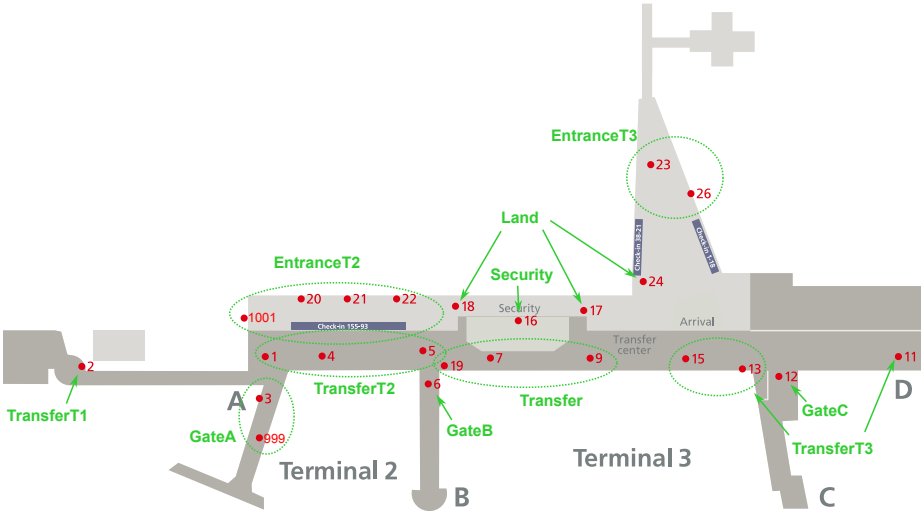
A Data Warehouse Solution for Flow Analysis Utilising Sequential Pattern Mining

Department of Computer Science @ Aalborg University



Master's Thesis in Database Technology by
Simon Nicholas Moesby Tinggaard
Rune Leth Wejdling

Department of Computer Science **Aalborg University**

Title:
A Data Warehouse Solution for Flow Analysis Utilising Sequential Pattern Mining

Project period:
DAT6:
1st of February to 12th of June, 2009

Project Group:
Computer Science, d621a

Members:
Simon Nicholas M. Tinggaard
Rune L. Wejdling

Supervisor:
Torben Bach Pedersen

Copies:
5

Pages in Report:
57

Pages in Appendix:
3

Pages in total:
60

Abstract:

Tracking of visitors in indoor spaces has many applications, specifically when administrating large public areas, like airports and train stations. In this work we present a data warehouse solution, designed to store and facilitate analysis of large amounts of tracking data. The solution is designed specifically for a data set provided by Blip Systems A/S, tracking visitors in Copenhagen airport. We propose to utilise sequential pattern mining to precompute flow information and present three different approaches, to incorporating frequent patterns for flow analysis in the data warehouse. A prototype system is implemented to provide grounds for an extensive experimental study. The study discloses the strengths and weaknesses of the different approaches, and indicates in which cases the different approaches are applicable.

Preface

This master's thesis was written in the spring of 2009 by Simon Nicholas M. Tinggaard and Rune L. Wejdling, with guidance and supervision of Torben Bach Pedersen. The thesis is the result of our work at the Database and Programming Technologies (DPT) research unit at Aalborg University (AAU). The work in this thesis is based on a data set produced by Blip Systems A/S, a company located in North Jutland, Denmark. Blip Systems A/S is a software development firm specialised in Bluetooth marketing and tracking solutions.

This thesis focusses on developing a data warehouse solution facilitating flow analysis on visitor movement in Copenhagen airport. The data warehouse solution utilises some ideas and techniques presented in our pre-master thesis [9]. Section 3.2 describing the bouncing problem in the Blip Systems data set is a revised edition inspired by the work in our pre-master thesis. The data warehouse schema presented in Section 4 extends a simplified version of the schema presented in our pre-master thesis and the algorithms presented in Section 6.3.1 are improved versions of the algorithms introduced in our pre-master thesis.

We would like to thank our supervisor Torben Bach Pedersen for his support and Lars Tørholm Christensen from Blip Systems A/S for his cooperation, during the preparation of this thesis.

Contents

1 Introduction

In recent years, there have been advancements in the area of tracking and monitoring moving objects, using spatiotemporal data collection. Research within the area includes, tracking cars moving around in cities to find shared patterns for carpooling [6], identifying frequently used (hot) routes in road networks [14], tracking of people moving in indoor spaces [2, 3, 5, 11, 15], and tracking products in a supply chain [4, 7, 8]. Different technologies are used to track the moving objects, depending on the environment in which the tracking system is deployed. Global Positioning System (GPS) is mostly used if the objects are tracked in outdoor environments, where the position of the object can be reported at anytime and anywhere. When tracking objects in indoor spaces, technologies such as RFID [2, 7, 15, 21] and Bluetooth [5, 22], are more applicable. These technologies rely on a series of strategically placed sensor points (e.g., RFID readers or Bluetooth access points), that track the position of objects within the range of the sensor points. This makes these technologies fit for use in environments where the signal to the GPS satellites may be interrupted, e.g., indoor spaces.

The spatiotemporal data collected by the tracking and monitoring systems has many different applications including, real-time tracking systems that assist in, e.g., problem detection and prevention, and location based and context aware services. Among these applications the collected data is also being used for historical analysis, e.g., to provide statistics for staff planning, assessing market trends, and analysing the shopping habits of customers. The amount of applications are many.

The advancement in the area of tracking systems, has lead to research in the challenges of storing and analysing the huge amount of spatiotemporal data generated by these tracking systems. The solutions range from highly specialised storage models [7, 15] to solutions based on a relational or multidimensional databases [4, 8]. One of the challenges in analysing the spatiotemporal data, and the main subject of the current work, is to find patterns in the way the tracked objects behave and identify common trends [1, 16, 17, 19].

Application The task of planning and maintaining the internal layout of large indoor spaces like shopping malls, airports, train stations, and other public areas, is becoming increasingly complex. The ability to track how visitors move through and around inside a building, is an essential part of the planning process. The tracking information can be used to find information on areas of congestion, which are used in, e.g., the planning of internal layout changes, the pricing of advertising spaces, and other applications. Apart from the benefits of historical analysis, a visitor tracking system can provide useful information for real-time monitoring systems and location based services.

The current work is based on a application concerning tracking of visitors in an airport, more specifically Copenhagen airport (CPH). Blip Systems A/S [22] (Blip) in cooperation with Copenhagen Airports A/S [23], collects spatiotemporal tracking data on visitors at CPH, using a Bluetooth tracking system. The tracking systems consists of a number of Bluetooth access points placed at strategic locations in CPH, tracking Bluetooth devices moving around the airport. This information can be used to track the movement of the visitors carrying the devices. The system currently collects close to 750,000 tracking

records per day. Given a steady increasing number of Bluetooth enabled devices tracked by the system, and a request for increased accuracy, by the airport administration, this number is likely to grow. In order to get some useful movement information from this amount of data and make it accessible to the airport administration, an efficient storage and analysis solution is needed.

In previous work [9] we proposed a data warehouse solution to store and facilitate a statistical analysis of the tracking data from the airport. In the current work we develop an extended data warehouse solution, for flow analysis, by exploiting the fact that every visit, to the airport, can be represented as a sequence of locations, which represent the route the visitor has travelled through the airport. When working with flow analysis of the visitors, these sequences can be utilised to find patterns in the movement of visitors. These patterns can in turn be used to answer the following two questions raised by CPH's administration: *How many visitors traverse a given path and where are they seen before and after?* and *How do the visitors diverge from a given path?*, where a path is a sequence of one or more selected locations, e.g., "*Security Check* and *Gate A*". Answering these questions could assist CPH in making decisions like where to put different advertisements, what rent to charge for the stores in the different areas, and which areas tend to be congested.

The previous questions, specifically the first, can be answered by implementing modified versions of the FlowCube [7] or S-OLAP [15] warehouse systems. Both systems are designed to provide flow analysis in large amounts of tracking data. Our current work differs from these systems by providing a design that is simple to implement in an RDBMS, and therefore simpler to integrate in an existing data warehouse solution. This evades the added complexity of implementing a specialised data warehouse to provide flow analysis, and enables the use of existing online analytical processing (OLAP) tools.

Research in the area of identifying common trends includes mining association rules [1, 19]. Association rules is similar to the first of the above mentioned question, e.g., the rule stating that visitors traversing the path "*Entrance* and *Security Check*" are with a certain probability seen later at "*Gate A*", is similar to answering that a certain percentage of the visitors traversing the path are later seen at "*Gate A*". Our current work differs from mining association rules by answering the second question, as presented above, though related by the fact that both rely on frequent pattern mining [16, 17].

Contributions In the current work we present a data warehouse solution, supporting flow analysis, for the tracking data collected by Blip at CPH. The solution is designed to be implemented in an RDBMS and support the use of existing OLAP tools. We present three different approaches to store and index frequent patterns found in the tracking data. An ETL system is designed to load the tracking data into the data warehouse, while cleansing information skewing the flow analysis results from the tracking data. We implement a prototype flow count query processor able to answer the flow analysis questions presented above, using each of the three different indexing approaches. An experimental study is performed with the implementation to evaluate the performance and effectiveness of the three approaches. We show that each approach is applicable under different scenarios, depending on the requirements for recall, execution time, and space usage.

2 Problem Definition

The goal of the current work is to develop a data warehouse solution that assists answering the previously introduced questions. To answer those questions we must first be able to answer the following (simpler) question: " *Where have the visitors been before, in between, and after passing a given series of locations?*"

Assuming we have a set L of all possible locations denoted as $\{l_1, ..., l_n\}$, we introduce the following definitions:

Definition 1. A **sequence** α is an ordered list of locations from L denoted by $\langle l_1 l_2 ... l_n \rangle$, where l_i is called an **element** of the sequence α.

Definition 2. A **sequence database** S is a set of tuples $\{(id, s)\}$, where id is a sequence id and s is a sequence.

Definition 3. A sequence α is a **subsequence** of a sequence β if $|\alpha| \leq |\beta|$ and all elements in α exist in β in the same sequential order, e.g., given the sequence $\gamma = \langle l_1 l_2 l_3 \rangle$, then $\langle l_1 l_2 l_3 \rangle$, $\langle l_1 l_2 l_3 \rangle$, $\langle l_1 l_2 \rangle$, $\langle l_1 l_3 \rangle$, $\langle l_2 l_3 \rangle$, and $\langle l_2 \rangle$ are some of the subsequences of γ, but, e.g., $\langle l_2 l_1 \rangle$ is not a subsequence of γ.

Definition 4. A sequence β is a **supersequence** of α if α is a subsequence of β.

Definition 5. A sequence β is said to **contain** a sequence α if α is a subsequence of β. The **support** of α in a sequence database S is the number of sequences in S that contain α, denoted as $support(\alpha)$.

Definition 6. A sequence α is a (frequent) **pattern** in a sequence database S if α is contained in at least $min_{support}$ sequences in S, i.e., $support(\alpha) \geq min_{support}$, where $min_{support}$ is a positive integer.

Since analysts are mostly interested in statistically significant information, e.g., things that happen frequently, we propose to use frequent patterns to perform flow analysis on the Blip data set described in Section 3.

To be able to conduct flow analysis on the Blip data set, we introduce the notion of a **visit**. A visit describes a visitor spending an amount of time in the airport. Visitors will typically spend from a few hours to a day in the airport, depending on the type of visitor. A visitor arriving with a plane and entering the country will typically only spend the time it takes to get through customs and fetch her luggage. Whereas a visitor in transit can spend from a few hours to a day waiting for the next flight.

A visitor can visit the airport multiple times, and thereby generate multiple visits. The notion of a visit is described in the following definition:

Definition 7. A **visit** identifies the collection of tracking records produced by a device (a.k.a. visitor) at the airport, describing when and where a device has been tracked, from the visit is created, until the visit is closed. A visit is created when no open visits exist for the tracked device, and it is closed when the time since the device was last tracked exceeds a given threshold. Through the collection of tracking records the following information is accumulated in a visit: The entry location and time of entry, the exit location and time of exit, the length of the visit, and a sequence of locations at which the device was tracked during the visit. The sequence of locations generated during a visit is known as

the **visit sequence**. The entry and exit locations are used to characterise the visit as follows: *Departing* if the device enters the airport from land and exits through a gate, *Arriving* if the device enters through a gate and exits through a land entrance, *Transit* if the device enters and exits the airport by a gate, and *KissAndFarewell* if the device enters and exits the airport at one of the land entrances.

The visits and their visit sequences can be utilised to construct a sequence database for flow analysis. The visits included in the sequence database can be selected with the attributes relevant to the specific analysis, e.g., construct a sequence database only containing visit sequences collected in a specific time period or with a specific characterisation.

The goal of the current work is to develop an efficient method to store and compute the answers to the **Flow Count Query** as presented in the following definition.

Definition 8. Given a sequence $s = \langle l_1 ... l_n \rangle$ and a sequence database S, the **Flow Count Query** (FCQ) computes the following:

1. The set R of all supersequences r of s, where r is of the form $\langle l_1 ... l_{k-1} ? l_k ... l_n \rangle$ for $1 \leq k \leq n+1$, where ? is a wildcard representing any location from L.
2. The support of each supersequence r in the sequence database S and the support of s in S.

The support counts returned by FCQ enables the analyst to answer questions like: *"Where do people go after leaving the security area and before they leave through a specific gate?"*.

3 The Blip Data Set

The current work is based on a data set collected by Blip at Copenhagen airport. We have worked together with Blip in order to better understand the data in relation to the problems introduced in Section 2. This section describes the source data set and some data issues introduced in the collection process.

3.1 Source Data

This section describes the structure and contents of the Blip data set. The data set consists of the following three tables:

bluetoothaddress The `bluetoothaddress` table contains records for all bluetooth devices tracked by the system listed with the following attributes (`id`, `bluetoothAddress`, `cod`, `mobileManufacturer`, `mobileModel`). The `bluetoothaddress` attribute is the unique bluetooth address of the given device. The `cod` attribute is the class of device, it is an internal classification of different types of bluetooth devices, e.g., *smartphones*, *PDAs*, or *mobile phones*. The `mobileManufacturer` attribute contains the name of the mobile manufacturer, e.g., *SonyEricsson*, *Nokia*, or *Samsung*. The `mobileModel` attribute contains the specific model of the device, e.g., *E50*, *K800i*, or *6300*. Only `bluetoothaddress` is available throughout the data set, the rest of the attributes are only available

for some device records. The following table shows an example of a complete device record.

id	bluetoothAddress	cod	mobileManufacturer	mobileModel
259838	006225251A4B	12564	SonyEricsson	P1i
...	...	...	...	...

lbszone The `lbszone` table contains a record for each bluetooth access point used in the system. Every access point is represented with an `id` and a `zone` attribute, where `zone` is a unique name for the access point. An example of this is the zone with id *16* and the name *ms-spopos1.16* and as it can be seen on the map shown in Figure 1, this zone is covering the security area at CPH. An example of a zone record is shown in the following table.

id	zone
16	ms-spopos1.16
...	...

tracking The `tracking` table contains the tracking records collected by the system. A tracking record contains the following attributes, (`id`, `enterTime`, `leaveTime`, `maxRssi`, `maxRssiTimestamp`, `zone_fk`, `user_fk`). The `enterTime` and `leaveTime` are timestamps describing when the device was first and last seen by the access point. The `maxRssi` is the highest signal strength recorded at that access point and `maxRssiTimestamp` is a timestamp describing when it was recorded. The attributes `zone_fk` and `user_fk` are foreign keys to the `lbszone` and `bluetoothaddress` tables, respectively.

The following table shows an example of a tracking record. The tracking record shows that a device, with user id: *259838*, is tracked at a zone, with zone id: *16*, for 14 minutes 32 seconds, with a maximal signal strength of *-71*.

id	enterTime	leaveTime	maxRssi	maxRssiTimestamp	zone_fk	user_fk
50	2008-04-23 15:10:49	2008-04-23 15:25:21	-71	2008-04-23 15:12:43	16	259838
...	...	...	...	...	...	...

The current data set is collected in the period from the April 23$^{\mathsf{rd}}$ 2008 to October 16$^{\mathsf{th}}$ 2008. During this period Blip experienced some outages in the form of failing access points, resulting in reduced coverage in large parts of the period. The data collected in the period from April 24$^{\mathsf{th}}$ 2008 to May 24$^{\mathsf{th}}$ 2008 is the most complete and is used in the remainder of the current work. In this period Blip recorded 131,304 unique Bluetooth devices using 25 access points, resulting in over 14.2 million tracking records.

