MASTER

Performance-aware conformance checking on material handling systems
visualization methodology of performance of routes in a baggage handling system

Turu Pi, A.

Award date:
2019

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
Performance-aware Conformance Checking on Material Handling Systems

Visualization methodology of performance of routes in a Baggage Handling System

Master thesis

Anna Turu Pi

Supervisors:
Dr. Dirk Fahland
Ir. Hilda Bernard
Ir. Koen Verhaegh

Eindhoven, July 2019
Abstract

Conformance Checking is concerned with finding deviations between a process model and event data. Either the process model or the log data is considered the ground truth, and the goal of Conformance Checking is to model the process closer to reality or detect unexpected deviations in the process. Our goal is to apply a Conformance Checking methodology on a Baggage Handling System. This is a logistics system composed of mainly a network of conveyors, but also some manual processes. The goal is to detect the routes taken in the system and perform Conformance Checking to detect performance deviations. One of the challenges encountered was the lack of a process model available, which required to propose a new technique to do Deviation Detection. Another challenge encountered was incomplete event data which hindered detecting route variants.

The method we propose converges the Process Mining research and the Graph Database technologies. Having incomplete routes, and having the ground truth about the system structure, we applied Pattern Matching techniques to complete the event traces. Having complete routes, our method consisted of comparing the performance of the routes variants with the ideal variants, which were user-defined by domain knowledge. Lastly, we propose Information Visualizations to present the performance of the process variants to the user. Since the process variants are physical routes within a Baggage Handling System, we provide the process variants in a network over the system’s layout. Our method has been tested on a Baggage Handling System and evaluated by process analysts to verify that the information obtained answers their use cases and gives a level of detail of the performance of the routes that were not available before.

With this project, we contribute to the transparency of Data Visualization. We propose a new technique that includes verification in Data Mining. We provide a methodology for transparent and complex data analysis. Our Process Mining methodology includes different levels of aggregation to also provide access to fine-grained data to the user. Our methodology also allows for user input in customizing the data visualization. In conclusion, the goal is not to provide a black box, e.g. a machine learning model, but a transparent data visualization.

Keywords: Process Mining, Conformance Checking, Deviation Detection, Property Graphs, Graph Databases, Visual Analytics, Transparent Process Mining
Preface

This master thesis is the result of my specialization in Process Mining Analytics at Eindhoven University of Technology within the Mathematics and Computer Science department and in collaboration with a Material Handling Systems company. I tremendously appreciate all the knowledge acquired in this two-year Master of Science in Big Data Management and Analytics.

My background was a four year Bachelor of Science in Industrial Management Engineering, and I find that the Process Mining specialization was the perfect continuation for my studies. I have enjoyed working with a Material Handling Systems company and I appreciate all that I have learned about logistics during the graduation project.

First of all, I would like to thank Dirk Fahland as my supervisor for providing me with the opportunity to work on this project. I am extremely thankful for his thoughtful and continuous guidance throughout the graduation project and all his support, which led to successful results.

Likewise, I am grateful to Hilda Bernard and Koen Verhaegh for their continuous support and dedication supervising my progress. I also want to thank Vadim Denisov for his previous work in Material Handling Systems and his support on my project. Then I would like to thank my colleagues for being so welcoming and providing such a positive working environment, and my manager for providing me with the opportunity to work in the team. Special thanks to Özge Köroğlu and Marie Heinrich for their companionship throughout our graduation projects.

Also, I am sincerely grateful to my fellow students and friends for their positive attitude and help during our studies, especially during the last two years where I have learned a lot from them. Last but not least, I would like to thank my family for their support in continuing my studies. All the effort was worth reaching this point.

Anna Turu Pi

Eindhoven,

Contents

Contents vii
List of Figures xi
List of Tables xiii
Listings xv

1 Introduction 1
1.1 Motivation and Context ................................. 1
  1.1.1 Business Motivation ................................ 1
  1.1.2 Scientific Motivation ................................ 2
1.2 Research Problem ......................................... 2
1.3 Methodology .............................................. 3

2 Preliminaries 5
2.1 Process Mining ............................................. 5
  2.1.1 Event Logs ........................................... 6
  2.1.2 Process Mining Techniques ........................... 6
2.2 Semantic Data Management ................................. 9
  2.2.1 Property Graphs ..................................... 10
  2.2.2 Graph Databases ..................................... 10
  2.2.3 Graph Query Language ............................... 11
  2.2.4 Towards a Standard Graph Query Language ......... 12

3 Use Case 13
3.1 Domain of Baggage Handling Systems ....................... 13
  3.1.1 BHS as Complex Systems .............................. 13
  3.1.2 High-level Process .................................. 13
  3.1.3 Low-level Process .................................. 15
  3.1.4 Layout Graph ....................................... 16
  3.1.5 Routing ............................................ 18
  3.1.6 Lack of Process Model ............................... 21
3.2 Problem Description ....................................... 21
  3.2.1 Examples of Process Variants ....................... 22
  3.2.2 User Objectives .................................... 24
  3.2.3 Benefits for the User ................................ 25
  3.2.4 Business Problem Definition ....................... 26
3.3 Overview of Data Sources .................................. 26
  3.3.1 Data Source for Bag Traces .......................... 26
  3.3.2 Lack of a Business Process Model ................... 26
3.4 Data Understanding ........................................ 28
  3.4.1 Event Logs Storage .................................. 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.2</td>
<td>Routing Information</td>
<td>32</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Property Graph Definition of the Routing Database</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>Data Science Problem</td>
<td>35</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Performance-aware Conformance Checking (PA-CC)</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>Definition of Technical Sub-problems</td>
<td>37</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Improving Data Quality for Bag Traces</td>
<td>37</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Transparent Process Mining</td>
<td>38</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Information Visualization</td>
<td>38</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Problem Definition</td>
<td>38</td>
</tr>
<tr>
<td>3.6.5</td>
<td>Analytics Objectives</td>
<td>39</td>
</tr>
<tr>
<td>3.6.6</td>
<td>Scope Limitations</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>Solution proposal</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>Motivation for Graph Database Approach</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Event Log Pre-processing</td>
<td>42</td>
</tr>
<tr>
<td>4.3</td>
<td>Complete Traces</td>
<td>42</td>
</tr>
<tr>
<td>4.4</td>
<td>Aggregation of Bag Traces into Routes</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>Visual Analytics of Routes</td>
<td>43</td>
</tr>
<tr>
<td>4.6</td>
<td>Data Pipeline</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>Event Logs Pre-processing</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Define Unique Items Identifier</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Remove Duplicate Events</td>
<td>46</td>
</tr>
<tr>
<td>5.3</td>
<td>Enriching Raw Data</td>
<td>47</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Parse Process Steps</td>
<td>47</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Locations Coordinates</td>
<td>47</td>
</tr>
<tr>
<td>5.4</td>
<td>Resulting Event Log</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>Complete Bag Traces through Pattern Matching</td>
<td>49</td>
</tr>
<tr>
<td>6.1</td>
<td>Build Graph from Routing Database</td>
<td>49</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Load Routing Data into a Graph Database</td>
<td>50</td>
</tr>
<tr>
<td>6.2</td>
<td>Complete Traces with Pattern Matching</td>
<td>51</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Input Data</td>
<td>51</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Challenges in Pattern Matching</td>
<td>51</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Final Algorithm</td>
<td>52</td>
</tr>
<tr>
<td>6.3</td>
<td>Output Data</td>
<td>54</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Storing Complete Traces in Tabular Form</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>Visual Analytics of Routes</td>
<td>57</td>
</tr>
<tr>
<td>7.1</td>
<td>The Problem of Routes Visualization</td>
<td>57</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Visualization Approach: Javascript Networks</td>
<td>57</td>
</tr>
<tr>
<td>7.1.2</td>
<td>The Challenge of Transparency of Information in Visualizations</td>
<td>58</td>
</tr>
<tr>
<td>7.2</td>
<td>Aggregation of Bag Traces into HL and LL Variants</td>
<td>59</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Aggregate Bags over HL Variants</td>
<td>60</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Aggregate Bags over LL Variants</td>
<td>62</td>
</tr>
<tr>
<td>7.3</td>
<td>Visualization of Process Variants</td>
<td>64</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Visualizations of HL Variants</td>
<td>64</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Visualizations of LL Variants</td>
<td>70</td>
</tr>
<tr>
<td>7.4</td>
<td>User Interface</td>
<td>75</td>
</tr>
<tr>
<td>8</td>
<td>Evaluation</td>
<td>77</td>
</tr>
<tr>
<td>8.1</td>
<td>Validation of Research Questions</td>
<td>77</td>
</tr>
<tr>
<td>8.2</td>
<td>Validation with the Stakeholder</td>
<td>78</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Stakeholder’s Use Cases</td>
<td>78</td>
</tr>
</tbody>
</table>
## 9 The Methodology

9.1 Data Preparation
9.1.1 Prepare a Graph Database
9.1.2 Pre-process Event Data
9.1.3 Query the Graph

9.2 Modeling
9.2.1 Aggregation of Bag Traces
9.2.2 Visualization of Process Variants

## 10 Conclusions

10.1 Contributions
10.1.1 Contributions to the Business
10.1.2 Scientific Contributions
10.2 Future Work

## Bibliography

## Appendix

A Complete Graph of Routing Database
A.1 Cypher queries to build the complete routing graph
A.2 Querying Real-time Routing
A.3 Complexity Comparison
A.4 Shortest Path Queries

B Tables Definition of the Routing Database

C Event Logs Pre-processing Algorithms
C.1 Algorithm to Reduce Bag Identifiers
C.2 Algorithm to Remove Duplicate Events
C.3 Algorithm to Parse Process Steps
C.4 Algorithm to Parse LG Coordinates

D Cypher Queries and Algorithms to Complete the Bag Traces in the Graph Database
D.1 Cypher Queries to Build the Routing Graph
D.2 Cypher query to include manual movements in the routing graph
D.3 Pattern Matching Algorithm to Complete the Bag Traces
D.4 Graph Data Model having Bag Traces Stored in the GraphDB

E Code to Obtain Information Visualizations
E.1 Assign Performance Metric as Edge Color
E.2 Pass Nodes and Edges to the JS `vis.Network`
E.3 Nodes and Edges Objects in JS `vis.Network`

F Variants Generation Algorithms
F.1 Algorithm to Aggregate Bag Traces into HL Variants
F.2 Algorithm to Aggregate Bag Traces into LL Variants

G User Interface Screenshots
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Phases of the CRISP-DM Process Model, from Wirth, R. et al. [34]</td>
</tr>
<tr>
<td>2.1</td>
<td>Relating the different terms, from general enclosing terms to more specific, from W. van der Aalst et al. [29]</td>
</tr>
<tr>
<td>2.2</td>
<td>Positioning of the three main types of Process Mining: discovery, conformance, and enhancement, from W. van der Aalst et al. [29]</td>
</tr>
<tr>
<td>2.3</td>
<td>The three basic types of Process Mining explained in terms of input and output: (a) discovery, (b) Conformance Checking, and (c) enhancement, from W. van der Aalst et al. [29]</td>
</tr>
<tr>
<td>2.4</td>
<td>Decision tree to select a Process Mining algorithm, from Lakshmanan, G. et al. [20]</td>
</tr>
<tr>
<td>2.5</td>
<td>Comparison of Pattern Matching in a Relational DB and in a Graph DB, from Neo4j [[22]]</td>
</tr>
<tr>
<td>2.6</td>
<td>Proposal of GQL, as a fusion of PGQL, G-CORE and Cypher, from The GQL Manifesto [1]</td>
</tr>
<tr>
<td>3.1</td>
<td>Diagram of the basic functions in a BHS</td>
</tr>
<tr>
<td>3.2</td>
<td>BHS as a Transition System</td>
</tr>
<tr>
<td>3.3</td>
<td>High-level good process examples</td>
</tr>
<tr>
<td>3.4</td>
<td>High-level bad process examples</td>
</tr>
<tr>
<td>3.5</td>
<td>Low-level process: Check-In</td>
</tr>
<tr>
<td>3.6</td>
<td>Low-level process: Sorter 1</td>
</tr>
<tr>
<td>3.7</td>
<td>Transport process of Sorter 1A</td>
</tr>
<tr>
<td>3.8</td>
<td>BHS layout without node categorization</td>
</tr>
<tr>
<td>3.9</td>
<td>Area in a BHS where each node is uniquely labeled by 3 identifiers</td>
</tr>
<tr>
<td>3.10</td>
<td>Module of a BHS composed of 3 areas</td>
</tr>
<tr>
<td>3.11</td>
<td>Layout Graph of an airport</td>
</tr>
<tr>
<td>3.12</td>
<td>Subsets of the BHS showing process steps</td>
</tr>
<tr>
<td>3.13</td>
<td>Dynamic Routing Scenarios</td>
</tr>
<tr>
<td>3.14</td>
<td>Example of subset of BHS with dynamic routing scenarios</td>
</tr>
<tr>
<td>3.15</td>
<td>Example of a BHS as a Petri Net</td>
</tr>
<tr>
<td>3.16</td>
<td>Example traces highlighted on the Layout Graph</td>
</tr>
<tr>
<td>3.17</td>
<td>Line Chart of system time per bag over low-level processes with a clear outlier</td>
</tr>
<tr>
<td>3.18</td>
<td>Line Chart of system time per bag over low-level processes</td>
</tr>
<tr>
<td>3.19</td>
<td>Example traces highlighted on the Layout Graph</td>
</tr>
<tr>
<td>3.20</td>
<td>Subset of a BHS. Different nodes in the BHS (scanners, tracking sensors, manual work) generate different logs</td>
</tr>
<tr>
<td>3.21</td>
<td>Subset of a BHS. The thick arrows represent the route of a bag, where only the pink arrows are present in the log</td>
</tr>
<tr>
<td>3.22</td>
<td>Routing Database</td>
</tr>
<tr>
<td>3.23</td>
<td>Graph schema of the Routing Database</td>
</tr>
<tr>
<td>3.24</td>
<td>Default routes and non-default routes in the Sorter 1A</td>
</tr>
<tr>
<td>3.25</td>
<td>Example of nodes LOCATION and ROUTE in the Sorter 1A</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

