen* have the same probability values and that they're completely ignoring a customers changing preferences over time.

Locations where a customer goes shopping might change over time. Reasons might be that a customer moves to another place, that the customer get a job in another city or educational reasons are shifting the customers main spending time to other places. Figure 3.4 shows a simplified example of *Jane Doe* who shops in spring (s_1, s_2, s_3) in *ZH MM Meilen* and in autumn in *VD M M EPFL* (s_4, s_5, s_6). Since the proportion between tuples containing *ZH MM Meilen* and tuples containing *VD M M EPFL* is the same, the probabilities is the same for all six sample tuples, each calculated with $3/6 = 0.5$.



ShopsIn^P

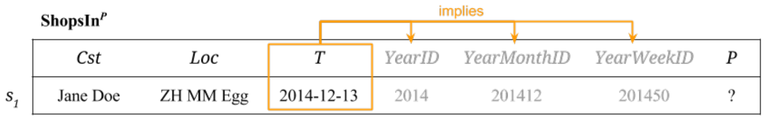
	Cst	*Loc*	*T*	*P*
s_1	Jane Doe	ZH MM Meilen	2014-03-01	0.5
s_2	Jane Doe	ZH MM Meilen	2014-03-05	0.5
s_3	Jane Doe	ZH MM Meilen	2014-03-10	0.5
s_4	Jane Doe	VD M M EPFL	2014-09-30	0.5
s_5	Jane Doe	VD M M EPFL	2014-10-03	0.5
s_6	Jane Doe	VD M M EPFL	2014-10-04	0.5

Figure 3.4: Disadvantage of the Approach

We don't know exactly why *Jane Doe* is mainly shopping in *ZH MM Meilen* in March and afterwards, around the beginning of October, she prefers to shop in *VD M M EPFL*, but it points out that it might be quite naive to let all existing events influence the probability of a single event, disregarding how far in time they have happened.

3.2 Shopping Preferences Over time

In order to deal with the rather undesired temporally unlimited influence of events on each other, we've decided to calculate the probabilities with the help of specific scopes. With a scope we define a window in which an event is only influenced by other events within this window. We define three different calculation scopes since with that we're able to decide later which calculation scope or combination of calculation scopes is best to use for the final probability values. To be specific about the different scopes, we perform calculations on a yearly, a monthly and a weekly basis. In order to be able to do that, we need the year-, month- as well as the week- numbers which are all implicitly given by the timestamp T and can be derived from the **Calendar** relation from Figure 2.4. For an easier understanding, we display these attributes explicitly as additional attributes in our targeted **ShopsIn** relation. For example, in Figure 3.5 we have Jane Doe shopping in *ZH MM Egg* on *2014-12-13* with a yet uncalculated probability value. We also display the temporal IDs, e.g. the $YearWeekID$ with a value of *201450* means that the time-point *2014-12-13* is in the *50th* week of the year *2014*.

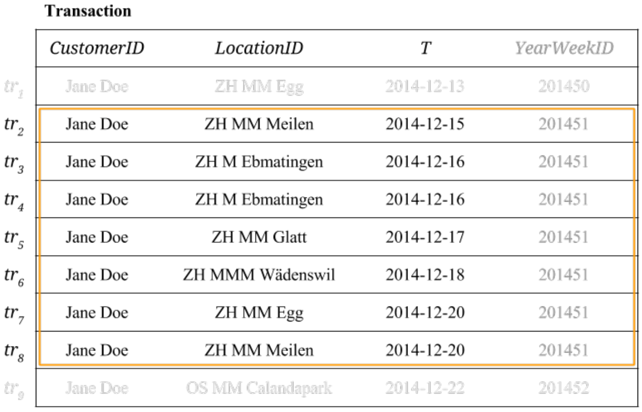


ShopsIn^P

	Cst	*Loc*	*T*	*YearID*	*YearMonthID*	*YearWeekID*	*P*
s_1	Jane Doe	ZH MM Egg	2014-12-13	2014	201412	201450	?

Figure 3.5: The Implicit Scope Attributes

In terms of keeping the visualization and the calculations clearly arranged, we show examples only in a weekly scope and therefore also only use the $YearWeekID$ from the scope attributes to distinguish which events affect the probabilities of certain event groups. Figure 3.6 shows some sample events of the $Transaction$ relation which we will use as source tuples for the probability calculations of some result events for the targeted **ShopsIn** relation. As visualized we will use the 7 source events tr_2 to tr_8 happening in the *51th* week of *2014*.



Transaction

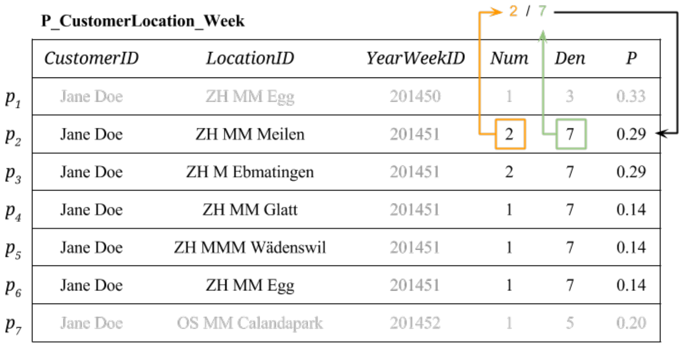
	CustomerID	*LocationID*	*T*	*YearWeekID*
tr_1	Jane Doe	ZH MM Egg	2014-12-13	201450
tr_2	Jane Doe	ZH MM Meilen	2014-12-15	201451
tr_3	Jane Doe	ZH M Ebmatingen	2014-12-16	201451
tr_4	Jane Doe	ZH M Ebmatingen	2014-12-16	201451
tr_5	Jane Doe	ZH MM Glatt	2014-12-17	201451
tr_6	Jane Doe	ZH MMM Wädenswil	2014-12-18	201451
tr_7	Jane Doe	ZH MM Egg	2014-12-20	201451
tr_8	Jane Doe	ZH MM Meilen	2014-12-20	201451
tr_9	Jane Doe	OS MM Calandapark	2014-12-22	201452

Figure 3.6: Sample tuples for Calculating Jane Doe's Probabilities in Week 51

For the weekly calculation scope, we compute the portion of how many times a certain customer has been shopping at the location of the current tuple during the whole week with respect to the total amount of times this customer has been shopping during the whole week. The calculation formula to get the probability P for every $s_i \in \boldsymbol{s}$ looks now as follows:

$$s_i.P \;=\; \frac{|\{s \mid s \in \boldsymbol{s} \wedge s.Cst = s_i.Cst \wedge\, s.Loc = s_i.Loc \,\wedge\, s.YearWeekID = s_i.YearWeekID\}|}{|\{s \mid s \in \boldsymbol{s} \wedge s.Cst = s_i.Cst \,\wedge\, s.YearWeekID = s_i.YearWeekID\}|}$$

Figure 3.7 shows the detailed computation of the probabilities with respect to the weekly scope whereas we keep the results temporarily in a separate relation $P\_CustomerLocation\_Week$. The attributes Num and Den represent the numerator and the denominator of a tuples current probability calculation. For example, since the two tuples tr_2 and tr_8 from Figure 3.6 represent events of Jane Doe shopping at ZH MM Meilen in the *51th* week of *2014*, the attribute Num of tuple p_2 in Figure 3.7 has the value 2. Vice versa, the denominator Den of p_2 is of value 7 due to the corresponding total amount of events given by the tuples tr_2 to tr_8 placed in the *51th* week of *2014*. By calculating $Num = 2$ divided by $Den = 7$ we finally get the probability $P = 0.29$ for the tuple p_2.



P_CustomerLocation_Week

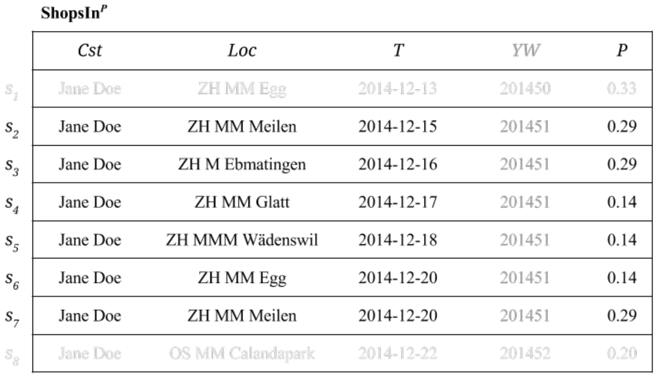
	CustomerID	*LocationID*	*YearWeekID*	*Num*	*Den*	*P*
p_1	Jane Doe	ZH MM Egg	201450	1	3	0.33
p_2	Jane Doe	ZH MM Meilen	201451	2	7	0.29
p_3	Jane Doe	ZH M Ebmatingen	201451	2	7	0.29
p_4	Jane Doe	ZH MM Glatt	201451	1	7	0.14
p_5	Jane Doe	ZH MMM Wädenswil	201451	1	7	0.14
p_6	Jane Doe	ZH MM Egg	201451	1	7	0.14
p_7	Jane Doe	OS MM Calandapark	201452	1	5	0.20

Figure 3.7: Example of Probability Calculations in a Weekly Scope

Given the above, we're able to derive a **ShopsIn** relation with the probabilities calculated in a weekly scope. Figure 3.8 shows the $\textbf{ShopsIn}^P$ relation that results from assigning the probability values from the $P\_CustomerLocation\_Week$ relation from Figure 3.7 to the events from the **Transaction** relation from Figure 3.6. Let $\boldsymbol{p}$, $\boldsymbol{tr}$ and $\boldsymbol{s}$ be abbreviations for $P\_CustomerLocation\_Week$, **Transaction** and $\textbf{ShopsIn}^P$ respectively, the relational algebra expression for deriving the $\textbf{ShopsIn}^P$ relation looks as follows:

$$\boldsymbol{x} \leftarrow (\rho_{Cst,Loc,T,YW,Num,Den,P}(\boldsymbol{p})) \bowtie_\theta ({}_{Cst,Loc,T,YW}\vartheta(\rho_{Cst,Loc,T,YW}(\boldsymbol{tr}))),$$
$$\theta \equiv p.Cst = tr.Cst \wedge p.Loc = tr.Loc \wedge p.YW = tr.YW$$

$$\boldsymbol{s} \leftarrow \pi_{tr.Cst,tr.Loc,tr.T,p.YW,p.P}(\boldsymbol{x})$$



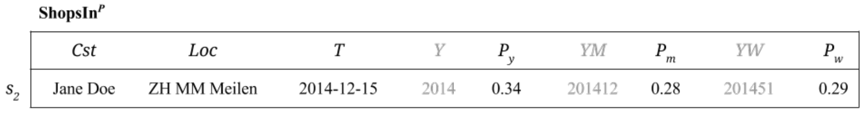
ShopsIn^P

	Cst	*Loc*	*T*	*YW*	*P*
s_1	Jane Doe	ZH MM Egg	2014-12-13	201450	0.33
s_2	Jane Doe	ZH MM Meilen	2014-12-15	201451	0.29
s_3	Jane Doe	ZH M Ebmatingen	2014-12-16	201451	0.29
s_4	Jane Doe	ZH MM Glatt	2014-12-17	201451	0.14
s_5	Jane Doe	ZH MMM Wädenswil	2014-12-18	201451	0.14
s_6	Jane Doe	ZH MM Egg	2014-12-20	201451	0.14
s_7	Jane Doe	ZH MM Meilen	2014-12-20	201451	0.29
s_8	Jane Doe	OS MM Calandapark	2014-12-22	201452	0.20

Figure 3.8: **ShopsIn** Relation with Weekly Ccoped Probabilities

Before we're joining $\boldsymbol{p}$ with $\boldsymbol{tr}$, we're already renaming the attributes of these relations to the final attribute names they should have in $\boldsymbol{s}$ and for $\boldsymbol{tr}$ we perform even a grouping before the join. The grouping is necessary e.g. for duplicate tuples like tr_3 and tr_4 in the **Transaction** relation from Figure 3.6. Then we join $\boldsymbol{p}$ with $\boldsymbol{tr}$ on the attributes Cst, Loc and YW. Take note, that in the formula above we use the intermediary relation $\boldsymbol{x}$ only for better visualizing purposes. After the join we use a projection to get finally the desired attributes we want to have in the $\mathbf{ShopsIn}^P$ relation.

To get an idea how it looks like when we do calculations with respect to all three scoping attributes, Figure 3.9 shows event s_2 from Figure 3.8 but additionally extended with the results of the probability calculations on the yearly and monthly basis. We name the probability columns according to their different calculation scopes P_y, P_m and P_w for yearly, monthly and weekly scopes respectively.



ShopsIn^P

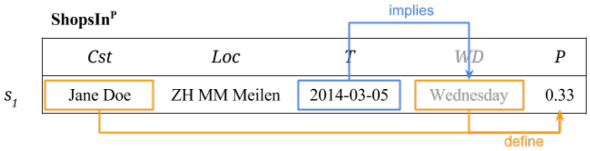
	Cst	*Loc*	*T*	*Y*	P_y	*YM*	P_m	*YW*	P_w
s_2	Jane Doe	ZH MM Meilen	2014-12-15	2014	0.34	201412	0.28	201451	0.29

Figure 3.9: ShopsIn relation with scoped probabilities

We do not give examples for the yearly and monthly based calculations here since the calculations are directly derivable from the given example of the calculations on a weekly basis.

3.3 Weekday Periodicity

The second and rather different approach to calculate the probabilities for our targeted $ShopsIn$ relation depends on the weekdays given by the tuples' dates. We want to examine how likely a customer tends to shop on specific weekdays. The weekdays are actually implied by the timestamp itself and can be derived from the $Calendar$ relation, as shown earlier in Figure 2.4. For an easier understanding, we display the weekdays as an additionally attribute WD for $ShopsIn$ as shown in Figure 3.10. The sample tuple represents the fact that *Jane Doe* has been shopping at *ZH MM Meilen* on the *5th March in 2014* and the probability for this event has been *0.33*. The weekday WD points out that the event has happened on a Wednesday.