3.2 The Bouncing Problem

The tracking data is collected by 25 strategically placed Bluetooth access points that scan for active Bluetooth devices once every second. To get a good coverage of the important areas of the airport, the range of some access points overlap, which introduces a problem we refer to as the **bouncing problem**. The bouncing problem occurs when a series of tracking records makes it look like the device is bouncing back and forth between two or more access points.

An example of this problem is shown in Figure 2 where the grey areas are areas where the access points overlap. When a device is traversing though the

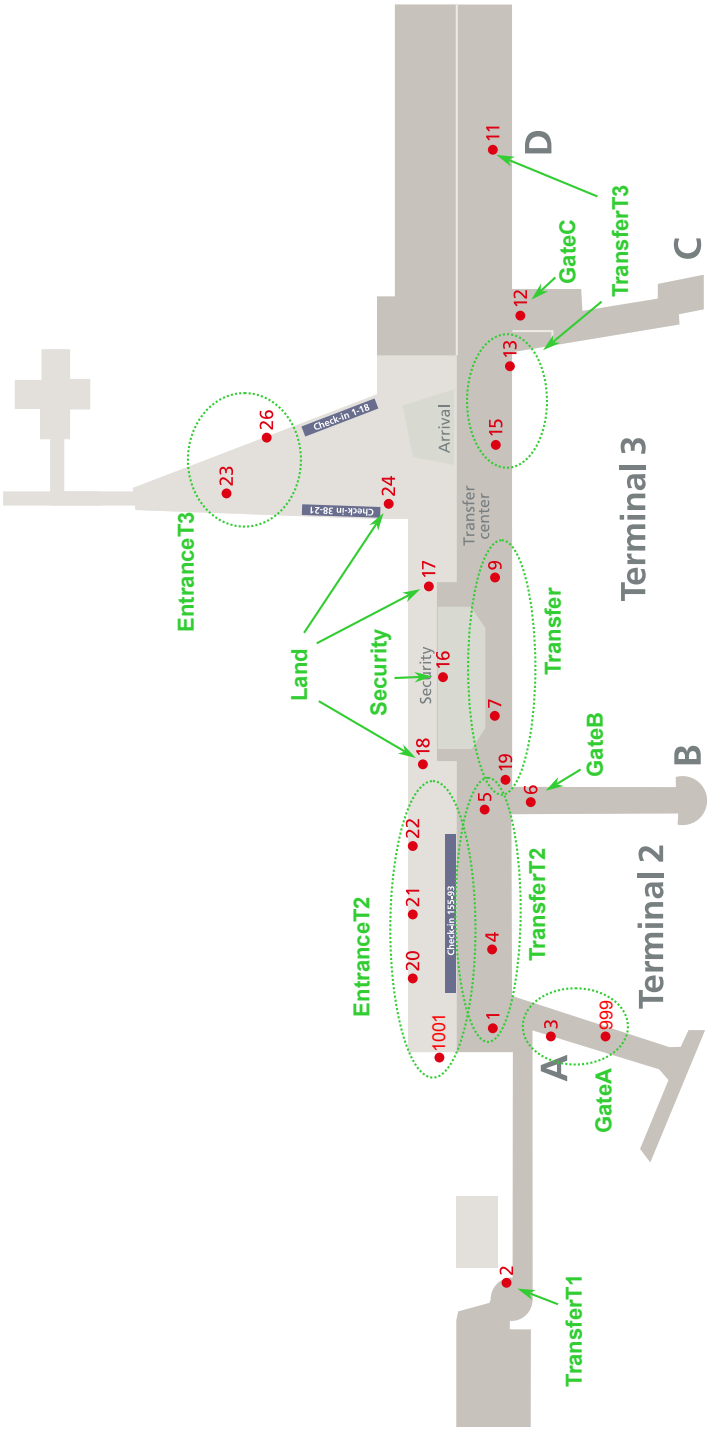


Figure 1: Blip access point placements in CPH. The red dots represent access points, and the green text/circles represent our zone divisions. For simplicity the "`ms-spopos1.`" prefix is removed from the access point names.

areas covered by AP1, AP2, AP3, and AP4 as indicated by the red line, the device will be tracked by two or three access points when located in the grey areas.

The data gathering application used by Blip is not designed to handle devices that are tracked in more than one location at the same time. This means that when a device is located in an area that is covered by more than one access point, the data gathering application creates **bounce records**. A bounce record is a tracking record where the **dwell time**[1] is less than or equal to some system specified bounce threshold. The bounce records are usually created when a device is located in range of two or more overlapping access points. Bounce records can also appear when a device moves past a access point, just at the edge of the access points range. This will result in the device only tracked by the access point form zero to a few seconds, also creating a bounce record.

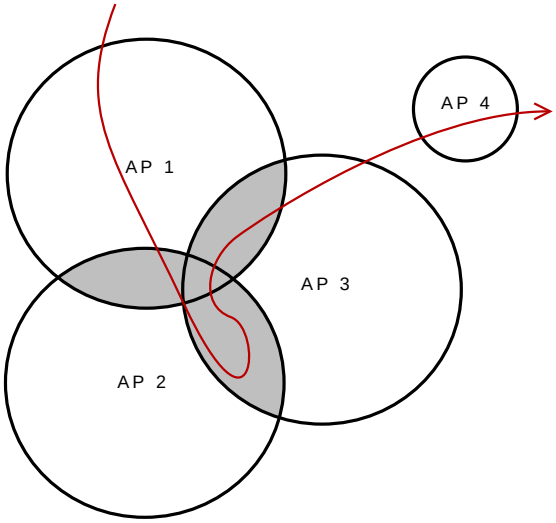


Figure 2: A bouncing problem scenario.

The bounce records can have a very negative effect on the flow analysis if they result in the generation of a visit sequence with many repeating patterns like: ⟨... AP1 AP2 AP1 AP2 ...⟩. This often happens when the device was located in an area covered by both access points AP1 and AP2. A repeating pattern makes it look like the device was moving back and forth between the two access points, even though it could have been stationary or just moving from one area to the other. In order to detect these bouncing records two factors must be taken into account. First, the dwell time of the tracking record, and second, the time in-between[2] the tracking records.

To reduce the impact of this problem we have excluded access point 14, from the remainder of this current work, as described in Section 6.1. The bouncing problem is further addressed in Section 6.3.1, where we introduce a flow optimised Bounce Detection and Elimination algorithm.

[1]The **dwell time** of a tracking record, is defined as (`leaveTime - enterTime`).
[2]This is introduced later as **idle time** in Section 6.3.1.

3.3 Data Set Zone Divisions

To introduce some higher levels of abstraction in the movement of visitors we introduce a **location hierarchy**, built on top of the access points in the `lbszone` table. The location hierarchy consists of 4 levels: **Location**, **Zone**, **Area**, and **Site**. **Location** is the bottom level in the hierarchy and represents the actual access points. **Zone** is a collection of one or more locations represented by a Zone name and is used to represent a small section of the airport, e.g., *EntranceT3* and *TransferT3*. **Area** is a collection of one or more Zones represented by a Area name, and is used to represent a larger section of the airport, e.g., *Land* or *Transfer*. **Site** is the top level of the hierarchy and includes one or more areas. **Site** is represented by a Site name, and is used to represent the actual airport, e.g., *CPH*.

The Location and Zone levels in CPH are marked on the map shown in Figure 1. A table showing all levels of the hierarchy is presented in Appendix A. In the remainder of this work, we use the Zone level as the lowest level of abstraction in visitor movement. This can help eliminate some of the noisy data produced by the bouncing problem, and thereby improve the quality of the flow analysis with the cost of a reduction in the granularity of the movement.

4 Data Warehouse Schema

In this section we present the data warehouse schema shown in Figure 3, and discuss the less trivial parts of the schema. The proposed multidimensional database schema [12] is designed to accommodate the data from the Blip data set, using the basic assumption that, in some aspects, space is not as costly as complexity. The design is a mixture between a star and a snowflake schema, preferring the star schema for its low complexity [13], and using snowflake schema where needed.

All dimension tables contain a `ID` attribute as a surrogate key. This key has no reference to the original data set, it is a unique key assigned to each row, when the data is loaded into the warehouse.

Fact Tracking The `Fact_Tracking` fact table contains the tracking records, with foreign keys to the relevant entries in the dimension tables. Each row contains a tracking record with foreign keys to the date, time, location, visit, and classification dimensions representing all the attributes of the tracking record, as well as the two base facts `DwellTime` and `MaxRSSI`.

The classification attribute is used to classify the state of the tracking record. This is mainly to note if the tracking record has been modified when loaded into the data warehouse. This will be further discussed in Section 6, specifically in the transformation phase.

Time and Date Dimensions The tables `Dimension_TimeOfDay` and `Dimension_Date` tables are used to model time and date respectively. Besides the attributes needed to model date and time, the tables include two string attributes. These attributes enable classification of a date or time, e.g., classify a specific day as a *holiday* or a specific time of day as *morning rush hour*.

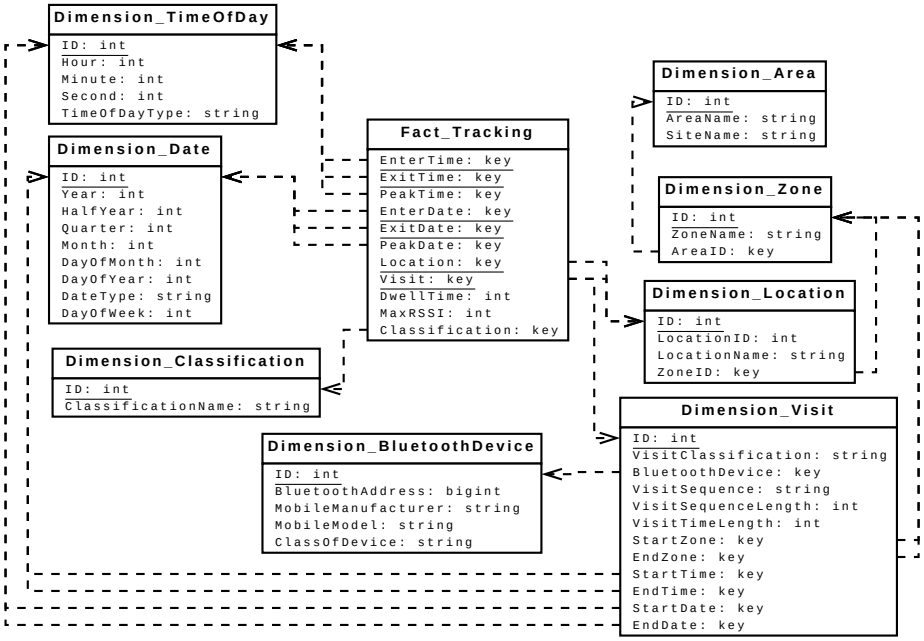


Figure 3: Data warehouse schema.

Time and date are split into two dimensions in order to save a significant amount of rows. In this way the date dimension contains a few records per day and the time dimension a few records per second of a day, depending on the number of classifications in both cases. If the two dimensions where joined, it would result in a few rows for every second of every day, which would make the dimension grow heavily (> 31.5 million rows per year). The cost of splitting the dimensions is an extra foreign key, for each time stamp, in the fact table and visit dimension, which is considered a better solution in this case.

Location, Zone, and Area Dimensions The location dimensions represent the access points present in the source data set. The dimensions represent a four level hierarchy, as presented in Section 3.3, split into the three tables: `Dimension_Location`, and its two outrigger dimension, `Dimension_Zone` and `Dimension_Area`.

The `Dimension_Location` table contains the actual access points present in the source data set, represented with a `LocationName` and a `ZoneID`. The `Dimension_Zone` table enables the access points to be divided into different "zones" represented with a `ZoneName` and a `AreaID`. This is used to group access points with the same physical placement in the airport. The `Dimension_Area` table contains the different sections (areas) of the airport. The `AreaName` is the name of the section, and `SiteName` is the name of the airport. The area concept is used to group different zones together.

The location dimensions are split into three tables to allow linking to a specific zone or area without also linking to a specific location. This is required later on in the visit dimension.

Bluetooth Dimension The `Dimension_Bluetooth` table contains the Bluetooth devices present in the source data set, with all the available attributes.

Classification Dimension The classification dimension (`Dimension_Classification`) is used to classify a specific tracking record as, e.g., *OK* or *ModifiedBounceRecord*. This dimension allows the analyst to select if records that have been altered when loaded into the data warehouse should be included in the analysis or not.

Visit Dimension The visit dimension (`Dimension_Visit`) is used to identify, classify, and associate each tracking record with a specific visit. The visit dimension is inserted between the two tables `Fact_Tracking` and `Dimension_BluetoothDevice`, to stress that a Bluetooth device cannot be tracked in the airport without being associated to one or more visits.

The dimension contains four foreign keys that represent the period in which the visit took place (`StartTime`, `EndTime`, `StartDate`, and `EndDate`), two foreign keys to identify the start and end zone of the visit (`StartZone`, `EndZone`), and an outrigger `BluetoothDevice` to identify the tracked Bluetooth device. The amount of zones traversed and the time period of a specific visit is stored using the two attributes `VisitSequenceLength` and `VisitTimeLength`. The sequence of zones traversed during the visit is stored as a string in the `VisitSequence` attribute. It should be noted that this attribute is only necessary in some of the approaches presented later. The last attribute `VisitClassification` is used to characterise a specific visit using the classifications introduced in Section 2 (*Departing*, *Arriving*, *Transit*, and *KissAndFarewell*). These classifications enable the analyst to, e.g., limit the flow analysis to visitors that are actually departing from the airport.

5 Pattern Indexing Approaches

In this section we propose three different approaches to implementing flow analysis, in the data warehouse, that all build on a general frequent pattern approach, as described in the following sections. First, the prerequisites for utilising frequent patterns are presented. Then the three different approaches are presented.

5.1 Frequent Pattern Index Overview

The idea behind the three approaches is to exploit precomputed frequent patterns to make an efficient flow analysis. The frequent patterns can be used to either locate preaggregated results or limit the search space, when querying the visit sequence database. Given the results in [16] and [17], we have chosen to use the PrefixSpan algorithm as presented later on in Section 6, since this algorithm has still shown to be one of the fastest algorithms available for frequent pattern mining. The use of frequent patterns imply that only patterns with a support above a certain threshold will be stored.

To store these frequent patterns we extend the data warehouse, described in Section 4, with a new pattern dimension (`Dimension_Pattern`). This dimension contains the following attributes: An `ID` to identify the pattern, and a

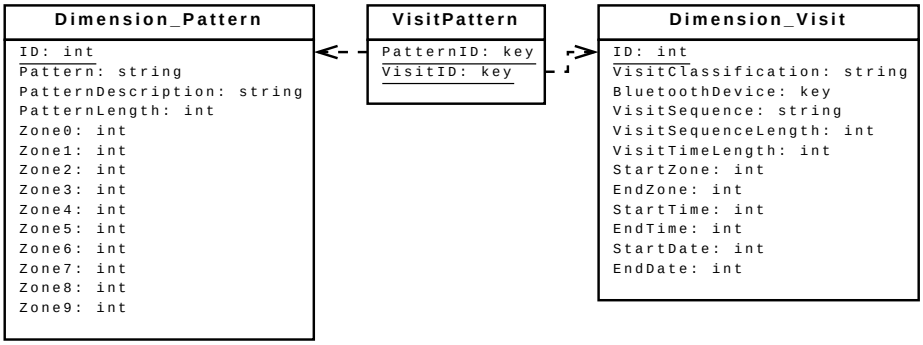


Figure 4: Visit Pattern Association schema.

`Pattern` and `PatternDescription` which holds the specific pattern and its textual description, respectively. The `PatternLength` attribute stores the amount of elements in the pattern, and the ten zone attributes (`Zone0 - Zone9`) store each element of the pattern. These zone attributes are combined to make a browsable pattern hierarchy when building a multidimensional OLAP cube on the dimension. Based on the current data set we have decided to only support patterns of length up to 10 in the pattern hierarchy, since longer patterns do not occur frequently.

The pattern dimension is shown as part of Figure 4 and can be linked to the existing data warehouse in different ways, as described in the following sections.

5.2 Visit Pattern Association

The main idea behind the Visit Pattern Association (VPA) approach is to limit the search space used to compute the results needed to answer the FCQ. The approach is inspired by the Inverted Indexing Approach presented in [15].

The VPA approach is based on a new bridge table (`VisitPattern`) added between the pattern dimension and the visit dimension, as shown in Figure 4. The bridge table is responsible for associating each pattern in the pattern dimension with the visits, where the visit sequence contains the specific pattern. This makes it a many-to-many relationship, since a visit sequence can contain many patterns, and a pattern can be found in many visit sequences.