3.26 Example of a blocking rule ........................................ 35
3.27 Classical CC vs PA-CC ........................................... 37
4.1 Total Data Pipeline .................................................. 44
5.1 Pipeline for Event Logs pre-processing ............................ 45
6.1 Routing Tables retrieved. Use location_id as the node identifier .... 49
6.2 Property Graph generated from the tables NODES and SEGMENTS .... 50
6.3 Routing graph ....................................................... 50
6.4 Concept of inserting manual movements in the graph ................. 51
6.5 Data Pipeline of the Pattern Matching Algorithm .................... 53
6.6 Result of querying one bag trace in the graph ...................... 54
7.1 Example of a vis.Network ........................................... 58
7.2 Example of a dynamic HTML file displaying 3 time metrics in a network ... 59
7.3 Data Pipeline of the High Level process classification described in Algorithm 6 ... 60
7.4 Data Pipeline of the Low-Level process classification ................ 62
7.5 Data Pipeline for visualization of High Level processes ............... 64
7.6 Line Chart of In System Time per bags over a High Level Process ........ 66
7.7 Graph visualization of total time per bag (3 bags) .................. 66
7.8 Graph visualization of total time per bag (2 bags). Further information in the hovering notes .......... 67
7.9 Graph visualization of total time per bag (10 bags) ................ 68
7.10 Graph visualization of total time per bag (10 bags) - Closer look .......... 69
7.11 Graph visualization of total time per bag (10 bags) - Incidents ........... 69
7.12 Comparison of performance between 2 bags - observe deviation ........ 70
7.13 Data Pipeline of the Low-Level visualizations .................... 71
7.14 Line Chart of In System Time per bags over Low Level Processes in the Check In ... 72
7.15 Graph visualization of total time per bag (10 bags) - Incidents .......... 73
7.16 Graph visualization of total time per bag (10 bags) - Incidents .......... 73
7.17 Graph visualization of total time per bag (10 bags) - Incidents .......... 74
7.18 Application Pipeline ............................................... 75
A.1 Inner joins among the routing tables ................................ 91
A.2 Patterns searched in real-time routing queries ....................... 93
B.1 Node types: configuration of incoming and outgoing edges per node .......... 96
D.1 Interpretation of the MATCH and the RESULT clause of Listing D.3 .......... 106
D.2 Interpretation of the MATCH and the RESULT clause of Listing D.4 .......... 107
D.3 Interpretation of the MATCH and the RESULT clause of Listing D.5 .......... 107
D.4 Data Model of the property graph. The information about the bag traces is stored in the MOVED_TO edges .......... 110
D.5 Example of a trace in the graph .................................. 110
G.1 Menu 1 - Settings ................................................ 118
G.2 Menu 2 - Create Workspace ....................................... 118
G.3 Menu 2 - Select a date to evaluate ................................ 118
G.4 Menu 3 - Main Menu ............................................ 118
G.5 Menu 4 - Steps in Data Preparation ............................... 118
G.6 Menu 4 - Requisites in Data Classification .......................... 118
G.7 Menu 5 - Steps in Data Classification ................................ 119
G.8 Menu 6 - Steps in High-level Classification ......................... 119
G.9 Menu 6 - Steps in Low-level Classification ......................... 119
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Event data from 3 bag traces</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Event Log from high-level processes, identified by ID1</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>Event Log from low-level processes, identified by ID2</td>
<td>29</td>
</tr>
<tr>
<td>3.4</td>
<td>Slice of Bag Information Report, for bag with ID2 = 75480489</td>
<td>29</td>
</tr>
<tr>
<td>3.5</td>
<td>Slice of Bag Information Report, where a bag misses a flight. ID2 remains the same. The bag is assigned a new ID1</td>
<td>30</td>
</tr>
<tr>
<td>3.6</td>
<td>Slice of Tracking Report 1, for bag with ID2 = 75480489</td>
<td>30</td>
</tr>
<tr>
<td>3.7</td>
<td>Slice of Tracking Report 2, for bag with ID2 = 75480489</td>
<td>30</td>
</tr>
<tr>
<td>3.8</td>
<td>Union of Tracking Report 1 and Tracking Report 2 for the bag with ID2 = 75480489. The rows that belong to Tracking Report 2 are highlighted.</td>
<td>31</td>
</tr>
<tr>
<td>3.9</td>
<td>Event Log from high-level processes, identified by ID1 and including ID2</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>Preprocessed_logs.csv</td>
<td>47</td>
</tr>
<tr>
<td>6.1</td>
<td>complete_traces.csv</td>
<td>55</td>
</tr>
<tr>
<td>7.1</td>
<td>Attributes of complete_traces.csv used in the Data Classification</td>
<td>60</td>
</tr>
<tr>
<td>7.2</td>
<td>Mockup of the table HL_sequences</td>
<td>61</td>
</tr>
<tr>
<td>7.3</td>
<td>Mockup of the table bags_per_HL_sequences</td>
<td>61</td>
</tr>
<tr>
<td>7.4</td>
<td>Mockup of the table traces_HL_sequences</td>
<td>62</td>
</tr>
<tr>
<td>7.5</td>
<td>Mockup of the table routes</td>
<td>63</td>
</tr>
<tr>
<td>7.6</td>
<td>Mockup of the table bags_per_route</td>
<td>63</td>
</tr>
<tr>
<td>7.7</td>
<td>Mockup of the table traces_routes</td>
<td>63</td>
</tr>
<tr>
<td>C.1</td>
<td>Iterations of the algorithm &quot;Reduce bag identifiers&quot;</td>
<td>100</td>
</tr>
</tbody>
</table>
Listings

| A.1 | Cypher queries to load the routing tables in the graph | 92 |
| A.2 | Real-time routing query, where the destination is not provided | 93 |
| A.3 | Real-time routing query, where the destination is provided | 94 |
| A.4 | All the path between a pair of consecutive nodes in the log | 95 |
| A.5 | Shortest path between a pair of consecutive nodes in the log using Dijkstra’s shortest path algorithm | 95 |
| C.1 | Snippet of the XML file with the definition of each node | 102 |
| C.2 | Parse the BHS Definition file to obtain the process steps of the nodes in the LG | 102 |
| C.3 | Snippet of the SVG file with the coordinates of each node | 103 |
| C.4 | Parse the SVG file to obtain the coordinates of the nodes in the LG | 104 |
| D.1 | Cypher queries to load the routing graph | 105 |
| D.2 | Cypher query to create manual movements | 105 |
| D.3 | Cypher Pattern Matching query for the first event of a trace | 106 |
| D.4 | Cypher Pattern Matching query for every event of a trace except the first | 106 |
| D.5 | Cypher Pattern Matching query to end a trace | 107 |
| D.6 | Python functions that return the cypher queries with parameters, to be executed | 108 |
| E.1 | Assign color to edges. Choose a palette. Assign the minimum and maximum values to the minimum and maximum performance. Each possible performance is a color in-between | 111 |
| E.2 | Pass nodes and edges to the JS vis.Network | 112 |
Chapter 1

Introduction

This master thesis is the result of a graduation project as part of the Master of Science in Big Data Management and Analytics, an Erasmus Mundus initiative, coordinated by the consortium of Université Libre de Bruxelles (ULB), Universitat Politècnica de Catalunya (UPC), Eindhoven University of Technology (TU/e), Technische Universiteit Berlin (TUB) and Université François Rabelais de Tours (UFRT). This graduation project has been supervised in the Analytics for Information Systems (AIS) Research Group from TU/e.

In this chapter, we start by justifying the motivation and the context of this thesis in Section 1.1. Then we formulate the research problem and break it down into more detailed research questions in Section 1.2. We finally describe the methodology we used and the steps that were taken in Section 1.3.

1.1 Motivation and Context

This master thesis has been developed in collaboration with a Baggage Handling Systems company. In this section we start by introducing why is a Baggage Handling System company interested in the results of this master thesis. Then we point our how does this research contribute to the state of the art in Process Mining.

1.1.1 Business Motivation

Our use case is a given Baggage Handling System (BHS). The process engineers in the company analyze the data generated about the bags that are processed in the system (event data) to improve the understanding of how is the system performing, e.g. "Where and when do traffic jams take place?", "Which are the bottlenecks in the system?", "Are bags lost in the system?", "Which bags arrive late to their flight and with which delay?"

However, the data the process engineers have currently available is incomplete data, i.e. they have information about bags passing through some specific locations, but not through all the locations the trace went through. That is because having a large BHS where thousands of bags enter the system every day, it would be too expensive and unrewarding to store the complete event data.

The KPIs that the process engineers use are computed/estimated through heuristic since the complete event data is not available. In other words, the information about which route each bag goes through is not stored, only some events per bag trace are stored.
CHAPTER 1. INTRODUCTION

However, even if the process engineers had the complete sequence of locations a bag went through, it would be a complex task for the user to make sense out of a long sequence of location codes. The process engineers required the proposal of an understandable approach to visualize the data.

We have been proposed to research a methodology to obtain the process variants of the bag traces in the system and to include a technique to assess the performance over the system of each route variant. Having information and visualizations about how the process variants perform in the system can help them detect bottlenecks, traffic jams, and unbalanced load in parallel routes.

1.1.2 Scientific Motivation

In this section, we justify why did we consider necessary to find a new approach to perform Conformance Checking in the system we studied, i.e. why did we propose a new methodology instead of reusing Process Mining techniques available in the state of the art. So far, no one has proposed a technique to complete event traces that have missing events, using ground truth information about which sequences of events are possible. The use case we implemented was a use case where a minority of the events created were stored for analytical purposes due to the large size of the system.

In this BHS use case, we propose a Process Mining methodology that considers the process in two levels of abstraction; we relate each trace to a high-level process and a low-level process. The methodology we propose takes into consideration the complexity of Material Handling Systems and obtains the many routes that take place.

These are all arguments that lead to the proposal of this master thesis, but the main justification to contribute in the Conformance Checking techniques instead of implementing a methodology already studied, was the lack of a process model. As W. van der Aalst et al. describe in their work, Process Mining is both data-driven and process-centric: it provides an answer to a wide range of conformance and performance questions using a combination of event data and process models. Classical Conformance Checking techniques rely on comparing the traces in an event log with a process model already defined to detect deviations. In her work, Xixi Lu reviewed the available Deviation Detection techniques, where some worked around the lack of a process model, and she proposed a new technique herself. However, those techniques used the event data as the ground truth. They did not apply for the Material Handling System we present, where the traces are incomplete, as the majority of the events are not stored.

Also, we provide a new form of visualizing the performance of a process, since we provide information visualization to analyze a system where the physical layout has a big impact on the performance of the process. The layout of the transport routes and the distance the items must travel affects the time performance of the traces. Therefore, in our methodology, we provide coverage of how should the process data of a Material Handling System be visualized to satisfy the needs of the process analyst.

1.2 Research Problem

In this section, we define the research problem and expand it into a list of research questions. The research problem is as follows:

*Given an event log of a Material Handling System, we would like to develop a methodology to provide insights on the performance of the process variants.*
CHAPTER 1. INTRODUCTION

It is assumed that each record of the event log contains a timestamp, an identifier, and a field that defines the event as a unique step within the process that can be considered the activity. It is also assumed that there is information about the high-level process related to each step available and that the events can be related to a position in the layout of the system.

As the research problem is not concrete enough to be evaluated, we expand it into the following list of research questions (RQ):

RQ1. Given event traces with missing events and given ground truth information about the consequentiality among events, i.e. given information about possible sequences of events, we aim to complete the event traces by finding the missing sequences of events.

RQ2. Given complete traces, we aim to group them by two kinds of process variants: sequences of high-level events and sequences of low-level events. We aim to have access to information about the load of each variant and the time performance of that route.

RQ3. Given bag traces and process variants, we aim to provide visualizations of their performance that can be interpretable for the user.

RQ4. Given bag traces, we want to visualize their routes over the Baggage Handling System.

RQ5. Given bag traces, we want to visualize the time performance of the traces and the process variants along the entire process.

RQ6. Given event data and structured data about the consequentiality of events, we aim to provide an interactive tool to obtain the performance of process variants and visualize it.

1.3 Methodology

We choose to follow CRISP-DM (CRoss Industry Standard Process for Data Mining)[34] (refer to the CRISP-DM Model in Figure 1.1a) for this thesis. In Chapter 3 we describe Business Understanding (Section 3.1 Domain of Baggage Handling Systems) and Data Understanding (Section 3.4 Data Understanding). In Chapter 4 we introduce the method selected to approach the Data Preparation and the Modeling, which is further developed in the subsequent chapters. The Data Preparation that our method requires consists of three steps:
CHAPTER 1. INTRODUCTION

1. The first step is Build Graph from Routing Database (Section 6.1). It consists of building a Graph Database with the ground truth of the interconnections of components of the BHS. Selecting the right database does not only depend on storage requirements, but it is also dependent on the analytics goals. We discovered that to solve our Process Mining problem of performance Deviation Detection, it was necessary to enrich the data available by querying the ground truth in a Graph Database.

2. The second step is Event Logs Pre-processing (Chapter 5). Having initial raw data from multiple event logs, we must identify the events that belong to each trace (Section 5.1). Since we retrieve logs from different sources, the event data also requires data cleaning, removing duplicates and verifying that the events in a trace are consistent with each other (Section 5.2). The event data must also be enriched with other information that will be required in later steps of the process (Section 5.3).

3. The third and final step of the Data Preparation consists of finding and fixing the missing paths in the traces by querying the graph (Section 6.2). For each pair of events in a trace, there might be a sequence of events not registered in the log. The graph build in the first step contains the events connected, where each connection is a possible movement in the system. To each pair of events in the trace, we query the shortest path between them to obtain the most feasible route that was taken.

Having completed traces, we enter the Modelling phase by aggregating the traces into process variants (Section 7.2). This has been done over low-level (LL) events and high-level (HL) events, obtaining LL and HL process variants. The performance of the process variants has been measured over the time performance of the bags in the system (e.g. duration of each event, duration of the trace, cumulative duration in the system, remaining time in the system...). After obtaining the performance metrics of the variants, we provide the user with information visualizations of the performance of the routes (Section 7.3). Our focus was to provide the user with the highest level of understanding possible. We visualize the routes over the system’s layout to give a better understanding of which function is executed in each part of the route.

We finally test if the routes’ performance information and the visualizations obtained can answer process performance questions from the BHS analysts. We perform this evaluation in Chapter 8. The deployment of this tool has been in a Proof of Concept application (Section 7.4). We provide the user with a user interface to select the data to include in the model and to select the parameters to customize the outcome. The user interface can be seen in Appendix G.

In conclusion, in chapters 4, 5, 6 and 7 we justify the decisions taken in each of the phases based on our implementation with a BHS use case. After we were satisfied with the outcome, we summarized the methodology proposed in Chapter 9. The model in Figure 1.1b shows how we enclose the phases of Data Preparation and Modeling to be reproduced.
Chapter 2

Preliminaries

In this chapter, we explain fundamental concepts to fully understand this thesis and its context. We refer to two main areas of research: Process Mining and Semantic Data Management. Process Mining is the area of research which comprehends our research problem. We define some terms that are commonly used in this thesis, such as event logs and traces. We situate our problem in more detail in the area of Conformance Checking, and we present related work on Deviation Detection. It is important to assess why we didn’t choose one of the related methods to solve our problem and decided to start a new technique. The next area of research is Semantic Data Management. We introduce its concept since the technology we used to solve our problem is comprised in this area. We build a Graph Database and we use the benefit of using a graph data model and graph query languages.

In order to describe the two areas of research previously mentioned, we dedicate Section 2.1 to the introduction of Process Mining and Section 2.2 to the introduction of Semantic Data Management.

2.1 Process Mining

In this section, we introduce the area of Process Mining. We situate the context of Process Mining, i.e. its position among broader fields. We start by describing basic terms in Section 2.1.1, and then in Section 2.1.2 we drill deeper into the main types of Process Mining.

As declared by members of the IEEE Task Force on Process Mining in The Process Mining Manifesto, "Process Mining is a relatively young research discipline that sits between Computational Intelligence and Data Mining on the one hand, and process modeling and analysis on the other hand. The idea of Process Mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in today’s (information) systems.”[29]

The goal of Process Mining is to provide answers to conformance and performance questions about processes. It uses data about the executions of the process and the modeling of the process to provide feedback about the real behaviour of the process. Figure 2.1 situates Process Mining among its related areas of research. All the techniques that aim to provide decision making support are positioned within the concept of Business Intelligence (BI). Process Intelligence is the combination of BI and Business Process Modeling since its focus is to improve processes and their management. Within this smaller branch, Process Mining comprises the Process Intelligence techniques that use event data to achieve its goals.[29]
CHAPTER 2. PRELIMINARIES

2.1.1 Event Logs

The data required to perform Process Mining on a process are event logs. We can create an event log by extracting data from a process. An event log consists of a list of records, where each record contains data about an event (e.g. when did that event occur, which activity took place, who performed this activity, which resources were used). Cases are individual instances of a process, and a trace of events is recorded per each case. A trace is the recorded sequence of events that took place within a case. For event logs to be used to describe processes, they must meet certain assumptions:

- The execution of a process generates cases, i.e. unique instances of the process.
- A case consists of a set of events, i.e. each event corresponds to a unique case.
- Events within a case have a chronological order.
- Events can contain attributes. Common attributes are activity, time, costs, and resource.

2.1.2 Process Mining Techniques

In this section, we mention the three main types of Process Mining. The goal of this Master Thesis is to find a Conformance Checking methodology, more specifically, we need to develop a Deviation Detection technique. For this purpose, we first introduce existing Deviation Detection techniques in Section 2.1.2, to consider if our research problem is already covered in the state of the art. In Section 2.1.2 we also mention how does the state of the art survey existing techniques and recommends an existing technique based on the conditions of our research problem.