ShopsIn^P

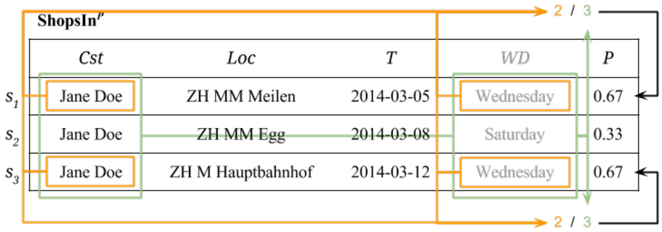
	Cst	*Loc*	*T*	*WD*	*P*
s_1	Jane Doe	ZH MM Meilen	2014-03-05	Wednesday	0.33

Figure 3.10: The Attributes Cst and WD are used to calculate P

In order to get the probability values of the customer's likeliness to shop at specific weekdays, we need to calculate the portion of how many times a customer has been shopping on this weekday of the current tuple compared to the total amount of times the customer went shopping, no matter which weekday. The calculation formula to get the probability P for every $s_i \in \boldsymbol{s}$ looks as follows:

$$s_i.P \;=\; \frac{|\{s \mid s \in \boldsymbol{s} \wedge s.Cst = s_i.Cst \wedge s.WD = s_i.WD\}|}{|\{s \mid s \in \boldsymbol{s} \wedge s.Cst = s_i.Cst\}|}$$

Figure 3.11 shows an example with two more rows of Jane Doe shopping at *ZH MM Egg* on *2014-03-08* which is a *Saturday* and at *ZH M Hauptbahnhof* on *2014-03-12* which is a *Wednesday*. Since Jane Doe has been shopping at *Wednesday* two times of the overall three times, the two tuples s_1 and s_3 get each a probability of $2/3 = 0.67$, whereas the calculation is also visualized in the Figure.

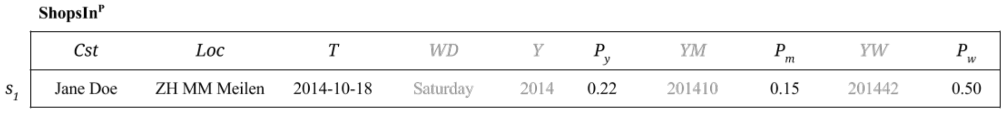

ShopsIn^P

	Cst	*Loc*	*T*	*WD*	*P*
s_1	Jane Doe	ZH MM Meilen	2014-03-05	Wednesday	0.67
s_2	Jane Doe	ZH MM Egg	2014-03-08	Saturday	0.33
s_3	Jane Doe	ZH M Hauptbahnhof	2014-03-12	Wednesday	0.67

Figure 3.11: Deriving P from the values of WeekdayID

Unfortunately we have very similar issues here like in the first approach before, where we examined a customer's preference of specific locations. All of *Jane Doe*'s events taking place on *Wednesday*'s have the same probability values. To lower this phenomena, we will again apply the three different calculation scopes on this probability calculations as we've done it in the section before for the first approach. Since the procedure stays the same and it is possible to apply it directly on this type of probability calculation, we won't deliver any further examples for that and will directly look at a final result event of $\textbf{ShopsIn}^P$.

The sample event s_1 from Figure 3.12 means that we look at an event for Jane Doe shopping at ZH MM Meilen on *2014-10-18* which is a *Saturday*. The probabilities state, what the probability has been that Jane Doe shops on a *Saturday*. P_y sets the scope on the current year *2014*, P_m sets the scope on the current month *October* of *2014* and P_w sets the scope on the current week number *42* of *2014*.


ShopsIn^P

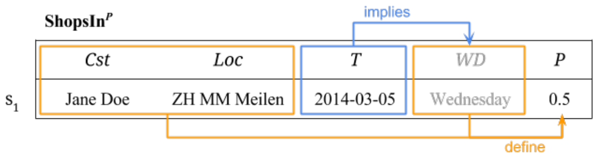
	Cst	*Loc*	*T*	*WD*	*Y*	P_y	*YM*	P_m	*YW*	P_w
s_1	Jane Doe	ZH MM Meilen	2014-10-18	Saturday	2014	0.22	201410	0.15	201442	0.50

Figure 3.12: $\textbf{ShopsIn}$ Relation with Scoped Probabilities

3.4 Location Preferences and Weekday Periodicity

The approaches that we've shown before seem quite reasonable methods to get probability values for the $\textbf{ShopsIn}$ relation. It might be quite straightforward to combine them since the probability of where a customer tends to shop sounds very likely influenced by the current weekday and vice versa. Since the exact science of the people's shopping behaviour would by far exceed the scope of this paper, we have to state some naive assumptions about a customer's daily routine. We believe that the routine is formed to a certain extend by a customer's obligations. We assume that a customer's daily routine and his obligation not only indicate on what weekdays a customer tends to shop, but also that the weekdays give us a hint where a customer is more likely to shop. Or in the other direction, that given a customer's purchase in a certain location, we could tell on which weekday(s) this event might be more likely.

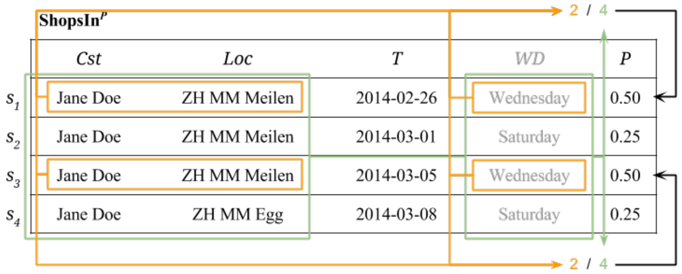
So with that, the third approach combines the two approaches from before and therefore makes the probabilities depend on the values of the customers Cst, locations Loc as well as on the values of the weekdays WD. Again as before, the weekdays are implied by the timestamp itself and for an easier understanding we additionally visualize them as a separate column as shown in Figure 3.13. The sample tuple represents the fact that Jane Doe has been shopping at ZH MM Meilen on the *5th March in 2014* which has been a *Wednesday* and the probability for this event has been *0.5*.



	Cst	Loc	T	WD	P
s1	Jane Doe	ZH MM Meilen	2014-03-05	Wednesday	0.5

Figure 3.13: The Attributes Cst, Loc and WD are used to calculate P

In order to get the probability values of the customer's likeliness to shop at specific locations on specific weekdays we need to calculate the portion of how many times a customer has been shopping at this location on this weekday of the current tuple compared to the total amount of times the customer went shopping, no matter which location or weekday. Figure 3.14 shows an example with three more rows. Jane Doe is shopping at *ZH MM Meilen* on *2014-02-26* which is a *Wednesday*, at *ZH MM Meilen* on *2014-03-01* which is a *Saturday* and also at *ZH MM Egg* on *2014-03-08* which is a *Saturday*. Since *Jane Doe* has been shopping at *ZH MM Meilen* on *Wednesday*'s two times of the overall four times, the two tuples s_1 and s_3 get each a probability of $2/4 = 0.5$. The calculation is also visualized in the Figure.

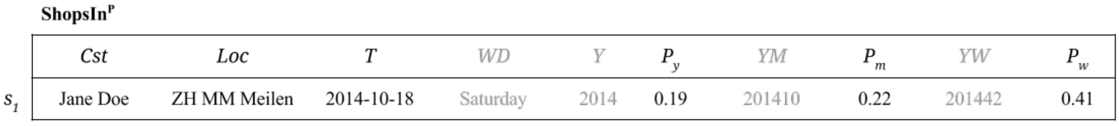


	Cst	Loc	T	WD	P
s1	Jane Doe	ZH MM Meilen	2014-02-26	Wednesday	0.50
s2	Jane Doe	ZH MM Meilen	2014-03-01	Saturday	0.25
s3	Jane Doe	ZH MM Meilen	2014-03-05	Wednesday	0.50
s4	Jane Doe	ZH MM Egg	2014-03-08	Saturday	0.25

Figure 3.14: Calculating Probabilities

As the example shows, we have again some tuples with same probability values, so we will reapply the three different calculation scopes to this probability calculations as we've done it before. Similarly as before, we do not repeat any examples since the procedure stays the same and therefore jump right through to a result example.

Almost identically as in the other two approaches, the sample event s_1 from Figure 3.15 means that we look at an event for Jane Doe shopping at *ZH MM Meilen* on *2014-10-18* which is a *Saturday*. The probabilities state, what the probability has been that Jane Doe shops in ZH MM Meilen on a *Saturday*. P_y, P_m and P_w represent again the probabilities according to their respective scopes Y, YM and YW.



ShopsIn^P

	Cst	*Loc*	*T*	*WD*	*Y*	P_y	*YM*	P_m	*YW*	P_w
s_1	Jane Doe	ZH MM Meilen	2014-10-18	Saturday	2014	0.19	201410	0.22	201442	0.41

Figure 3.15: ShopsIn relation with scoped probabilities

3.5 Evaluating Probability Calculations

The probability calculations for the **ShopsIn** relation reveal some reasonable considerations behind the computations but also some aspects that need to be investigated more deeply in future work. On the one hand the dependency of the probabilities on the combined attributes $CustomerID$, $LocationID$ and $WeekdayID$ received some positive feedback. On the other hand, as the final results of the $ShopsIn^{TP}$ relation has shown, a daily granularity for the tuples reveals a data set that is a bit to perforated. That means that the $ShopsIn^{TP}$ relation we finally got, has to many temporal gaps in a customers shopping behaviour at specific locations. This is not always bad, but with respect to a direct usage in the next chapter's discussed PTA algorithm, the amount of tuples that can be merged together is far too little. Also the calculation scopes are a bit too static. In this chapter we have shown yearly, monthly and weekly scopes that were quite easily calculable with the given helper tables in the database. But after some late considerations it makes sense to set the scopes more dynamically as a window around a tuples current timestamp. E.g. if we have a given time point *2014-10-18* and want to consider a yearly scope, taking the year 2014 as scope for the calculations is the approach given in this chapter. An other but probably better approach follows the idea to set e.g. a yearly scope in a way that we consider for the calculations the half year before the date *2014-10-18* and the half year after that date. But as said previously, this investigations need to be pursuit in future work.

4 The PTA

This chapter describes how the parsimonious temporal aggregation algorithm (PTA [1]) is modified in order to reduce the enormous mass of data that comes along with our investigation of Migros' data set. So far in this project, we have a huge amount of time point timestamped probabilistic data and the PTA helps us to reduce the amount of tuples in the desired relations by forming intervals which follow the flow of the data. At the same time the PTA also ensures that the introduced error which emerges from the reduction is minimal. Gordevicius et al. [1] have introduced different PTA approaches and presented their respective algorithms that are designed for general-purpose programming languages. However, in order to be able to use a PTA algorithm in the Teradata database environment of the Migros' database infrastructure, a nearly pure SQL approach is needed. First we show the obstacles that prevent a direct adaptation of the existing algorithms to probable equal SQL versions. After that, we show some very basic principals of the PTA algorithm which we then reuse for a naive algorithmic approach that can be adapted to a simple SQL approach. We illustrate the performance related drawbacks of this basic SQL algorithm before we finally introduce some alternative modifications that drastically speed up the SQL version of the PTA algorithm.

4.1 Obstacles of Existing PTA Algorithms

Given the implementation of PTA using a general purpose programming language Gordevicius et al. [1] also shown the advantages regarding the performance while keeping the merging introduced at a low as possible level. Unfortunately these algorithms rely on programming language constructs and mechanisms that are not that easily to reflect in a pure SQL environment.

Initially, a dynamic programming approach has been introduced, that depends on constructs like matrices. Matrices are not naturally given structures in SQL and some non-trivial algorithmic transformations would be needed to conceptually mirror these matrices that this algorithm relies on. In an environment like in Migros, where we have billions of tuples, it is also clear that we can't try to directly apply database tables to represent matrices. The matrices can become enormously huge, since their two dimensions are given by the number of tuples of the input relation and the target size that the result relation should have.

Furthermore, a greedy algorithm has been proposed, that streams input tuples and inserts them into a heap for further merging procedures. Streaming data is also not a natural concept of common SQL languages and mirroring a heap data structure in a table would need some thoughtful considerations on how to map it in SQL.

Both approaches have in common that they keep the data or a part of the data that will be merged in main memory and the same holds true for the matrices or heap structures. In a pure SQL environment some rethinking is needed since intermediary results of iterative merging steps will be stored in the database and are considerably more expensive for this reason. Especially in the data dimensions of Migros it is clear that excessive amounts of iterative inserts, updates and deletes need to be avoided in order to keep the algorithm with good performance.

So even though it is not easily or effectively to reuse some mechanisms of the introduced algorithms, some basic concepts can be kept. In the following sections we will show the basic concepts that our modified PTA adopts and in which way these concepts are reused in our own SQL based version of the PTA.

4.2 Basic PTA Principles

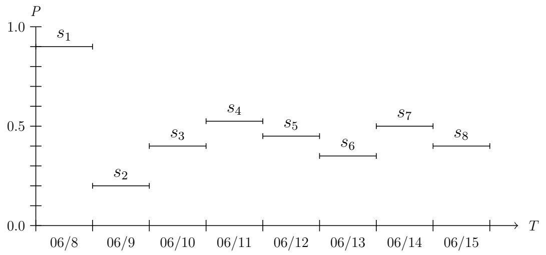
For the scope of this project, we assume a linearly ordered, discrete time domain, Ω^T. A time interval is represented as $T = [Ts, Te] \in \Omega^T \times \Omega^T$ and is a contiguous set of time points, where Ts and Te are the inclusive start an end point respectively. The time interval that is associated with each tuple represents the tuple's valid time. Since we're in the domain of temporal probabilistic databases, tuples represent events and the probability $P \mapsto [0, 1]$ states the probability of an event for each time point in the time interval of the event. The schema of a temporal probabilistic relation is denoted as $R = (A_1, ..., A_m, Ts, Te, P)$, where the non-temporal and non-probabilistic attributes are represented as $A_1, ..., A_m$, the time interval is represented as Ts, Te and the probability is represented as P. To shorten notations we use the abbreviation $\boldsymbol{A} = A_1, ..., A_m$ for all non-temporal/non-probabilistic attributes of a relation. For $r.A_1 = s.A_1 \wedge ... \wedge r.A_m = s.A_m$ we write the abbreviated version $r.\boldsymbol{A} = s.\boldsymbol{A}$.