The `VisitPattern` table is a new fact table in the data warehouse, used to locate the visit sequences that contain a given pattern and thereby giving an exact visit count or a limited search space. As described in Section 2, the query should compute visit counts for all the frequent supersequences of the given query sequence. These visit counts are easily found for all supersequences present in the pattern dimension, by counting the associations in the `VisitPattern` fact table. If a supersequence is not present in the pattern dimension, a subsequence of that supersequence, present in the pattern dimension, is used to select a limited set of visit sequences that might contain the supersequence, given the Apriori property [1]. A one-pass scan of the limited set of visit sequences can then compute the amount of sequences that contain each of the missing supersequences.

Pros: The search space limitation approach is motivated by the fact that the amount of tracked users is large which results in a heavily growing visit dimension and therefore a large amount of visit sequences. This results in a large search space when searching for a specific query sequence. By utilising the precomputed associations in the `VisitPattern` fact table, this search space can be greatly reduced or even eliminated if all patterns necessary to answer the query are present in the pattern dimension.

Another key feature of this approach is that, since each pattern is linked to all visits containing the pattern, all attributes in the visit dimension can be utilised to query the patterns. An example of this would be to only select visits classified as *departing* or selecting visits that start in a certain zone.

Cons: The amount of rows in the `VisitPattern` fact table can become very large, due to the fact that all patterns are associated with all the visits in which they occur. An example of this could be a pattern only containing a central location where most visitors pass by. This pattern will then be associated with up to 100% of the visits, creating a huge amount of rows in the fact table.

The selection of the specific visit sequences, by random reads in the database, is potentially more costly than performing linear scan of all visit sequences in the sequence database [20]. This means that the method is only effective when the amount of extracted sequences is greatly reduced. An example of this could be if the selected pattern is present in about 90% of the visits then it would be faster to select all visit sequences to utilise sequential read. But if the pattern is only present in about 5% of the visits, we assume that random reads would perform better.

5.3 Pattern Count Aggregation

The Pattern Count Aggregation (PCA) approach is based on preaggregations in a new fact table and is motivated by the ability to quickly find exact visit counts, as needed to answer an FCQ. The fact table contains a preaggregated count of visits for each pattern during a given time interval and a given set of preaggregated dimension attributes. The aggregation time interval can be set to any level in the time or date hierarchies, depending on the desired time granularity. Figure 5 shows an example with a daily time aggregation interval including the visit dimension classification attribute. The PCA approach provides an efficient lookup when the query level is coarser than, or equal to the aggregation time interval, e.g., if a query requests the weekly count of visits for a given pattern, then a daily time interval could simply be summed up to weeks. If the query level is finer than the aggregation time interval, the aggregated count is unusable.

The aggregation interval is application specific and a balance has to be found between the desired granularity and the amount of space used to store the aggregated values. The fact table contains one row per pattern for each aggregation interval, which is multiplied by the amount of distinct values for each visit dimension attribute preaggregated.

Pros: This approach is motivated by the efficient query performance gained by utilising the preaggregated counts. The amount of rows in the fact table is greatly reduced, compared to the Visit Pattern Association.

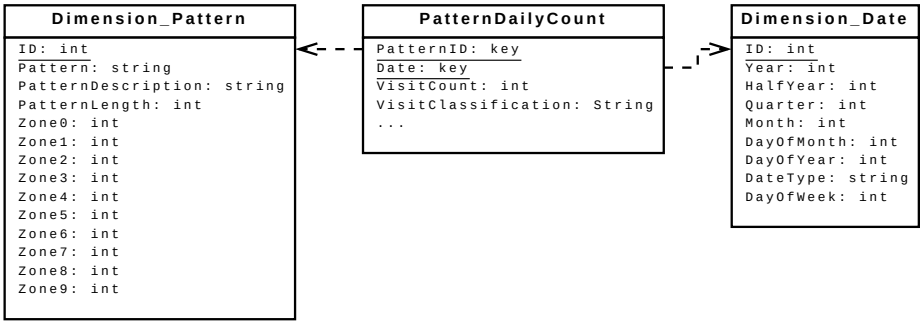


Figure 5: Pattern Count Aggregation schema.

Cons: The lack of ability to locate the actual visit sequences that contain the pattern, limits the possibility of using the aggregation as a selection filter and thereby extract the visits sequences used to compute patterns that fall under the minimum support threshold.

The PCA approach has a potential penalty in the size of the fact table if the aggregation interval and the minimum support threshold are set too low and the amount of preaggregated visit dimension attributes is large.

5.4 Pattern Aggregation Hybrid

The Pattern Aggregation Hybrid (PAH) approach is a combination of the two previous approaches. The idea is to utilise strengths of both approaches. We propose to use the PCA approach to answer FCQs concerning patterns that can be preaggregated within the specified support threshold and use the visit pattern associations of the VPA approach to associate a domain specific set of patterns with the visits containing these. The associations are then used to answer FCQs targeting patterns that do not pass the specified support threshold. The data warehouse schema of this approach is shown in Figure 6.

This approach utilises the fact that the PCA approach is an efficient way of storing precomputed counts, and thereby also efficient in returning results to a given sequence query. The storage space used by the PCA approach to store very frequent patterns is low compared to the VPA approach, but when the support gets low both approaches tend grow fast in the amount of rows in the fact tables.

We propose to use the VPA approach to associate a domain specific set of patterns with the visits containing them. By domain specific we mean that this depends on the application in which the PAH approach is used. In our PAH implementation described in Section 6.5 we associate all patterns of length three with the visits containing them. This is based on the fact that associating too short, and thereby too frequent patterns, will result in a large amount of associations. And too long patterns will degrade the usability of the approach.

By making these associations we maintain the ability to extract the greatly reduced amount of visit sequences containing the patterns, and utilise these for a result computation at an even lower support threshold.

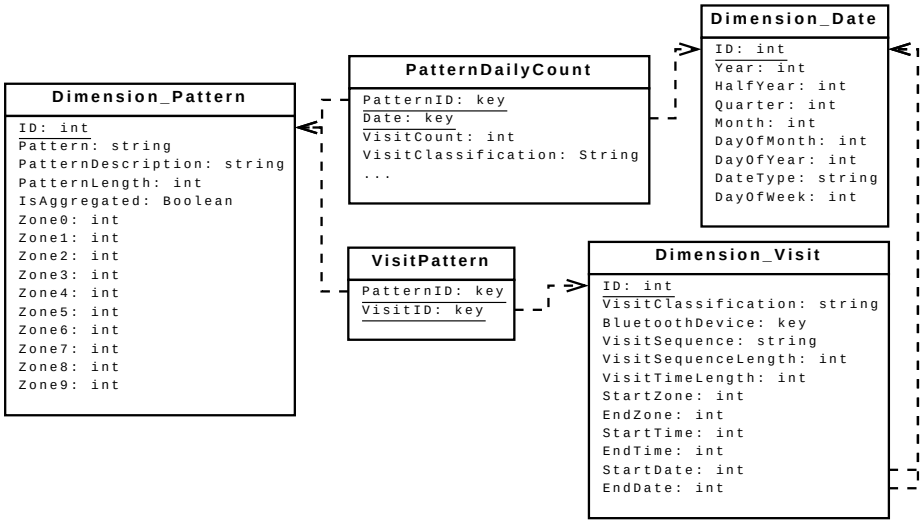


Figure 6: Pattern Aggregation Hybrid schema.

Pros: This approach utilises the strengths of both approaches by providing the efficient storage from the PCA approach, and the limitation in the search space of patterns with a low support threshold from the VPA approach.

Cons: This is a more complex model. The patterns that are stored as preaggregated counts are only present for a predefined amount of attributes from the visit dimension and a predefined time level, where patterns that are associated with the visits containing them, supports the use of all visit dimension attributes.

The decision of which patterns to associate can be complex since it depends on the domain in which the system is deployed. By only looking at the support threshold when deciding which patterns should be associated, there is a possibility that some patterns could be poorly represented in the system, resulting in a varying result quality. These issues will be discussed further in Section 8.

6 Extract Transform Load

In this section we present our Extract Transform Load (ETL) system. The ETL system is designed and implemented using the *good practice* techniques presented in [13]. The architecture of the ETL is shown in Figure 7. It is divided into three phases: The extraction phase responsible for extracting the data from the source database, the transformation phase responsible for cleaning and transforming the data to match the data warehouse, and the load phase responsible for loading the transformed data into the data warehouse.

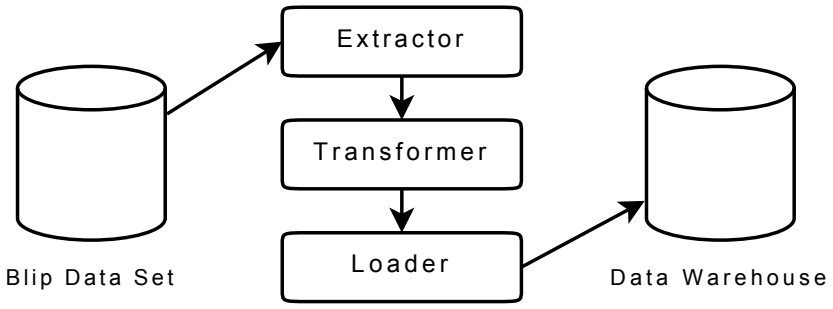


Figure 7: ETL Architecture.

The ETL system is implemented as a multi-threaded C# application executing each phase in a separate thread. This enables us to pipeline the whole process of extracting, transforming and loading the source data into the data warehouse. We choose to implement our own hand-coded ETL system instead of using an ETL tool, because the transformations required in the transformation phase are highly specialised and would be complex to implement using an ETL tool.

Between each phase of the ETL system the data is stored in thread-safe queues enabling the phases to work concurrently by dequeuing data from the previous phase as soon as it is ready, and enqueuing data for the next phase when it is processed.

To exemplify the effect of the different phases we use the data from our current implementation as a running example throughout this section.

The following Section 6.1 describes the extraction phase, Section 6.2 gives an overview of the transformation phase where Section 6.3, Section 6.4, and Section 6.5 describe the three subphases of the transformation phase. Section 6.6 describes the load phase of the ETL.

6.1 Extraction Phase

The Extractor shown in Figure 8 is responsible for extracting the data from the source database described in Section 3. The Extraction phase consist of three steps: First all locations and Bluetooth devices present in the source database are extracted and passed to the Loader for prepopulation of the respective dimensions. When finished all tracking records are extracted utilising the following selection criteria to ensure that only complete and meaningful data get extracted, since incomplete data could skew the later flow analysis.

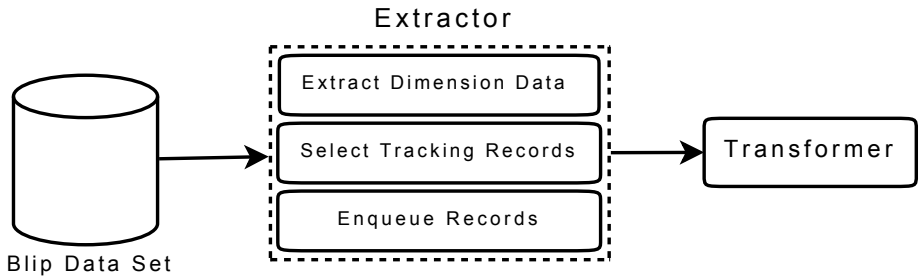


Figure 8: ETL Extractor.

Selection Criteria The extracted tracking records have to pass the following criteria. The first criteria ensures that only data from the specified time period is extracted. The second criteria is introduced to exclude location 14, that mostly generated noisy data. The access point at location 14 is physically located close to location 13 and 15 (shown on the map in Figure 1 on page 6), but tests have shown it to have bouncing problems with locations in many different parts of the airport, skewing the flow analysis. The decision to exclude the location was made in cooperation with Blip, and has shown to have a positive effect on flow analysis.

The third criteria concerns data that is incomplete, and the forth and fifth criteria concerns data that is considered incorrect since it contains one or more invalid time attributes.

1. The records must have an `enterTime` within the time period April 24^{th} 2008 to May 23^{th} 2008, both days included.
2. The records must have `zone_fk` different from 14.
3. All attributes must be complete, no *null* values.
4. The records must have `leaveTime` $\geq$ `enterTime`.
5. The following must apply to `maxRssiTimestamp`: `enterTime` $\leq$ `maxRssiTimestamp` $\leq$ `leaveTime`.

In our running example with the Blip data set, the selection criteria has the following effects. A total of 14.2 million records are recorded in the time period (first criteria), after zone_fk 14 (second criteria) is removed a total of 12.2 million records are left. 46,302 records contain *null* values (third criteria) and 1,844,982 have invalid timestamps (forth and fifth criteria), which leaves a total of 10,342,331 records extracted from the data set.

All tracking records that pass the selection criteria are ordered by ascending `enterTime` and enqueued as ready for the Transformer. The tracking records are currently of the form: (*EnterTime*, *ExitTime*, *PeakTime*, *BluetoothDevice*, *Location*, *MaxRSSI*, *Classification*), where *EnterTime*, *ExitTime*, and *PeakTime* represent `enterTime`, `leaveTime`, and `maxRssiTimestamp` respectively, *BluetoothDevice* represents `user_fk`, *Location* represents `zone_fk`, *MaxRSSI* represents `maxRssi`, and *Classification* represents their current classification (thus so far all are classified as *OK*).

6.2 Transformation Phase Overview

The transformation phase is responsible for cleansing and transforming the extracted data. The cleansed data is then used to compute all the information needed to match the data warehouse schema described in Section 4. An overview of the transformation phase is shown in Figure 9.

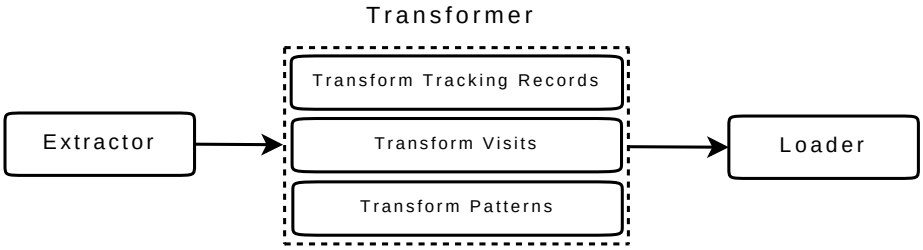


Figure 9: ETL Transformer Overview.

The transformation phase is as shown in Figure 9 divided into the following three subphases with separate responsibilities:

Transform Tracking Records This subphase is responsible for transforming the tracking records and detecting visits. The tracking records and detected visits are sent to the next subphase. The steps of this subphase are described in Section 6.3.

Transform Visits This subphase is responsible for making the final cleansing of the detected visits before enqueuing the visits and tracking records for loading, and computing the frequent patterns from the set of detected visits. Once the patterns are cleansed the patterns are enqueued for loading and sent to the next subphase. The steps of this subphase are described in Section 6.4.

Transform Patterns This subphase includes three different steps that can be executed independently of each other. Each step is responsible for computing the data necessary for one of the indexing approaches presented in Section 5. The steps of this subphase are described in Section 6.5.

6.3 Transformation of Tracking Records

The transformation subphase called Transform Tracking Records described in this section concerns the cleansing and transformation of tracking records and the detection of visits. An overview of this subphase is shown in Figure 10.

The Transformer receives a queue of tracking records from the Extractor. This queue is first processed by the Bounce Detection step described in the following section. The resulting tracking record queue is then processed by the Collapse Detection step described in Section 6.3.2. The tracking record queue returned by the Collapse Detection step is then used in the detection of visits in Section 6.3.3.

The following sections contain a description of the different steps executed in this subphase.

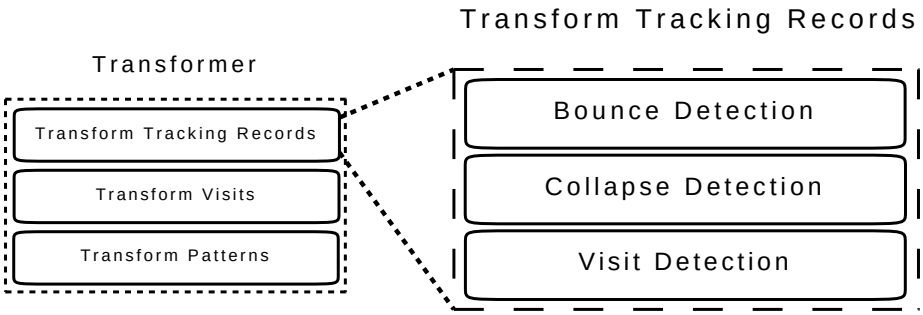


Figure 10: ETL Transformation subphase *Transform Tracking Records*.