Figure 2.2 shows the role of each of the Process Mining types have based on their relationship with the model and the logs. Having event data available and not knowing which is the model of the process, the goal of Process Discovery is to extract the model of a process from the event log. Having a process model defined and event data from the execution of the process, Conformance Checking consists of detecting deviations between the model and the log. Lastly, after detecting how the event data deviates from the model, Model Enhancement consists of improving the model of the process to reflect better the reality. Our project is positioned within the branch of Conformance Checking. Conformance Checking is mainly used to detect control flow deviations. The goal of our research problem is to detect performance deviation. Therefore, we continue by presenting Deviation Detection techniques, to assess if they fit to solve our research problem.
CHAPTER 2. PRELIMINARIES

Figure 2.2: Positioning of the three main types of Process Mining: discovery, conformance, and enhancement, from W. van der Aalst et al. [29]

Deviation Detection

Figure 2.3 shows that, in order to detect deviations in a process, Conformance Checking requires to have an event log and a model available. The main limitation to perform Classical Conformance Checking in our research problem is that there is no process model defined (see Section 3.1.6 Lack of Process Model). In her work, Xixi Lu reviewed four existing Deviation Detection techniques and proposed a new Deviation Detection[21]. Her method managed to detect deviations from event logs only, without requiring a precise normative model of the process. Her method was based on finding patterns and trusting the relations among events [21]. Below, we list the four Deviation Detection methods reviewed by Xixi Lu and we assess to which extent would they cover our use case.

Figure 2.3: The three basic types of Process Mining explained in terms of input and output:(a) discovery, (b) Conformance Checking, and (c) enhancement, from W. van der Aalst et al. [29]

DDT1 - Conformance Checking

Description. It contains a set of techniques that check process conformance of recorded executions against a normative process and identify where observed behavior does not fit the process model and thus deviates from some normative, prescribed behavior [30].

Limitation. It requires a normative process in the form of a process model [30]
DDT2 - Discover a model of frequent behavior

*Description.* It uses a discovered model of most frequent behaviour. This method also checks where log deviates against the model[21].

*Limitation.* In the case of the Material Handling System, a discovered model of "the most frequent behavior" would not provide the optimal routes. It is more meaningful to use expert knowledge to define the happy routes (i.e. the routes with the least exceptional behaviour).

DDT3 - Discover a model of clustered traces

*Description.* This method considers that having a process where there are many process variants, it is underfitting to compare the event data to one normative process. In this technique, the traces are first clustered or classified to create the process model used for Conformance Checking[21].

*Limitation.* This approach would benefit a complex process like a Material Handling System. The traces could be clustered withing two levels of detail: low-level routes within a high-level event, or high-level routes. However, this method does not cover the limitation that the traces are incomplete, and the storage of events is so sparse in this use case, that the resulting clustered traces would not be representative of the process. Nevertheless, this technique can be applied to the data after completing the traces in our method.

DDT4 - Detect deviations comparing traces

*Description.* This method does not discover a process model, it detects deviating traces by comparing traces with each other, directly extracted from the log[21].

*Limitation.* This method does not use domain knowledge to decide which are the intended traces. A Deviation Detection based on outlier behavior does not find the best performing routes. Again, it does not provide a solution to the incomplete traces problem.

We observe that the Deviation Detection techniques reviewed[21] work around the lack of a model by using the traces as the ground truth to discover a model or compare traces with one another. In the BHS use case we present (see Chapter 3), the traces have many missing events, making them an unreliable description of routes. Being our goal to investigate transport routes, it is essential to have complete traces, in order not to incorrectly categorize which trace represents which route. In the next section, we assess if other Process Mining techniques work around the limitations of our use case.

Related work

Leveraging Process-Mining Techniques[20] proposes a decision tree[19] to select the most adequate Process Mining algorithm for each research problem (see Figure 2.4). For Event Logs with noise and that require a Petri Net[25][12] output and that can contain duplicate tasks and invisible tasks, a Genetics Miner is the approach that covers those conditions. However, Genetics Miner mines a Petri Net representation1 of the process model from the traces in the event data.[31] For that, it uses a genetics algorithm; its search technique mimics the evolution of biological systems. This complex search method manages to nine process models that can contain all kind of constructs, such as sequence, choice, parallelism, loops, duplicate tasks, etc. It can also handle noise in the initial data. The downside of such a complex search algorithm is that it requires a large amount of computational time.

---

1A Petri Net is a bipartite graph consisting of places and transitions.[32] It is used to represent a process. Tokens are units of resources that flow through the network. When an event is executed, a transition is fired. For a transition to be fired, it requires to have tokens available in all its preceding places. When a transition is fired, one token is destroyed in each of its preceding places and generated in the succeeding places.
2.2 Semantic Data Management

In this section, we introduce the area of research of Semantic Data Management. It has a wide scope, and it comprehends from Knowledge Graphs, used in Semantic Web to relate concepts used in different domains, to Graph Databases, whose purpose is to provide an alternative storage and querying technology for problems that are not easily solved in Relational Databases. The focus of our technique are Graph Databases and the potential of Graph Queries, but we would like to start by introducing some concepts that have common ground in Semantic Data Management.

We’d like to start by providing some basic definitions. Then in Section 2.2.1, we present Property Graphs, the mathematical foundation behind Graph Databases. We continue introducing Graph Databases in Section 2.2.2 and the Graph Query Language in Section 2.2.3, which can perform queries that are not possible in Relational Databases, thanks to data being highly interconnected in Graph Databases.

Within Semantic Data Management, there are two main graph families: Knowledge Graphs and Property Graphs. Knowledge Graphs have less structure, which allows them to have higher flexibility of semantics. They are used in the Semantic Web[6][10] to relate different terms from different sources. We recall Knowledge Graphs because throughout the thesis we refer triples due to their simplicity as a data structure. The triple is the basic construct of a Knowledge Graph, e.g. the Resource Description Framework² (RDF). The structure of a triple is (subject, predicate, object). A set of triples form an RDF graph.[33] In the end, the structure of a triple fits any

²RDF:https://www.w3.org/RDF/.
CHAPTER 2. PRELIMINARIES

directed graph, since it is equivalent to (start node, edge, end node).

2.2.1 Property Graphs

A graph is a set of nodes and edges. A Property Graph allows to include meta-data in the nodes and edges in the form of attributes. A Labelled Property Graph also includes labels (the nodes and edges can be classified). The nodes (called entities) can be classified of nodes called labels and the edges (called relationships) can be categorized by types. Both nodes and edges can have attributes (called properties). Edges must also have a direction, i.e. a start node and an end node. Each node an edge has a unique identifier.[24]

In Foundations of Modern Query Languages for Graph Databases, R. Angles et al. differentiate between edge-labelled graph and Property Graph as follows: edge-labelled graph is the base model considered in the Graph Database literature and the Property Graph, a model commonly employed in practice, contains nodes and edges in labelled graphs which can be annotated with additional meta-information. [4] We can conclude that Property Graphs are the basis over which the Graph Databases are modelled. Therefore, it is possible to perform graph queries on them, founded in graph patterns. Graph patterns[18] consist of matching graph-structured queries against a Graph Database. [4]

Basic Graph Patterns are equivalent to conjunctive queries[26] without projections. However, database languages, which are based on the relational algebra, allow more complex Pattern Matching[14] queries. In Section 2.2.2, we show an example of a Pattern Matching query (see Figure 2.5). In large volumes of data, querying complex patterns is more efficient in a graph than in a relational structure. However, the main benefit of querying Property Graphs are the queries that traverse through the graph. Navigational graph patterns[28] match paths of arbitrary length in the graph[4]. A common problem that among them is to query the shortest path[13] from one node to another of the graph. In this master thesis, we will use shortest path queries to solve our research problem (see Chapter 4).

2.2.2 Graph Databases

The main technology that has been used in this master thesis to solve the research problem is Graph Databases: we loaded the system data in a Graph Database and queried the event data on it. In this section we introduce Graph Databases, how do they differ from Relational Databases.

Relational Databases[11] are the most traditional databases, widely used as a standard in the industry. Their properties ensure high data integrity. The properties of Relational Databases (ACID properties[17]) meet the requirements of OLTP (On-line Transaction Processing), thus they are adequate in systems designed to store transactions. However, Relational Databases don’t fit the needs of analytics systems (On-line Analytical Processing, OLAP). OLAP tend to require massives volumes of data and complex queries. Executing such complex queries over big volumes of data in Relational Databases require a high computational complexity. OLAP introduced the need for having NoSQL Databases.

NoSQL Databases work in the opposite principle than ACID; they fulfill the BASE properties[9]. As Biffl S. et al. mention in Semantic Web Technologies for Intelligent Engineering Applications, "NoSQL Databases provide better horizontal scalability[8] and are more flexible for schemaless operations when compared to traditional Relational Databases. They offer hierarchical representations of structures similar to those used in ontologies. Consequently, dependencies and relations between models stored in an ontology may be represented in the same way for model instances in the database while avoiding potential schema mismatches".[6] In other words, NoSQL databases are schemaless, one of the limitations for Relational Databases to adapt its structure to data that
CHAPTER 2. PRELIMINARIES

Among the NoSQL technologies, Graph Databases\cite{26} are the ones that allow representing more complexity of data. It supports complex queries not available in any other DBMS. Among Graph DBMS in the market, Neo4j has remained the leading technology over the years\cite{2}. It developed a relevant part of the literature in Graph Databases, and was the pioneer in proposing a standard Graph Query Language.

![Pattern in a Relational DB](image1)

![Pattern in a Graph DB](image2)

Figure 2.5: Comparison of Pattern Matching in a Relational DB and in a Graph DB, from Neo4j \cite{22}

Graph Databases support pattern matching queries and navigational queries. Figure 2.5 compares the complexity of querying a pattern in a Relational Database and a Graph Database. For a Graph Database, this is a fast query, where after doing an index lookup on the node "Alice", it only needs to retrieve three edges. Meanwhile, reproducing the same query in a Relational Database is equivalent to performing natural joins between the related tables. In a problem with a high volume of data, it is not advised to solve these queries with a Relational Database. Furthermore, Graph Databases allow querying navigational graph patterns, i.e., queries without a fixed length, for example, the shortest path query\cite{13}. The shortest path query is impossible to perform in a Relational Database since Relational Databases have the limitation that the number of joins must be fixed. It is possible to perform a recursive query, but still, in many cases, it returns an incomplete execution. Meanwhile, the computational complexity of Dijkstra’s shortest path algorithm\cite{13} is $O(\|\text{Nodes}\|^2)$. To solve our research problem, we concluded that it was necessary to use the shortest path algorithm, therefore our approach consisted in using Graph Databases.

2.2.3 Graph Query Language

The goal of Graph Query Languages is to reduce the computational complexity in specific queries. Graph Query Languages support complex pattern matching.\cite{4} In this project, we execute some pattern matching queries in a Graph Database. Cypher is a query language for Graph Databases and it is the closest to a standard for graphs in the Database industry. As Green A. et al. describe, "Cypher is a query language for Property Graphs. It was originally designed and implemented as part of the Neo4j Graph Database, and it is currently used in a growing number of commercial systems, industrial applications and research projects. [...] (Cypher) features in particular pattern matching, filtering, and most relational operations on tables."\cite{15}
We want to finalize by mentioning in Section 2.2.4 that there still doesn’t exist a standard Graph Query Language, but it is a future goal. Meanwhile, cypher is being used as the standard graph language by many leading software companies.

### 2.2.4 Towards a Standard Graph Query Language

In the Database industry, a standard for Graph Query Languages has still not been adopted. In an open letter, Neo4j introduced the proposal to define a standard for Graph Query Language to the database industry.[16] This proposal matured and increased in acceptance, and it is now described in The GQL Manifesto[3]. It aims to fusion properties from the three main Graph Query Languages in the market[23]: Cypher, PGQL and G-CORE (see Figure 2.6). Nevertheless, a standard for Graph Query Languages is still an open letter.

![Figure 2.6: Proposal of GQL, as a fusion of PGQL, G-CORE and Cypher, from The GQL Manifesto](image)

12 Performance-aware Conformance Checking on Material Handling Systems
Chapter 3

Use Case

In this chapter, we introduce the use case of detecting deviations in process variants of a Baggage Handling System. We start introducing how the system works. Afterwards, we observe the limitations of the available data and the lack of a process model to perform existing Deviation Detection techniques. Then, we propose the method of Performance-aware Conformance Checking (hereinafter PA-CC), and list the objectives of this project.

Section 3.1 Domain of Baggage Handling Systems introduces how the BHS works. In Section 3.2 Problem Description we present the interest of the user in detecting process variants and understanding their performance in relation with the process. Then we introduce how our event data looks like in Section 3.3 Overview of Data Sources, going into more detail about all the data related to this project in Section 3.4 Data Understanding, where we explore the available data, and how it challenges our method. We finally present in Section 3.5 Data Science Problem the need to propose a new Deviation Detection technique, and we break down the objectives of this method in Section 3.6 Definition of Technical Sub-problems where we list the tasks we tackle in the following chapters.

3.1 Domain of Baggage Handling Systems

Section 3.1.1 BHS as Complex Systems introduces the level of complexity of Baggage Handling Systems, which can be split in High-Level Processes (see them described in Section 3.1.2) and Low-Level Processes (see them described in Section 3.1.3). Section 3.1.4 explains how the two kinds of processes are linked on a physical layout level, and Section 3.1.5 explains how the two levels are linked in the decision process. We conclude in Section 3.1.6 that there is no business model available to describe the ideal process of the BHS. This leads to the problem description in Section 3.2.

3.1.1 BHS as Complex Systems

The Baggage Handling System of an airport (BHS) is the system that registers baggage in an airport, analyzes its content, and transports it to the flight to be boarded (see Figure 3.1). It is a network formed by conveyor belts and other types of equipment: scanners, screening machines, storage units, etc. From the moment the bag is registered, the system keeps track of the location of the bag, its destination, and the processes it requires to complete.

3.1.2 High-level Process

In a BHS, the events are classified into high-level (HL) events. The process each bag goes through from start to end is a sequence of high-level events or process steps. Figure 3.2 shows the Transition
CHAPTER 3. USE CASE

Figure 3.1: Diagram of the basic functions in a BHS

System of a BHS, where each transition is a process step: check-in, Sorter 1, scanner, etc. Those high-level events are grouped as sub-areas of the BHS.

Figure 3.2: BHS as a Transition System

Examples of HL Processes

The high-level process the bag undertakes depends on where the bag is introduced into the system and which is its final destination. Each terminal in an airport is an independent BHS, thus, a bag will be introduced to the system in a Check-In desk, or it will be transported from another terminal. In consequence, the two expected destinations are either boarding a flight or being transported to another terminal.

In Figure 3.3 and Figure 3.4 we present some high-level process examples. This system will attempt to let the bag follow the simplest process given its initial conditions, e.g. which are its starting point and its destination. If a bag needs to be directed to another terminal, its shortest process possible is Figure 3.3a. If a bag arrives from another terminal and has to board a flight in the current BHS, its simplest process is Figure 3.3b. In the case when the bag arrives too early, it is directed to storage until it can be boarded. Early bags that do not require to switch terminals are expected to follow the process in Figure 3.3c or Figure 3.3d.

Determining Factors

Many factors determine the high-level process that a bag follows. The entry point and exit point determine the first and last high-level event of the process the bag goes through. The intermediate steps the bag goes through are conditioned by the situation in the system and particular conditions of the bag, such as:

- The time of arrival: The bag will be directed to the storage if the final gate to board the flight is not available yet (seen in Figure 3.3d).
• **The load of the system**: Every bag must be scanned. The bags are directed to scanners available from the Sorter 1 loop. If no scanner is available, the bag recirculates in the Sorter 1 (seen in Figure 3.4a).

• **Unexpected incidents**:  
  – If a bag is not scanned correctly, it requires to be scanned again (seen in Figure 3.4a).  
  – If the content of the bag raises some alarm, it must be directed to manual inspection (seen in Figure 3.4a).  
  – If the bag is not scanned correctly after many attempts, it must be manually inspected (seen in Figure 3.4a).  
  – If the shape or state of the bag interrupts the flow in the system, it must be removed and reintroduced in the system.  
  – If a bag is lost in tracking and found again, it could have been manipulated, and for security, it must be sent to Sorter 1 and scanned again (seen in Figure 3.3b).

### 3.1.3 Low-level Process

Each high-level event in the BHS (e.g. Check-in, Sorter 1, Scanner, Storage, etc.) is a subprocess itself. The transport process or low-level (LL) process is this subprocess within an area where each baggage item must travel through a sequence of physical locations from a starting point (source node) to an end point (sink node). Each process step has many sources and sinks, and the bags could take infinite possible routes in a subprocess if the subprocess contains loops.