Example 1 *As a running example we use the* **ShopsIn** *relation in Figure 4.1a which records shopping trips: a customer (Cst), the shopping location (Loc), and the time period (inclusive [Ts, Te] in a daily granularity) in which for each day there is the probability (P) that the customer will shop at this very specific location. For instance, tuple s_1 states that there is a 0.9 probability that Ann shops at HB on the 8th of June. In the graphical illustration in Figure 4.1b, the timestamps of the consecutively adjacent tuples s_1 to s_8 are are shown as horizontal lines and the vertical positioning is proportional to the probability values.*

ShopsIn

	Cst	*Loc*	Ts	Te	P
s_1	Ann	HB	06/08	06/08	0.9
s_2	Ann	HB	06/09	06/09	0.2
s_3	Ann	HB	06/10	06/10	0.4
s_4	Ann	HB	06/11	06/11	0.525
s_5	Ann	HB	06/12	06/12	0.45
s_6	Ann	HB	06/13	06/13	0.35
s_7	Ann	HB	06/14	06/14	0.5
s_8	Ann	HB	06/15	06/15	0.4

(a) **ShopsIn** Relation



(b) Graphical Illustration of **ShopsIn**

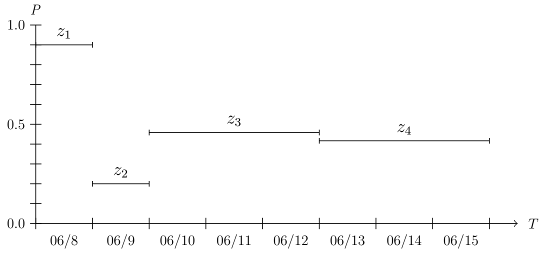
Figure 4.1: The **ShopsIn** Relation with some Sample Data

Figure 4.2a and Figure 4.2b show the result when we apply the PTA algorithm where the *result size is reduced to 4 tuples. Four separate merging steps on the input relation in Figure 4.1a are necessary to get the desired result size. For instance, we're using two mergings for s_3, s_4 and s_5 to get z_3 as one of the final PTA result tuples. The probability of z_3 is computed by averaging the probabilities of s_3, s_4 and $_5$ over each day, i.e.,* 0.4, 0.525 *and* 0.45 *for each day respectively, yielding the value* 0.45833.

ShopsIn

	Cst	*Loc*	*Ts*	*Te*	*P*
z_1	Ann	HB	06/08	06/08	0.9
z_2	Ann	HB	06/09	06/09	0.2
z_3	Ann	HB	06/10	06/12	0.45833
z_4	Ann	HB	06/13	06/15	0.4166

(a) The reduced **ShopsIn** Relation

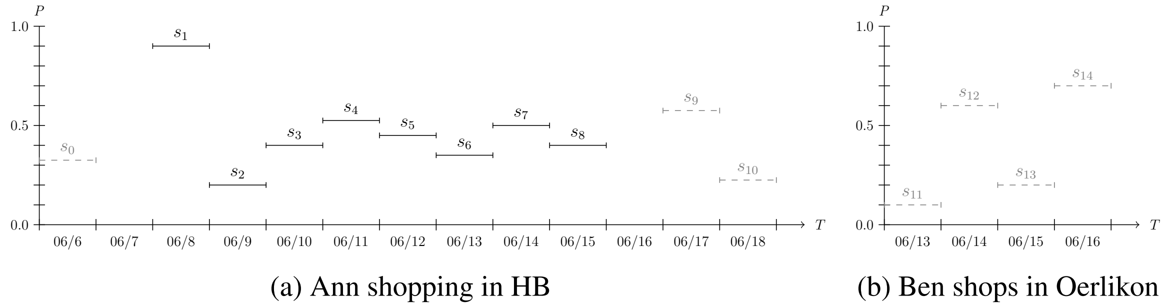


(b) Graphical illustration of the reduced relation

Figure 4.2: The **ShopsIn** relation after applying the PTA

Usually multiple groups of consecutively adjacent tuples exist in an input relation that is going to be reduced by the PTA algorithm. We will term such groups *adjacency-groups* from now on and later we will talk about the advantage of knowing the existence of *adjacency-groups*.

Example 2 *In Figure 4.1 and Figure 4.2 we have omitted other* adjacency-groups *for better visualizing purposes, but now we will show how it looks like when we have multiple* adjacency-groups *in the data. Figure 4.3a shows that we can have several* adjacency-groups *for the same customer making purchases in a certain location, i.e. Ann making also other purchases in HB which are represented by the tuples* s_0, s_9 *and* s_{10}. *In Figure 4.3b we additionally like to show that there generally exist more records in the* **ShopsIn** *relation that state the fact that also other customers are making purchases in certain locations. E.g. the tuples* $s_{10}, s_{11}, s_{12} ands_{13}$ *represent the fact that a customer called Ben might be shopping at Oerlikon with different probability values on each day between the 13th and 16th of June. But for now we will ignore other possible adjacency groups and we will only focus on the tuples* s_1 *to* s_8.



(a) Ann shopping in HB

(b) Ben shops in Oerlikon

Figure 4.3: Other groups of consecutively adjacent tuples

4.2.1 The Merging of Adjacent Tuples

One of the most basic concepts of the PTA [1] algorithm is that it accomplishes the reduction by repeatedly merging adjacent tuples together. Two tuples are defined to be adjacent if they hold two specific conditions. First, they need to have the same values for the grouping attributes and second, that the start-point of one of the two tuples immediately follows the end-point of the other tuple. Let $\boldsymbol{r}$, R, $\boldsymbol{A}$ be as previously introduced, then two tuples $r_i, r_j \in \boldsymbol{r}$ are *adjacent*, $r_i \prec r_j$, iff

1. $r_i.\boldsymbol{A} = r_j.\boldsymbol{A}$
2. $r_i.Te = r_j.Ts - 1$

Example 3 *In our example data 4.1 we see that e.g. s_1 and s_2 have the same grouping attributes $\boldsymbol{A} = \{Cst, Loc\}$ with $s_1.Cst = s_2.Cst =$ Ann and $s_1.Loc = s_2.Loc =$ HB. Additionally also the second adjacency constraint holds for these two tuples; the start-point Ts of s_2 immediately follows the end-point Te of s_1, i.e. $s_2.Te = s_1.Ts - 1 = 06/08$. Therefore, s_1 and s_2 are adjacent $s_1 \prec s_2$. Since all the tuples in our example have the same values for the grouping attributes and furthermore are ordered by the attribute Ts, it is immediately clear that we have 7 adjacent tuple-pairs $s_1 \prec s_2$, $s_2 \prec s_3$, $s_3 \prec s_4$, $s_4 \prec s_5$, $s_5 \prec s_6$, $s_6 \prec s_7$ and $s_7 \prec s_8$, which of course can be written in short as $s_1 \prec s_2 \prec s_3 \prec s_4 \prec s_5 \prec s_6 \prec s_7 \prec s_8$.*

$PTA_{parallel}$ *tuple-pair* and *merge-pair* interchangeably and write $r_i \oplus r_j$ to denote the merging of the two adjacent tuples $r_i, r_j \in \boldsymbol{r}, r_i \prec r_j$ which is defined as

$$r_i \oplus r_j = (r_i.A_1, ..., r_i.A_m, r_i.Ts, r_j.Te, P_{new}),$$

$$P_{new} = \frac{|r_i.T| * r_i.P + |r_j.T| * r_j.P}{|r_i.T| + |r_j.T|}$$

Example 4 *The merging $s_1 \oplus s_2$ of the two tuples s_1 and s_2 from Figure 4.1 would result to the new tuple $z_1 = ($Ann$,$HB$, 06/08, 06/09, 0.55)$. Ann and HB are given by the values of the current grouping attributes Cst and Loc, $06/08$ is given by the inclusive start time point $s_1.Ts$ and $06/09$ is given by the inclusive end time point $s_2.Te$. The new aggregated value $z_1.P = 0.55$ is calculated by the time weighted mean of $s_1.P$ and $s_2.P$. The calculations are as follows:*

$$z_1.P_{new} = \frac{|s_1.T| * s_1.P + |s_2.T| * s_2.P}{|s_1.T| + |s_2.T|} = \frac{1*0.9+1*0.2}{1+1} = \frac{1.1}{2} = 0.55$$

The syntax for the reduction ρ of a relation $\boldsymbol{r}$ to a certain size c is denoted as $\rho(\boldsymbol{r}, c)$. As long as the size of $\boldsymbol{r}$, i.e. $|\boldsymbol{r}|$, hasn't reached the desired result size c, we continue to merge tuples in relation $\boldsymbol{r}$.

4.2.2 The Sum Squared Error and Merging Order

In order to keep the introduced error of the reductions as small as possible, one concept of the PTA is to merge the most similar adjacent tuples together. In the context of this paper, it means that the PTA needs to merge together tuples that have the most similar probability values in P, weighted with respect to their time interval. With the computation of the sum squared error, SSE, that would emerge from merging tuple-pairs together, the algorithm is able to determine the most similar candidate merge-pair. The general calculation formula for the SSE has been introduced by Gordevicius et al. [1] and we slightly adapt it to the current temporal probabilistic context. Let $\boldsymbol{r}, R, \boldsymbol{A}$ be defined as before, $\boldsymbol{z} = \rho(r, \cdot)$ be a reduction of $\boldsymbol{r}$, and let for each $z \in \boldsymbol{z}, \boldsymbol{r}_z = \{r | r \in \boldsymbol{r} \wedge z.\boldsymbol{A} = r.\boldsymbol{A} \wedge r.T \subseteq z.T\}$ be the set of all input tuples that are merged into z. The $SSE(\boldsymbol{r}, \boldsymbol{z})$ of reducing $\boldsymbol{s}$ to $\boldsymbol{z}$ is

$$SSE(\boldsymbol{r}, \boldsymbol{z}) = \sum_{z \in \boldsymbol{z}} \sum_{r \in \boldsymbol{r_z}} |r.T|(r.P - z.P_{new})^2$$

Example 5 *To get the SSE that results from merging s_1 and s_2, the PTA computes the squared distance (over the aggregation result P_{new}) between z_1 and the input tuples s_1 and s_2. The detailed computation is as follows:*

$$\begin{aligned} SSE(\{s_1, s_2\}, z_1) &= |s_1.T| * (s_1.P - z_1.P)^2 + |s_2.T| * (s_2.P - z_1.P_{new})^2 \\ &= 1 * (0.9 - 0.55)^2 + 1 * (0.2 - 0.55)^2 \\ &= 1 * 0.35^2 + 1 * (-0.35)^2 \\ &= 0.1225 + 0.1225 \\ &= 0.245 \end{aligned}$$

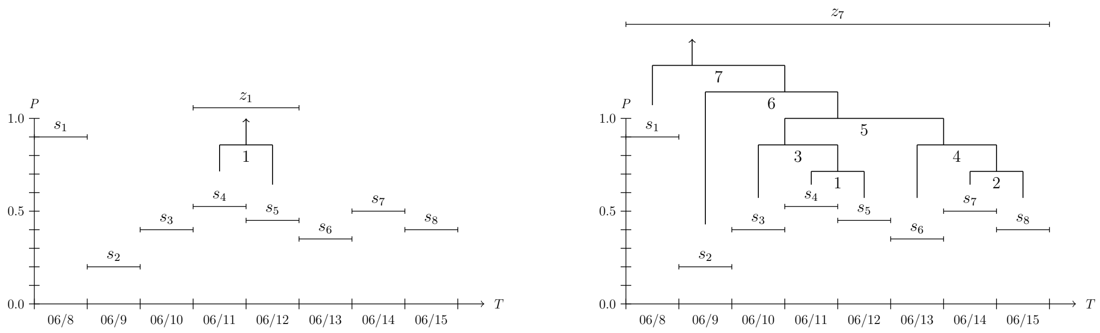
After calculating the SSE for each possible merge-pair candidate, the PTA chooses the tuple-pair that has the smallest SSE. If the decision is ambiguous, i.e. two or more merges would have the same resulting SSE, the PTA algorithm then chooses to merge the tuple-pair that has the earliest starting time point.