6.3.1 Bounce Detection

This section presents the Bounce Detection step which is responsible for minimising the bouncing problem described in Section 3.2. In the beginning of this step the dwell time and idle time of each record is computed and stored on the specific tracking records. These records are then processed by the Bounce Detection algorithm shown in Algorithm 1. The form of the tracking records is now extended with three new attributes, *DwellTime*, *IdleTime*, and *BluetoothAddress*. *DwellTime* we recall from Section 3.2 and *IdleTime* is defined below. *BluetoothAddress* is the Bluetooth address of the device that was tracked by the tracking record. The Bluetooth address is used to identify devices in the ETL system, because the `user_fk` attribute from the Blip data set is no longer needed and therefore not stored in the data warehouse.

To describe the algorithm a clear definition of the following terms is needed. The **bounce threshold** B_t is an implementation specific threshold set according to the constraints of the given data set. **Idle time** denotes the time since a device was last tracked. We recall the **bounce record**, presented in Section 3.2, as a tracking record with dwell time $< B_t$ for $B_t > 0$. We define a **bounce region** as a time span in which a device only generates bounce records and the idle time between the records is less than B_t. The terms bounce record and bounce region are clarified by the following example.

Example Figure 11 presents an example of a device that moves through the area presented in Figure 2 on page 7. At time t0 the device is detected by AP1, and at time t1 the device starts to bounce between AP1 and AP2, which starts the bounce region, until t2. As the device moves around the area covered by multiple access points, the bounce region continues. The device is next being tracked through a number of bounce records between AP2 and AP3 in the timespan from t2 to t3. In the timespan from t3 to t4, the device is detected by three different access points, AP1, AP2 and AP3. The last part of the bounce region is from t5 to t6, where the device is detected by both AP1 and AP3. At time t5 the bounce region ends because the device produces a tracking record with a dwell time $> B_t$ in AP3. At time t6 the device makes a clean shift from AP3 to AP4 with no bounce records.

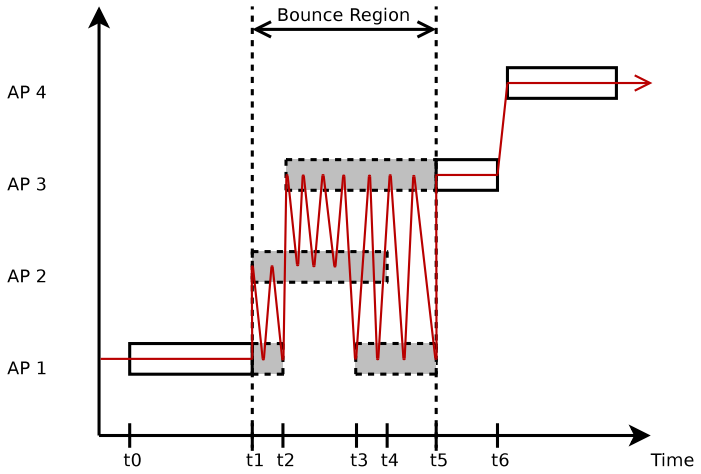


Figure 11: Time line example showing the tracking history of the device movement shown in Figure 2. Each solid white box represents a clean tracking record, and the greyed boxes represent many bounce records. The red line presents an example of the tracking record distribution.

Bounce Detection Algorithm The Bounce Detection algorithm is shown in Algorithm 1, and is designed to locate and process bounce regions in a queue of tracking records.

From Line 2 to 19 every tracking record tr is processed to detect bounce regions and eliminate the bounce records in these. On Line 3 in the algorithm, the queue BE of all previous bounce records from BQ with the same Bluetooth address as the current tr, is created. On Line 4 the dwell time of tr is compared to B_t. If the dwell time of tr is larger than B_t the current bounce region is ended and all bounce records in BE are removed from BQ. On Line 7 and 9 the Bounce Elimination algorithm presented later in this section, is run on the queue BE and an $AddToCollapseQueue$ function is run on the current tr. $AddToCollapseQueue$ takes a processed tracking record and adds it to a queue of records that are ready for Collapse Detection presented in Section 6.3.2. On Line 11 to 16 the algorithm handles the case where dwell time of tr is less than or equal to B_t. If the idle time of tr is unknown or less than B_t we add tr to BQ, which continues the current bounce region or starts a new one. Otherwise, we end the current bounce region by, removing the records in BE from BQ, running the Bounce Elimination algorithm on BE, and adding tr to BQ to start a new bounce region.

To process all bounce records left in BQ the set BT of the Bluetooth addresses of the devices that still have records in BQ is created on Line 20. From Line 21 to 25 all the remaining bounce records in BQ are processed.

Bounce Elimination Algorithm The Bounce Elimination algorithm is shown in Algorithm 2. The algorithm takes a queue of bounce records detected as a bounce region and eliminates the bounce records by combining them into as few records as possible.

On Line 1 the algorithm handles the case where there is only 1 bounce record

Algorithm 1: `BounceDetection(`TRQ`)`, detects bounce regions within a queue TRQ of tracking records ordered by ascending enter time. Uses the system specific bounce threshold B_t in seconds.

```
1  BQ.clear();
2  while tr ← TRQ.dequeue() do
3  |  BE ← Queue of tracking records t from BQ where
   |  t_BluetoothAddress = tr_BluetoothAddress;
4  |  if tr_DwellTime > B_t then
5  |  |  if BE.length() > 0 then
6  |  |  |  Remove all elements in BE from BQ;
7  |  |  |  BounceElimination(BE) ;
8  |  |  end
9  |  |  AddToCollapseQueue(tr) ;
10 |  else
11 |  |  if tr_IdleTime < B_t ∨ tr_IdleTime = Unknown then
12 |  |  |  BQ.enqueue(tr);
13 |  |  else
14 |  |  |  Remove all elements in BE from BQ;
15 |  |  |  BounceElimination(BE);
16 |  |  |  BQ.enqueue(tr);
17 |  |  end
18 |  end
19 end
20 BT ← Set of all unique BluetoothAddress in BQ;
21 foreach bt ∈ BT do
22 |  BE ← Queue of tracking records t from BQ where
   |  t_BluetoothAddress = bt;
23 |  Remove all elements in BE from BQ;
24 |  BounceElimination(BE);
25 end
```

in BE. The bounce record is simply added unmodified to the collapse detection queue, since there are no other records to combine it with.

If there is more than one record in BE, the start and end time of the bounce region is found on Line 5 and 6. The start time is set to the minimum enter time and the end time is set to the maximal exit time of all the records in the bounce region. On Line 7 the queue L of all locations in the bounce region is created by utilising the *GetOrderedLocations* function. *GetOrderedLocations* takes a bounce region queue as input and returns a queue of the locations visited in the bounce region. The locations are ordered by the first time they are visited in the bounce region queue. Except if a location in the queue is the last locations to be visited before the device enters the bounce region. Then this location is set as the first location in the queue. In this way the bounce record can possibly be joined with the previous tracking record in the latter Collapse Detection.

In the case that only one location is visited in the bounce region, a new tracking record is created and added to the collapse queue on Line 8 to 15.

If more than one location is part of the bounce region a new tracking record

Algorithm 2: `BounceElimination(`BE`)`, given a queue BE of bounce records it creates new records that match all locations and time span of the bounce region.

```
1  if BE.length() = 1 then
2  |  br ← BE.dequeue();
3  |  AddToCollapseQueue(br);
4  else
5  |  region_Start ← br_EnterTime, where br is the element i BE with the
   |  lowest enter time;
6  |  region_End ← br_ExitTime, where br is the element in BE with the
   |  highest exit time;
7  |  L ← GetOrderedLocations(BE);
8  |  if L.length() = 1 then
9  |  |  br ← Any element from BE;
10 |  |  br_EnterTime ← region_Start;
11 |  |  br_ExitTime ← region_End;
12 |  |  br_DwellTime ← br_ExitTime − br_EnterTime;
13 |  |  br_Classification ← "ModifiedBounceRecord";
14 |  |  AddToCollapseQueue(br);
15 |  else
16 |  |  region_Total ← region_End − region_Start ;
17 |  |  t_Offset ← 0;
18 |  |  while loc ← L.dequeue() do
19 |  |  |  t_Count ← The number of elements br in BE where
   |  |  |  br_Location = loc;
20 |  |  |  t_DwellTime ← (t_Count / |BE| * region_Total);
21 |  |  |  br ← Any element br from BE where br_Location = loc;
22 |  |  |  br_EnterTime ← region_Start + t_Offset;
23 |  |  |  br_ExitTime ← br_EnterTime + t_DwellTime;
24 |  |  |  br_DwellTime ← t_DwellTime;
25 |  |  |  br_Classification ← "ModifiedBounceRecord";
26 |  |  |  AddToCollapseQueue(br);
27 |  |  |  t_Offset ← t_Offset + t_DwellTime;
28 |  |  end
29 |  end
30 end
```

for each location is created on Line 18 to 28. The value of $region_{total}$ represents the total time span of the bounce region in seconds and t_{offset} is the amount of dwell time already allocated to already processed locations in the current bounce region. On Line 19 the total amount of tracking records for the given location is counted in order to make a weighted distribution of the $region_{total}$ on Line 20. This is done to divide the total time of the bounce region according to the amount of tracking records produced for a given location. On Line 21 to 25 the new tracking record is created and classified as a *ModifiedBounceRecord*. The record is then added to the load queue and t_{offset} is updated.

In our running example the Bounce Detection step, with $B_t = 20$ seconds,

removes 2,707,844 records from the tracking record queue, leaving a total of 7,634,487 records.

The queue of tracking records generated by the Bounce Detection and Bounce Elimination algorithms is then given as input to the Collapse Detection step, presented in the following section.

6.3.2 Collapse Detection

This step is responsible for reducing the total amount of records loaded into the data warehouse by combining repeating tracking records into fewer records. The step is motivated by the following assumption: If a device is tracked at a location and tracked at that same location at some point later in time, without being tracked at any other locations in between, we can combine the records into one longer record. We can not determine where the device has been in-between the two tracking records and we therefore assume it has stayed at the same location. In order to avoid the case with visitors leaving the airport and returning through the same entry at a later time, we specify a **collapse threshold**. If the idle time between two records exceeds the collapse threshold, they will not be joined.

This step keeps track of where and when all devices were last seen. This knowledge is then used to join the tracking records for all devices, that comply with the following criteria:

- The tracking records are from the same location and device.
- The device has not been tracked at other locations in-between the tracking records.
- The idle time between the tracking records does not exceed the collapse threshold.

The collapse threshold is set to 5 minutes in the running example, removing 2,811,614 records from the tracking record queue, leaving a total of 4,822,873 records.

The tracking record queue computed by this step is then given as an input to the Visit Detection step, presented in the following section.

6.3.3 Visit Detection

In this step the $VisitDetection$ algorithm, shown in Algorithm 3, is utilised to detect visits in the queue of tracking records passed on by the Collapse Detection step.

To describe the algorithm we need to introduce and refine a few data types. Recall the visit from Section 4 of the form: (*ID*, *BluetoothAddress*, *VisitSequence*, *StartTime*, *EndTime*), where *ID* is the surrogate key for the visit in the data warehouse, *BluetoothAddress* is the Bluetooth address of the device tracked by this visit, *VisitSequence* is the sequence of locations visited during the visit, and *StartTime* and *EndTime* contain the start and end times of the visit. To associate each tracking record to a specific visit, a *VisitID* attribute is now added to the tracking records. The attribute will contain the surrogate key to the visit the tracking record is part of. To maintain the state of

the devices in the system, we introduce a **known device** record of the form: (*BluetoothAddress*, *Visit*, *LastSeen*).

From line 1 to 28 the algorithm iterates through all tracking records in the tracking record queue while keeping track of which devices have been seen before and when they have been seen before. On line 2 it checks if the device is seen before and extracts the known device record *td* on line 3. *td* is removed from the set of known devices DS on line 4 to be updated and is added again on line 19. On line 5 to 11 the algorithm handles the case where the time, from the device last was seen, to the current record exceeds the visit threshold V_t. This ends the current visit and starts a new visit with the current *tr* as the first record. To end the current visit td_{visit}, the $CompleteVisit$ function is executed with the visit and the device last seen time as input. The $CompleteVisit$ function completes the visit by adding and updating the following attributes on the visit: *EndTime* of the visit is set to the device last seen time (which is the exit time of the last tracking record in the visit), *VisitSequenceLength* is computed from the length of the visit sequence, *VisitTimeLength* is computed from the start and end time, and *StartZone* and *EndZone* are set to the first and last element in the visit sequence.

The completed visit is added to the set of visits VS and a new visit is created with the $NewVisit$ function taking the values of the current tracking record as input. The $tr_{VisitID}$ is updated with the new visit id v_{ID}, the visit is applied to the known device record *td*, and the $td_{LastSeen}$ attribute is updated with the exit time of the current tracking record.

Line 12 to 17 handles the case where the idle time before the tracking record is less than the visit threshold. In this case the visit sequence is update, the $tr_{VisitID}$ is set to the id of the current visit, and the devices last seen value is updated. Line 20 to 26 handles the case where the device has not been seen before. A new known device record is created with the Bluetooth address of the tracking record, and a new visit is associated with the known device record and the tracking record. On line 29 to 33 all visits contained in the known device records in DS, are completed and added to VS, since there are no more tracking records. On line 34 the algorithm returns the set of all detected visits VS.

In our running example a total of 247,366 visits are detected in the 4.8 million tracking records. The set of detected visits computed by this step and the tracking record queue, are given as input to the Transform Visits subphase.

6.4 Transformation of Visits

This subphase is responsible for computing the frequent patterns needed by the indexing approaches, which are loaded in the next subphase of the Transformer. An overview of this subphase is shown in Figure 12.

The steps shown in Figure 12 extracts and cleans all visit sequences before inserting them into a sequence database. The sequence database is then written to a file and given as an input to a PrefixSpan implementation [10, 16] which returns all patterns satisfying a specified support threshold. The set of frequent patterns computed by PrefixSpan is then cleaned before it is passed to the next transformation phase and enqueued for the Loader.

Algorithm 3: `VisitDetection`(TRQ), detects and returns the set VS of all visits in a queue TRQ of tracking records. Uses a set DS of known device records to keep track of all devices and the visit threshold V_t to determine when a new visit starts.

```
1  while tr ← TRQ.dequeue() do
2      if td ∈ DS|td_BluetoothAddress = tr_BluetoothAddress then
3          td ← The element td from DS where
           td_BluetoothAddress = tr_BluetoothAddress;
4          DS ← DS\{td};
5          if tr_EnterTime − td_LastSeen > V_t then
6              td_Visit ← CompleteVisit(td_Visit, td_LastSeen);
7              VS ← VS ∪ {td_Visit};
8              v ← NewVisit(tr_BluetoothAddress, tr_Location, tr_EnterTime);
9              tr_VisitID ← v_ID;
10             td_Visit ← v;
11             td_LastSeen ← tr_ExitTime;
12         else
13             v ← td_Visit;
14             v_VisitSequence.append(tr_Location);
15             tr_VisitID ← v_ID;
16             td_Visit ← v;
17             td_LastSeen ← tr_ExitTime;
18         end
19         DS ← DS ∪ {td};
20     else
21         td ← NewKnownDevice(tr_BluetoothAddress);
22         v ← NewVisit(tr_BluetoothAddress, tr_Location, tr_EnterTime);
23         tr_VisitID ← v_ID;
24         td_Visit ← v;
25         td_LastSeen ← tr_ExitTime;
26         DS ← DS ∪ {td};
27     end
28 end
29 foreach td ∈ DS do
30     DS ← DS\{td};
31     td_Visit ← CompleteVisit(td_Visit, td_LastSeen);
32     VS ← VS ∪ {td_Visit};
33 end
34 Return VS;
```

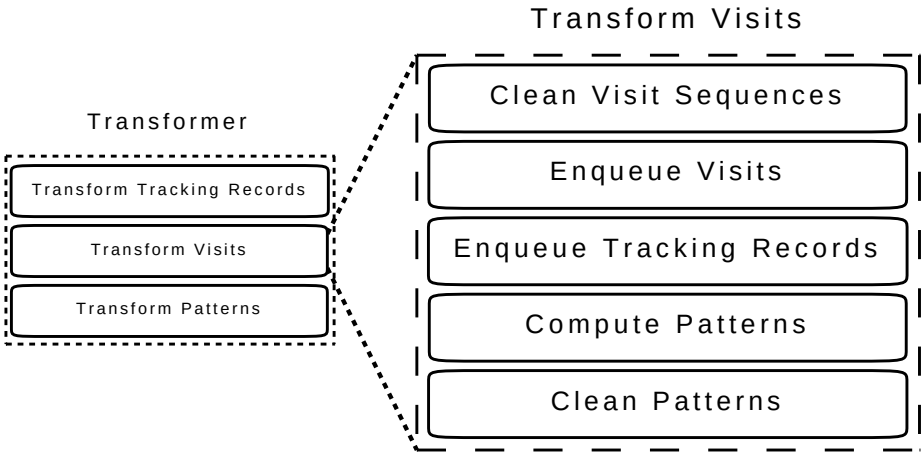


Figure 12: ETL Transformation subphase *Transform Visits*.