As shown in Figure 3.5, when a bag starts the Check-In subprocess, it can start in one of the many Check-In desks available. The sequence of movements the bag goes through since it enters Check-In until it starts the next high-level event is a low-level process. The low-level process can also be called a transport process: the low-level events are the locations the bag passes through, and the process describes the physical route the bag followed.

Figure 3.6 is a simplification of the Sorter 1 subprocess. This BHS has two Sorters of type 1 available: Sorter 1A and Sorter 1B. They perform the same functions, but there is more than one available to parallelize the work. The bags are directed to one or the other for *load balancing*. The function of Sorter 1 is to distribute the bags among the scanners available since every bag
requires to be scanned. Depending on the destination of each bag, the sorter decides to send each bag either to a scanner, to an inspection station, to Sorter 2, to Sorter 3 or to another terminal.

### 3.1.4 Layout Graph

The transport process is more complex than it is shown in Figure 3.5 and Figure 3.6. Figure 3.7 shows the transport process in Sorter 1A. The transport process contains all the physical movements through the equipment in the BHS. The representation used in Figure 3.7 is the **Layout Graph (LG)**. The LG describes a complete Transition System; it shows every conveyor in the system and every junction between conveyors. The conveyors are represented by arrows, called **segments**, and the junctions are **nodes**. Each node represents a **location** in the airport; it is an atomic event in the system. The transport process a bag goes through is the sequence of nodes it passes through.

We describe a transport process as each low-level process in the BHS. In Figure 3.5 and Figure 3.6, we focus on the Check-In subprocess and the Sorter 1 subprocess. However, BHS can be seen as a unique network. The LG is the layout that links the transport process to the BHS as a whole network. Figure 3.8 shows the LG of an entire BHS.
Hierarchical System

The Layout Graph is a floorplan of the BHS of an airport. Looking at Figure 3.8, it seems difficult to identify where each high-level event occurs in the process. To identify both high-level events (e.g. check-in) and low-level events (nodes), the system uses a hierarchical labeling method. The system can be hierarchically itemized into process steps, areas, zones, and equipment, being each equipment an indivisible/atomic location in the system. Every equipment in the system is labeled by a combination of 3 identifiers: the area and zone it is located in, and the equipment category it belongs to. Figure 3.9 is an example of a BHS module that displays the 3 level of identifiers each node has. Figure 3.10 shows a subset of a BHS composed of 3 areas.

The Layout Graph is the standard representation of our BHS. Using those 3 levels of identifiers, it is possible to identify any location in the system. Figure 3.11 shows the LG again, but this time the nodes are color-coded based on their Area ID. Figure 3.12 contains two subsets of the Layout Graph. In Figure 3.12a we can see the Check-In sub-process, where each line of desks is labeled and colored with a different Area ID. Figure 3.12b shows the Sorter 1 sub-process, where the Sorter 1A and Sorter 1B are displayed in black and grey. We can see many entrances and exits connected to the two sorters in green. The red nodes are inspection stations, and the eight rows
CHAPTER 3. USE CASE

3.1.5 Routing

LL Process Dependant of HL Process

From the high-level point of view, the system decided for the sequence of high-level events each bag must follow based on the information about the bag. The components in the system need to communicate the high-level information with each other to know where each bag is supposed to end up and decide for a low-level route.
CHAPTER 3. USE CASE

Figure 3.11: Layout Graph of an airport

Figure 3.12: Subsets of the BHS showing process steps

Routing Decision Process

In Section 3.1.3 we have seen that the Layout Graph describes the possible movements the bags can take in the BHS. The bags can move from one node to another following the outgoing edges. However, that still does not explain how the system decides which outgoing edge to take.

The routing of each bag from one location to the next is decided dynamically, based on

- the physical connections,
- the availability of the equipment,
- and the load distribution.
Dynamic Routing Scenarios: When a bag is located in a node, the system must direct it to a node in one of the outgoing edges. There are 3 possible scenarios (Figure 3.13):

- Scenario 1: only 1 outgoing segment
- Scenario 2: 2 outgoing segments, one is assigned as default (includes destination when the non-default is chosen)
- Scenario 3: 2 outgoing segments, none is assigned as default (needs to include a destination)

In Figure 3.14, we show examples of the three scenarios. There is no preference to enter Sorter1A or Sorter 1B as they perform the same functions. Hence, the two outgoing edges of node "Divert to Sorter 1" are set to TRANSPORT_DEFAULT = No (Scenario 3). For the divert to decide where to direct the bag, a destination is specified in the logs.

Some examples of Scenario 1 are the nodes "Entry to Sorter 1A", "Sorter 1A conveyor 2", "Merge from Scanner 3", "Merge from Scanner 2" and all the scanners; since they only have one outgoing edge, it is assigned as TRANSPORT_DEFAULT = No. It is clear that the bag only has one possible movement.

Scenario 2 takes place when an activity has preference over another. By default, a bag loops around the sorter, unless it is assigned to enter a scanner. This case is depicted in the nodes "Divert to Scanner 1", "Divert to Scanner 2", and "Divert to Scanner 3". The segments that move along the sorter are assigned TRANSPORT_DEFAULT = Yes, while the segments that connect with the scanners are assigned TRANSPORT_DEFAULT = No. If the bag is supposed to enter a scanner, the logs specify a concrete scanner as a destination.
Manual Movements not included in Layout Graph

The Baggage Handling System (BHS) is a network of conveyors that transport the bags along the airport, and perform a series of activities that the bag requires before boarding into the flight: identification, scanning, inspection, storage, routing to another terminal, routing to the assigned flight...

The BHS itself is the process model that defines the possible processes each bag can encounter (plus manual movements, which are not displayed in the system but are also part of the process). Figure 3.15 represents a small BHS, represented as a Petri Net[32], where the conveyors are explicitly part of the system, and the manual movements that are not represented in the network but are possible events in the process. It is a possible variant of the process that a worker manually moves a bag from place p2 to place p6.

![Figure 3.15: Example of a BHS as a Petri Net](image)

3.1.6 Lack of Process Model

In this section, we conclude that LG is a physical process model of the entire system. It describes every possible movement. This model is not equivalent to a business process model. A business process model describes the ideal process, where ideal denotes the process the company attempts not to deviate from. Therefore, we aim to create a model of the best routes in the system. This is the business process model that will be used to check for performance deviations.

3.2 Problem Description

In this section, we present the interest in detecting the variants of both HL and LL processes. The stakeholder in this project are the Process Engineers that work analyzing the event data of the BHS to provide insights on the system performance. We start in Section 3.2.1 by presenting some HL process variants and LL process variants, and how is their performance affected based on the route taken. Then we describe the goal of the stakeholder in Section 3.2.2, and break down how such objective benefits the user in Section 3.2.3. We conclude summarizing what was observed in this section into the Business Problem Definition in Section 3.2.4.
3.2.1 Examples of Process Variants

High-level Process Variants

In Section 3.1.2, we have seen that a BHS allows for different sequences of high-level functions to be executed. In a BHS, many different work-flows take place at the same time. We aim to detect optimal work-flows and non-optimal work-flows. Taking a look at the routes highlighted in Figure 3.16 we can observe that depending on the HL process that the bag takes, its route will be longer or shorter. Below we show some examples of high-level process and explain in which situation would each take place. We highlight how some of these processes look in the floorplan in Figure 3.16 to notice the difference in the distance among variants.

![Diagram showing different process routes]

(a) Trace of a bag that is sent to the flight directly after being scanned

(b) Trace of a bag directed to storage

(c) Trace of a bag that recirculates in the Sorter 1

Figure 3.16: Example traces highlighted on the Layout Graph

\[ \text{Check In} \rightarrow \text{Sorter 1} \rightarrow \text{Scanner} \rightarrow \text{Sorter 1} \rightarrow \text{Sorter 3} \rightarrow \text{Flight} \]  (3.1)

\[ \text{Check In} \rightarrow \text{Sorter 1} \rightarrow \text{Scanner} \rightarrow \text{Sorter 1} \rightarrow \text{Sorter 2} \rightarrow \text{Storage} \rightarrow \text{Sorter 2} \rightarrow \text{Sorter 3} \rightarrow \text{Flight} \]  (3.2)
CHAPTER 3. USE CASE

Check In → Sorter 1 → Sorter 1 → Sorter 1 → Scanner → Sorter 1 → Sorter 3 → Flight

(3.3)

Check In → Sorter 1 → Scanner → Sorter 1 → Scanner → Sorter 1 → Sorter 3 → Flight

(3.4)

Check In → Sorter 1 → Scanner → Sorter 1 → Inspection → Sorter 1 → Sorter 3 → Flight

(3.5)

• Process 3.1, is the fastest high-level process possible. See an example trace in Figure 3.16a.

• Process 3.2 shows a case of a bag with an early arrival. See an example trace in Figure 3.16b.

• Process 3.3 takes place when there is a high load in Sorter 1 and the bag can’t be scanned in the first round. The bag recirculates in Sorter 1.

• Process 3.4 takes place if the bag is misread in the scanner. See an example trace in Figure 3.16c.

• Process 3.5 shows a case when the scanner raises an alarm and the bag must be further inspected.

Low-level Transport Process Variants

Let’s look at bags that follow the same HL variant but different LL routes. Figure 3.17 shows a small subset of bags in the HL events "Sorter 1" and "Scanner”. It compares the bag traces by total time in the process (we can categorize them as fast or slow bags), and the number of movements (they can be split by short and long traces) in the subprocess. In Figure 3.17, the green line is a great outlier, which makes it impossible to discern the other bag traces. 3.18 shows the same traces except for the outlier, which reduces the scale of the y-axis and allows us to discern each bag trace. We observe:

![Figure 3.17: Line Chart of system time per bag over low-level processes with a clear outlier](image-url)
• One clear outlier (dark green line in Figure 3.17), very slow bag and very long trace: bag ID = A. 3 hours 21 minutes (03:21:00). In Figure 3.19a we see its route: It recirculates twice in the Sorter 1, it was scanned once and stopped twice in an inspection station.

• Slow bag, long trace: bag ID = B, 87 moves, 7 minutes (00:07:14). It is the bronze line in the line chart in Figure 3.18. In Figure 3.19b we see its route: it recirculated once in the Sorter 1, was scanned once and stopped once in an inspection station.

• Slow bag, short trace: bag ID = C, 49 moves, 6 minutes (00:05:59). It is the grey line in the line chart in Figure 3.18. In Figure 3.19c we can see its route: the bag did not need to recirculate in the Sorter 1, it never entered a scanner and but it was manually inspected, which is a longer activity than a scan.

• Fast bag, long trace: bag ID = D, 76 moves, 2 minutes (00:02:38). It is the dark green line in Figure 3.18. In Figure 3.19d we see its route: it was scanned only once and it did not need to recirculate in Sorter 1. It entered and left the sorter from the same point, doing a complete loop.

• Fast bag, short trace: bag ID = E, 25 moves, 2 minutes (00:02:27). It is the orange line in the line chart in Figure 3.18. We can see its route in Figure 3.19e: it is the shortest route, because it was scanned once, it did not recirculate in the Sorter 1 and its exit was right after the scanners.

Figure 3.18: Line Chart of system time per bag over low-level processes

3.2.2 User Objectives

The user defines the goal as detecting which are optimal and non-optimal routes. The high-level process a bag goes through can be classified as a default route or not. The goal is for the bags to encounter the minimal number of incidents. Processes 3.1 and 3.2 are considered default routes: given the initial conditions, the bag followed the optimal process along the airport. Process 3.3 is a non-optimal route because the re-circulation could be avoided by managing the load in the system better, e.g. by balancing the load among the scanners available or using more scanners.

So, looking at the variants, they could be classified into happy flow and non-happy flows. In Business Process Modeling, a happy flow is a sequence of activities that is executed if the process does not encounter exceptions. [7] In the BHS problem, the domain-experts manually decide which
are default routes and which are deviating routes. The goal of this thesis is not to automate the
detection of default and deviating routes, but to automate the route generation (Section 7.2), and
provide visual analytics for the domain-experts to observe the performance of the routes (Section
7.3).

3.2.3 Benefits for the User

We can summarize the benefits of detecting the optimal and non-optimal for the user as the fol-
lowing. Knowing this information about HL routes can help improve the decision making process
in the BHS to assign better performant sequences of HL events to the bags. From the LL process
point of view, knowing the performance of each LL variant can help understand how the LL route
taken affects the performance of the bag, it can help to detect incidents (e.g. traffic jams) in
the system, and propose improvements of the physical routing system. Below we detail how not
having this information available affects the performance in the BHS.

Detecting the performance of HL routes can help improve the decision-making process in the
BHS. The system should try to assign each bag the sequence of HL events possible that is closer
to the optimal. For example, Process 3.2 is an optimal route when the bag enters the system with
much time to spare. However, many times the bag is directed to storage when it would not be
necessary. One of the goals of obtaining information about the flow in the system is to detect
when the route followed is non-optimal.
CHAPTER 3. USE CASE

The bags exit the BHS when they are loaded into the flight (or directed to another terminal). Due to the concurrence of flights scheduled, a bag cannot be directed to the exit gate until the exit gate is assigned to its flight. Therefore, when a bag is scanned in Sorter 1, the system checks if its exit gate is available. If the exit gate is available, the bag is immediately directed to the flight, see process 3.1. Otherwise, the bag is sent to storage until boarding time, see process 3.2.

With the information they have available, the process engineers try to detect if there are many cases where a bag is directed to the storage, and closely after, their gate is made available. Those bags circulate unnecessarily in the system when they could be sent to the gate and arrive at the right time. If those cases can be detected and bags can be directed sooner to their exit, the load in the system will decrease, decreasing the traffic jams. Further, the In-System-Time of the bags would be reduced, diminishing the chance of missing the flight.

3.2.4 Business Problem Definition
We can conclude that we aim to obtain a process model to observe how the bag traces deviate from the model. The model must describe "ideal" routing and performance to enable the user to detect/observe deviations from routing and performance. Therefore, we aim to:

• Understand each process separately
• Understand the flows of the system as a whole, which can bring further insights, as
  – Understand how the load of the system affects the performance
  – Detect bottlenecks in the system

In the next sections, we study which is the data available that is needed for this project, and how its quality helps of challenges the project. The Business Problem we start defining here in a generic manner, is then detailed into concrete objectives in Section 3.6.

3.3 Overview of Data Sources
In this section, we take a first look at how the event data of the BHS looks like. The quality of the data available will affect the possible techniques we can apply in this case. Our BHS generates event data to keep track of which activities each bag passes through. Having event data, we can build a log to reproduce the bag traces. In Table 3.1 we see the events from 3 bag traces.

3.3.1 Data Source for Bag Traces
In Table 3.1 we observe a list of three traces, identified by 3 different Case ID values (column ID2). Each row in the table represents an event that took place: an item (a bag, in this case) performed an activity at a certain point in time. The column Timestamp show when did each event took place. Then we have other columns available, and we must decide which data do we want to use as an activity. The activity defines which event took place, i.e. what was the process performed during the event. In the BHS, we wish to understand further how the routing of bags along the system work. Therefore, what we want to analyze as an activity are the locations in the system that a bag passes through. In the case of our event log, the location in the system is defined by three fields: AreaID, ZoneID and EquipmentID.