Example 6 *The SSE calculations for the first merging iteration in our running example are shown in Figure 4.4. Each tuple represents a possible candidate merge-pair for the first merge of two adjacent tuples. Each of them is expanded with the new probability value in P_{new} and the SSE that would result from the merge of the two corresponding tuples. As the results show, the merge of s_4 and s_5 would produce the smallest error of SSE $= 0.0028125$.*

	Cst	*Loc*	*Ts*	*Te*	P_{new}	*SSE*
$s_1 \oplus s_2$	Ann	HB	06/08	06/09	$\frac{1*0.9+1*0.2}{1+1} = 0.55$	$1 * (0.9 - 0.55)^2 + 1 * (0.2 - 0.55)^2 = 0.245$
$s_2 \oplus s_3$	Ann	HB	06/09	06/10	$\frac{1*0.2+1*0.4}{1+1} = 0.3$	$1 * (0.2 - 0.3)^2 + 1 * (0.4 - 0.3)^2 = 0.02$
$s_3 \oplus s_4$	Ann	HB	06/10	06/11	$\frac{1*0.4+1*0.525}{1+1} = 0.4625$	$1 * (0.4 - 0.4625)^2 + 1 * (0.525 - 0.4625)^2 = 0.0078125$
$s_4 \oplus s_5$	Ann	HB	06/11	06/12	$\frac{1*0.525+1*0.45}{1+1} = 0.4875$	$1 * (0.525 - 0.4875)^2 + 1 * (0.45 - 0.4875)^2 = 0.0028125$
$s_5 \oplus s_6$	Ann	HB	06/12	06/13	$\frac{1*0.45+1*0.35}{1+1} = 0.4$	$1 * (0.45 - 0.4)^2 + 1 * (0.35 - 0.4)^2 = 0.005$
$s_6 \oplus s_7$	Ann	HB	06/13	06/14	$\frac{1*0.35+1*0.5}{1+1} = 0.425$	$1 * (0.35 - 0.425)^2 + 1 * (0.5 - 0.425)^2 = 0.01125$
$s_7 \oplus s_8$	Ann	HB	06/14	06/15	$\frac{1*0.5+1*0.4}{1+1} = 0.45$	$1 * (0.5 - 0.45)^2 + 1 * (0.4 - 0.45)^2 = 0.005$

Figure 4.4: SSE Calculations

Since the calculated result is distinct, the PTA merges the tuples s_4 and s_5 to the resulting tuple z_1, as shown in Figure 4.5a. Note that the vertical positioning of the resulting tuple does not not represent it's newly calculated probability value of 0.4875.



(a) First Best Merge

(b) Complete Merging Order

Figure 4.5: **ShopsIn** relation

4.2.3 PTA Size and Error Bound

The concept of the PTA allows to choose from two different stop criteria regarding the dimension of the reduction, i.e. how many tuples are being merged together until the PTA stops the reduction procedure. The two different possibilities to choose from are:

1. A certain size bound c, as already shown exemplary in Figure 4.2a. With that we indicate the desired size we'd like the final result relation to have. Of course, the data of the input relation can maximally be reduced to a certain size c_{min}, depending on the given data.

2. An error bound ε, that limits the PTA to only merge tuples until a certain total error has been reached. In our project, we don't focus on this approach.

Example 7 *For our running example, we can determine that c_{min} equals 1, since all input tuples have the same values in the grouping attributes and are consecutively adjacent. Therefore we can freely choose the size bound c to be between the input relation size $|\boldsymbol{s}| = 8$ and $c_{min} = 1$. If we repeat the merge procedure as many times as possible, which reveals to be 7 merging steps since we have 8 input tuples, we end up with the final result tuple z_7. The individual merging steps that lead to this result are shown in Figure 4.5b, whereas the numbers below the horizontal merging lines indicate the order of the executed merging steps. This illustration helps as well to determine what would happen if we choose a different size bound. For example if we'd choose $c = 3$, we would then get the final result tuples $\{z_1, z_2, z_3\}$ whereas $z_1 = s_1, z_2 = s_2, z_3 = (s_3 \oplus s_4 \oplus s_5 \oplus s_6 \oplus s_7 \oplus s_8)$. z_3 is the result tuple that would result from the first 5 merging steps.*

4.3 A Basic PTA Algorihtm in an SQL Context

In this section, we show the PTA's most basic concepts, that we've described in Section 4.2 before, in an algorithmic context. We map the algorithm to an SQL context, investigate the algorithm's performance bottlenecks and show some performance measurements that make clear that the basic SQL approach is not sufficient.

4.3.1 The Basic PTA Algorithm

As we have seen in it as well in Section 4.2, the step by step merging of tuples until the input relation reaches a certain size bound c requires a certain amount of iterations. Roughly said, each iteration contains the calculations for the sum squared errors and the actual merge of the most similar candidate tuples. Figure 4.6 shows the algorithm $PTA_{basic}(\boldsymbol{r}, c)$ which comprises the PTA's most basic concepts. Let $\boldsymbol{r}$ be the input relation that should be reduced by the PTA and c be the size to which we want $\boldsymbol{r}$ to be reduced. The schema of relation $\boldsymbol{r}$ is $R = (\boldsymbol{A}, Ts, Te, P, SSE)$, whereas the SSE attribute is used to keep track of the increasing SSE when tuples are merged together. Of course, In the very beginning, all tuples have an SSE of 0 since no tuples have been merged together and therefore no error has been introduced. The algorithm starts with a while loop which depicts the fact that we merge tuples as long as the size bound c is not reached. After that, the two for loops are needed to compose the possible candidate merge-pairs. For each candidate merge-pair we calculate the new probability value p and the emerging sse and then store this merge-pair into $\boldsymbol{z}$. The schema Z of relation $\boldsymbol{z}$ is the same as for $\boldsymbol{r}$ and therefore $Z = R = (\boldsymbol{A}, Ts, Te, P, SSE)$. After having all candidate merge-pairs together, we choose the actual merge-pair, $z_{\oplus}$, to be merged with respect to the constraints we have described in Subsection 4.2.2. We add $z_{\oplus}$ to the relation $\boldsymbol{s}$ and then also remove the two tuples that actually have been used as the source tuples for the newly merged tuple $z_{\oplus}$. Finally. as soon as the relation s reaches the size bound c, we quit the main loop and return the reduced result relation as output.

```
**Algorithm:** $PTA_{basic}(\boldsymbol{r}, c)$
**while** $|\boldsymbol{r}| > c$ **do**
    $\boldsymbol{z} \leftarrow \emptyset$;
    **foreach** $r_i \in \boldsymbol{r}$ **do**
        **foreach** $r_j \in \boldsymbol{r} \wedge r_j.\boldsymbol{A} = r_i.\boldsymbol{A} \wedge (r_j.Ts - 1 = r_i.Te)$ **do**
            $p \leftarrow \frac{|r_i.T| * r_i.P + |r_j.T| * r_j.P}{|r_i.T| + |r_j.T|}$;
            $sse \leftarrow |r_i.T|(r_i.P - p)^2 + |r_j.T|(r_j.P - p)^2 + r_i.SSE + r_j.SSE$;
            $z \leftarrow \{(r_i.\boldsymbol{A}, r_i.Ts, r_j.Te, p, sse)\}$;
            $\boldsymbol{z} \leftarrow \boldsymbol{z} \cup z$;
    $z_{\oplus} \leftarrow \{z \mid z \in \boldsymbol{z} \wedge \nexists z' \in \boldsymbol{z}(z'.SSE < z.SSE \vee (z'.SSE = z.SSE \wedge z'.Ts < z.Ts)\}$;
    $\boldsymbol{r} \leftarrow \boldsymbol{r} \cup z_{\oplus}$;
    $\boldsymbol{r} \leftarrow \boldsymbol{r} - \{r \mid r \in \boldsymbol{r} \wedge r.\boldsymbol{A} = z_{\oplus}.\boldsymbol{A} \wedge r.Ts = z_{\oplus}.Ts \wedge r.Te < z_{\oplus}.Te\}$;
    $\boldsymbol{r} \leftarrow \boldsymbol{r} - \{r \mid r \in \boldsymbol{r} \wedge r.\boldsymbol{A} = z_{\oplus}.\boldsymbol{A} \wedge r.Te = z_{\oplus}.Te \wedge r.Ts > z_{\oplus}.Ts\}$;
**return** $\boldsymbol{r}$;
```

Figure 4.6: The PTA_{basic} algorithm comprising the basic PTA concepts

4.3.2 Mapping to SQL

The algorithm in Figure 4.6 compraises some loop-constructs that need some further explanation on how they're mapped to SQL and the Migros Teradata Environment. Regarding the two nested for loops it doesn't get too complicated since we can represent these with a self join expression:

$$\rho_{\boldsymbol{r_1}}(\boldsymbol{r}) \bowtie_{(r_1.\boldsymbol{A}=r_2.\boldsymbol{A})\wedge(r_2.Ts-1=r_1.Te)} \rho_{\boldsymbol{r_2}}(\boldsymbol{r})$$

The while loop in contrary is not directly implementable with SQL. In each Iteration of the while loop the algorithm has to do it's computation on a new state of the relation $\boldsymbol{r}$ due to the tuple-pairs that get merged in each of the iterations of the while loop. For the realization of the while loop we use Teradata BTEQ, a general-purpose command-based program, that is used to submit SQL queries to a Teradata Database. The SQL queries are wrapped into macros which can then be run via an `execute` command. Other BTEQ-SQL-Files can be included with the `run` command and with the help of some simple `if`, `goto` and `label` directives we can construct a while loop.

Example 8 *Figure 4.7 shows a simplified excerpt of how the BTEQ while-mechanism is implemented. We need two separate files, whereas the* `main.sql` *basically only needs to call the* `loop.sql`*. Inside the* `loop.sql` *we execute the macro* `mergeTuples` *which contains the SQL code that mirrors everything inside the while loop from Figure 4.6. Then we make a check for a certain errorcode, which is set when the size bound is reached. If the condition evaluates to false, we continue with the* `run file=/loop.sql` *command to rerun* `loop.sql` *from the beginning. But if the condition evaluates to true, we jump to the end of the file via the* `goto` *and* `label` *directives. BTEQ quits the current* `loop.sql` *file and continues proceeding in the* `main.sql` *file. Since nothing happens there after the* `run file=/loop.sql` *command, BTEQ stops running the main script and with that the algorithm has reached it's end.*

```
-- <main.sql> : Main BTEQ script
.run file=/loop.sql

-- <loop.sql> : Loop BTEQ script
execute mergeTuples;
.if errorcode = 3513 then .goto ExitLoop
.run file=/loop.sql
.label ExitLoop
```

Figure 4.7: BTEQ While Loop

4.3.3 Complexity and Performance of the Basic SQL Approach

Given $n = |\boldsymbol{r}|$ as the amount of tuples in the input relation and c as the desired size we want relation $\boldsymbol{r}$ to be reduced, the runtime complexity for the PTA_{basic} algorithm is basically given by the three nested loops:

1. the while loop that is run $n - c$ times
2. the outer for loop that runs through all n tuples
3. the inner for loop that runs through all n tuples

The join is computed theoretically using a nested loop and this leads us to a complexity that is cubic to the number of tuples n. Especially with respect to the very big data set that we're confronted with in the Migros Environment, this polynomial complexity is highly undesired:

$$(n - c) * n * n \approx O(n^3).$$

Running the PTA_{basic} algorithm on some sample data sets proves us that the algorithm is far too slow. The measuring results are shown in Figure 4.8. None of the test runs, each of them with different sizes n of the input relation $\boldsymbol{r}$, has finished the merging procedure in a reasonable time. Therefore the size bound c, which we choose to be c_{min} for all three test runs respectively, hasn't been reached. Measuring the number of iterations I that happen in a minute I/M allows us to predict the approximate time T that the algorithm would have needed to reach the end of its computations.

	n	c	$n - c$	I/M	T
#1	$166'773$	$99'266$	$67'507$	60/60	$\approx 19h$
#2	$3'397'074$	$2'047'580$	$1'349'494$	35/60	$\approx 27d$
#3	$84'531'997$	$50'887'420$	$33'644'577$	3/60	$\approx 21y$

Figure 4.8: Performance of the PTA_{Basic} algorithm

Calculating the SSE for each merge-pair and merging the candidate tuple-pair becomes slower the higher n gets. With an increasing n, the number of iterations (n - c) augments proportionally. Therefore two general improvements need to be targeted for the algorithm:

1. Increasing the number of merges per iteration so that less iterations have to be performed in total.
2. Optimizing the calculations for the SSE so that a single iteration needs less time.

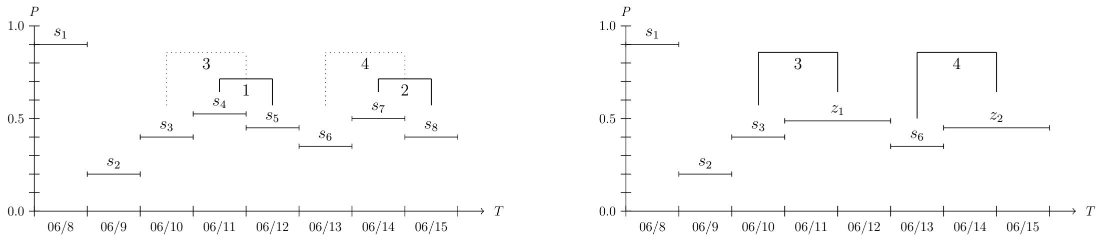
4.4 The Parallel PTA

In this section we present new algorithmic concepts that drastically improve the performance of the PTA in Migros' Teradata environment. First we investigate how to decrease the amount of iterations. Afterwards, we show how we can minimize the duration of the SSE calculations and with that the duration of a single iteration.

4.4.1 Conditions for Current Merging

As we have seen in the previous section, one major overhead in the PTA algorithm is the huge amount of iterations that will downgrade the performance. The solution is the execution of multiple mergings simultaneously in a single iteration.