6.4.1 Clean Visit Sequence

All the visits detected in the Visit Detection step contain a sequence of locations traversed. Since we decided to raise the abstraction level to zones instead of locations as described in Section 3.3, the location ids in the visit sequences are now converted to the corresponding zones ids.

After the conversion the visit sequences can contain a lot of repeating zones caused by multiple detections in the same zone. An example of this could be if a device was tracked at location 20, 21, and 22 which all refer to the zone named *EntranceT2* then the sequence would contain three entries for this specific zone. This results in longer sequences which again makes the pattern computation task more complex without supplying new information.

The sequences have to be cleansed for repeating zones. This is done in this step by only extracting the non repeating parts of the sequence. An example of this is the sequence $\langle l_1 l_1 l_1 l_2 l_1 \rangle$. In this sequence the step first extracts l_1 then skips the following l_1 elements, before extracting the l_2 element, and the subsequent l_1 element. The resulting sequence being $\langle l_1 l_2 l_1 \rangle$.

As an effect of the conversion into zones and the cleansing of the visit sequences, the *VisitSequenceLength*, *StartZone*, and *EndZone* attributes of each visit are updated accordingly.

The set of visits is now ready to be loaded and is enqueued for the Loader as well as the queue of tracking records from the previous subphase. This is the two steps shown as *Enqueue Visits* and *Enqueue Tracking Records* in Figure 12. The tracking records are not enqueued for the Loader in the previous subphase do to dependency with the visits.

In our running example approximately 11 elements are in average removed per visit, resulting in an average visit sequence length of 8.5 elements. The set of visits is now passed to the Compute Patterns step.

6.4.2 Compute Patterns

In this step all visit sequences are extracted from the set of visits and written to a sequence file. The sequence file is then given as input to the PrefixSpan implementation, along with the system specified support threshold. PrefixSpan computes and returns the set of all frequent patterns satisfying the given support threshold, in the sequence file.

In our running example we compute patterns from the visit seqenence set for one day at a time. This is done to simulate the case where data is importet on a daily basis.

The set of frequent patterns computed by PrefixSpan is extracted from the output file and given as an input to the Clean Patterns step.

6.4.3 Clean Patterns

This step is responsible for removing patterns, from the frequent pattern set, which do not provide significant information, i.e., other patterns exist which can provide approximately the same results. After some initial experiments with the Blip data set, we discovered a large amount of frequent patterns with repeating zones and zone pairs. On account of this observation, to reduce the amount of patterns, the following pattern cleansing rules are introduced.

Frequent patterns containing repeating zones and repeating zone pairs, e.g., the repeating zone $\langle l_1 l_1 \rangle$ and the repeating zone pair $\langle l_1 l_2 l_1 l_2 \rangle$, are removed from the pattern set. In this case *repeating* implies that no other zone elements exist in the frequent pattern between the repeating zone pairs. Given the Apriori property [1] a subsequence of the patterns will be present in the frequent pattern set, with equal or higher support.

Example 1 Two patterns, (a) $\langle l_1 l_1 l_1 l_1 \rangle$ and (b) $\langle l_1 l_2 l_1 \rangle$, containing the zone l_1. Pattern (a) will be removed because the zone is repeating, but pattern (b) will not.

Example 2 Two patterns, (a) $\langle l_1 l_2 l_1 l_2 \rangle$ and (b) $\langle l_1 l_2 l_3 l_1 l_2 \rangle$, containing the zone pair $l_1 l_2$. Pattern (a) will be removed because the zone pair is repeating, but pattern (b) will not.

The set of cleansed patterns is now updated with an pattern id for each pattern, so the patterns are of the form: (*ID*, *Pattern*). In our running example a total of 112,403 frequent patterns are computed with a support $\geq 1\%$. Of these only 7,850 pass this step. The set of patterns is then enqueued for the Loader and passed to the Transform Patterns subphase.

6.5 Transformation of Patterns

This section describes the steps needed to compute and load the index data for the three indexing approaches, presented in Section 5. Each step is responsible for computing data for one of the indexing approaches. Depending on which approach is selected in the ETL system, only the matching step is executed.

6.5.1 Initialise the VPA Approach

This step is responsible for computing the associations, in the `VisitPattern` table, for all patterns. The computations are described in Algorithm 4, and performed by iterating through all patterns, while searching for visit sequences containing the selected pattern. The set $VPAS$ of visit pattern associations, computed by the algorithm, is then enqueued for the Loader to be loaded into the data warehouse.

Algorithm 4: `VPAComputation(`VS`, `PS`)` computes and returns the set $VPAS$ of visit pattern associations given a set VS of visits, and a set PS of patterns.

```
1 foreach p ∈ PS do
2 |   foreach v ∈ VS do
3 |   |   if v_sequence contains p_pattern then
4 |   |   |   VPAS ← VPAS ∪ {(v_ID, p_ID)};
5 |   |   end
6 |   end
7 end
8 return VPAS;
```

6.5.2 Initialise the PCA Approach

This step is responsible for computing the daily pattern counts needed to form the `PatternDailyCount` table. The daily computation is performed, as described in Algorithm 5, by iterating through all dates represented in the set of visit. For each day the algorithm iterates though all patterns and counts the visits containing the specific pattern. This count is then inserted into the pattern daily count set PDC along with the date and the pattern id.

When finished, the set PDC of pattern daily counts is enqueued for the Loader to be loaded into the data warehouse.

6.5.3 Initialise the PAH Approach

This step is responsible for computing the daily pattern counts needed to form the `PatternDailyCount` table, and the visit pattern associations needed to form the `VisitPattern` table. The daily pattern count computation is performed as described in the previous Initialise the PCA Approach step.

As described in Section 5.4, concerning the PAH approach, we need to decide which patterns to associate and which to preaggregate. We decided to associate all patterns of length three with the visits containing them. This was decided since the patterns of length three in most cases represent a great reduction in the search space, and since most of the shorter patterns are already preaggregated. By associating patterns of length three the amount of rows in the PAH `VisitPattern` table ranges from 5.4 down to 2.2 million rows at a support thresholds of 1 to 10 percent respectively.

Algorithm 6 describing this step iterates through all dates represented in the visit set. For each day the algorithm iterates though all patterns and if the

Algorithm 5: `PCAComputation(`VS`, `PS`)` computes and returns the set PDC of pattern daily counts given a set VS of visits and a set PS of patterns.

```
1  D ← GetDates(VS) ;
2  foreach d ∈ D do
3  |  foreach p ∈ P do
4  |  |  pdc ← 0;
5  |  |  foreach v ∈ VS|v_StartDate = d do
6  |  |  |  if v_sequence contains p_pattern then
7  |  |  |  |  pdc ← pdc + 1;
8  |  |  |  end
9  |  |  end
10 |  |  if pdc > 0 then
11 |  |  |  PDC ← PDC ∪ {(p_ID, d, pdc)} ;
12 |  |  end
13 |  end
14 end
15 return PDC;
```

pattern is of length three the algorithm iterates through all visits of that day to calculate all visit pattern associations. These associations are then added to the set $VPAS$ of visit pattern associations. If the pattern is not of length three the algorithm iterates through all visits to count the amount of visit sequences that contain the specific pattern. This count it added to the set of pattern daily counts PDC along with the pattern id and the specific date.

When done, the set PDC of pattern daily counts and the set of visit pattern associations $VPAS$, is passed to the Loader to be loaded into the data warehouse.

6.6 Load Phase

The Loader is responsible for loading the extracted and transformed data into the data warehouse. This section provides a description of the different steps of the Loader shown in Figure 13.

Prepopulate Dimensions The Loader starts by populating the date, time, location, and Bluetooth device dimensions. The dimension information is bulk loaded into the data warehouse and hash tables containing the corresponding surrogate keys are maintained in main memory to speed up the loading of visits and tracking records.

The date dimension is populated to match the outer constraints of the data loaded and the time dimension is populated to match every second of a day. The location and its outrigger dimensions are populated according to the zone division presented in Section 3.3. The Bluetooth device dimension is populated with the Bluetooth devices that have produced tracking records in the extracted part of the data set.

Algorithm 6: `PAHComputation(`PS`, `VS`)` computes and returns the set PDC of pattern daily counts and the set $VPAS$ of visit pattern associations given a set VS of visits and a set PS of patterns.

```
1  D ← GetDates(VS) ;
2  foreach d ∈ D do
3  |  foreach p ∈ P do
4  |  |  if |p_Pattern| = 3 then
5  |  |  |  foreach v ∈ VS|v_StartDate = d do
6  |  |  |  |  if v_sequence contains p_Pattern then
7  |  |  |  |  |  VPAS ← VPAS ∪ {(v_ID, p_ID)};
8  |  |  |  |  end
9  |  |  |  end
10 |  |  end
11 |  |  else
12 |  |  |  pdc ← 0;
13 |  |  |  foreach v ∈ VS|v_StartDate = d do
14 |  |  |  |  if v_sequence contains p_Pattern then
15 |  |  |  |  |  pdc ← pdc + 1;
16 |  |  |  |  end
17 |  |  |  end
18 |  |  |  if pc > 0 then
19 |  |  |  |  PDC ← PDC ∪ {(p_ID, d, pdc)} ;
20 |  |  |  end
21 |  |  end
22 |  end
23 end
24 return PDC, VPAS;
```

Load Visit The set of visits enqueued for the Loader is updated with the correct Bluetooth device, locations and date surrogate keys, and bulk loaded into the data warehouse. When finished a hash table of the visit surrogate keys assigned by the data warehouse is created.

Load Tracking Records When the Loader receives a queue of tracking records this step updates each tracking record with the correct surrogate keys for date, time, location, classification and visit. When this is done the records are bulk loaded into the data warehouse.

Load Patterns The set of patterns enqueued for the Loader is bulk loaded into the data warehouse and a hash table containing the surrogate keys is created to speed up the loading of pattern counts and pattern associations.

Load Pattern Counts This step is responsible for loading the pattern counts into the data warehouse, if the PCA og PAH approach is selected, while maintaining the correct pattern surrogate keys.

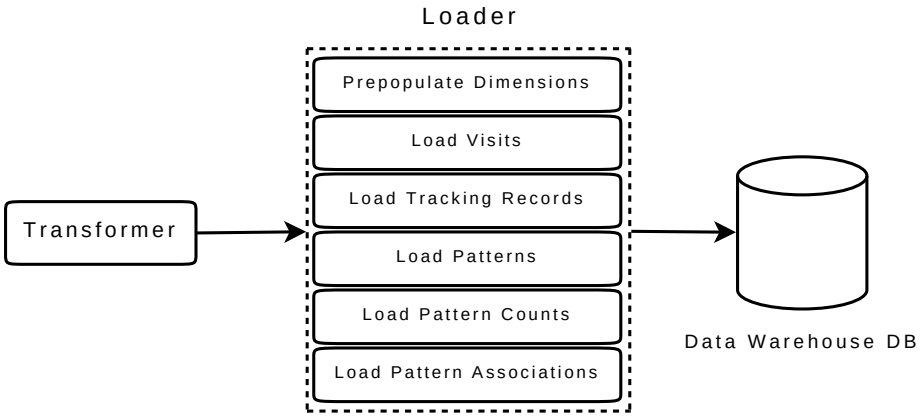


Figure 13: ETL Loader.

Load Pattern Associations This step is responsible for loading the visit pattern associations into the data warehouse if the VPA or PAH approach is selected. Before loading the associations, the pattern and visit ids are updated with the correct surrogate keys. When finished the associations are bulk loaded into the data warehouse.

7 FCQ Processor Implementation

In this section we present a brief overview of the implemented FCQ Processor prototype. First, the different OLAP cube designs are presented, to support each of the three indexing approaches. Second, we present a step by step description of the steps needed to process a Flow Count Query (FCQ) on the different OLAP cube designs.

7.1 OLAP Cube Design

The data warehouse schema presented in Section 4, combined with the schema extensions presented for each of the pattern indexing approaches, are used as the base for each of the OLAP cubes presented in the following paragraphs. The presented cubes only include the dimensions and measures required to answer a FCQ, as presented in Section 2.

In the current prototype the data warehouse is implemented in Microsoft® SQL Server® 2008 (MSSQL) and the OLAP cubes are built in Microsoft® Analysis Services using MOLAP.

VPA Cube The VPA Cube consists of the following three dimensions: `Dimension_Pattern` representing all patterns in the cube, `Dimension_Visit` representing all visits in the cube, and the `Dimension_Date` is indirectly a part of the cube, represented as the `StartDate` attribute in the visit dimension. The dimensions for the exceeding attributes in the visit dimension are not included in the current implementation, in order to reduce the size of the cube and thereby make it comparable to the other approaches. The cube contains a

measure (`Visit Pattern Count`), which is an aggregated count of the number of associations in the underlying `VisitPattern` fact table. This measure is used to compute the amount of visits for a given pattern, or vice versa, within the scope of the selected dimension.

PCA Cube The PCA cube consists of the following two dimensions: `Dimension_Pattern` representing the patterns and `Dimension_Date` representing the date on which the pattern count have been detected. The cube is built by combining the two dimensions using the `PatternDailyCount` fact table in the underlying relational database and aggregating the `VisitCount` values for each date as a `Visit Count` measure. In the current implementation the `VisitClassification` attribute is excluded, to reduce the size of the cube.

PAH Cubes The OLAP implementation of the PAH approach consist of two cubes, as indicated by the two fact tables shown in Figure 6. The **PAH-Association** cube is built using the `VisitPattern` fact table and is implemented using the same dimensions as the VPA cube. The **PAH-Aggregation** cube is built using the `PatternDailyCount` fact table and is implemented using the same dimensions as the PCA cube.

7.2 FCQ Processor Description

In the following we present the steps needed to process a FCQ on each of the different cubes presented in the previous section. In all three cases the input and the first steps are the same. The input to the processor is a query sequence qs and a time period.

In all steps the time period is used, as a selection criteria in the date dimension, to limit the results to the given time period. The following three steps are the same for all three processors:

1. Compute the supersequence set SS of qs, by generating all possible supersequences of qs with length $|qs| + 1$, and add qs to SS.
2. Extract the set P of patterns represented in the pattern dimension of the cube.
3. Extract the set PS of supersequences from SS that are present in the set P.

VPA FCQ Processor The following steps are specific to executing a FCQ on the VPA Cube.

4. The set of visit counts for the supersequences in PS is computed by selecting the matching patterns in the pattern dimension and by using the `Visit Pattern Count` measure to compute the amount of visits.
5. A set of patterns, each matching a subsequence of a supersequence left in SS, is selected from the pattern dimension. If multiple subsequences of a supersequence are represented as patterns in the pattern dimension, the pattern associated with the least amount of visits is chosen to minimise the search space as much as possible.

6. The set of visit sequences containing the patterns found in the previous step, is extracted from the visit dimension.

7. The set of visit sequences is then scanned for the supersequences left in SS and the amount of visit sequences containing each of the supersequences, is computed.

8. When done, all supersequences in PS and SS, and their visit counts, are returned as the answer to the FCQ.

PCA FCQ Processor The following steps are specific to executing a FCQ on the PCA cube.

4. The set of visit counts of the supersequences in PS is computed by selecting the matching patterns in the pattern dimension and by using the `Visit Count` measure to compute the amount of visits.

5. When done, all supersequences in PS, and their visit counts are returned as the answer to the FCQ.

PAH FCQ Processor The following steps are specific to executing a FCQ on the PAH cubes. The first three steps (1-3) are performed on the PAH-Aggregation cube.

4. The set of visit counts of the supersequences in PS is computed by selecting the matching patterns in the pattern dimension and by using the `Visit Count` measure of the PAH-Aggregation cube to compute the amount of visits.

5. The set PS is stored as $PS2$ and Step 2 and 3 are repeated using the PAH-Association cube to compute a new pattern set PS.

6. The set of visit counts of the supersequences in PS is computed by selecting the matching patterns in the pattern dimension and by using the `Visit Pattern Count` measure to compute the amount of visits.

7. The steps 5-7 of the VPA FCQ Processor are then executed on the PAH-Association cube.

8. When done, all the supersequences in PS, all the supersequences in $PS2$, the supersequences from SS that could be computed, and their visit counts, are returned as the answer to the FCQ.