3.3.2 Lack of a Business Process Model
We can understand the BHS layout as the physical model that defines which sequences of events (bag movements) are possible. However, there is no business process model assigned: the process each bag goes through is decided along the way (depending on its final destination, the results of scanning its contents, the load of the system at that moment, the waiting time until
Table 3.1: Event data from 3 bag traces

<table>
<thead>
<tr>
<th>ID2</th>
<th>Timestamp</th>
<th>AreaID</th>
<th>ZoneID</th>
<th>EquipmentID</th>
<th>Trigger</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>10465742</td>
<td>01/01/2018 11:56:53</td>
<td>3194</td>
<td>1</td>
<td>97</td>
<td>Route with Destination</td>
<td>31</td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:56:57</td>
<td>3132</td>
<td>1</td>
<td>98</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:57:04</td>
<td>3104</td>
<td>2</td>
<td>33</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:57:06</td>
<td>3104</td>
<td>2</td>
<td>34</td>
<td>Default Route</td>
<td>31</td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:57:47</td>
<td>3100</td>
<td>7</td>
<td>5</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:58:07</td>
<td>3100</td>
<td>11</td>
<td>3</td>
<td>Route with Destination</td>
<td>31</td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:58:16</td>
<td>3100</td>
<td>13</td>
<td>3</td>
<td>Route with Destination</td>
<td>31</td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:58:28</td>
<td>3100</td>
<td>15</td>
<td>7</td>
<td>Route with Destination</td>
<td>31</td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:58:35</td>
<td>3108</td>
<td>2</td>
<td>1</td>
<td>Route with Destination</td>
<td>31</td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 11:59:38</td>
<td>3243</td>
<td>4</td>
<td>1</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 12:00:11</td>
<td>3243</td>
<td>6</td>
<td>1</td>
<td>Route with Destination</td>
<td>31</td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 12:00:24</td>
<td>3260</td>
<td>4</td>
<td>4</td>
<td>Default Route</td>
<td></td>
</tr>
<tr>
<td>10465742</td>
<td>01/01/2018 12:01:10</td>
<td>3260</td>
<td>9</td>
<td>96</td>
<td>Route with Destination</td>
<td>31</td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:20:18</td>
<td>3196</td>
<td>1</td>
<td>97</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:20:22</td>
<td>3133</td>
<td>1</td>
<td>98</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:20:40</td>
<td>3104</td>
<td>2</td>
<td>38</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:21:14</td>
<td>3100</td>
<td>7</td>
<td>5</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:21:34</td>
<td>3100</td>
<td>11</td>
<td>3</td>
<td>Default Route</td>
<td>206</td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:21:43</td>
<td>3100</td>
<td>13</td>
<td>3</td>
<td>Default Route</td>
<td>206</td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:21:55</td>
<td>3100</td>
<td>15</td>
<td>7</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:22:02</td>
<td>3108</td>
<td>2</td>
<td>1</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:23:05</td>
<td>3243</td>
<td>4</td>
<td>1</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:23:35</td>
<td>3243</td>
<td>6</td>
<td>1</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:23:48</td>
<td>3260</td>
<td>4</td>
<td>4</td>
<td>Default Route</td>
<td></td>
</tr>
<tr>
<td>10598075</td>
<td>01/01/2018 11:24:38</td>
<td>3260</td>
<td>9</td>
<td>94</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:16:34</td>
<td>3192</td>
<td>1</td>
<td>97</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:16:38</td>
<td>3180</td>
<td>1</td>
<td>98</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:16:48</td>
<td>3154</td>
<td>2</td>
<td>24</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:16:50</td>
<td>3154</td>
<td>2</td>
<td>25</td>
<td>Default Route</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:16:55</td>
<td>3154</td>
<td>2</td>
<td>30</td>
<td>Default Route</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:17:33</td>
<td>3150</td>
<td>6</td>
<td>5</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:17:53</td>
<td>3150</td>
<td>10</td>
<td>3</td>
<td>Default Route</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:18:02</td>
<td>3150</td>
<td>12</td>
<td>3</td>
<td>Default Route</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:18:14</td>
<td>3150</td>
<td>14</td>
<td>7</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:18:23</td>
<td>3158</td>
<td>2</td>
<td>1</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:19:30</td>
<td>3242</td>
<td>4</td>
<td>1</td>
<td>Transport</td>
<td></td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:19:59</td>
<td>3242</td>
<td>6</td>
<td>1</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:20:24</td>
<td>3260</td>
<td>5</td>
<td>4</td>
<td>Default Route</td>
<td></td>
</tr>
<tr>
<td>10633251</td>
<td>01/01/2018 10:21:11</td>
<td>3260</td>
<td>9</td>
<td>94</td>
<td>Route with Destination</td>
<td>206</td>
</tr>
</tbody>
</table>
3.4 Data Understanding

In this chapter, we explore the data available in our BHS that can be used in our performance-aware Deviation Detection method. We explore the information available and the quality of it, to decide which steps our methodology must follow, and which is the data pipeline required to fulfill all tasks.

In Section 3.4.1 we present the Event Log data: Data Complexity explains which data is available, Data Sources presents in which format the event data is stored, and Data Quality: Incomplete Bag Traces presents the rationale for which data is stored. Section 3.4.2 presents the routing data that can be used to complete the missing event data, and in Section 3.4.3, we propose a graph data model that contains all the routing information. In contrast to a Relational Database this depicts a visual approach to understand how the data is connected.

3.4.1 Event Logs Storage

The components in the BHS exchange information with one another about the state of the bags and the system. When the bags are transported through the BHS, many logs are generated by different components with messages used by the system: tracking messages, error messages, information messages... For operational purposes, those messages are read on the fly by other components. For analytical purposes, some of those messages are stored in reports.

Data Complexity

In order to control a BHS, there are high-level systems, keeping track of each bag along the entire airport, and low-level systems, keeping track of the bag through every single location. The code of the label physically attached to the bag is the identifier used in the high-level system (from now on known as ID1).

High-level data, ID1 The high-level system stores the personal details of the passenger and the information about the flight. Each bag is supposed to be identified by a unique ID1 along the entire BHS. However, there can occur exceptions when a bag requires to be relabeled, e.g. a bag misses the flight and it is assigned to a new flight. In those cases, from that moment on, the bag is assigned a new ID1. In Table 3.2, the bag with ID1 = 1 is assigned to a new flight and relabeled as ID1 = 4.

<table>
<thead>
<tr>
<th>ID1</th>
<th>Timestamp</th>
<th>Name</th>
<th>Flight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01/01/2019 07:02:46</td>
<td>Jane Doe</td>
<td>AA 4356</td>
</tr>
<tr>
<td>2</td>
<td>01/01/2019 07:32:18</td>
<td>John Doe</td>
<td>HV1298</td>
</tr>
<tr>
<td>3</td>
<td>01/01/2019 07:51:13</td>
<td>Richard Roe</td>
<td>BA 9922</td>
</tr>
<tr>
<td>4</td>
<td>01/01/2019 09:08:24</td>
<td>Jane Doe</td>
<td>AA 1133</td>
</tr>
</tbody>
</table>

Table 3.2: Event Log from high-level processes, identified by ID1

Low-level data, ID2 The low-level systems control smaller areas in the airport. A low-level system assigns a unique identifier to each bag (from now on known as ID2), and tracks each movement of the bag along this area of the airport. The BHS contains many low-level systems, and when a bag switches low-level systems, it is registered as a new item and assigned a new ID2. Therefore, it is usual that a bag is identified by multiple ID2 values while it moves along the airport (see Table 3.3).
CHAPTER 3. USE CASE

ID2 Timestamp Location Message
1 01/01/2019 07:02:46 010101 Registered
1 01/01/2019 07:04:18 010201 Directed to Sorter 1
2 01/01/2019 07:06:13 030101 Scanned
2 01/01/2019 07:08:24 041506 Directed to Sorter 2

Table 3.3: Event Log from low-level processes, identified by ID2

Data Sources

The event data we presented in Section 3.4.1 is stored in different tables in a database, depending on the source of the logs. Here we present the three tables that store event logs that are significant for our project. Every day, The Bag Information Report, the Tracking Report 1 and the Tracking Report 2, collect information from the bags that enter the system.

Bag Information Report

The logs that inform about changes in the attributes of the bag are stored in the Bag Information Report. They include the personal details of the passenger, the physical properties of the bag, the flight it is directed to, the final destination, etc. When some of that information changes, a new record is inserted in the file. In the case of the bag missing the flight, its ID1 will change (it is directly linked to the bag tag) since it has to be relabeled to board another flight. It is possible to keep track of the bag by relating ID1 values with ID2 values. The ID2 value of a bag changes more often than ID1: it changes every time the bag is moved to an area controlled by a different programmable logic controller (PLC).

Table 3.4 shows the logs corresponding to a single bag that we can find in the Bag Information Report. Some attributes, such as dimensions of the bag, and passenger’s personal details, have been excluded. In Table 3.5, we show a case of a bag being retagged, and we observe that it conserved the ID2 value. Nevertheless, the moment the bag is retagged, it is assigned a new ID1 value.

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Timestamp</th>
<th>Activity</th>
<th>On_Schedule</th>
<th>Message_Content</th>
<th>Identify_Status</th>
<th>Flight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:36:37</td>
<td>Overrule</td>
<td></td>
<td>Planning_overrule</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:36:37</td>
<td>Scan_1</td>
<td>OnTime_1q_flight</td>
<td>FirstScan</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:36:37</td>
<td>Scan_1</td>
<td>OnTime_1q_flight</td>
<td>FirstScan</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:36:37</td>
<td>Scan_1</td>
<td>OnTime_1q_flight</td>
<td>FirstScan</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:36:37</td>
<td>Scan_1</td>
<td>OnTime_1q_flight</td>
<td>FirstScan</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:40:41</td>
<td>To_flight</td>
<td>OnTime_1q_flight</td>
<td>Route_1q_flight</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:40:41</td>
<td>To_flight</td>
<td>OnTime_1q_flight</td>
<td>Route_1q_flight</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:45:20</td>
<td>To_flight</td>
<td>OnTime_1q_flight</td>
<td>Route_1q_flight</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:45:20</td>
<td>To_flight</td>
<td>OnTime_1q_flight</td>
<td>Route_1q_flight</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:45:47</td>
<td>To_flight</td>
<td>OnTime_1q_flight</td>
<td>Route_1q_flight</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
<tr>
<td>0125222283</td>
<td>75480489</td>
<td>01/01/2018 11:46:07</td>
<td>To_flight</td>
<td>OnTime_1q_flight</td>
<td>Route_1q_flight</td>
<td>Identified</td>
<td>AA_0202</td>
</tr>
</tbody>
</table>

Table 3.4: Slice of Bag Information Report, for bag with ID2 = 75480489

Tracking Report 1 and Tracking Report 2

When a bag is tracked by some sensor or scanned by a scanner, logs are stored in those files, stating the location in the airport (AreaID, ZoneID, EquipmentID), the time of the event, and the identifier of the bag (ID2). If the bag must be directed to a segment different than the one by default, the field Destination is included, to specify a segment in the Routing Database. The logs are stored in different reports depending on the source that generated them. Here we observe logs stored in Tracking Report 1 and Tracking Report 2.
CHAPTER 3. USE CASE

Table 3.5: Slice of Bag Information Report, where a bag misses a flight. ID2 remains the same. The bag is assigned a new ID1

Table 3.6: Slice of Tracking Report 1, for bag with ID2 = 75480489

Table 3.7: Slice of Tracking Report 2, for bag with ID2 = 75480489

Table 3.8: The union of the logs in both files for the bag with ID2 = 75480489.
### Data Quality: Incomplete Bag Traces

**Logs are only stored in critical locations** From all logs generated by the BHS, only a subset is stored for analytical purposes. The logs are sparsely stored due to the high volume of logs that the system generates. The amount of logs stored has been reduced to the ones that provide the most significant information for data analysts to look at to **limit the storage requirements to a minimum**.

The event logs can be obtained from different sources; the information about the location of the bags can be retrieved from different messages that the system generates when it interacts with the bag. As depicted in Figure 3.20, different nodes in the BHS (scanners, tracking sensors, manual work) generate different logs. Hence the generation of logs depends on the location and the activity taking place. Furthermore, not all logs generated are stored for posthoc analysis. The system generates a tracking message in every location, but only the critical messages to track the bag routing are stored.

Then, we retrieve the messages stored from different sources. **Tracking Report 1** contains messages related to tracking (tracking message, change of direction, etc.) and **Tracking Report 2** contains messages that inform about operations performed on the bag (scanning message, inspection message, etc.). To have the most complete input data about the bag traces, we will have to **union the logs from both sources**.

**Duplicated logs or incoherences among data sources** One rule that must be met when generating the bag traces is that a bag can not be located in more than one place at the same time. Repeated timestamps can be caused because an event might launch several messages at the same time. A data source might link a message to the current node while another relates the event to the previous node, causing an incoherence in the log. We have found several cases of pairs of locations in the system that generate logs with duplicate timestamps. Those logs must be found and rectified to complete the bag traces.

**Routing information to complete bag traces** Nevertheless, the traces will still be incomplete. In Figure 3.21 we depict in pink the part of the trace that can be drawn from the content of the logs. Knowing the information in the logs, the pink arrows show the parts of the trace that can be discovered. However, knowing the dynamic routing rules, the trace can be built. The completed trace is shown in Figure 3.22.

---

**Table 3.8**: Union of Tracking Report 1 and Tracking Report 2 for the bag with ID2 = 75480489. The rows that belong to Tracking Report 2 are highlighted.

<table>
<thead>
<tr>
<th>ID2</th>
<th>Timestamp</th>
<th>AreaID</th>
<th>Trigger/Activity</th>
<th>Destination</th>
<th>EquipmentID</th>
<th>ZoneID</th>
</tr>
</thead>
<tbody>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:38:38</td>
<td>3050 Transport</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:38:44</td>
<td>3050 Route with Destination</td>
<td>63</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:39:03</td>
<td>5540 Default Route</td>
<td>4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:40:04</td>
<td>5540 Transport</td>
<td>95</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:40:08</td>
<td>5540 Route with Destination</td>
<td>64</td>
<td>94</td>
<td>22</td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:40:20</td>
<td>5536 Transport</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:40:41</td>
<td>5536 Scan</td>
<td>99</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:40:59</td>
<td>5536 Route with Destination</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:41:18</td>
<td>5540 Default Route</td>
<td>3</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:42:01</td>
<td>5540 Transport</td>
<td>95</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:42:11</td>
<td>5540 Route with Destination</td>
<td>3</td>
<td>94</td>
<td>43</td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:43:11</td>
<td>5542 Transport</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:42:20</td>
<td>5542 Transport</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:42:36</td>
<td>5542 Transport</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:43:04</td>
<td>5544 Route with Destination</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:43:36</td>
<td>5570 Default Route</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:45:19</td>
<td>5570 Route</td>
<td>3</td>
<td>99</td>
<td>37</td>
</tr>
<tr>
<td>75480489</td>
<td>01/01/2018</td>
<td>11:45:19</td>
<td>5570 Route with Destination</td>
<td>3</td>
<td>80</td>
<td>10</td>
</tr>
</tbody>
</table>
CHAPTER 3. USE CASE

Figure 3.20: Subset of a BHS. Different nodes in the BHS (scanners, tracking sensors, manual work) generate different logs.

The complete trace is shown by the set of thick arrows. All the black arrows of the trace are missing events.

Process Mining techniques require all of the events of the traces to be available to actually discuss and compare routes and visualize them in the system for the analyst. We conclude that to complete the bag traces, we need to retrieve the event logs and routing information, which explains how the route of each bag is decided.

Figure 3.21: Subset of a BHS. The thick arrows represent the route of a bag, where only the pink arrows are present in the log.

3.4.2 Routing Information

In this section, we retrieve tables from the Routing Database and explain the information they contain. The necessary information to understand and reproduce the routing logic of our BHS is comprised in the Data Model displayed in Figure 3.22. In Appendix B we define the tables and their fields that are essential for the routing to take place.

In the next subsection, we transform this relational data model into a property graph and...
CHAPTER 3. USE CASE

Figure 3.22: Routing Database

implement it into a Graph Database using Neo4j. The goal is to prove that the routing database represents the Layout Graph of our BHS accurately and that following a route in the system is consistent with the routing rules.

3.4.3 Property Graph Definition of the Routing Database

In this subsection, we show the data model of the routing database in a property graph[4]. Henceforth, a node will be identified by the concatenation of the identifiers area_id, zone_id, equipment_id. This notation is already used in the Layout Graph, but in the Routing Database, the primary key is node_id.

\[ location_id = area_id \| zone_id \| equipment_id \]

Figure 3.23 depicts the schema of the resulting property graph. We renamed the nodes as locations since the term node can be confused with the vertices in the graph. Each row from the table SEGMENTS is an edge between a pair of LOCATION nodes. Each segment is also represented as a node labeled as ROUTE. The goal of having ROUTE as an entity is to connect the segments with its usages since the edges could not be treated as entities. To reduce the data model, the preferences and the destinations are attributes in the nodes of label USAGE. The nodes of type USAGE are connected to nodes of the label RULE, which consecutively are connected to a pair of nodes of the label STATUS. Each node and edge contains the minimal attributes required for the routing.