Example 9 *As already seen before in Figure 4.5b and now highlighted in Figure 4.9a, we see that in the first iteration s_4 and s_5 are merged and in the second iteration we do the same for s_7 and s_8. Since these two tuple-pair candidates do not overlap each other, it seems allowed that these two merging steps could also happen at once in the same iteration, without messing up the original order and changing the result. It even doesn't matter which of the two tuple-pairs are merged first. As shown in Figure 4.9b, the thing that matters is that s_4 and s_5 are merged together before we merge the resulting tuple z_1 with s_3 and that s_7 and s_8 are merged before the merging of z_2 with s_6. So an algorithm needs to be defined that merges $s_4 \oplus s_5$ and $s_7 \oplus s_8$ in the same iteration.*



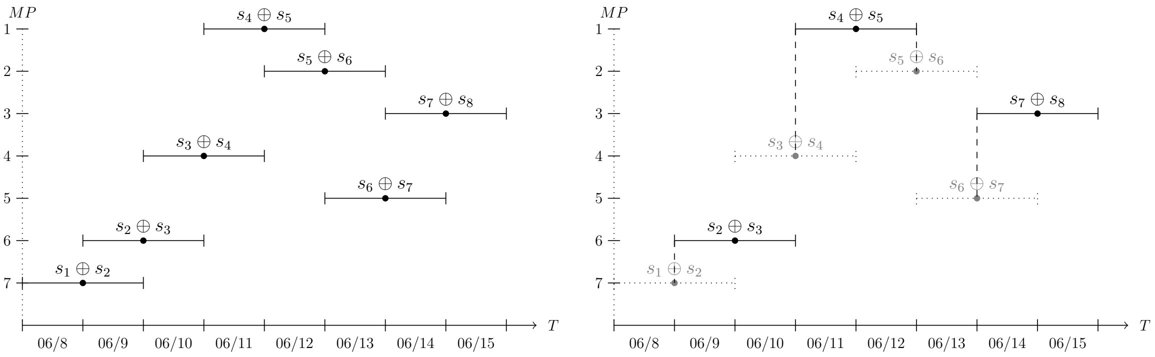
(a) Merging steps 1 and 2

(b) Merging steps 3 and 4

Figure 4.9: Multiple mergings in a single iteration

In order to determine which tuple-pairs should be merged concurrently, one part of the approach is to order the tuple-pairs according to their merging order. We use a new integer attribute that we term merge priority MP that stores the merging order for the merge pairs. As previously described in Subsection 4.2.2 the merging order is determined by the SSE, and if the decision is ambiguous, we order merge-pairs with equal SSE according to their starting time point Ts. The main idea now is to merge the tuple-pairs with a higher priority, given that they do not overlap with some merge-pairs that have already been chosen for actual merges. We call this idea the *Parallel Merge Strategy* and leave it to this natural language definition for now. As we will see later, implementing this idea is not that trivial.

Example 10 *Using the SSE calculations we've made in Figure 4.4 we're able to order the tuple-pair candidates according to their merge-priority MP, as shown in Figure 4.10a. Note that the y-coordinates now only state the merge-order from top to bottom and do not reflect the merge-pairs' probabilities P. E.g. as we already know, s_4 and s_5 are the first two tuples that should be merged and thus the merge-pair $s_4 \oplus s_5$ is visualized as the topmost tuple-pair. Now if we proceed further, the tuple-pair with the next highest MP is $s_5 \oplus s_6$. But since s_5 has already been chosen to be merged with s_4 before, we do not take this tuple-pair as a possible candidate merge-pair. As shown in Figure 4.10b, we appropriately visualize this fact with a dotted line of the merge-pair and a vertically dashed line that shows the overlapping boundary. Inspecting all remaining merge-pairs in the same manner in the order of the MP reveals us the three merge-pair candidates $s_2 \oplus s_3$, $s_4 \oplus s_5$ and $s_7 \oplus s_8$. Comparing this result to the complete merging order in Figure 4.5b shows us that this approach of determining concurrent tuple-pair merges is not sufficient yet. Whereas choosing the pairs $s_4 \oplus s_5$ and $s_7 \oplus s_8$ fits* *the expectations that these tuple-pairs should be merged in the same iteration, the tuple-pair $s_2 \oplus s_3$ shouldn't be chosen. Actually, one of its tuples, s_3, should be merged together with z_1 as shown in Figure 4.5b. Only some merging steps after that, and only if the size bound hasn't been reached yet, z_5 will be merged together with s_2.*



(a) Order tuple-pairs according to their MP

(b) Choosing merge-pair candidates

Figure 4.10: Trying multiple merges at once

The algorithm is not allowed to execute a merging step x of two tuples, where at least one of these tuples would have been merged with a different tuple in a merging step that would actually precede merging step x. In order to prevent that, we need to calculate in advance the SSE's of merging three adjacent tuples together to a merge-triplet. In favor for a better differentiation of the terms, we call the SSE of merge-pairs SSE_2 and the SSE of merge-triplets SSE_3. Only the merge-pairs that have a SSE_2 that is strictly smaller than the smallest SSE_3 of a merge-triplet, are possible candidates for being merged in a single iteration. We label the constraint as well as the value with SSE_{MAX}.

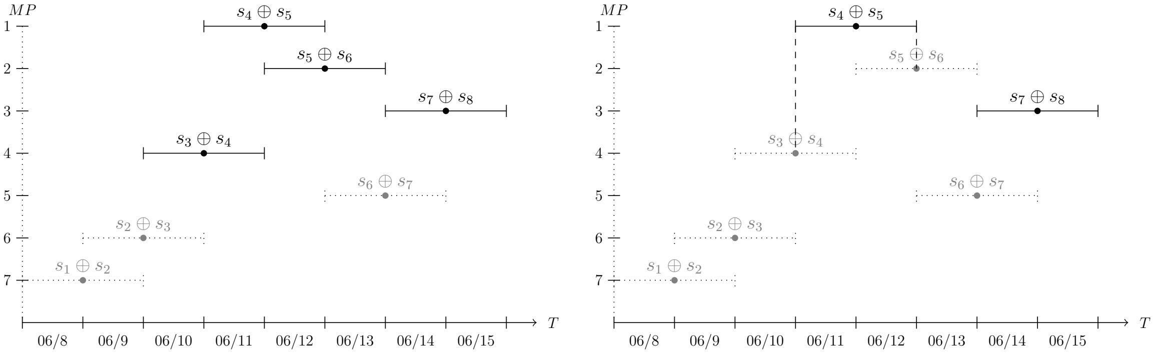
Example 11 *Figure 4.11 shows that the merge of $s_3 \oplus s_4 \oplus s_5$ has among all SSE_3 the smallest value and therefore $SSE_{MAX} = 0.00791667$.*

ShopsIn$_{SSE_3}$

	Cst	*Loc*	*Ts*	*Te*	*P*	*SSE₃*
$s_1 \oplus s_2 \oplus s_3$	Ann	HB	2015-06-08	2015-06-10	0.500000000000	0.260000000000
$s_2 \oplus s_3 \oplus s_4$	Ann	HB	2015-06-09	2015-06-11	0.375000000000	0.053750000000
$s_3 \oplus s_4 \oplus s_5$	Ann	HB	2015-06-10	2015-06-12	0.458333333333	0.007916666667
$s_4 \oplus s_5 \oplus s_6$	Ann	HB	2015-06-11	2015-06-13	0.441666666667	0.015416666667
$s_5 \oplus s_6 \oplus s_7$	Ann	HB	2015-06-12	2015-06-14	0.433333333333	0.011666666667
$s_6 \oplus s_7 \oplus s_8$	Ann	HB	2015-06-13	2015-06-15	0.416666666667	0.011666666667

Figure 4.11: Calculating the SSE_3 of Merge-Triplets and finding SSE_{MAX}

Comparing this value with the SSE_2s of the merge-pairs, which we've already calculated in Figure 4.4, gives us the tuple-pairs $s_3 \oplus s_4$, $s_4 \oplus s_5$, $s_5 \oplus s_6$ and $s_7 \oplus s_8$. All of them have a SSE_2 stricty smaller than SSE_{MAX} and thus fulfill our new SSE_{MAX} constraint, which is also



(a) Applying the SSE_{MAX} constraint (b) Applying the *Parallel Merge Strategy*

Figure 4.12: Combining the SSE_{MAX} constraint with the *Parallel Merge Strategy*

shown in Figure 4.12a. The merge-pairs $s_1 \oplus s_2$, $s_2 \oplus s_3$ and $s_6 \oplus s_7$ are eliminated by the SSE_{MAX} constraint.

If we now reapply the Parallel Merge Strategy from the previous Subsection 4.4.1 we finally get the desired choosing of the merge-pairs $s_4 \oplus s_5$ and $s_7 \oplus s_8$. Note that we do not have to check any overlappings regarding the merge-pairs $s_1 \oplus s_2$, $s_2 \oplus s_3$ and $s_6 \oplus s_7$ since they're already dismissed by the SSE_{MAX} constraint.

4.4.2 The Algorithm of the *Parallel Merge Strategy*

In the interest of precisely defining the logic behind the *Parallel Merge Strategy* we have to specify some mechanisms to make the improved algorithm work. We term the algorithm that implements the *Parallel Merge Strategy* $PTA_{parallel}$. Concerning some explanations a new example needs to be introduced.

Example 12 *In Figure 4.13 we assume to have a set of consecutively overlapping merge-pairs with all having the same grouping attributes. All of these tuple-pairs fulfill the SSE_{MAX} condition and thus all of them are candidates for being possibly merged together. Again, the timestamps of the merge-pairs are shown as horizontal lines. The merge-pair identifier above each merge-pair, e.g. m_1, is enumerated from 1 to 25 accordingly to the starting time points of the merge-pairs on the timeline. We equip the tuple-pairs with an attribute $TimeID$ that mirrors these values. Note that this time, the vertical positioning is only a locally relative representation of the merging priority MP. As we will see later, it is sufficient to only visual this minimal aspect. The vertically higher positioned merge-pairs have a lower MP number than their immediate timely overlapping neighbours that are placed lower and thus have a higher preference to be actually merged. The MPs are written below the respective merge-pairs. Other time specifications, e.g. specific labels on the the timeline, are completely omitted since they're not needed for the understanding of the algorithm.*

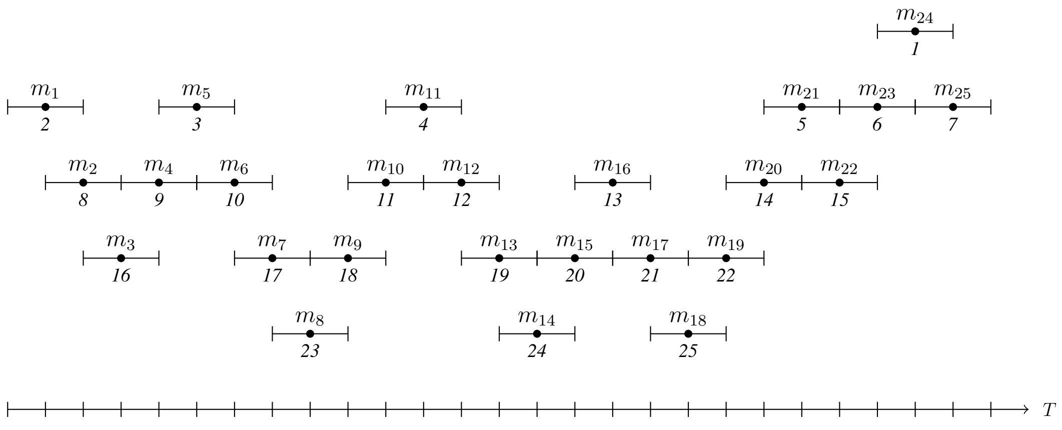


Figure 4.13: New example having 25 consecutively overlapping merge-pairs

First of all, we have to mark the $top = \top$ merge-pairs, which are the ones that have a MP value that is lower than the MP of the immediately following and preceding merge-pairs. Similarly, we also flag $bottom = \perp$ merge-pairs that have a MP that is higher as the MP of the immediately following and preceding merge-pairs. The two sets of tuples are defined as follows:

$$\begin{aligned}
&\boldsymbol{m}^{\top} = \{m \mid m \in \boldsymbol{m} \ \wedge \ \forall m' \in \boldsymbol{m}(\\
&\quad ((m.Ts < m'.Ts \ \wedge \ m.Te > m'.Ts) \vee (m.Ts > m'.Ts \ \wedge \ m.Ts < m'.Te)) \\
&\quad \wedge\, m.MP < m'.MP \\
&)\}
\end{aligned}$$

$$\begin{aligned}
&\boldsymbol{m}^{\perp} = \{m \mid m \in \boldsymbol{m} \ \wedge \ \forall m' \in \boldsymbol{m}(\\
&\quad ((m.Ts < m'.Ts \ \wedge \ m.Te > m'.Ts) \vee (m.Ts > m'.Ts \ \wedge \ m.Ts < m'.Te)) \\
&\quad \wedge\, m.MP > m'.MP \\
&)\}
\end{aligned}$$

The second line in both the definitions $\boldsymbol{m}^{\top}$ and $\boldsymbol{m}^{\perp}$ fetches overlapping tuples for each top or bottom merge-pair respectively. The third line in the definition for $\boldsymbol{m}^{\top}$ makes sure that for a top merge-pair $m^{\top}$ the value of the merge-priority MP is lower than the values for the merge-pairs it overlaps. Vice versa, in the third line of the definition for the bottom merge-pairs $\boldsymbol{m}^{\perp}$, the value for the merge-priority MP of a bottom merge-pair $m^{\top}$ must be higher than in the overlapping merge-pairs.