The prototype application shown in Figure 14, displays a map of CPH airport, where each zone is marked with a circle and a check box. The idea is that an analyst will plot a query sequence qs by selecting one, or more check boxes, and perform the FCQ on the plotted sequence. The result computed by one of the above FCQ processors is then displayed on the map. As shown in Figure 14 the selected qs is $\langle$ *Security* $\rangle$ and the two counts on each zone is the percentage of visits *before (red)* and *after (green)* the selected qs respectively.

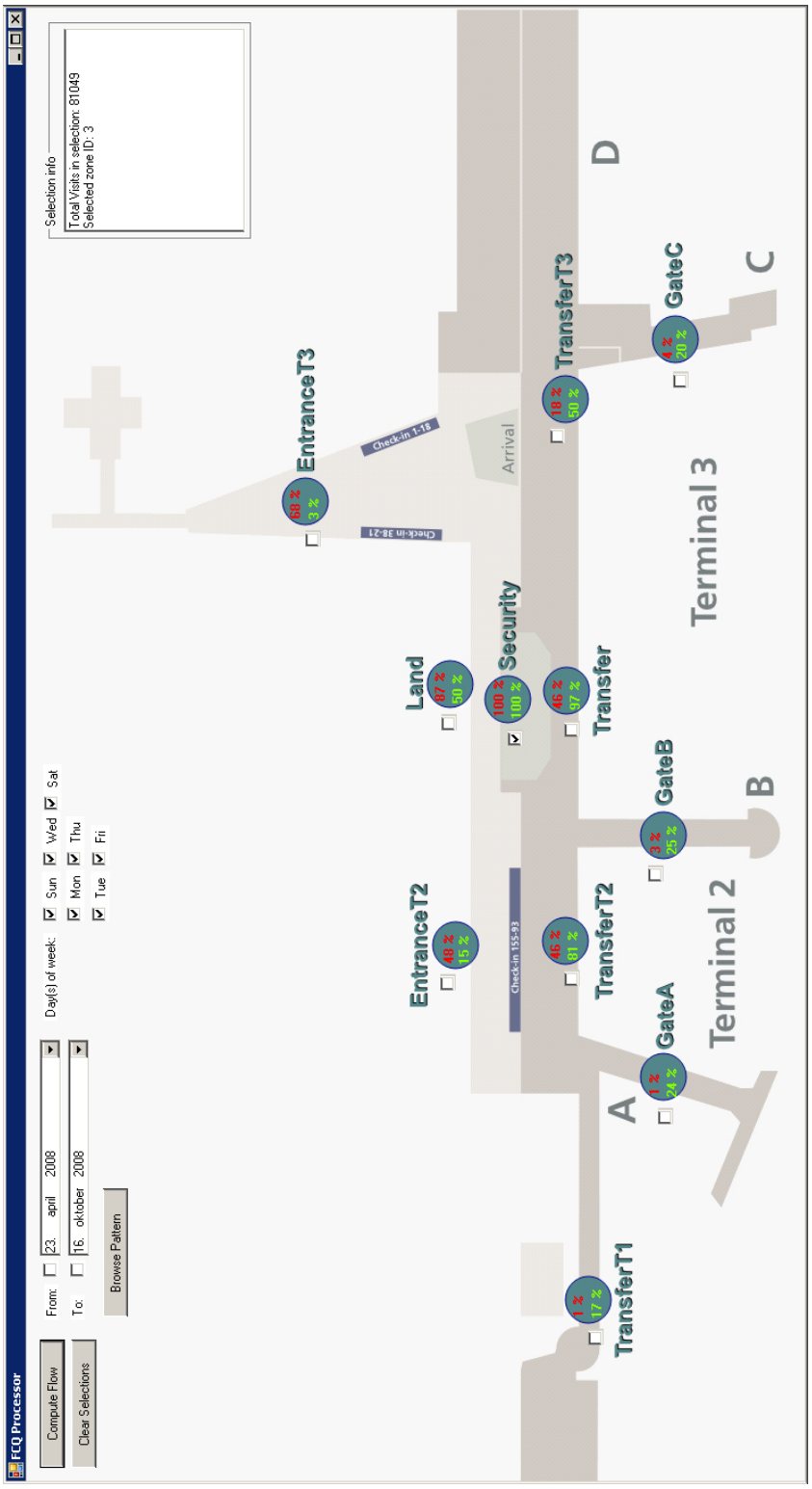


Figure 14: Screenshot of the implemented FCQ Processor prototype.

8 Experimental Study

In this section we present a series of experiments performed on the three indexing approaches presented in Section 5, using the implemented prototype of the FCQ Processor described in Section 7. The Blip data set presented in Section 3 is used as source data and an implementation of the ETL system presented in Section 6 is used to load the data set.

Section 8.1 describes the experimental settings, followed by Section 8.2 describing the recall experiment concerning the effectiveness of the three indexing approaches. Section 8.3 describes the experiments performed to measure the execution time of each approach, followed by Section 8.4 concerning the space usage of each approach.

8.1 Experimental Setting

All experiments are performed on a modern desktop computer, with an Intel® Core™2 Duo E6750 dual core CPU clocked at 2.66GHz and 8GB main memory, running Windows Server® 2008 64bit Edition.

The data used in the following experiments is the most complete part of the Blip data set, which is from the time period of April 24^{th} 2008 to May 24^{th} 2008, as described in Section 3. A total of 247,366 visits are registered in this period.

We have designed twelve query sequences for the experiments. Six frequent and six infrequent query sequences to study the performance of the three approaches in both scenarios. The query sequences are shown in Table 1 and Table 2 along with their support in the visit sequence database. The support is computed by scanning the visit sequences directly in the visit dimension of the data warehouse.

The twelve query sequences, both frequent and infrequent, are designed to: (1) represent a wide range of patterns in the data set, through an even distribution of support and length, and (2) simulate a series of queries that would be relevant for a visitor flow analyst in the airport.

	Support	**Query Sequence**
qs_1	47.47%	⟨ *Transfer* ⟩
qs_2	32.02%	⟨ *Security Transfer* ⟩
qs_3	28.02%	⟨ *Land Security Transfer* ⟩
qs_4	20.94%	⟨ *EntranceT3 Land Security Transfer* ⟩
qs_5	17.13%	⟨ *GateC* ⟩
qs_6	11.46%	⟨ *GateC TransferT3* ⟩

Table 1: The set of *frequent* query sequences used in the experiments. Support is the support of the sequence in the visit sequence database, and Query Sequence represents the query sequences as a sequence of Zone names.

The experiments are performed on 10 different OLAP cubes built on 10 different data warehouses with the Blip data set loaded using support thresholds ranging from 1% to 10%. This range of support thresholds is chosen to match the outer bounds of the infrequent query sequence set, and thereby showing the effect of having the sequences included in pattern dimension, or not.

	Support	Query Sequence
qs_7	9.36%	⟨ *EntranceT2 TransferT3* ⟩
qs_8	8.00%	⟨ *Land GateA* ⟩
qs_9	6.58%	⟨ *Security Transfer GateC* ⟩
qs_{10}	4.90%	⟨ *EntranceT2 TransferT2 TransferT1* ⟩
qs_{11}	2.76%	⟨ *EntranceT2 Security TransferT3 GateC* ⟩
qs_{12}	1.01%	⟨ *GateA TransferT2 EntranceT2* ⟩

Table 2: The set of *infrequent* query sequences used in the experiments. Support is the support of the sequence in the visit sequence database, and Query Sequence represents the query sequences as a sequence of Zone names.

8.2 Recall Study

A precision and recall study is commonly used to measure the effectiveness of an information retrieval system [18]. Precision and recall measures the systems ability to retrieve relevant information, and it is assumed that a higher precision and recall gives a more effective system.

The recall study performed, is designed to evaluate the quality of the results computed when using the three different indexing approaches. A precision study has been omitted since none of the approaches can return false positives and the precision is therefore given to be 100% in all approaches.

In the following experiments we measure recall as follows. The recall of an approach is the percentage of material actually retrieved in an answer to a FCQ.

We measure two types of recall: **Pattern Count** (PC) recall and **Visit Count** (VC) recall. The PC recall study measures the ratio of all relevant supersequences returned by the FCQ for a given query sequence:

$$\text{PC Recall} = \frac{PC_{Computed}}{PC_{Total}} \cdot 100 \tag{1}$$

$PC_{Computed}$ is the amount of supersequences computed by the FCQ Processor and PC_{Total} is the total amount of supersequences of a given query sequence.

The VC recall measures the ratio of visit counts (support) returned by the FCQ for a given query sequence.

$$\text{VC Recall} = \frac{VC_{Computed}}{VC_{Total}} \cdot 100 \tag{2}$$

$VC_{Computed}$ is the sum of the visit counts for the supersequences computed by the FCQ Processor. VC_{Total} is the total sum of visit counts for all supersequences of the given query sequence.

Recall Study Overview

In this section, the two types of recall (PC and VC) are evaluated in the following two scenarios using both the frequent and infrequent set of query sequences.

1. The FCQ Processor is set to compute **all supersequences** of the given query sequences. This experiment is performed to show the overall strength and weaknesses in the three approaches. The results of this experiment are presented in Section 8.2.1

2. The FCQ Processor is set to compute the **15 most frequent supersequences** (Top15). This experiment is designed to show a more realistic case where only the more statistically significant results are computed. We choose to do a Top15 experiment since, a Top20 would be too close to the total amount of supersequences computed for most of the query sequences, and a Top10 would produce too few supersequences to give a usable result. The results of this experiment are presented in Section 8.2.2

In this recall study the VPA approach is only included for comparison, and we only evaluate on the PCA and PAH approach, since the VPA approach gives a 100% recall at all the included levels of support threshold. This is due to the fact that VPA per definition finds a set of subsequences that are represented in the pattern dimension, and thereby is able to extract and scan the limited amount of visit sequences for all supersequences of the query sequence. The high recall of the VPA approach gives some penalties in the form of, longer execution time and a higher space usage. This is addressed in the later experiments regarding execution time and space usage.

8.2.1 Recall of All Supersequences

This section first presents the results of the experiment using the set of frequent query sequences, while computing all supersequences to answer a FCQ, and later presents the results using the infrequent query sequences.

All Supersequences of Frequent Query Sequences Figure 15 shows the average PC recall of each indexing approach, given the set of frequent query sequences. As seen in the figure the recall of PCA and PAH declines as the support threshold increases. This is due to the fact that PCA can only compute results for the supersequences that are included in the pattern dimension, and thereby aggregated. PAH produces a slightly higher PC recall than PCA, because it is able to scan for additional supersequences, by using the limited set of patterns that have been associated with the visit dimension.

The average VC recall for each approach is shown in Figure 16. This experiment shows that the VC recall of the PCA and PAH approach is significantly higher than the PC recall shown in the previous experiment result. This is due to the fact that most of the patterns missed by PCA and PAH in the PC recall study, are less frequent, and thereby have a low visit count.

This VC recall experiment shows that even though the PCA approach only computes counts for less than 70% of the supersequences (PC recall) at 1% support threshold, the most frequent supersequences are still included, giving a VC recall close to 100%. Considering the two recall studies, primarily the VC recall study, the PCA approach is considered applicable with a support threshold up to 5%, where the PAH approach is considered applicable using support threshold of up to 7%.

All Supersequences of Infrequent Query Sequences This study evaluates the performance of the three approaches, using the set of infrequent query sequences. The average PC recall for each approach is shown in Figure 17. As expected this experiment shows that the PCA approach is only efficient on the more frequent query sequences, since the recall starts below 40% at a support

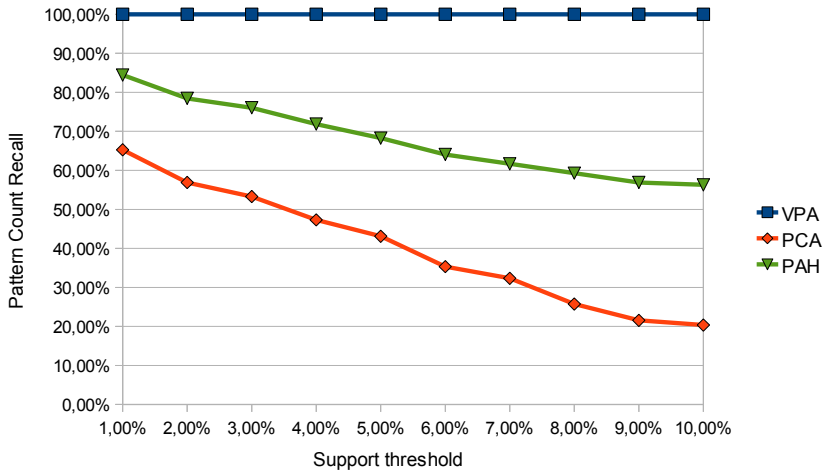


Figure 15: Average PC recall for all frequent query sequences, with all supersequences computed.

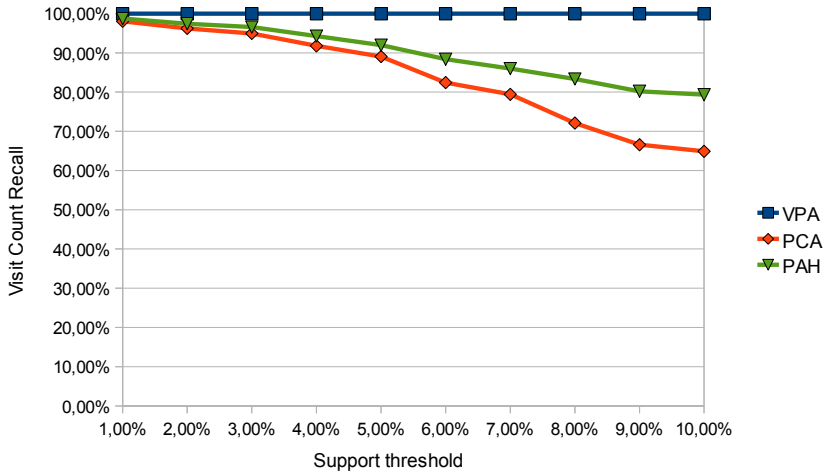


Figure 16: Average VC recall for all frequent query sequences, with all supersequences computed.

threshold of 1%. This experiment also gives a clear view of the benefits obtained by the visit associations in the PAH approach, since the PAH approach in most cases gives a much higher recall than the PCA approach. The figure shows that the recall of the PAH approach drops between support thresholds of 7% and 8%, this is due to the limited amount of patterns associated in the PAH approach when the support threshold gets higher. This shows that on average this is the point where the approach looses the associations needed for the visit extraction, on this set of query sequences.

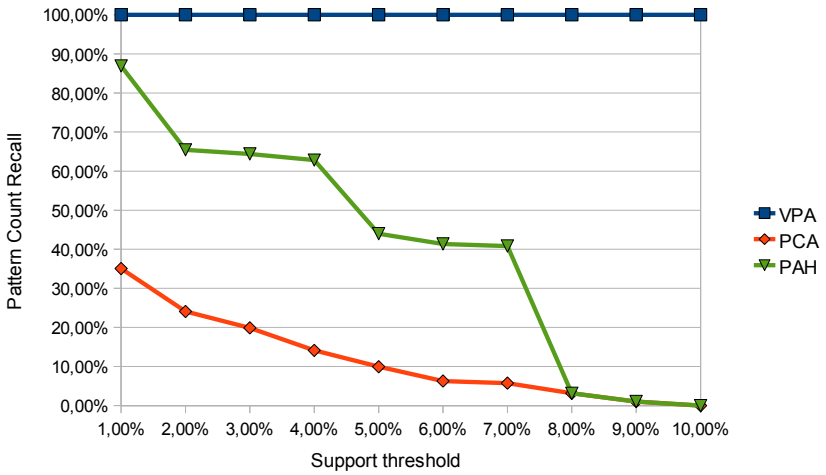


Figure 17: Average PC recall for all infrequent query sequences, with all super-sequences computed.

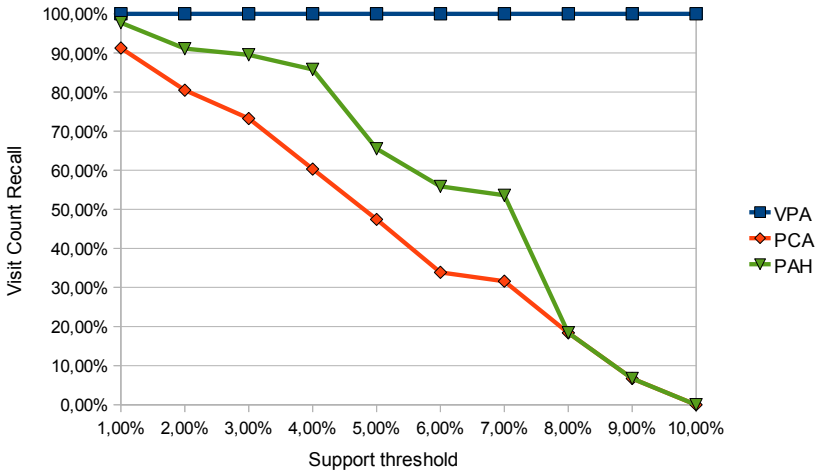


Figure 18: Average VC recall for all infrequent query sequences, with all super-sequences computed.