**Build the Routing Graph**  We loaded the routing tables to a Graph Database with the schema described above using Neo4j. Neo4j is one of the Graph Databases that supports more essential graph queries; it supports node/edge adjacency, reachability in fixed-length paths, regular simple paths, shortest path, and pattern matching queries. Also, it has remained the market leader until the present.[5]

Therefore, we proceed to use Neo4j to build our property graph in Neo4j and we will use it to perform routing queries. In Appendix A.1 we provide the cypher queries used to load the tables into the graph. We obtain the database schema presented in Figure 3.23. The size of the graph is 93,071 nodes of 5 labels, and 204,088 edges of 7 types. We observe in Figure 3.25 and Figure 3.26 how some of this data is seen in the Neo4j console, and how is it interpreted.
Testing the graph   Having domain knowledge about how the BHS works, we verify with some examples that the routing information is consistent with the ground truth of the system. For example, to choose between Sorter 1A or Sorter 1B, they have the same level of preference, i.e. none is selected as the sorter by default. This can be seen in the first divert of Figure 3.24, because the two outgoing edges from the node "Divert to Sorter 1" have the property transport_default = No (i.e. scenario 3 from Figure 3.13). However, looping through the Sorter 1 is the default route versus entering a scanner or an inspection station. If the bag has to enter a scanner, it does not have to follow the default route, thus the event log provides a destination, and connecting the usage with the location returns the right route. In Figure 3.24 we display a subset of nodes LOCATION, and we highlight the value of the property transport_default of each edge. It shows that in the first divert, none of the outgoing segments is the default route. The nodes that only have one outgoing edge do not require to set the edge as default either. It is only in the case when one route will be chosen over the others, continue in Sorter 1 instead of entering a scanner that transport_default is set to true.

In Figure 3.25 we display the same example as in Figure 3.24, but we include the nodes ROUTE to show how they are equivalent to the edges SEGMENT_TO. Each route connects to its start_node with the edge FROM, and to its end_node with the edge TO. Figure 3.26, shows a close-up view of the divert in the node "Divert to Scanner 1". As shown in Figure 3.24, the edge to "Divert to Scanner 2" is the default segment. Therefore, the bag will only be directed to "Scanner 1" if the event log includes destination = 63. We see that route "r_05" is not connected to any node USAGE since the default segments do not require it. Also, what the blocking rule shows, is that if the Scanner 1 returned an error status between levels 3 and 7, after 15 seconds, the route to Scanner 1 would be blocked, and the bag would follow the default route and be assigned again the destination of another of the scanners.
Conclusion of Data Understanding

In this chapter, we conclude that the Event Data available has the potential to describe each bag trace. Using the routing information it is possible to deduce the missing events for each bag trace. Having the physical structure of the BHS and the routing information, the information we are missing to compare our bag traces with the optimal variants is the list of ideal routes. The requirement of obtaining the ideal routes leads us to Section 3.5 Data Science Problem.

3.5 Data Science Problem

In this section, we propose Performance-aware Conformance Checking as the method to detect deviations in process variants in a complex system where no process model is available.

In this BHS problem, we lack a process model to detect the deviations of the process variants. The proposal for a new technique to obtain the process model relies on the following arguments:

1. The physical model cannot be used to compare a bag trace to an optimal variant because the physical model allows all possible movements - any bag will conform to the physical movements.

2. Discovering a business process model from the data with standard techniques would also just return a physical model based on the most frequent routes despite they are not necessarily the fastest/optimal.

In absence of a model, we propose to do deviation mining/detection (see Section 2.1.2), which requires 3 steps: identify and group different trace variants by behavior and performance, identify

Performance-aware Conformance Checking on Material Handling Systems
“ideal” or “optimal” variants, and compare all traces against these optimal variants to identify deviants.

3.5.1 Performance-aware Conformance Checking (PA-CC)

The first goal of this project is to aggregate the bag traces to generate the routes taken and the load of each. By aggregating the bag traces we find the main process variants. We aim to detect in which routes the bags’ performance deviates from the bags that follow the process model routes.

For this purpose, we propose the method of Performance-aware Conformance Checking (PA-CC). We propose that Performance-aware Conformance Checking consists of discerning the default routes in the BHS. We understand the default routes as the ones that experience the least exceptional behaviour.

The classical Conformance Checking techniques consist of measuring the quality of a process model concerning an event log. [32] In this BHS problem, the event log is not the ground truth of what took place since we know it is incomplete. The goal of Conformance Checking is to verify the quality of the imputed model where the real process is unknown. Our goal is not to see if the discovered process model is equivalent to the original process model. In our case, the ground truth is the physical layout, i.e. the real model, but our goal is to see how the BHS is used, based on the route and the performance of each trace.

The method underlying our Performance-aware CC proposal consists of differentiating the default routes from the other routes. The question that remains to be answered is how the default routes help to evaluate the performance of the bag traces. Consequently,

- The default routes will become the baseline of the Baggage Handling process, i.e. the model all the items aim to follow.
- By recognizing the default routes, it is possible to identify other routes variants and compare their behaviour to the baseline.

Therefore, Figure 3.27 shows how our PA-CC method differs from the classical CC method. Classical CC aims to detect control flow deviations in the traces by comparing the logs with a business process model. In other words, it aims to detect when the events in the logs and in the model differ. In the use case of Baggage Handling, the system contains a large number of possible events, and it is in the nature of the process that bags take different routes (e.g. using the system efficiently by load balancing). The goal of our PA-CC is not to detect the deviations in the control flow, but to detect when the performance of the routes differs from the model by taking longer or slower routes. To be able to perform this analysis, a model of the routes must be created since it does not exist yet.
3.6 Definition of Technical Sub-problems

Our starting point to provide PA-CC is that there is not a business process model defined apart from the physical model, the BHS itself. Thus we have to use the event logs to see the possible routes in the system. For that, the data needs to be precise enough, we must be able to track each bag along the entire BHS. Thus, it is one of the main goals to complete the event log data. In Section 3.6.1 we present how to improve the quality of the Event Log data. In Section 3.6.2 we introduce the requirements for providing Transparent Data Mining: having a large amount of data and not having any human-made ground-truth model available there is a high chance of misinterpreting the data and of overlooking errors in the method. Also, Section 3.6.3 introduces the requirement to understand the data about routes and HL processes having a large and complex system, which can be addressed providing adequate information visualizations.

3.6.1 Improving Data Quality for Bag Traces

Label logs with unique bag identifiers (Chapter 5)  In order to identify the logs that belong to each bag, we need to relate the ID1 and ID2 values that belong to the same bag. The high-level event logs also insert a record every time a bag is assigned a new ID2 (see Table 3.9). To track each bag throughout the system, we need to relate the high-level and low-level identifiers. Both types of identifiers can change along the system. The logs belonging to each bag must be labelled with a unique identifier.

<table>
<thead>
<tr>
<th>ID1</th>
<th>Timestamp</th>
<th>Name</th>
<th>Flight</th>
<th>ID2</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/2019 07:02:46</td>
<td>Jane Doe</td>
<td>AA 4356</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>01/01/2019 07:04:18</td>
<td>Jane Doe</td>
<td>AA 4356</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>01/01/2019 07:51:13</td>
<td>John Doe</td>
<td>HV1298</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>01/01/2019 09:08:24</td>
<td>Richard Roe</td>
<td>BA 9922</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.9: Event Log from high-level processes, identified by ID1 and including ID2

Find the complete route of each bag trace following a physical path. Handle noise in the logs (Chapter 6)  Since we union logs from different data sources, we can find incoherences in the data, such as an event is stored by consecutive pieces of equipment instead of by a unique piece of equipment. Also, to recognize that two bag traces belong to the same transport process variants, they both need to include all the physical locations they passed through.
CHAPTER 3. USE CASE

3.6.2 Transparent Process Mining

We aim to create a clear methodology that provides the user with Transparent Process Mining: the user interacts with the data pipeline and understands the results obtained at each step. Hence, our methodology cannot contain black boxes.

Requirement for a user interface to customize the BHS analysis  The data analyst has access to domain knowledge. For our methodology to be useful for the user, it must allow to perform different analyses, to filter the data and aggregate it to observe concrete parts of the process and concretes points in time. Therefore, it is a requirement to provide a user interface where the analyst decides which files, parameters, and filters to use. The user interface should also allow the pipeline to be performed in steps since some steps require a longer time to execute and only need to be performed once, while others, like classification, take less time and will be performed more frequently.

Observation of fine-grained events  The BHS is a complex system with high-level and low-level processes, and the user must have access to its fine-grained data. One of the tasks of the analyst is to reach conclusions about which equipment needs to be looked at for maintenance, and thus, they must provide the operators with exact locations in the system. Therefore, our methodology cannot aggregate the activities in the process to reduce the spaghetti model. The events must never be treated as a black box. For this purpose, our methodology provides intuitive information visualizations to the user. The user is familiar with the Layout Graph of the airport (see Figure 3.11), and we propose to plot the information over the well-known structure of the system.

3.6.3 Information Visualization

Our methodology includes information visualizations that fulfill the following requirements:

- **BHS structure understanding** (which area/process are we looking at)
- **Performance metric understanding**: load in the system, In-System-Time, bags movements, etc.
- Observe **deviation in performance** of bags among process variants, or among events
- Allow for **user input**: The user must be able to decide which data to look at, to choose among possible parameters (bag identifiers, time of the day, location in the airport), and to choose among possible visualizations and calculations.

3.6.4 Problem Definition

The technical problem definition that we propose in this project is summarized as follows:

- Given event data (as described in Section 3.4.1) and routing information (as described in Section 3.4.2) we aim to:
  - Extract *proper event logs* from different sources. (Chapter 5)
  - Complete bag traces finding the sequence of physical locations each bag passed through (Chapter 6)
  - Classify bags into:
    - High-level process variants
    - Transport process variants
  - Aggregate the performance of bags over routes (Section 7.2) to compare:
CHAPTER 3. USE CASE

3.6.5 Analytics Objectives

Below, we further split the analytics and visualization objectives into a complete list of analytics provided for the user:

- Provide visual analytics to let a user identify intended/ideal routes and deviating routes and compare them (see Chapter 7)
- Discover the routes taken, both in the HL process (see Table 7.2 in Section 7.2.1: HL_sequences) and LL processes (see Table 7.5 in Section 7.2.2: routes). Discovering the HL routes allows the user to classify them between default routes and deviating routes, using their domain knowledge to detect the exceptional behavior. Discovering the LL routes provides the user with insights of how the system is being used, how the load balancing system is distributing the bags and how the paths are being used.
- Having identified default routes and deviating routes, provide the user with access to the data of the load of each route (see Table 7.3 in Section 7.2.1: bags_per_HL_sequence, and Table 7.6 in Section 7.2.2: bags_per_route).
- Provide performance-based visualizations to compare bags along their HL process (see Section 7.3.1) and closer looks to the performance of LL routes (see Section 7.3.2)
- Compare the performance of bags along a HL process (see Figure 7.6 in Section 7.3.1) and in more detail, compare the performance for each bag movement to visually detect bags that belong to different categories (see Figure 7.14 in Section 7.3.2).
- Visualize the routes taken in an interpretable form for the user (i.e. over the Layout Graph layout). (All JS visualizations in Section 7.3)
- After the user determined with domain knowledge that a bag trace represents the default behavior and another deviates from it, provide the user with a visualization of the deviation between their performances (see Section 7.3.1, Figure 7.12).
- Detecting incidents in the system (see Figure 7.11)
- Observe the performance of physical routes, based on the load of bags they transported and the time performance of the bags (see Section 7.3.2)

Finally, we test if the performance-aware Conformance Checking approach can answer process performance questions from the BHS analysts. We perform this evaluation in Chapter 8.

3.6.6 Scope Limitations

The PA-CC provides a methodology to detect and visualize the routes used in sections of the BHS and their performance. The scope of this project includes aggregation of bag traces and their time performance into routes, and visualization of routes along the system. The scope does not include data interpretation nor domain knowledge, hence it does not include the analysis of the current state of a BHS. Nevertheless, this methodology can be used by domain experts to observe the process variants and understand the performance of the system. Statistics of the data and other widely-used calculations are not included either as they are considered trivial and can be
performed a posteriori.

The final goal of our methodology is to detect which process variants deviate from the intended process. This is not an outcome that can be automatically obtained from our methodology. Here we want to discuss to what degree is this outlier detection automatic, in other words, how is it decided in our methodology which are the intended routes. The goal of Section 7.2 is to generate the HL & LL variants. Then, the sequences of HL or LL events are given to the user, the process engineer. The process engineer decides which sequences are default routes, i.e. without exceptional behavior, and which deviate. In conclusion, the detection of deviating routes is not automated.

The scope of this thesis is to explore the PA-CC problem and provide visual analytics tools for domain-experts. We provide automatic support for routes generation and time-performance per route generation. Fully-automatic detection of default routes and deviating routes in the BHS process as in classical CC is out of the scope of this thesis. The scope of this thesis is then summarized as follows:

*The scope of this thesis is to explore the performance-aware CC problem and provide some semi-automatic support for deviating process variants detection. The scope also considers providing visual analytics tools for domain-experts to observe the performance of each process variant. Fully-automatic detection as in classical CC is out of the scope of this thesis.*
Chapter 4

Solution proposal

In this chapter, we introduce the approach chosen to develop a method that solves the problems presented in Section 3.6. We proceed to summarize the main problem to solve that led us to choose to work with Graph Databases: For a given bag trace, we have an incomplete route. We want to obtain the complete bag trace that includes all the locations in the BHS layout that the bag passed through and they must be chronologically sorted. The Routing Information explains how are the locations interconnected and can, therefore, be used to find the possible paths that the bag followed between each pair of locations in the trace.

In Section 4.1 we justify why finding the right path that each bag followed is a problem that must be solved querying a Graph Database rather than a relational database. We continue by listing some of the pipeline requirements in each of the following stages of our methodology that lead to the pipeline of our method in section 4.6. Those stages are 4.2 Event Log Pre-processing, 4.3 Complete Traces, 4.4 Aggregation of Bag Traces into Routes, and 4.5 Visual Analytics of Routes.

4.1 Motivation for Graph Database Approach

The goal of this section is to justify why completing the bag traces with traversal queries in a Graph Database is a more feasible approach than trying to complete the missing parts of the trace with a relational approach. Our main argument is that the queries required to find the connections between locations where length of the path is unknown is practically unfeasible in a relational database due to its computational complexity.

To query the routing between a pair of nodes, the Traversal Navigation is the most important property to choose a Property Graph over a relational database. For real-time routing, i.e. finding the next node when a bag is located in a starting node, we require a pattern matching query. This query is formalized as a graph isomorphism. Those algorithms are generally hard to compute and are NP-complete. To recreate a bag trace, our goal is to find a connection between a pair of nodes. That is a reachability query, formalized as a traversal query. Generally, the computational cost of a shortest path query is $O(|Vertices|^2)$.

Shortest Path Query

Our goal is to find the missing sequence of locations between every pair of events registered in a trace. The query to efficiently hop through a sequence of nodes in a graph without knowing the cardinality of the sequence is the Shortest Path query. When the use case requires a shortest path query, working with a Graph Database always beats the relational database in efficiency. The other operations we could perform would not be able to handle a flexible number of hops.
in the data. Below we display how this query performed in a Graph Database beats equivalent operations in tabular form, therefore justifying our approach in this problem.

Computational complexity of the shortest path query in a relational Database: The shortest path query is impossible to perform in a relational database, since they have the limitation that the number of joins must be fixed. It is possible to perform a recursive query, but still in many cases, it returns an incomplete execution.

Computational complexity of the shortest path query in a Graph Database: The computational complexity of Dijkstra’s shortest path algorithm is $O(|N|N^2)$.