Example 13 *In Figure 4.14 we show the new state of the merge-pairs that results from the application of the top and bottom merge-pair rules. The merge-pair m_1 for example is a top $m_1^{\top}$ merge-pair. Since m_1 overlaps with the merge-pair m_2 and m_1 has a lower MP number than m_2, it must be a top merge-pair. Similarly merge-pair m_3 is a bottom $m_3^{\perp}$ merge-pair. It overlaps the preceding merge-pair m_2 and the following merge-pair m_4 and since both of them have lower MP numbers than m_3 we define m_3 to be a bottom merge-pair.*

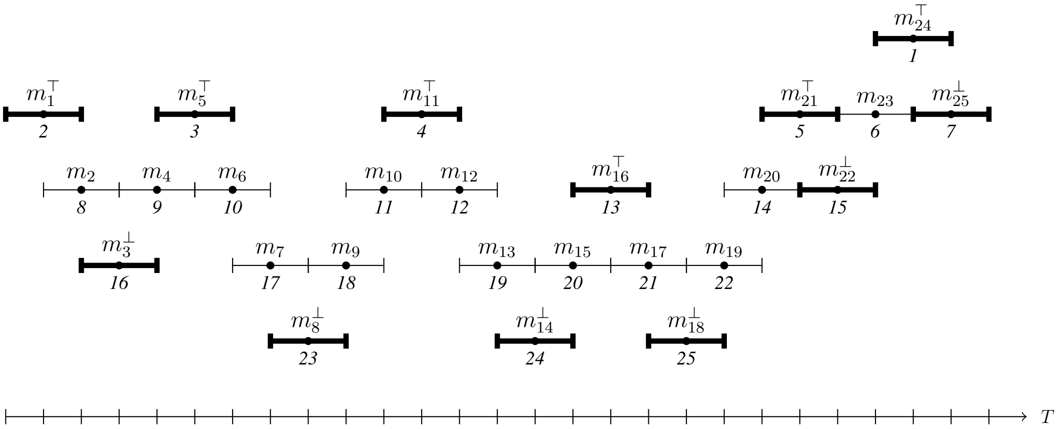


Figure 4.14: Finding top and bottom merge-pairs

For each bottom-flagged merge-pair $m^{\perp}$, we now check if the nearest left and the nearest right top-flagged merge-pair $m^{\top}$ have both odd or both even $TimeID$ attribute values. All merge-pairs that do not fulfill this condition are removed from further considerations.

$$\begin{aligned} m_l^{\top} = \{m^{\top} \mid m^{\top} \in \boldsymbol{m}^{\top} \ \wedge \ & m^{\top}.Ts < m^{\perp}.Ts \\ & \wedge \nexists m^{\top\prime} \in \boldsymbol{m}^{\top}(m^{\top\prime}.Ts < m^{\perp}.Ts \wedge m^{\top\prime}.Ts > m^{\top}.Ts) \\ \} \end{aligned}$$

$$\begin{aligned} m_r^{\top} = \{m^{\top} \mid m^{\top} \in \boldsymbol{m}^{\top} \ \wedge \ & m^{\top}.Ts > m^{\perp}.Ts \\ & \wedge \nexists m^{\top\prime} \in \boldsymbol{m}^{\top}(m^{\top\prime}.Ts > m^{\perp}.Ts \wedge m^{\top\prime}.Ts < m^{\top}.Ts) \\ \} \end{aligned}$$

$$\boldsymbol{m} \leftarrow \boldsymbol{m} - \{m^{\perp} \mid m^{\perp} \in \boldsymbol{m}^{\perp} \mid m_l^{\top}.TimeID \ \% \ 2 \ \neq \ m_r^{\top}.TimeID \ \% \ 2\}$$

The first definition specifies on the first line that the nearest left top merge-pair $m_l^{\top}$ is a top merge-pair $m^{\top}$ on the left of the current bottom merge-pair $m^{\perp}$. The second line says that no other top merge-pair $m^{\top\prime}$ on the left exists that is closer to the current bottom merge-pair $m^{\perp}$ than the nearest left top merge-pair $m_l^{\top}$. In the second definition, we give vice versa the corresponding rules for the nearest top right top merge-pair $m_r^{\top}$. The third definition requires that a bottom merge-pair $m^{\perp}$ that should be removed has a $m_l^{\top}$ and a $m_r^{\top}$ with one of them having a $TimeID$ that is odd and the other having a $TimeID$ that is even.

Example 14 *In Figure 4.15 the bottom merge-pair $m_{14}^{\perp}$ has tuple-pair $m_{11}^{\top}$ as the nearest left top merge-pair. The top merge-pairs $m_1^{\top}$ and $m_5^{\top}$ are far further away from $m_{14}^{\perp}$. Vice versa we define for $m_{14}^{\perp}$ the tuple-pair $m_{16}^{\top}$ as the nearest right top merge-pair. Whereas $m_{11}^{\top}$ has an odd $TimeID$ of* 11, *$m_{16}^{\top}$ has an even $TimeID$ of* 16 *and so therefore we remove $m_{14}^{\perp}$ from $\boldsymbol{m}$. Applying the new rules to all bottom merge pairs $m^{\perp} \in \boldsymbol{m}^{\perp}$ reveals that the bottom merge-pairs with the $TimeIDs$* 18 *and* 22 *have to be removed as well. With the removal of$m_{14}^{\perp}, m_{18}^{\perp} and m_{22}^{\perp}$ we reveal 4 newly formed subgroups of consecutively overlapping merge-pairs. For example the subgroup $\#2$ comprises the merge-pairs m_{15}, m_{16} and m_{17} that consecutively overlap each other.*

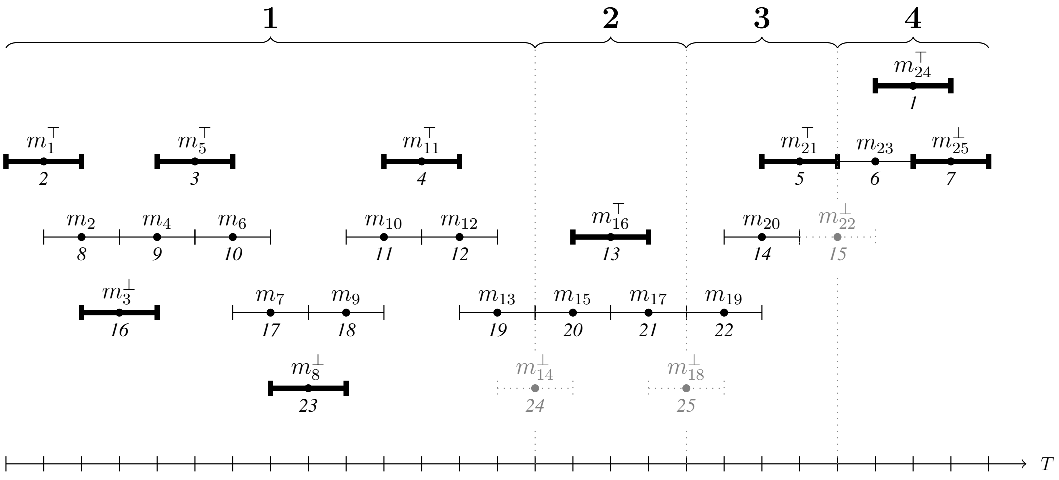


Figure 4.15: The creation of subgroups with consecutively overlapping merge-pairs

Accordingly we assign the subgroup numbers to a new attribute *SubGroupID* for each of the remaining merge-pairs. In each of the newly revealed subgroups, we search the top merge-pair with the lowest *MP* attribute value $m_{sg}^{\top}$ and determine whether its *TimeID* is odd or even. This computation is needed in order to finally define which merge-pairs of a subgroup will be chosen for actual merges. All merge pairs that have a *TimeID* not equally odd or even like in the merge-pair $m_{sg}^{\top}$ have to be removed from further considerations.

$$
\begin{aligned}
m_{sg}^{\top} = \{&m^{\top} \mid m^{\top} \in \boldsymbol{m}^{\top} \\
&\quad \wedge \nexists m^{\top\prime} \in \boldsymbol{m}^{\top}(m^{\top\prime}.\mathit{SubGroupID} = m^{\top}.\mathit{SubGroupID} \wedge m^{\top\prime}.\mathit{MP} < m^{\top}.\mathit{MP}) \\
\}&
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{m} \leftarrow \boldsymbol{m} - \{&m \mid m \in \boldsymbol{m} \mid m.\mathit{SubGroupID} = m_{sg}^{\top}.\mathit{SubGroupID} \\
&\qquad \wedge m.\mathit{TimeID} \,\%\, 2 \neq m_{sg}^{\top}.\mathit{TimeID} \,\%\, 2 \\
\}&
\end{aligned}
$$

The first definition fetches for each subgroup the topmost merge-pair $m_{sg}^{\top}$, i.e. the merge-pair that has the lowest *MP* attribute value in a subgroup. The second definition then removes in each subgroup the merge-pairs m that are not equally odd or even compared to the topmost merge-pair $m_{sg}^{\top}$ of the corresponding subgroup.

Example 15 *In Figure 4.16 the merge-pair m_1 with $MP = 2$ is the one with the smallest MP of subgroup #1. It's TimeID is 1, which is an odd number and so all merge-pairs in subgroup #1 that have an odd TimeID will be kept for concurrent merging. We apply the same mechanism as well to the subgroups #2, #3 and #4 and determine that the merge-pairs $m_1, m_3, m_5, m_7, m_9, m_{11}, m_{13}, m_{16}, m_{19}, m_{21}$ and m_{24} are finally chosen to be actually merged simultaneously.*

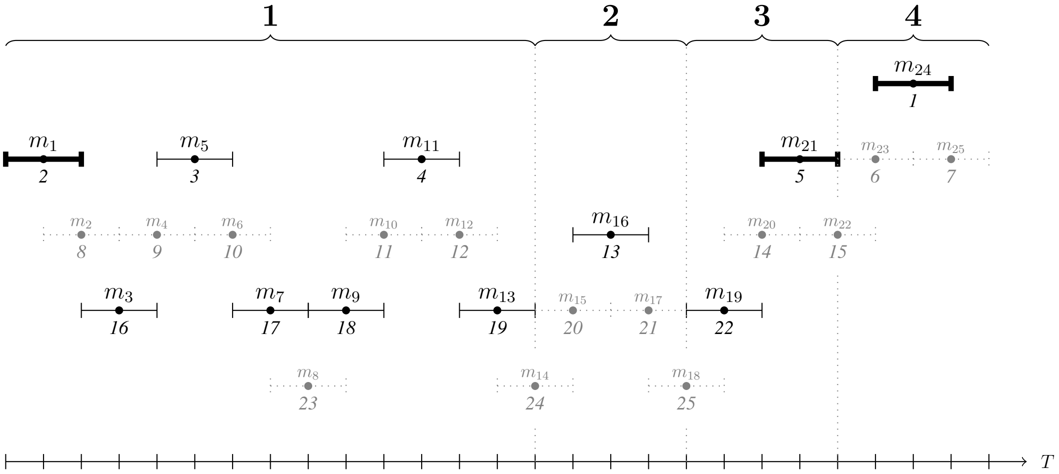


Figure 4.16: Final choosing of the merge-pairs to be merged simultaneously

4.5 Implementation & Performance of the $PTA_{parallel}$

So far we've increased the number of merges per iteration and therefore reduced the total amount of iterations that are needed for the result of the algorithm to reach a desired size bound c. We show how we optimize the calculations for the sum squared error so that a single iteration needs less computation time. As the following Subsection 4.5.1 shows, the achieving improvements are very implementation specific. After we have outlined the main mechanisms of our implementation, we show in Subsection 4.5.2 experimental measurements that show the performance gain of our new approach.

4.5.1 Optimizing the SSE Calculations

In the PTA_{basic} algorithm the SSE of merging two tuples is recalculated from scratch for all merge-pair candidates in each iteration and that means that we perform in every single iteration the self join on all of the tuples of the input relation, as described in Subsection 4.3.2. Regarding the probable relation size in the Migros domain, it is clear that this takes a while. Even worse, in the $PTA_{parallel}$ approach, we also need to calculate the SSE for merge-triplets as well and this is actually a self join on a self join, i.e. has the form $r \bowtie r \bowtie r$. If we'd need to recalculate this as well for every single iteration, we probably loose all the performance improvements we've hopped to gain in the previous sections.

We will make use of 4 additional relations besides the main input relation in order to be able to apply some specific mechanisms to save some computation time. The schemas of these five relations are shown in Figure 4.17a. All relations except the **Result** relation have a new attribute *AdjGrpID* that stores the adjacency-group a tuple belongs to. Remember that we've introduced the concept of *adjacency-groups* in Section 4.2. The **Main** relation stores the input tuples with all the attributes we already know, but additionally, the value of the *AdjGrpID* attribute states the adjacency-group a tuple belongs to. The **MergePair** relation represents all merge-pairs that can be formed with the tuples from the **Main** relation.

Basically we've used such a relation already before in Section 4.2.2 which has been visualized in Figure 4.4. The only difference now is that the $\mathbf{MergePair}$ relation includes also the MP attribute that is needed for our *Parallel Merge Strategy* and so the $\mathbf{MergePair}$ relation looks as in Figure 4.17b. The $\mathbf{MergeTripletSSE}$ relation schema is similar to the relation shown in Figure 4.11 but this time we do not need to keep track of every single merge-triplet and it's corresponding SSE_3. Instead we only store the attributes Cst, Loc, $AdjGrpID$ and append the smallest sum squared error that can be found in each of these corresponding *adjacency-groups*. An example is shown in Figure 4.17c. The $\mathbf{ChosenMergePair}$ relation represents all the tuple-pairs we actually merge during an iteration. This means it contains in each iteration the result that emerges from applying the *Parallel Merge Strategy* to the $\mathbf{MergePair}$ relation. Since the tuples of the $\mathbf{ChosenMergePair}$ relation are actually a subset of the $\mathbf{MergePair}$ relation, the schemas of both relations are the same. The last relation $\mathbf{Result}$ contains, as the name indicates, the result that we return with our $PTA_{parallel}$ algorithm, whereas it's size $|\mathbf{Result}|$ requires to be equal to c.

1. $\mathbf{Main}(Cst, Loc, AdjGrpID, Ts, Te, P, SSE)$,
2. $\mathbf{MergePair}(Cst, Loc, AdjGrpID, Ts, Te, P, SSE, MP)$
3. $\mathbf{MergeTripletSSE}(Cst, Loc, AdjGrpID, SSE)$
4. $\mathbf{ChosenMergePair}(Cst, Loc, AdjGrpID, Ts, Te, P, SSE, MP)$
5. $\mathbf{Result}(Cst, Loc, Ts, Te, P)$

(a) The Schemas of the Implemented Relations

MergePair

Cst	*Loc*	*AdjGrpID*	*Ts*	*Te*	*P*	*SSE*	*MP*
Ann	HB	1	06/08	06/09	0.55	0.245	7
Ann	HB	1	06/09	06/10	0.3	0.02	6
Ann	HB	1	06/10	06/11	0.4625	0.0078125	4
Ann	HB	1	06/11	06/12	0.4875	0.0028125	1
Ann	HB	1	06/12	06/13	0.4	0.005	2
Ann	HB	1	06/13	06/14	0.425	0.01125	5
Ann	HB	1	06/14	06/15	0.45	0.005	3

(b) The $\mathbf{MergePair}$ Relation

MergeTripletSSE

Cst	*Loc*	*AdjGrpID*	*SSE*
Ann	HB	1	0.00791667

(c) The $\mathbf{MergeTripletSSE}$ Relation

Figure 4.17: Implemented Relations

Figure 4.18 describes the basic SQL scripts that we run on these five relations. The scripts can be divided into three main phases. First, we perform some initial steps that have to be computed only once. Then we start the iterative reduction procedure that uses as a loop mechanism a BTEQ while loop that is almost equal to the one we have introduced in Subsection 4.3.2. After having reached the size bound c, we then finally have to put together the final result relation.