The average VC recall for each approach, using infrequent query sequences, is shown in Figure 18. This experiment again shows that the VC recall of the PCA and PAH approach is significantly higher than the PC recall.

The two experiments on this set of query sequences, shows that the PCA approach is only applicable when the support threshold is at 1%, where the PAH approach is applicable using support threshold of up to 4%.

8.2.2 Recall of Top15 Supersequences

This section first presents the results of the experiment on the set of frequent query sequences, while computing Top15 supersequences and later presents the results using the infrequent query sequences.

Top15 Supersequences of Frequent Query Sequences The Top15 PC recall experiment for each approach, using frequent query sequences, is shown in Figure 19. This study shows that both the PCA and PAH actually give a higher recall in regards to the statistically significant results (Top15). The PAH approach has the same tendencies as in the prior study, but with a increased recall. The PCA approach also gives a higher recall even though it declines faster than in the results of the experiment computing all supersequences.

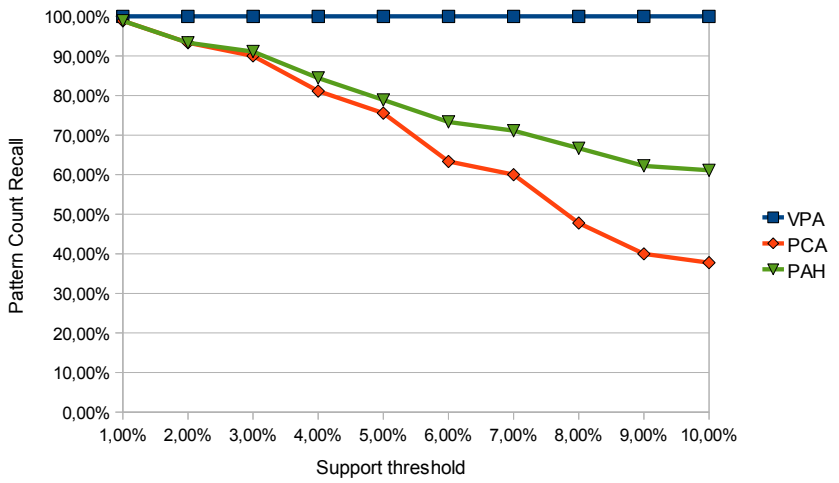


Figure 19: Average PC recall for all frequent query sequences, with Top15 supersequences computed.

The average VC recall for each approach using frequent sequence queries is shown in Figure 20. Again the experiment shows that the PCA and PAH approaches give a higher recall when only computing the statistically significant results (Top15).

Considering the two recall results presented, using frequent query sequences, the PCA and PAH approaches could be applicable for a support threshold of up to 6% and 8%, respectively.

Top15 Supersequences of Infrequent Query Sequences The Top15 PC recall for each approach using infrequent query sequences is shown in Figure 21.

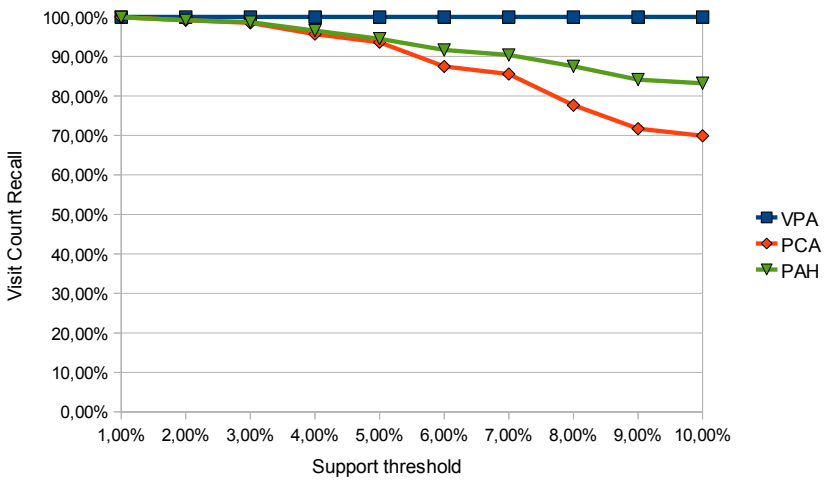


Figure 20: Average VC recall for all frequent query sequences, with Top15 supersequences computed.

The experiment shows that both the PCA and PAH gives a higher recall in regard to the statistically significant results (Top15). The PAH approach has the same tendencies as in the prior study regarding all supersequences, but with an increased recall. The PCA approach also gives a higher recall even though it declines faster than in the all supersequence study.

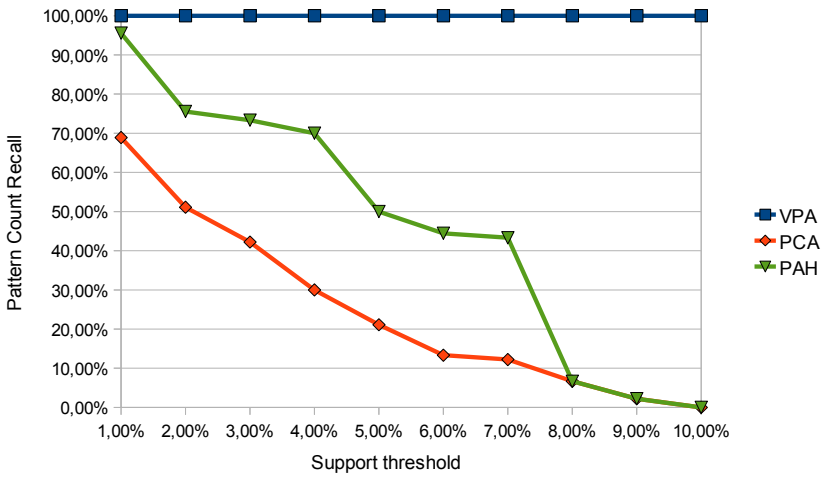


Figure 21: Average PC recall for all infrequent query sequences, with Top15 supersequences computed.

The average VC recall for each approach using infrequent sequence queries is shown in Figure 22. The experiment shows that the PCA and PAH approaches give a higher recall, when only computing the statistically significant results (Top15).

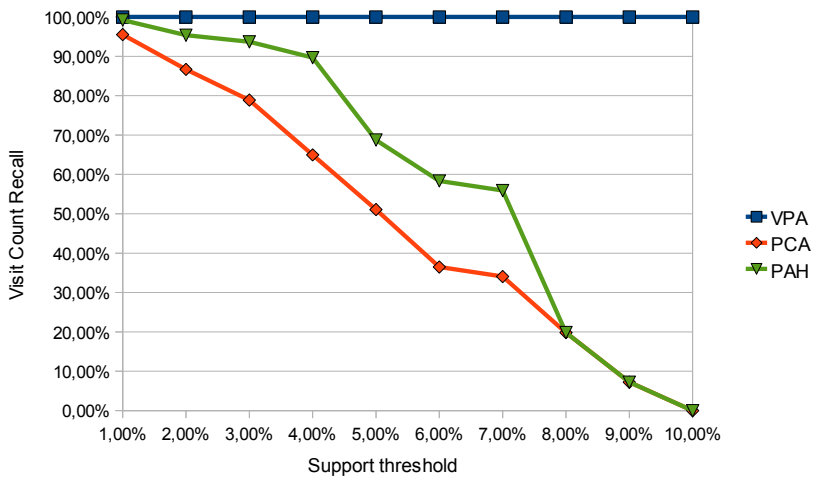


Figure 22: Average VC recall for all infrequent query sequences, with Top15 supersequences computed.

Considering the two recall results presented using infrequent query sequences, the PAH approach could still be applicable for a support threshold of up to 4%, but the PCA approach would only be applicable using a support threshold of up to 2%.

8.3 Execution Time

In this study we measure two types of execution time. The first experiment measures the performance of the three approaches compared to the support threshold. The second experiment measures the performance of the three approaches when querying on time periods of varying length.

Execution Time Compared to Support Threshold In this study we measure the average execution time of the three approaches on the ten cubes with support thresholds ranging from 1% to 10%. The execution time is the average time it takes to compute and return results for all supersequences of each of the twelve query sequences. The results are shown in Figure 23, and as expected it can be seen that the VPA approach is the slowest, followed by the PAH method. This is due to the fact that these two approaches utilise extraction and scan of a limited set of visit sequences. The PCA approach is significantly faster at all support thresholds. This is partly due to the fact that all results are precomputed and stored in the data warehouse, making the result computation less complex, and partly that the PCA approach has a relatively limited PC recall, when the support thresholds gets higher, which results in fewer queries to the cube than in the two other approaches.

Figure 23 shows that the VPA approach increases rapidly in execution time when the support threshold exceeds 7%. This shows the time penalty caused by extracting the increasingly larger set of visit sequences for the visit count computation. As a comparison, a total extraction of all visit sequences and a

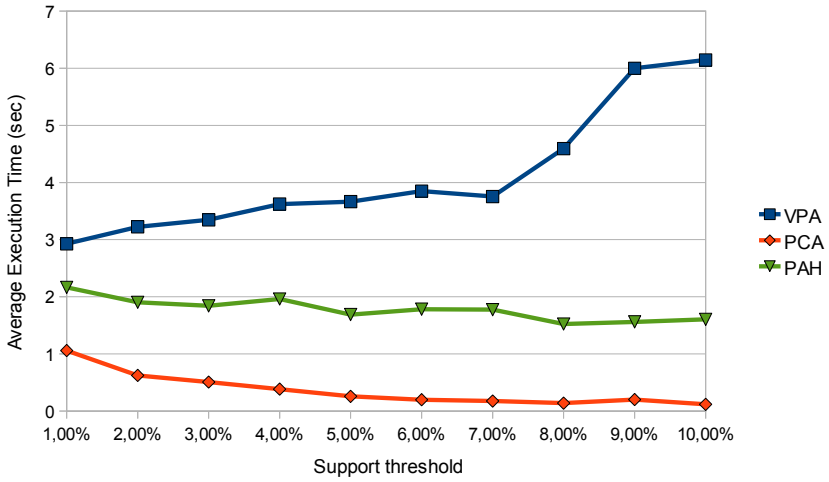


Figure 23: The average execution time, for the three approaches, when executing the query sequences on the data warehouse loaded with a varying support threshold.

linear scan for all supersequences for each query sequences executes on average in 20 seconds, making the VPA approach more than 70% faster, even at 10% support threshold. This result is excluded from the graph to improve readability.

Considering the execution time results presented, the PCA and PAH approach are considered applicable for all the presented support thresholds, but the VPA approach would only be applicable using a support threshold of up to 7%.

Execution Time on Increasing Time Periods In this study we measure the performance of the three approaches, when the query time period increases. This is done by measuring the average time it takes the three methods to compute and return results, for all query sequences on all support thresholds (from 1% to 10%), using different query periods ranging from 1 to 4 weeks.

Figure 24 shows the effect the time period has on the average execution time of each indexing approach. The figure shows that the PCA approach has an almost constant average execution time, where the PAH and VPA approaches increases when the query period increases. The figure again shows the VPA approaches time penalty in extracting the increasingly growing visit sequence set. Even though the results only range up to 3,5 seconds the experiment still shows that the VPA grows fast when the query period increases. The study indicates that, if the query period is multiplied up to one year, the VPA approach would use approximately 32 seconds to compute the results, where the PAH approach only would use approximately 12 seconds. Based on these observations we conclude that, the VPA approach does not scale as well as the PAH and PCA approaches.

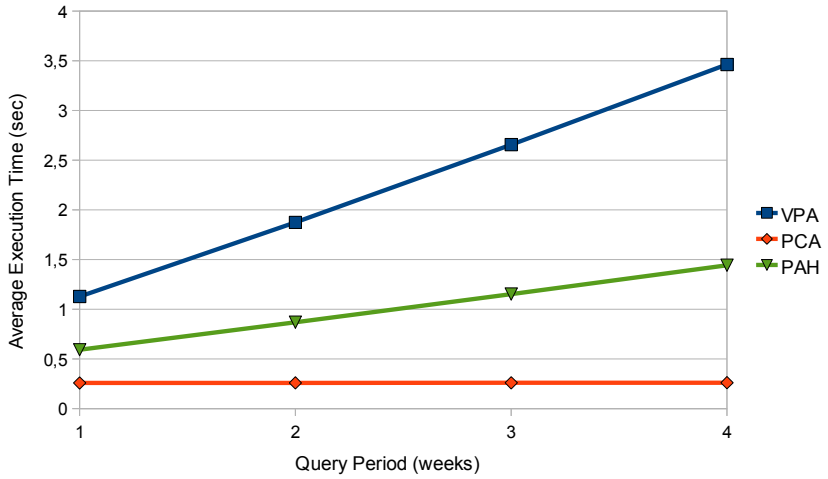


Figure 24: The average execution time, for the three approaches, when executing all query sequences at all support thresholds (from 1% to 10%), with query time periods ranging from 1 to 4 weeks.

8.4 Space Usage

In this study we measure the space usage of the three indexing approaches to see how well the different approaches scale compared to the amount of tracking records.

To measure the space usage of the three approaches we introduce a measure called **Row Measure**. As described next the **Row Measure** is computed differently on each approach to make the result comparable:

VPA Row Measure The row measure of the VPA approach is the amount of rows in the fact table `VisitPattern`. These rows contain two integer attributes: `PatternID` and `VisitID`.

PCA Row Measure The row measure of the PCA approach is the amount of rows in the fact table `PatternDailyCount` multiplied by 2 since this fact table contains four integer attributes: `PatternID`, `Date`, `VisitCount`, and `VisitClassification`, assuming the classification has been implemented as a classification dimension to avoid having a string in the fact table.

PAH Row Measure The row measure of the PAH approach is computed by adding: The amount of rows in the visit association fact table, that contains two integer attributes: `PatternID` and `VisitID`, with the amount of rows in the pattern count fact table, multiplied by 2 since this table contains four integer attributes: `PatternID`, `Date`, `VisitCount` and `VisitClassification`. Assuming the classification has been implemented as a classification dimension to avoid having a string in the fact table.

The row measure of each approach is shown in Figure 25. As shown, the VPA approach uses a huge amount of space at the low support thresholds, whereas both the PCA and PAH approaches have a more reasonable space usage, even at 1% support threshold.

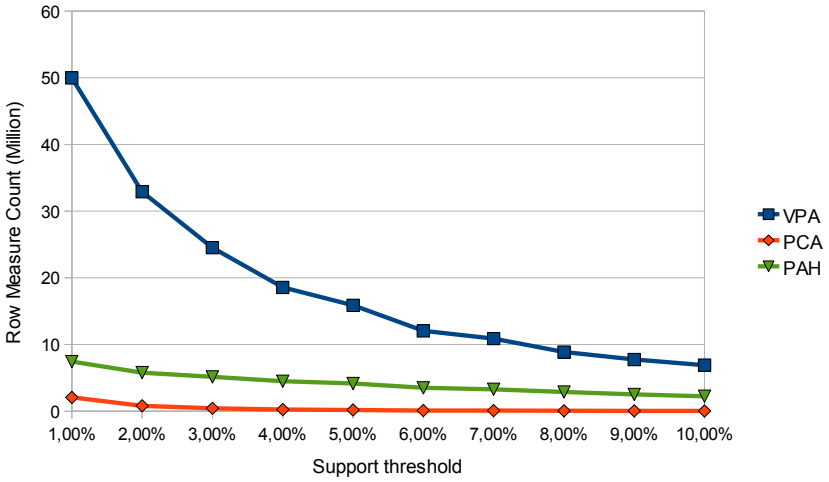


Figure 25: Space usage in row measure count for each of the three approaches, at the different support thresholds.

Comparing the space usage of the three approaches to the amount of tracking records in the `Fact_Tracking` fact table, gives the following information: The current `Fact_Tracking` table in the data warehouse includes a total of 4.8 million tracking records. This shows that the VPA grows at a rate of more than ten rows per new tracking record, loaded into the system, at 1% support threshold. With this taken into consideration, the VPA approach is only considered feasible at a support threshold higher than 6% where the space usage is increasing at a rate of less than three rows per tracking record. The space usage of the PCA and PAH approaches is significantly lower than the VPA and is therefore considered feasible even at a support threshold of 1%.