4.2 Event Log Pre-processing

In order to turn the raw data into usable data to describe traces of a process, it is required to transform this data along a sequence of steps, i.e. the Event Log Pre-processing. A common procedure to obtain traces from an event log is to select a trace identifier, split the events that belong to each trace, and sort the events of each trace chronologically. This returns traces, sequences of events that define a process. This stage of our methodology is explained in Chapter 5.

The main challenge we encounter in this stage is finding an identifier that represents a unique trace for each item. Currently, each bag is represented by a set of pairs (ID1, ID2). We need to label each bag with a unique identifier. For more details on how was this challenge tackled, see Section 5.1, and further contents are provided in Algorithm 1.

4.3 Complete Traces

In this section, following the approach described in Section 4.1, we list some of the requirements to build a graph and to query the missing paths of each trace in the graph.

Locations not available in the Routing Database The logs also happen to track some "intermediate locations" that are not present in the Routing Database nor the Layout Graph. They do not take part in the routing method. They create noise in our event log and must be removed. We develop an algorithm that takes as input the Event Logs and the Nodes table from the Routing Database and returns the Event Logs where the nodes are the inner join of both files. The full Algorithm 5 is in Appendix D.3.

Events in the log with duplicate timestamp One rule that must be met when generating the bag traces is that a bag can not be located in more than one place at the same time. Repeated timestamps can be caused because an event might launch several messages at the same time. A source file might link a message to the current node while another relates the event to the previous node, causing an incoherence in the log. We have found several cases of pairs of locations in the system that generate logs with duplicate timestamps. Those logs must be dealt with before proceeding with the pattern matching queries.

We remove duplicate events and fix the order of events with identical timestamps, see the method explained in Section 5.2 and full algorithm 2 in appendixes.

To find the shortest path, include duration as a cost To find the shortest path between each pair of nodes in the event log, we will use the average duration that it takes to travel along each segment as the cost metric in the Dijkstra algorithm. Thus, the attribute avg_duration in the table SEGMENTS from the Routing Database is a required field to include in the Routing Graph. See Section 6.1 for more details.
CHAPTER 4. SOLUTION PROPOSAL

For any two subsequent events in a log between two locations N1 and N2, there may be additional nodes the bag visited where the events are not recorded in the log. To complete the log, we run a shortest-path search from N1 to N2 in the LG graph and take avg_duration as a cost. For more details, see Section 6.2.3, where we show how we use Dijkstra’s shortest path algorithm with avg_duration as cost, both in cypher queries, and displaying the patterns obtained.

**Manual movements** As mentioned before in Section 3.1.5, manual movements are not included in the Routing Database. Since they are possible events in the BHS process, all possible manual movements will be inserted as segments in the Graph Database with the maximum value of avg_duration. To recognize which nodes allow a manual movement, the nodes in the graph must include the field node_type from the Routing Database. See Section 6.1.1 for more details.

4.4 Aggregation of Bag Traces into Routes

This section explains the data pipeline requirements for the step of aggregation of Bag Traces into Routes, described in Chapter 7.2. The main requirement is to relate the events in the logs to the high-level event that comprises them, for the bag traces to later be aggregated by HL process variants. Each node of the Layout Graph is categorized into a process step. We will consider the values of the field process step as the possible high-level events.

4.5 Visual Analytics of Routes

In this section we mention the data pipeline requirements to perform information visualization in Chapter 7.3. It is relevant to mention the approach followed to visualize the data since the visualization method chosen affects the rest of the data pipeline. The decisions to be taken are in which data structure is the data stored, and which is its implementation. As a Proof of Concept implementation, we chose to develop a command-line interface to control the steps of the method (implemented in Python). The program generates Javascript visualizations embedded in HTML files. Details given in Chapter 7.

One requirement was that the visualizations must be similar to the Layout Graph the engineers are familiar with. For this, we parse the coordinates from the LG file (given in XML format) and add them to the SVG generated by our python program. In Listing C.3, we can see a snippet of the resulting SVG file that will be parsed to obtain the nodes’ coordinates. The coordinates of the nodes in this file have to be passed to our final visualizations.

4.6 Data Pipeline

This section presents the data pipeline that is required to perform all the steps of the methodology (see Figure 4.1). It is required to mention it since every output is not directly followed by the step it will be used as input (e.g. we obtain the layout coordinates in the Pre-process of Event Logs, but this data is not required until later in the Visualization of Routes).
Figure 4.1: Total Data Pipeline
Chapter 5

Event Logs Pre-processing

In this chapter, we describe the Event Logs Pre-processing step of our methodology. After retrieving raw event data from different sources, we clean it (removing unnecessary fields and records), define trace identifiers and join the data from different sources. We also enrich the event data with added information that will be used in later steps. The outcome of this step are incomplete bag traces. They are incomplete because most of the events of each trace were not recorded, and at this stage, it has not been possible to obtain the missing information yet. These traces are completed with the sequences of missing events in Chapter 6.

In Figure 5.1 we show the pipeline of the procedure described in this chapter. We use event data: the event log Bag Information contains high-level information about each bag (see Table 3.4); the event logs Tracking Report 1 (see Table 3.6) and Tracking Report 2 (see Table 3.7) contain the low-level event data: some locations the bag passed through in the system. The low-level traces are not complete. One bag generates a high-level trace, and multiple low-level traces. All the data from one bag must be related with a unique identifier. We also have available the HL_system file. This is an XML file that relates each LL event to its corresponding HL event (see a snippet in Listing C.1). We also include a diagram of the coordinates of each node in the system, the physical_layout (see Listing C.3). We parse this SVG file to use the nodes coordinates in the visualizations in Chapter 7.

![Pipeline for Event Logs pre-processing](image)

Figure 5.1: Pipeline for Event Logs pre-processing

5.1 Define Unique Items Identifier

The Bag Information Report stores pairs of identifiers (ID1, ID2). For each bag, its last ID1 value is kept as the unique bag identifier. If a new record had an existing ID2 value and a new ID1 value, we must relabel the pairs that contained the old ID1 value with the new one to keep track of which events belong to a unique bag. Below we contextualize this data:
1. A bag is identified in the high-level systems by an ID1 value, and in the low-level systems by an ID2 value.

2. One bag might be assigned a new ID1 value or a new ID2 value. When that occurs, a new pair (ID1, ID2) that identifies the bag is stored in the Bag Information Report, with the timestamp when the change occurs.
   - Therefore, the pairs (1,1), (1,2), (2,2), (2,4), (4,4), are all labels of the same bag. The pair (3,3) identifies a different bag because it has no ID1 or ID2 values in common with the previous pairs.

3. The event logs are related to ID2 values. Our goal is to identify every ID2 value with a unique bag. For that, we will relate every ID2 value with the last ID1 value of each bag.

   For each bag, we iterate over the log and follow the “chain” of ID1 and ID2 identifiers as at every ID change, one of the IDs remains constant. Once we identified the chain, we replace the ID1 and ID2 values found later in the chain with the original ID1 and ID2 value originally assigned to the bag. See Appendix C.1 for the complete algorithm and an example.

   **See Algorithm 1 Algorithm to Reduce Bag Identifiers in Appendix C.1 Algorithm to Reduce Bag Identifiers. Table C.1 Iterations of the algorithm “Reduce bag identifiers” shows the result of iterating through the algorithm.**

5.2 Remove Duplicate Events

Our initial event data has some quality issues. Since we are joining event data from different sources, sometimes the events are duplicate. Exact duplicates are easy to handle. However, we also observe in the data that at certain locations, the events are sometimes stored twice, one for one location, and another for the next location in the route. In order to perform the shortest path queries in Chapter 6 without errors, we need to have the traces sorted correctly, and duplicate timestamps for different locations can appear in the wrong order. Below we describe the algorithm to solve this issue.

**Algorithm to Remove Duplicate Events**

The algorithm works as follows (see full Algorithm 2 in Appendix C.2.):

1. Remove exact duplicate records: find records with duplicate (ID1, timestamp, location_id) and keep only the first.

2. Search for each bag trace for events at different locations, but with the exact timestamp: separate the subset of records with duplicate (ID1, timestamp).

3. Observing the Layout Graph, identify for each pair of locations with duplicates which location precedes the other.

4. Reorder records accordingly where the first record goes 1ms back in time.

5. Join again with the rest of the logs.

   From an event log of 769,489 rows, 25,672 had a repeated timestamp in its bag trace. That corresponds to 3.33% of the events in the event log. After applying Algorithm 2 to modify their timestamps, the log did not contain any more duplicate timestamps.
5.3 Enriching Raw Data

This section comprises the enrichment of the event data by including extra information that will be used in further steps of the methodology.

5.3.1 Parse Process Steps

As mentioned in Section 4.4, in order to aggregate the bag traces into HL process variants, we need to relate each LL event to the HL that contains them. \textit{HL\textsubscript{system}} is an XML file that we parse to obtain the process\_step of each location\_id. The XML format of the file is described in Listing C.1. For further details, see Algorithm 3 in Appendix C.3.

5.3.2 Locations Coordinates

As mentioned in 4.5, we need to parse the SVG file \textit{layout\_graph}. In Listing C.3, we display an example of the metadata of a note. The attribute \texttt{transform} contains the coordinates where the node is located on the canvas. We parse the SVG file in Listing C.4. For further details, see Algorithm 4 in Appendix C.4.

5.4 Resulting Event Log

In the beginning of this chapter, Figure 5.1 expresses the resulting logs after the step of Algorithm 3 (columns [ID2, timestamp, activity, process\_step]) and the final\_bagtags from the step of Algorithm 1 (columns [ID1, ID2]). Those tables are joined on common identifiers (ID2) obtaining the file preprocessed\_logs (see Table 5.1). In the last step, all events can be related to a unique bag trace, and we obtain both, HL and LL information, about each event. This is the data format that will be used to query the graph in Chapter 6: We have incomplete traces, were each recorded event is a row. ID1 is the trace identifier, activity the LL event (each of the nodes in the graph), and process\_step is the corresponding HL\_event.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
ID1 & ID2 & timestamp & activity & destination & EVENTTIME & process\_step \\
\hline
integer & integer & yyyy-mm-dd & integer & integer & yyyy-mm-dd T & string \\
& & HH:MM:ss.SSS & & HH:MM:ss.SSS'Z' & \\
\hline
\end{tabular}
\caption{Preprocessed\_logs.csv}
\end{table}
Chapter 6

Complete Bag Traces through Pattern Matching

In this chapter we present our method of completing the bag traces by querying the traces in a Graph Database, where the Graph Database contains the possible movements along the system (meeting the requirements presented in Section 4.3). This method has 2 steps:

- Building a graph from the Routing Database
- Querying the graph with Pattern Matching

First, we present in Section 6.1 how the graph is built, and in Section 6.2 how we obtain the complete route every bag went through by querying the graph. Finally, in Section 6.3 we describe the output data that will be used in Visual Analytics of Routes.

6.1 Build Graph from Routing Database

In this section, we describe how we build a Graph Database that contains all possible sequences of events in our process, i.e. all possible routes within the BHS. We load the minimum data required from the Routing Database, and then we enrich the graph by including the segments that are manual movements.

Input Data

We use a subset of the tables in the Routing Database presented in Section 3.4.2 where we only consider nodes with location_id and node_type and segments with start/end-node and duration (see Figure 6.1).

![Diagram of Routing Tables]  

Figure 6.1: Routing Tables retrieved. Use location_id as the node identifier
6.1.1 Load Routing Data into a Graph Database

In Listing D.1 we show the cypher queries executed to turn the tables Nodes and Segments into a Graph Database. However, the table Segments only includes information about physical equipment, i.e. conveyors. Every other possible movement in the system, i.e. manual movements, must be included in our graph, as described in Section 3.1.5.

In Appendix D.1 we include the cypher query we executed to load the tables to the graph. After executing this query, we have the Layout Graph in a Graph Database. Figure 6.3 shows the content of that graph. The size of the graph is 1,553 nodes and 21,367 edges.

Figure 6.3: Routing graph

Include Manual Movements

Figure 6.4 depicts the concept of inserting manual movements in the graph. At this stage, the graph includes all events in the BHS that are part of the physical layout (events performed by equipment in the system). However, events performed by manual labor, e.g. manually transporting fragile items, are also part of the process and must be included in the graph. What this entails is that every node that does not have an outgoing segment must be connected with every node that has no incoming segments, because between each of those pairs of nodes, a manual process...
CHAPTER 6. COMPLETE BAG TRACES THROUGH PATTERN MATCHING

can occur.

As mentioned in Section 3.1.5, the BHS is composed of a network of conveyors, but also all possible manual movements are part of the process (see Figure 3.15). The manual movements do not appear in the Layout Graph as segments, but they must be included in our property graph. Looking at the existing types of nodes in Figure B.1, we observe that nodes with node_type = ENTRY are entry points and that nodes with node_type = EXIT are exit points. An entry point does not have any incoming edge, therefore all bags arriving in that node, arrived via a manual movement. An exit point does not have any outgoing edges, therefore bags that arrive to that node, are removed via a manual movement.

To obtain a complete model of all bag movements, we decide to enrich the graph with "manual movement" segments between each exit point and entry point. We assume that manual movements are slower than any other movement, therefore we assign them a very high duration (999s). In the shortest path query from Listing A.5, we consider that the maximum duration possible in a segment is 1000s. We create the segments that belong to manual movements in the graph with the maximum possible duration to be considered in our query. In Listing D.2 we show the Cypher query to create segments between pairs of nodes EXIT-ENTRY. After that, every possible movement in the BHS is included in the Graph Database.

6.2 Complete Traces with Pattern Matching

Having built a Graph Database with the routing information, we proceed to describe in this section how can we find the missing sequences of events in between each pair of events in the incomplete bag traces. We start this section by the starting point, i.e. the data available (see Section 6.2.1), then we comment the challenges faced while implementing this technique (see Section 6.2.2), and the final method is summarized in the final algorithm (see Section 6.2.3).

6.2.1 Input Data

To perform this step, we have the list of incomplete traces that where pre-processed in Chapter 5 (see Table 5.1, preprocessed_logs). We are also using the list of nodes that were available in the Routing Information. We will use it because the event logs might include location_id values that do not appear in the Routing Database. Those must be removed to avoid empty results in Pattern Matching. We will also use the list of pairs (location_id and node_type), to include in the graph the configuration of each node.

6.2.2 Challenges in Pattern Matching

In order to successfully complete the bag traces with the sequence of locations they passed through, our Pattern Matching algorithm encountered the challenges listed hereinafter. The final algorithm proposed in Section 6.2.3 already addresses them.
CHAPTER 6. COMPLETE BAG TRACES THROUGH PATTERN MATCHING

Mismatching Events between the Logs and the Ground Truth
We encounter a different level of detail in the event logs than in the routing information. The event log can include intermediate locations that are not significant for the routing system, therefore, those are not included in the Layout Graph or the Routing database. In the first attempt to query the bag traces in the Graph Database, we encounter empty results when a node is not part of the graph. To solve this error, the locations that are not in the Nodes table from the Routing database must be filtered out before starting the pattern matching queries. This step is included in our data pipeline in the Pattern Matching Algorithm 5. See details in Section 6.2.3.

Lack of Manual Movements in the Routing Information
The Routing Information only considers physical equipment as segments, e.g. conveyor belts. It was not possible to query the path in a bag trace that included any manual movement, because the connection between nodes was not found. Since all possible manual movements are part of the process, they are included in the graph. For further details on how this challenge is solved, refer to Section 6.1.1.

Events with Duplicated Timestamps
Having events with duplicated timestamps, we might query a pair of locations in the graph sorted incorrectly, which would return an empty result. This challenge is solved in Section 5.2, though not automated. In the Event Log Pre-processing, we need to determine the succession of pairs of events manually by visual inspection, but this manual step is only required once per BHS. For details refer to the procedure in Algorithm 2.

Consecutive Missing Events in the Trace
Following the rules for routing, we can find the next node the bag moved to for each location in the trace. However, the query would usually require multiple hops to connect one location in the trace with the next. Applying a recursive query to the Neo4j graph considering all the routing rules (see the Routing Decision Process in Section 3.1.5) has a very expensive execution.