1. Initialization
    - Populating the $\textbf{Main}$ relation with the input tuples from the $\textbf{ShopsIn}$ relation. For all tuples that have the same Cst and Loc values, assign a unique $AdjGrpID$ number that marks which tuples belong to the same adjacency-group.
    - For each *adjacency-group* which contains only one tuple, transfer its tuple in the $\textbf{Main}$ relation to the $\textbf{Result}$ relation since they can't be merged anyway.
    - Using a self join on the $\textbf{Main}$ relation, compute all merge-pairs that can be formed from the tuples of the $\textbf{Main}$ relation and store them into the $\textbf{MergePair}$ relation.
    - Using 2 consequent self joins, compute in each Cst, Loc group for all *adjacency-groups* the minimal SSE that results from merging three adjacent tuples from the $\textbf{Main}$ relation and store them into the $\textbf{MergeTripletSSE}$.
2. Iteration
    - The algorithms needed for the *Parallel Merge Strategy* determine the merge-pairs that we can simultaneously merge and store them into the $\textbf{ChosenMergePair}$ relation.
    - In the $\textbf{Main}$ relation delete the source tuples that were merged for the $\textbf{ChosenMergePair}$ relation and copy the chosen merge-pairs from the $\textbf{ChosenMergePair}$ relation into the $\textbf{Main}$ relation.
    - Delete all the merge-pairs from the $\textbf{MergePair}$ relation that are in the same *adjacency-groups* $AdjGrpID$ that also appear in the tuples that are in the $\textbf{ChosenMergePair}$ relation. Recompute only the new possible merge-pairs for these affected groups and therefore only a small subset of the $\textbf{Main}$ relation is actually joined together.
    - Similarly delete all the tuples from the $\textbf{MergeTripletSSE}$ relation that are in the same *adjacency-groups* $AdjGrpID$ that also appear in the tuples that reside in the $\textbf{ChosenMergePair}$ relation. The recalculation of the smallest SSE for these affected groups needs as well only a small subset of the tuples from the $\textbf{Main}$ relation to be part of the self join on a self join.
3. Finalize
    - If the size bound c has been reached and therefore no possible merge-pairs reside in the $\textbf{MergePair}$ relation, we can finally transfer all the tuples from the $\textbf{Main}$ relation to the $\textbf{Result}$ relation.

Figure 4.18: Implementation

4.5.2 Measurements of the Improved Approach

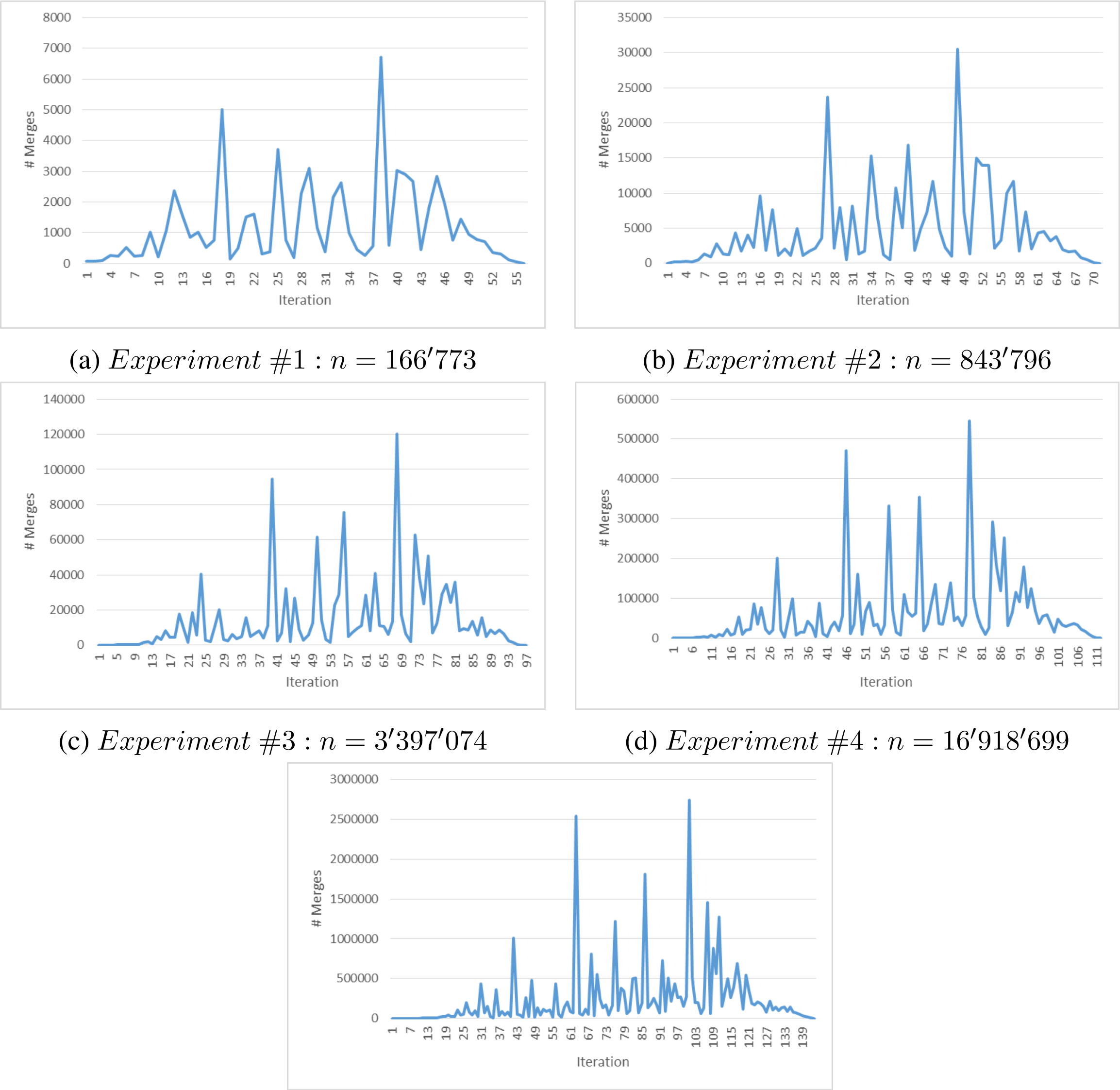
Running the new mechanisms implemented in the $PTA_{parallel}$ algorithm shows us drastic performance improvements. Since the algorithm is able to finish the computations in a reasonable time, we've measured the performance on five differently sized data sets. The results of the five experimental measurements $\#1, \#2, \#3, \#4$ and $\#5$ are shown in Figure 4.19. Let n and c be as before. I is again representing the amount of Iterations the algorithm has to process until the size bound c has been reached. The duration T is given in minutes and seconds.

	n	c	$n - c$	I	T
$\#1$	$166'773$	$99'266$	$67'507$	56	$2:46$
$\#2$	$843'796$	$506'334$	$337'462$	71	$3:03$
$\#3$	$3'397'074$	$2'047'580$	$1'349'494$	97	$4:49$
$\#4$	$16'918'699$	$10'179'069$	$6'739'630$	112	$12:01$
$\#5$	$84'531'997$	$50'887'420$	$33'644'577$	143	$21:35$

Figure 4.19: Performance of the $PTA_{parallel}$ algorithm

The algorithm shows impressive results and even the test run $\#5$ with the largest relation size $n = 84'531'997$ performs the $n - c = 33'644'577$ merging steps enormously quickly and reaches the final size $c = 50'887'420$ in about 21 minutes and 35 seconds. Having a look at the number of iterations $I = 143$ that were needed, shows us that we have decreased the number of iterations massively and that we have really huge numbers of concurrent merging steps per iteration.

The graphics in Figure 4.20 show the number of concurrent mergings that happen in each iteration. Surprisingly the shape of the curve stays approximately the same for all five experiments no matter how big the input relation is. Whereas the number of simultaneous merging steps is rather slowly increasing in the beginning, we get more of them after about a forth of the iterations. In the end, the amount of concurrent merge-steps is declining again. In each experiment we have two very prominent peaks e.g. in experiment $\#1$ in the iterations 18 and 38 we have in each 5003 and 6701 concurrent mergings respectively, whereas the iteration in the second peak performs on its own approximately 10% of all the merging steps in this test-run. Again, this last observation can be made in all our experiments, e.g. experiment $\#5$ has in iteration 101 nearly 3 million concurrent merging steps which is again close to 10% of the overall amount of mergings performed in this experiment. So the *Parallel Merge Strategy* is obviously very effective and a huge improvement compared to a sequential step by step merging procedure. Instead of having only one merge per iteration, as it is in the PTA_{basic} algorithm, we have multiple iterations per iteration in the $PTA_{parallel}$. For an input relation that has a size of $84'531'997$ tuples, the duration that the $PTA_{parallel}$ algorithm needs to complete is with approximately 21 minutes massively outperforming the PTA_{basic} algorithm that needs approximately 21 years. Important to mention is that the result of both algorithms is the same, but we do not prove this in this paper.



(a) $Experiment\ \#1 : n = 166'773$

(b) $Experiment\ \#2 : n = 843'796$

(c) $Experiment\ \#3 : n = 3'397'074$

(d) $Experiment\ \#4 : n = 16'918'699$

(e) $Experiment\ \#5 : n = 84'531'997$

Figure 4.20: Comparison of the Amount of Mergings per Iteration

5 Use Case Reference Query

In this chapter we will summarize the concepts that have been presented in this paper so far. We will do that by applying them for a use case for which Migros still searches approaches that work better than the ones they currently have. Migros is trying to improve the recommendation of complementary articles to specific articles e.g. when a customer is examining an article on their website and Migros would like to show additional articles that somehow fit to the present one. Until now the article recommendation is not based on a customer basis, instead it is based on the collected data of all purchases. Also the articles that are recommended tend to be improper since the finding of a meaningful solution is not that trivial as one might think. This chapter does not present a final best solution, instead the goal is to present an approach of how temporal probabilistic relations and the PTA can be used to try solving such a problem.

5.1 Preparing Relations for the Use Case

5.1.1 Deriving the Buys Relation and Calculating Probabilities

We will use the $\mathbf{TransactionArticle}$ relation that we've introduced in Section 2.2 as the source relation for the targeted relation that we need for the current use case. The $\mathbf{TransactionArticle}$ with two sample tuples is shown in Figure 5.1a and we derive the new $\mathbf{Buys}$ relation from it as shown in Figure 5.1b. We define $\boldsymbol{tra}$ to be an abbreviation for $\mathbf{TransactionArticle}$ and $\boldsymbol{b}$ be an abbreviation for $\mathbf{Buys}$. The relational algebra expression for deriving the $\mathbf{Buys}$ relation from the $\mathbf{TransactionArticle}$ relation looks as follows:

$$\boldsymbol{b} \leftarrow \rho_{Cst,Art,DateID}(\pi_{CustomerID,ArticleID,DateID}(\boldsymbol{tra}))$$

TransactionArticle

	LocationID	*DateID*	*TimeID*	*ArticleID*	*CustomerID*	*Sales*	...
...	...	...	...	...	...	...	...
tra_1	HB	06/07	160135	Salad	Ann	1.45	...
tra_2	Oerlikon	06/08	124803	Salad	Ann	1.45	...
...	...	...	...	...	...	...	...

(a) $\mathbf{TransactionArticle}$ Relation

Buys

	Cst	*Art*	*T*
...	...	...	...
b_1	Ann	Salad	06/07
b_2	Ann	Salad	06/08
...	...	...	...

(b) The $\mathbf{Buys}$ Relation

Figure 5.1: Deriving the $\mathbf{Buys}$ Relation

We reuse the attributes $CustomerID$, $ArticleID$ and $DateID$ to store tuples that represent facts about which customers Cst buy what articles Art at a certain day T. As already mentioned in the beginning of this paper, we will implicitly replace meaningless ID numbers with some exemplary values. The exemplary tuples b_1 and b_2 represent the fact that Ann buys Salad on the 7th as well as on the 8th August.

Different computation methods are possible to calculate the probabilities for the tuples of this relation. We will keep it rather puristic here since we'd like to primarily focus on the reference query in this chapter. For all tuples $b_i \in \boldsymbol{b}$ we do a simple division operation to calculate the probability P. First we count the number of tuples that have the same values given the grouping attributes $\boldsymbol{A} = (Cst, Art)$ as well as the same YW attribute and use it as the numerator. We've implicitly omitted theF attribute in the relations in Figure 5.1 but the computation method can be directly adopted from Section 3.2. Secondly we get the denominator by counting the number of tuples that have the same value in the Cst and YW attribute, which actually means we count the total amount of articles a customer buys within the same week. The calculation formula to get the probability P for every $b_i \in \boldsymbol{b}$ looks as follows:

$$b_i.P = \frac{|\{b \mid b \in \boldsymbol{b} \land b.\boldsymbol{A} = b_i.\boldsymbol{A} \land b.YW = b_i.YW\}|}{|\{b \mid b \in \boldsymbol{b} \land b.Cst = b_i.Cst \land b.YW = b_i.YW\}|}$$

Figure 5.2a shows the results when we apply the calculation formula on the tuples b_1 and b_2 and assuming that Ann has bought 16 additional articles and none of them is Salad. We get a probability P of $2/18 = 0.11$ for both tuples b_1 and b_2 due to the two Salads Ann has bought divided with a total of 18 articles that Ann has purchased. The tuples b_1 and b_2 represent the fact that the probabilities of Ann having bought Salad on the 8th and the 9th June respectively is 0.11 each. These probabilities already indicate a bit that the final probability values that emerge from the later defined reference query might become rather small.