8.5 Experimental Study Conclusion

This section describes our conclusions on the three indexing approaches with regards to the experiments performed in the previous sections. We discuss the pros and cons of the three approaches in regard to our observations in the experiments.

VPA Indexing Approach The strength and weakness of the VPA approach is clearly the fact that all patterns are associated with all visits containing them. This results in a 100% recall on all FCQ's, but it also demands a lot of space to be used for these associations. As shown in the space usage study the VPA approach is only considered feasible when the support threshold is set to 6% or more. The study regarding execution time shows that the VPA approach performs the best when the support is set to 7% or less. The execution time study also shows that the VPA approach does not scale as well as the two other approaches, since the average execution time used to compute the answer to an FCQ increases with a little less than a second, per week in the query time period. This does not sound as much, but as the query time period increases, this will become an issue.

The experimental study shows that the VPA approach is most suitable in applications where a 100% recall is required, and either space usage or execution time is not an issue. By adjusting the support threshold, the approach can perform fast with a high demand for space usage in the data warehouse, or perform slower with a more moderate space usage in the data warehouse.

PCA Indexing Approach The strengths of the PCA approach is the fast FCQ computation and the relatively small amount of space used. This can be seen in the execution time study, where this approach is not noticeably affected by the increasing query time period. The study concerning the space usage shows that, the PCA approach could easily be aggregated on a lower support threshold, which would improve the recall.

The weakness of the PCA approach is the varying recall. The recall study shows that, if the support threshold is set to 1%, then the PC recall ranges from 35% to almost 100%, and the VC recall ranges from 90% to 100%.

The experimental study shows that the PCA approach is most suitable in applications where a 100% recall is not required and where the VC recall is considered more important than the PC recall. Since the PCA approach is so fast and space efficient, a support threshold even lower than 1% can be considered.

PAH Indexing Approach The PAH approach provides a faster FCQ computation than the VPA approach and with a higher recall than the PCA approach.

The recall study shows that if the support threshold is set to 1% then the PC recall ranges from 85% to almost 100% and the VC recall ranges from 97% to 100%.

Like the PCA the PAH approach could be utilised with an even lower support threshold than 1%, since it would not demand an unacceptable amount of space. This would increase the amount of queries that can be answered, by utilising the preaggregated counts, which in turn would result in a higher recall and execution time.

The weakness of the PAH approach is, as introduced in Section 5.4, how to determine which patterns to include in the visit pattern associations.

As seen in this experimental study the PAH approach is most suitable in application where a close to 100% recall is prefered and where the VC recall is considered more important than the PC recall. The decision of which patterns to associate is then a factor that should be adjusted depending on the domain in which the approach is used.

9 Conclusion

The current work presents a data warehouse solution for flow analysis on indoor spatiotemporal tracking data collected by Blip Systems (Blip) at Copenhagen airport. The solution is designed to be implemented in an RDBMS and to support the use of OLAP tools, e.g., for statistical analysis of the tracking data. We present a prototype implementation of a flow count query (FCQ) processor that, utilising an OLAP cube generated on the data warehouse solution, is able to answer queries on flow information.

We design a multidimensional database schema and an ETL system capable of extracting, transforming, and loading the Blip data set into the data warehouse. The ETL system is designed to only extract complete and meaningful data and to use the extracted data to compute visit sequences utilised in the flow analysis. The transformation phase of the ETL system passes the data through a series of data cleansing steps, where noisy and faulty data, generated by the tracking system, and the total amount of tracking records are reduced. The current implementation of the ETL system extracted a total of 12.2 million tracking records from the Blip data set and reduced this amount with approximately 60%, down to 4.8 million tracking records. The remaining tracking information is divided into visits, that represent visitors at the airport. The tracking information of each visit is combined into sequences representing the locations traversed during the visit at the airport.

We propose to utilise the PrefixSpan sequential pattern mining algorithm to compute frequent patterns in the set of detected visit sequences. We present the following three different approaches, to store and index the frequent patterns utilised to answer an FCQ: The Visit Pattern Association (VPA) approach is designed to associate all frequent patterns computed, with the visits that contain the given pattern. The Pattern Count Aggregation (PCA) approach is designed to preaggregate the amount of visits containing the frequent patterns at a predefined time interval. The Pattern Aggregation Hybrid (PAH) approach is designed, as a combination of the two prior approaches, to utilise both visit pattern association and visit count preaggregation.

An experimental study is performed using a prototype implementation of the presented FCQ Processor, to study the recall, execution time, and space usage of the three approaches. The study shows that each approach is applicable in different applications depending on the requirements on recall, execution time, and space usage. Observations made through the experimental study indicates that the VPA approach is only applicable using a support threshold of 6% to 7%, where both the PCA and PAH approach are applicable at a support threshold of 1% or less. The VPA approach provides 100% recall on every sequence query with the penalty of increased execution time and space usage. The results at the previously mentioned support thresholds shows that the VPA approach has an approximately 400% longer execution time and a space usage approximately 600% higher than the PCA approach. The PCA approach is storage efficient and provides fast execution times, at the cost of a reduction in recall. The recall study of the PCA approach shows a visit count recall ranging from 90% to 100%, and a pattern recall ranging from 35% to almost 100%, using a support threshold of 1%. The PAH approach preserves the strengths of each approach at the cost of being more domain specific, i.e., the PAH approach needs to be customised to fit the application in which it is applied.

10 Future Work

A relevant subject for further research is the possibility of developing a generic version of the PAH approach to avoid the domain specialisation. This could enhance the possibility of providing a solution with a close to 100% recall, with a low execution time and space usage. Another relevant subject is the utilisation of virtual fact tables, in the generation of the OLAP cube, e.g., by introducing a dynamic view that computes the visit pattern associations of the VPA approach, instead of storing precomputed results. This would reduce the space usage in the data warehouse significantly, but at the cost of performance, when querying associations in the virtual table.

The following issue related to the current design of the approaches presented in Section 5, is left for future work. Depending on how the data is loaded into the data warehouse and how the frequent patterns are computed, it is likely that new frequent patterns will appear over time, and some existing patterns will cease to be frequent. This introduces a problem of deciding which patterns should be included, when updating the association and aggregation tables in the indexing approaches.

This issue could arise when loading visits into the data warehouse on a daily basis. On the first few days the same set of frequent patterns are found in the visits, but after a while some new patterns appear in the visit sequences. In this case there are two options: The new patterns are associated with the visits; (1) from the day they appear and all subsequent visits, or (2) all past, current, and subsequent visits. Option 2 is clearly the best possible solution, given the accuracy of results, when performing queries on patterns over time. Searching through all previous visits in the warehouse, on the other hand, can become an infeasible option, given the amount of previous data. Option 1 is faster to maintain, but more complex to query. It would require a *first seen* attribute on the pattern dimension, indicating from which point in time the pattern is included. This is simple to maintain in the ETL system, but is more complex to include in the results computation of a FCQ. Given the scope of the current work, this issue has not been further studied and is thereby left for future work.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*. Morgan Kaufmann, 1994.

[2] Alexandra Instituttet A/S. Spopos project. `http://www.spopos.dk`, 2007, Accessed: 01/06/2009.

[3] P. Bahl and V. N. Padmanabhan. RADAR: an in-building RF-based user location and tracking system. 2000.

[4] Sudarshan S. Chawathe, Venkat Krishnamurthy, Sridhar Ramachandran, and Sanjay Sarma. Managing RFID data. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004.

[5] Silke Feldmann, Kyandoghere Kyamakya, Ana Zapater, and Zighuo Lue. An indoor bluetooth-based positioning system: Concept, implementation and experimental evaluation. In *International Conference on Wireless Networks*, 2003.

[6] Győző Gidófalvi and Torben Bach Pedersen. Mining long, sharable patterns in trajectories of moving objects. *Geoinformatica*, 2009.

[7] Hector Gonzalez, Jiawei Han, and Xiaolei Li. Flowcube: constructing RFID flowcubes for multi-dimensional analysis of commodity flows. In *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006.

[8] Hector Gonzalez, Jiawei Han, Xiaolei Li, and Diego Klabjan. Warehousing and analyzing massive RFID data sets. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, Atlanta, GA, USA*. IEEE Computer Society, 2006.

[9] Jonas T. Hansen, Stig Jørgensen, Simon Nicholas M. Tinggaard, and Rune L. Wejdling. Pre-master Thesis: A Data Warehouse Solution for Analysis on Indoor Tracking Data, 2008.

[10] Illimine - partially open-source data mining package. `http://illimine.cs.uiuc.edu`, 2006, Accessed: 05/18/2009.

[11] Christian S. Jensen, Hua Lu, and Bin Yang. Indexing the trajectories of moving objects in symbolic indoor space. *11th International Symposium on Spatial and Temporal Databases*, 2009.

[12] Christian S. Jensen and Torben B. Pedersen. Multidimensional databases and OLAP. *Submitted to J. Hammer and M. Schneider (Ed.s): Handbook of Database Technologies, CRC Press, forthcoming*, 2007.

[13] Ralph Kimball and Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons, 2004.

[14] Xiaolei Li, Jiawei Han, Jae-Gil Lee, and Hector Gonzalez. Traffic density-based discovery of hot routes in road networks. In *Advances in Spatial and Temporal Databases, 10th International Symposium, SSTD 2007, Boston, MA, USA, July 16-18, 2007, Proceedings*. Springer, 2007.

[15] Eric Lo, Ben Kao, Wai-Shing Ho, Sau Dan Lee, Chun Kit Chui, and David W. Cheung. OLAP on sequence data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2008. ACM.

[16] Jian Pei, Jiawei Han, Behzad Mortazavi-asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering*, Washington, DC, USA, 2001. IEEE Computer Society.

[17] Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, and Umeshwar Dayal. Multi-dimensional sequential pattern mining. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, New York, NY, USA, 2001. ACM.

[18] Cornelis J. Van Rijsbergen. *Information Retrieval*. Butterworths, 1979.

[19] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating association rule mining with relational database systems: alternatives and implications. *SIGMOD Rec.*, 1998.

[20] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database Systems Concepts*. McGraw-Hill, fifth edition, 2006.

[21] Roy Want. *RFID explained: a primer on radio frequency identification technologies*. Morgan & Claypool Publishers, 2006.

[22] Blip Systems - Bluetooth Marketing Solutions. `http://www.blipsystems.com/`, 2008, Accessed: 04/15/2009.

[23] Copenhagen Airports A/S. `http://www.cph.dk/`, 2008, Accessed: 04/15/2009.

A Blip Data Set Zone Division Table

Location	Zone	Area	Site
1	TransferT2	Transfer	Copenhagen
2	TransferT1	Transfer	Copenhagen
3	GateA	Transfer	Copenhagen
4	TransferT2	Transfer	Copenhagen
5	TransferT2	Transfer	Copenhagen
6	GateB	Transfer	Copenhagen
7	Transfer	Transfer	Copenhagen
9	Transfer	Transfer	Copenhagen
11	TransferT3	Transfer	Copenhagen
12	GateC	Transfer	Copenhagen
13	TransferT3	Transfer	Copenhagen
15	TransferT3	Transfer	Copenhagen
16	Security	Transfer	Copenhagen
17	Land	Land	Copenhagen
18	Land	Land	Copenhagen
19	Transfer	Transfer	Copenhagen
20	EntranceT2	Land	Copenhagen
21	EntranceT2	Land	Copenhagen
22	EntranceT2	Land	Copenhagen
23	EntranceT3	Land	Copenhagen
24	Land	Land	Copenhagen
26	EntranceT3	Land	Copenhagen
999	GateA	Transfer	Copenhagen
1001	EntranceT2	Land	Copenhagen

Table 3: Table of Locations in CPH, showing their respective Zone and Area divisions. For simplicity the "`ms-spopos1.`" prefix is removed from the location names.

B Summary in Danish

Vi har under udarbejdelsen af denne kandidatafhandling, arbejdet inden for feltet analyse af personers bevægelse i indendørs arealer, for eksempel i en lufthavn. I denne forbindelse har vi samarbejdet med Blip Systems A/S (Blip), en nordjysk virksomhed, der har specialiseret sig i løsninger der benytter Bluetooth teknologien. Blip har i samarbejde med Copenhagen Airports A/S, implementeret et Bluetooth baseret sporingssystem, der overvåger besøgende i Køben- havns lufthavn (CPH). Overvågningen forgår ved at spore de besøgendes Bluetooth enheder, ved hjælp af målestationer placeret udvalgte steder i lufthavnen. Dette sporingssystem generer dagligt en stor mængde måledata, der kan benyttes til flere forskellige analyseopgaver. Eksempler på sådanne analyseopgaver inkluderer, analysering af køtider i forskellige dele af lufthavnen, og analyse af de besøgendes bevægelses mønstre.

I afhandlingen præsenterer vi et Data Warehouse (DW) design, som understøtter lagring og analyse af måledata, genereret af Blips sporingssystem, med henblik på analyse af de besøgendes bevægelsesmønstre. Strukturen i det præsenteret DW design er udviklet til at understøtte statistisk analyse af de store mængder data, ved hjælp af Online Analytical Processing (OLAP) værktøjer.

I afhandlingen præsenteres et (ETL) system der er designet til at tilpasse Blips måledata til strukturen i det præsenterede DW. Systemet eliminere ugyldigt data, reducere mængden af overflødigt data og detekterer *besøg* i lufthavnen. Et besøg definerer vi som en mængde måledata, der beskriver en Bluetooth enheds færden i en sammenhængende periode i lufthavnen. Et besøg dækker dermed over en persons tilstedeværelse og færden i lufthavnen.

Vi definerer en Flow Count Query (FCQ) som den forespørgsel, vi vil kunne besvare ved hjælp af vores DW: Givet en sekvens af lokationer og en tidsperiode, beregner en FCQ antallet af besøg der indeholder den forespurgte sekvens og hvor de er set, både før, efter og ind i mellem lokationerne i den forespurte sekvens.

Vi forslår at benytte Sequential Pattern Mining til at beregne ofte fremkomne mønstre i de detekterede besøg i lufthavnen. Vi præsenterer de følgende tre forskellige løsninger på hvorledes mønstrene kan lagres i det præsenterede DW, samt hvordan de benyttes i besvarelsen af en FCQ:

Visit Pattern Association (VPA) VPA metoden benytter associationer mellem de beregnede mønstre og de besøg mønstrene opstår i. Disse associationer forudberegnes og lagres, som bindinger i en tilføjelse til det præsenterede DW. Associationerne benyttes derefter til at beregne hvor mange besøg der indeholder et udvalgt mønster. Hvis dette mønster er blandt de forudberegnede ofte fremkomne mønstre, beregnes besøgs antallet direkte ved at tælle dets associationer. Hvis mønstret ikke er forudberegnet, udtrækkes en mængde af besøg, som med stor sandsynlighed indeholder mønstret, givet Apriori egenskaberne. Hvorefter der foretages en liniær søgning efter mønstret i denne mængde.

Pattern Count Aggregation (PCA) PCA metoden forudberegner værdierne for hvor mange besøg der indeholder et specifikt mønster for eksempelvis en dag. De forudberegnede værdier lagres i en udvidelse til det præsenterede DW. Antallet af besøg der indeholder et specifikt mønstre

kan herefter hentes direkte, ved at summere dagsværdierne op for den givne periode.

Pattern Aggregation Hybrid (PAH) PAH metoden er en hybrid af de to andre metoder (VPA og PCA) i det at metoden benytter både forudberegnede associationer og opsummerede værdier. Metoden kan derved benytte de opsummerede værdier, så længe et mønster er forudberegnet. Hvis et mønster ikke er forudberegnet kan associationerne bruges til at udtrække en mængde af besøg, til linær søgning, i samme stil som i VPA metoden.

Vi præsenterer og implementerer et prototype system der kan benytte de tre metoder til at beregne resultatet til en FCQ. Prototypen danner grundlag for en evaluering af de tre metoder hvor effektiviteten, tidsforbruget og pladsforbruget af hver metode vurderes.

Vi konkludere at de tre metoder har styrker inden for hvert deres felt, da VPA metoden altid giver et komplet resultat, på bekostning af et stort pladsforbrug og lang eksekveringstid. PCA metoden giver hurtige resultater og benytter betydeligt mindre plads end VPA metoden, dog på bekostning af et ikke helt komplet resultat. PAH metoden ligger imellem de to andre og giver derfor et bedre resultat end PCA ved brug af mindre plads og tid end VPA. Dog på den bekostning af metoden skal tilpasses det domæne hvori den benyttes.