$\mathbf{Buys}^P$

	Cst	Art	T	P
...	...	...	...	...
b_1	Ann	Salad	06/07	0.11
b_2	Ann	Salad	06/08	0.11
...	...	...	...	...

(a) Calculating Probabilities P

$\mathbf{Buys}^{TP}$

	Cst	Art	Ts	Te	P
...	...	...	...	...	...
$b_1 \oplus b_2$	Ann	Salad	06/07	06/08	0.11
...	...	...	...	...	...

(b) Applying the PTA

Figure 5.2: Further Transformations of the $\mathbf{Buys}$ Relation

5.1.2 Applying the PTA on the Buys Relation

As next we will apply the PTA algorithm on the $\mathbf{Buys}$ relation. Figure 5.2b shows the merging of the tuples b_1 and b_2 whereas the new tuple's value for Ts equals to the attribute value T of b_1 from the $\mathbf{Buys}^P$ relation and Te equals to the attribute value T of b_2. Figure 5.3a shows the temporal probabilistic table $\mathbf{Buys}^{TP}$ that contains some data we haven't shown before but which we made up now for exemplary purposes that are needed in the rest of this chapter. The two merged tuples from before in Figure 5.2b are now represented by the tuple b_5 of the $\mathbf{Buys}^{TP}$ relation. Additionally as already indicated by Figure 5.2b we now have intervals in the $\mathbf{Buys}^{TP}$ relation instead of the time points in the $\mathbf{Buys}^P$. E.g. tuple b_5 means that in each time point in the interval $[06/07, 06/08]$ there is a probability P of 0.11 that Ann buys a Salad.

$\mathbf{Buys}^{TP}$

	Cst	Art	Ts	Te	P
b_1	Ann	IceTea	06/02	06/04	0.29
b_2	Ann	Evian	06/08	06/13	0.33
b_3	Ann	Mozarella	06/08	06/10	0.277
b_4	Ann	Tomato	06/09	06/12	0.25
b_5	Ann	Salad	06/07	06/08	0.11
b_6	Ben	Chocolate	06/10	06/11	0.275

(a) The $Buys^{TP}$ relation

$\mathbf{TopLevelArticles}^{TP}$

	Cst	Art	Ts	Te	P
a_1	Ann	Evian	06/08	06/13	0.33

(b) The $\mathbf{TopLevelArticles}^{TP}$ Relation

Figure 5.3: $\mathbf{Buys}$ Relation after PTA

5.1.3 The $\mathrm{TopLevelArticles}^{TP}$ relation

The temporal probabilistic relation $\mathbf{TopLevelArticles}^{TP}$ is created out of a selection of the $\mathbf{Buys}^{TP}$ relation. The schema and reading of the relation is the same as in the $\mathbf{Buys}$ relation, but we claim that the top level articles that this relation contains, are those that have probabilities higher than 0.3. Given $\boldsymbol{a} = \mathbf{TopLevelArticles}^{TP}$ and $\boldsymbol{b} = \mathbf{Buys}^{TP}$ the corresponding relational algebra expression looks as follows:

$$\boldsymbol{a} \leftarrow \sigma_{p\,>\,0.3}(\boldsymbol{b})$$

From the $\mathbf{Buys}^{TP}$ relation, the only tuple that fulfills this constraint is b_2 since $b_2.P = 0.33$ and therefore $b_2.P > 0.3$ holds true. For that reason the $\mathbf{TopLevelArticles}^{TP}$ relation looks as shown in Figure 5.3b. In both of the relations $\mathbf{TopLevelArticles}^{TP}$ and $\mathbf{Buys}^{TP}$ we have tuples with rather low probability values and even the tuples b_2 and a_1 respectively with the highest values of all tuples don't seem quite high.

5.2 The Reference Query

The use case can be expressed in short with the following sentence: For a customer, which are the article combinations he tends to buy, given that they don't involve a top level article? The relational algebra expression to answer this question looks as follows:

$$\boldsymbol{z} \leftarrow (\rho_{\boldsymbol{b_1}}(\boldsymbol{b}) \bowtie_{\theta_1}^{TP} \rho_{\boldsymbol{b_2}}(\boldsymbol{b})) \;-^{TP}\; ((\boldsymbol{b} \bowtie_{\theta_2}^{TP} \boldsymbol{a}) \cup (\boldsymbol{a} \bowtie_{\theta_2}^{TP} \boldsymbol{b}))$$

$$\begin{aligned} \theta_1 &\equiv b_1.Cst = b_2.Cst \wedge b_1.Art <> b_2.Art \\ \theta_2 &\equiv b.Cst = a.Cst \wedge b.Art <> a.Art \end{aligned}$$

The temporal probabilistic self join $(\rho_{\boldsymbol{b_1}}(\boldsymbol{b}) \bowtie_{\theta_1}^{TP} \rho_{\boldsymbol{b_2}}(\boldsymbol{b}))$ of the $\mathbf{Buys}$ relation represents all article combinations that customers have bought. The expression $b_1.Art <> b_2.Art$ makes sure that we only join different articles and that we don't show combinations of the same article with itself, which wouldn't make any sense at all. The intermediary result $\boldsymbol{x}$ of this self join is shown in Figure 5.4a. The calculations to get the correct intervals and probabilities when using temporal probabilistic operators exceeds the scope of this paper.

$\boldsymbol{x}$

	Cst	Art_1	Art_2	Ts	Te	P
x_1	Ann	Evian	Mozarella	06/08	06/10	0.09141
x_2	Ann	Evian	Salad	06/08	06/08	0.03630
x_3	Ann	Evian	Tomato	06/09	06/12	0.08250
x_4	Ann	Mozarella	Evian	06/08	06/10	0.09141
x_5	Ann	Mozarella	Salad	06/08	06/08	0.03047
x_6	Ann	Mozarella	Tomato	06/09	06/10	0.06925
x_7	Ann	Salad	Mozarella	06/08	06/08	0.03047
x_8	Ann	Salad	Evian	06/08	06/08	0.03630
x_9	Ann	Tomato	Mozarella	06/09	06/10	0.06925
x_{10}	Ann	Tomato	Evian	06/09	06/12	0.08250

(a) $\boldsymbol{x} \leftarrow (\rho_{\boldsymbol{b_1}}(\boldsymbol{b}) \bowtie^{TP}_{\theta_1} \rho_{\boldsymbol{b_2}}(\boldsymbol{b}))$

$\boldsymbol{y}$

	Cst	Art_1	Art_2	Ts	Te	P
y_1	Ann	Evian	Mozarella	06/08	06/10	0.09141
y_2	Ann	Mozarella	Evian	06/08	06/10	0.09141
y_3	Ann	Evian	Salad	06/08	06/08	0.03630
y_4	Ann	Salad	Evian	06/08	06/08	0.03630
y_5	Ann	Evian	Tomato	06/09	06/12	0.08250
y_6	Ann	Tomato	Evian	06/09	06/12	0.08250

(b) $\boldsymbol{y} \leftarrow ((\boldsymbol{b} \bowtie^{TP}_{\theta_2} \boldsymbol{a}) \cup (\boldsymbol{a} \bowtie^{TP}_{\theta_2} \boldsymbol{b}))$

Figure 5.4: The intermediary results $\boldsymbol{x}$ and $\boldsymbol{y}$

Tuple x_1 states that Ann buys both, Evian and Mozarella in a day from the 8th June to the 10 June with a probability of 0.09141. When we compare it with tuple x_6 which gives us the probability for the combination Mozarella and Tomato, we see that the probability for x_1 is higher than x_6. Our intuition about the reference query however is that this fact changes when we compute the rest of the reference query. Our goal is to lower the impact of tuples that comprise top level articles and we hope that in the end the combination of Mozarella and Tomato shows us a higher probability than the combination of Evian and Mozarella since Evian is a top level article. Take note that we have some duplicate information by these tuples, e.g. the tuple b_4 states the same as b_1. Only the article names are swapped, which does not alter the meaning in the context of product combinations here. Of course there are ways to modify the reference query so that we don't get tuples with equal meaning, but the query would get much less readable.

The two temporal probabilistic joins that are united together $((\boldsymbol{b} \bowtie^{TP}_{\theta_2} \boldsymbol{a}) \cup (\boldsymbol{a} \bowtie^{TP}_{\theta_2} \boldsymbol{b}))$ represent the product combinations of the top level articles with the other articles. The union with the join of the mirrored input relations $\boldsymbol{a}$ and $\boldsymbol{b}$ is needed so that also both representations of the article combinations are present here as well, which is later necessary for the set difference operation. Again, the expression $b.Art <> a.Art$ in the θ_2 condition makes sure that we don't have combinations of the same article. In Figure 5.4b we show the intermediary result $\boldsymbol{y}$ for the current expression. The tuples from the intermediary result relation $\boldsymbol{y}$ can be read in the same way as the tuples from relation $\boldsymbol{x}$. E.g. the tuple y_1 states that Ann buys the combination of the products Evian and Mozarella in each day from the 8th June to the 10 June where the probability is 0.09141 for every day. Both of the intermediary result relations $\boldsymbol{x}$ and $\boldsymbol{y}$ have significantly lower probability values for their tuples compared to the ones in the input relations $\mathbf{Buys}^{TP}$ and $\mathbf{TopLevelArticles}^{TP}$.

Finally we perform the temporal probabilistic set difference $\boldsymbol{x} - \boldsymbol{y}$ of our reference query in order to lower the probability of product combination tuples that comprise a top level product in one of their article attributes, either in Art_1 or Art_2. The set difference operation results into the final relation $\boldsymbol{z}$ which is shown in Figure 5.5a. For a better visualization of which product combinations are most likely to be bought when we have lowered the probability of combinations that comprise a top level article, the result relation $\boldsymbol{z}$ has been ordered on the attribute P in a descending manner.

$\boldsymbol{z}$

	Cst	Art_1	Art_2	Ts	Te	P
z_1	Ann	Mozarella	Tomato	$06/09$	$06/10$	0.06925
z_2	Ann	Tomato	Mozarella	$06/09$	$06/10$	0.06925
z_3	Ann	Evian	Mozarella	$06/08$	$06/10$	0.0612447
z_4	Ann	Mozarella	Evian	$06/08$	$06/10$	0.0612447
z_5	Ann	Evian	Tomato	$06/09$	$06/12$	0.055275
z_6	Ann	Tomato	Evian	$06/09$	$06/12$	0.055275
z_7	Ann	Mozarella	Salad	$06/08$	$06/08$	0.03047
z_8	Ann	Salad	Mozarella	$06/08$	$06/08$	0.03047
z_9	Ann	Salad	Evian	$06/08$	$06/08$	0.024321
z_{10}	Ann	Evian	Salad	$06/08$	$06/08$	0.024321

(a) $\boldsymbol{z} \leftarrow \boldsymbol{x} - \boldsymbol{y}$

$\boldsymbol{z'}$

	Cst	Art_1	Art_2	Ts	Te	P
z'_1	Ann	Mozarella	Tomato	$06/09$	$06/10$	0.06925
z'_2	Ann	Evian	Mozarella	$06/08$	$06/10$	0.0612447
z'_3	Ann	Evian	Tomato	$06/09$	$06/12$	0.055275
z'_4	Ann	Mozarella	Salad	$06/08$	$06/08$	0.03047
z'_5	Ann	Salad	Evian	$06/08$	$06/08$	0.024321

(b) Cleaning up $\boldsymbol{z}$

Figure 5.5: Final Result Relation $\boldsymbol{z}$

Again we have at any one time two tuples here that are not real duplicates but actually state the same info. E.g. tuple z_1 and tuple z_2 both state that the probability for Ann buying the product combination of Mozarella and Tomato is 0.06925 in each day of the interval $[06/09, 06/10]$. For a better visualization and analysis of the result relation z we remove such duplicates and represent it accordingly as relation $\boldsymbol{z'}$ in Figure 5.5. As we've already indicated and estimated in the computation steps before, the probabilities have all become very low. Even though they're all quite low, we can draw some conclusions from the relative difference between the probability values. The article combination z'_1 has the highest probability of all the tuples of the current relation and therefore states that of all article combinations that Ann buys, the combination of Mozarella and Tomato is the most probable one with a probability of 0.06925 for the 9th and the 10th of June. At least this tuple matches our intuition since we would have expected from the query that the combination of Mozarella and Tomato has a higher probability than Evian and Mozarella. The only other tuple that does not contain the top level article Evian is the result tuple z'_4 which comprises the combination of Mozarella and Salad. Unfortunately this combination is less likely as the ones stated by z'_2 and z'_3 which contain Evian as one of their articles and thus the reference query does not fully mirror our intuition of the final result.

6 Conclusion

In this paper we've introduced the most important relations in Migros' database system which comprise the data that is collected from the purchases that customers make. We have shown different methods to calculate probabilistic versions of these relations and then we've used a newly developed PTA approach to transform the probabilistic relations into interval timestamped probabilistic relations. In the end we have applied the introduced concepts of this paper to show a way to solve a real life use case that Migros has to deal with.

Bibliography

[1] Juozas Gordevicius, Johann Gamper, and Michael H. Böhlen. Parsimonious temporal aggregation. In EDBT, 2009.

[2] A. Dignös, M. H. Böhlen, and J. Gamper. Temporal alignment. In Proceedings of the 2012 international conference on Management of Data, SIGMOD '12, pages 433–444. ACM, 2012.