In order to query the graph efficiently, instead of applying a complex recursive query, we follow the assumption that the default path is the fastest among the possible. Hence, we apply Dijkstra’s shortest path algorithm between each pair of locations in the log. One of the requisites that always holds in our BHS is that if the bag does not follow the route by default it must be recorded in the log. We validated this assumption by querying over 30 thousand traces without encountering errors or empty results. Refer to the cypher query in Listing A.5 and Section 4.1 for more details.

Memory Limitations of the Neo4j Console and Lack of Automation
Working with the Neo4j console requires to write one query at a time. Because of heap size limitations, the queries must be as short as possible (one Dijkstra algorithm per query). For every pair of locations in a trace, we execute a query and return results. This procedure requires too many worker hours and must be automated.

6.2.3 Final Algorithm
To deal with the lack of automation from the Neo4j console, our approach is to connect to the graph via py2neo. Therefore, we connect to the graph remotely inside of a procedure that iterates through each trace and each event in the trace. The final pattern matching algorithm is the method we used to complete the bag traces, tackling the challenges encountered. We describe it below; for the complete method refer to Appendix D.3.
CHAPTER 6. COMPLETE BAG TRACES THROUGH PATTERN MATCHING

Figure 6.5 shows the steps followed by the Pattern Matching algorithm. Having the output of the Event Log Pre-processing, i.e. the preprocessed_logs, we remove any location that is not included in the database. We then create a list of bags identifiers, i.e. bag_list, through which we iterate through later. In the event log, we sort each trace chronologically, and we remove consecutive duplicate events. Then, the bag traces are ready to be queried. We iterate through bag_list to complete each trace. For each bag identifier, i.e. ID1, we select its incomplete list of events. For each event in the list, we execute a query in the graph. We can see the result of querying a bag trace in Figure 6.6.

We recognize the start and the end of each trace by the virtual events START and END. In the example from Figure 6.6, we can see that for the first event, we connect the START event to the first event. We can see the cypher query query\_START in Listing D.3 and its interpretation in Figure D.1. For every other event in the incomplete trace, we query the shortest path with the previous event. We can see the cypher query query\_ELSE in Listing D.4 and its interpretation in Figure D.2. There is no information available about at which time the bag reached each of the missing locations. Therefore, we assign to each location the latest possible time the bag could have reached there. We can see that the bag 001 completed the event A at time $t_A$, and completed C at time $t_C$. The bag completed B between A and C, so we assign time $t_C$. When the algorithm finishes iterating through all the events, the trace is closed by the event END. For details refer to the cypher query query\_END in Listing D.5 and its interpretation in Figure D.3.

To reproduce our method, we provide the Python functions that return the parametrized cypher queries in Listing D.6. The full algorithm is described in Algorithm 5.
6.3 Output Data

In order to store the resulting complete traces, we explored two approaches:

1. Storing the bag traces in the Graph Database
2. Storing the bag traces in a relational table

The approach of storing the data in the Graph Database (explained in Appendix D.4) did not scale with growing data. Therefore, we decided to use a relational approach.

The advantage of storing the traces in the graph would be to use Neo4j algorithms for data classification. Using centrality metrics, we could find significant locations in the BHS, e.g. locations with the highest betweenness. However, those classifications were not part of our objectives and that approach had many inconveniences that made it unfeasible, those being:

- The graph would grow considerably.
- Every use of the bag traces requires to query the graph.
- This data model would be very inefficient for queries since the bag identifiers and the timestamps would require many lookups. Having those fields as edge properties would make the execution plan more complex.
- The aggregations of bag traces into process variants had a better fit in a tabular structure.

6.3.1 Storing Complete Traces in Tabular Form

Each bag movement is stored as a record in the table `complete_traces`. Each record represents a triple\([6]\) (source_node, edge, target_node) and contains all the information that will be added in the classifications and visualizations.

- source_node: we provide the following information:
  - source_id: unique identifier
  - source_name: location_id (AreaID.ZoneID.EquipmentID)
  - source_label: AreaID
  - source_keys: location_id, node_type
  - source_xy: coordinates from the LG
• **target_node**: we provide the following information:
  - **target_id**: unique identifier
  - **target_name**: location_id (AreaID.ZoneID.EquipmentID)
  - **target_label**: AreaID
  - **target_keys**: location_id, node_type
  - **target_xy**: coordinates from the LG

• **trace_id**: trace identifier; ID1 if we show a trace, LL_class if we show a route (defined in the next step, Chapter 7)

• **metric**: value of the time metric used to color the edges (defined in the next step, Chapter 7)

• **edge_roundness**: value assigned to the edge to avoid overlap (defined in the next step, Chapter 7)

In the visualizations, the trace_id, the metric and the edge_roundness are not direct inputs from complete_traces.

<table>
<thead>
<tr>
<th>attributes</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>source_id</td>
<td>1551</td>
</tr>
<tr>
<td>source_label</td>
<td>STAR</td>
</tr>
<tr>
<td>source_name</td>
<td>START</td>
</tr>
<tr>
<td>source_keys</td>
<td>{'location_id': 'START'}</td>
</tr>
<tr>
<td>ID1</td>
<td>001</td>
</tr>
<tr>
<td>process_step</td>
<td>CHECK_IN</td>
</tr>
<tr>
<td>timestamp</td>
<td>yyyy-mm-dd HH:MM:ss</td>
</tr>
<tr>
<td>target_id</td>
<td>1203</td>
</tr>
<tr>
<td>target_label</td>
<td>3106</td>
</tr>
<tr>
<td>target_name</td>
<td>31061403</td>
</tr>
<tr>
<td>target_keys</td>
<td>{'node_type': 'DVT', 'location_id': '31061403'}</td>
</tr>
</tbody>
</table>

Table 6.1: complete_traces.csv

This data structure provides a high level of flexibility to visualize it as a network, and to display different metrics. It has the flexibility to perform different aggregations (i.e. group by segment or by route, calculate time_lapse per traces, per segment, or per process_step, etc.) and to apply multiple filters (i.e. filter by time, process_step, ID1, etc.).
Chapter 7

Visual Analytics of Routes

In this chapter, we describe how we provide to the user the visual analytics to identify intended routes and deviating routes and compare them. In the previous chapter, we obtained the complete bag traces from the Graph Database. In these chapters, we explain the procedures to obtain the aggregated data and visualize it to view and understand every HL and LL process.

In Section 7.1 we explain how we aimed to visualize the data and the visualization method chosen and the challenges to view the data. In Section 7.2 we explain the different aggregations that were required, and which insights do each of them provide, both for HL processes (see Section 7.2.1) and LL processes (see Section 7.2.2).

Then Section 7.3 details the visualizations obtained, which goal does each accomplish and how can they be interpreted. Finally, Section 7.4 explains how the user can interact with the application to customize the visualizations they want to obtain.

7.1 The Problem of Routes Visualization

In this section, we present which technology we use to obtain visual analytics of the routes, and how we structure the data to make it possible. That is detailed in Section 7.1.1, and then in Section 7.1.2 we observe the challenges encountered for the visualizations to be meaningful for the user, which leads us to split our work into two parallel pipelines for HL processes and LL processes in the next Section 7.2 and to consider a user interface an essential part of our methodology, further detailed in Section 7.4.

7.1.1 Visualization Approach: Javascript Networks

In this section, we present how to obtain the clearest visualizations of traces and routes along the system by using network visualizations. Our approach has been to use Javascript networks from the package vis.js. We tuned some features to use the potential of the vis.Network objects to include in the visualization the data we want to see (e.g. to see in the visualization which is each route, which is their load, the time measurements, in which part of the system did each event occur, etc.).

Having the complete traces available in the format presented in Section 6.3 (see Table 6.1), we added some more fields to each row, and we passed the data to a Javascript network in the form of a Nodes object and an Edges object. For more details, see in Appendix E.2 how is each row transformed into Nodes and Edges objects. Further details about the properties of those objects are provided in Appendix E.3.
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

Include Visualization Features

For the data to be visualized successfully as in Figure 7.1, we first need to assign to the records some fields that will be used as edge properties.

- For the edges between each pair of nodes not to overlap each other, We must assign a roundness value to each edge depending on the number of edges per pair of nodes. Each edge between each pair of nodes must have a different value to be placed with different angles of curvature.

- We aim to color each edge based on its time measurement. For that, we assign a color to each edge as a value in a gradient from the minimum time value to the maximum time value included in the data. We further detail this process in Appendix E.1: Listing E.1 shows how the color is assigned to the edges. Given a palette as a gradient, the minimum and maximum values of the gradient are assigned to the extreme time values in the data. Then, each possible time measurement can be picked as a color in-between.

- To represent the load of bags that travel through each route, we assign a thickness value to each edge, where the thickness is equivalent to the number of bags per edge.

In Figure 7.1 we can see how the color of the node depends on the area it belongs to. The edge color represents the duration of the bags in each route, and the edge thickness is related to the number of bags per route. We also successfully avoided edge overlap, by assigning different roundness values to the edges, based on how many edges were present between each pair of nodes.

The resulting networks are displayed in HTML files. The files are static if they only show one network, but they can also be dynamic files, including different visualization for different time metrics (see Figure 7.2).

7.1.2 The Challenge of Transparency of Information in Visualizations

In Figure 7.9, we are displaying the traces of 10 bags along the airport. We can observe that the visualizations can easily get cluttered when we display many bag traces. It might be in the interest of the data analyst to observe a smaller section of the airport or a smaller subset of traces. Also, instead of observing bag traces independently, they can be grouped by the routes they followed. For this purpose, in Section 7.2, we present the possible data classifications to aggregate the bag traces over High-Level processes or Low-Level processes.

In order for the user to obtain the right data interpretation, they need to parameterize their classification (which bags or routes do they want to include, which process steps to include, which
metrics, etc. In Section 7.4 we present the options provided in the user interface to parameterize the data classification and visualizations.

7.2 Aggregation of Bag Traces into HL and LL Variants

In this section we describe how, using the information available from Chapter 6, we generate the process variants as a feature of the bag traces (i.e. which sequence of events did each bag take); we aggregate the bag traces into variants and compute performance metrics to be used in the visualizations in Section 7.3.

As an outcome of this step, we have available the data about which process variants took place in the system, which bags performed each and which are the time measurements for each variant. This is the information we provide to the domain expert to assess which are default routes. As mentioned in Section 3.6.6, automatic outlier detection of process variants is out of scope. Our current methodology is a semi-automated process, where the user detects with domain knowledge which are the intended routes among the list we provide, and they can then proceed to observe the performance of the process variants in our visualizations (which are described in Section 7.3).

The data received as input in this step is the output of Chapter 6 (see Table 6.1, complete_traces). The table complete_traces has many fields, some of them are irrelevant for the classification, and will only be used later in the visualization. Table 7.1 shows a simplified version of complete_traces, only including the subset of columns relevant for the aggregation of routes into variants.

In Section 7.1 we identified the need to observe the low-level physical movements and then high-level processes into different visualizations. We approach this problem by splitting the work
### Table 7.1: Attributes of complete_traces.csv used in the Data Classification

<table>
<thead>
<tr>
<th>ID1</th>
<th>source_node</th>
<th>target_node</th>
<th>timestamp</th>
<th>process_step</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>00001</td>
<td>00002</td>
<td>07:04:05</td>
<td>step_1</td>
</tr>
<tr>
<td>001</td>
<td>00002</td>
<td>00003</td>
<td>07:04:29</td>
<td>step_1</td>
</tr>
<tr>
<td>001</td>
<td>00003</td>
<td>00004</td>
<td>07:04:48</td>
<td>step_1</td>
</tr>
<tr>
<td>001</td>
<td>00004</td>
<td>00005</td>
<td>07:05:27</td>
<td>step_1</td>
</tr>
<tr>
<td>001</td>
<td>00005</td>
<td>00006</td>
<td>07:05:52</td>
<td>step_2</td>
</tr>
<tr>
<td>001</td>
<td>00006</td>
<td>00007</td>
<td>07:06:34</td>
<td>step_2</td>
</tr>
</tbody>
</table>

The table 7.1: Attributes of complete_traces.csv used in the Data Classification

into two parallel pipelines; in Section 7.2.1 we show the aggregations of bag traces over HL process variants, and in Section 7.2.2 we explain the equivalent aggregations over LL routes.

### 7.2.1 Aggregate Bags over HL Variants

In this section, we describe how we obtain the tables that answer some analytics objectives from Section 3.6.5. We describe the data pipeline that is then the input of the visualizations in Section 7.3.1.

![Figure 7.3: Data Pipeline of the High Level process classification described in Algorithm 6](image)

Figure 7.3 describes the process of aggregating the bag traces into HL variants. The full algorithm 6 can be found in Appendix F.1. The output of this data pipeline are 3 tables. They fulfill analytics objectives presented in Section 3.6.5:

- The table HL_sequences (see Table 7.2) fulfills the objective to discover the routes taken in the HL process. Discovering the HL routes allows the user to classify them between default routes and deviating routes, using their domain knowledge to detect exceptional behavior.

- The table bags_per_HL_sequences (see Table 7.3) fulfills the objective to, having the user identified default routes and deviating routes, provide the user with data of the load of each route.

- The table traces_HL_sequences (see Table 7.4) provides the input for Section 7.3.1, to provide performance-based visualizations to compare HL routes.
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

Data Pipeline

Having as input bag traces, we create sequences of process steps, i.e., sequences of HL events. Then, we create HL process variants: we assign to each bag the sequences of process steps it performed. Based on the number of bags that pursued each HL process variant, we assign an HL class to each HL sequence, being HL class = 0 the class with the higher load. As the class goes ascending, the number of bags per class goes descending. This results in the table HL sequences (see Table 7.2). Joining this result with the previous intermediate result, we assign to each bag the HL sequence it pursued, but also the class of the route and the load of the route. This results in table bags per HL sequences (see Table 7.3).

From the reduced bag traces (HL traces), we calculate time metrics per process step. Joining these HL traces with the HL process variants, we obtain the table traces HL sequences (see Table 7.4).

Output

Below, we provide snippets of the three tables that result from the output of the data pipeline described in Figure 7.3. It is relevant to take a look at which information does each table provide, since they can already be used by the final user to answer some of the analytics objectives, and they are the input of the next visualizations in Section 7.3.1.

Table 7.2 answers the objective of discovering the HL routes taken, and providing them to the user to identify default routes and deviating routes. Just by observing the sequence of process steps that a bag went through, a domain-expert can understand how much exceptional behavior the bag went through (e.g., recirculating in the Sorter 1 due to high load on the system, or being scanned repeatedly due to unsatisfying results).

<table>
<thead>
<tr>
<th>HL_class</th>
<th>HL_sequence</th>
<th>number_of_bags</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>step_1 - step_2 - step_3 - step_4</td>
<td>974</td>
</tr>
<tr>
<td>1</td>
<td>step_1 - step_3 - step_2 - step_3 - step_4</td>
<td>653</td>
</tr>
<tr>
<td>2</td>
<td>step_1 - step_2 - step_5</td>
<td>281</td>
</tr>
<tr>
<td>3</td>
<td>step_1 - step_2 - step_4 - step_2 - step_5</td>
<td>139</td>
</tr>
</tbody>
</table>

Table 7.2: Mockup of the table HL sequences

Table 7.3 provides the complete list of which bags followed each HL variant. In order to provide performance metrics of each bag along the HL process, and to aggregate them by process variant, we require the content of Table 7.4. The structure of the table traces HL sequences is very convenient to represent time metrics over each bag movement, being each bag movement an edge from a source node to a target node. We keep information about the bag identifier (ID1), the HL variant that the bag completed, and the load of the HL variant. We can either display time

<table>
<thead>
<tr>
<th>ID1</th>
<th>HL_class</th>
<th>HL_sequence</th>
<th>number_of_bags</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>0</td>
<td>step_1 - step_2 - step_3 - step_4</td>
<td>974</td>
</tr>
<tr>
<td>002</td>
<td>0</td>
<td>step_1 - step_2 - step_3 - step_4</td>
<td>974</td>
</tr>
<tr>
<td>003</td>
<td>1</td>
<td>step_1 - step_3 - step_2 - step_3 - step_4</td>
<td>653</td>
</tr>
<tr>
<td>004</td>
<td>1</td>
<td>step_1 - step_3 - step_2 - step_3 - step_4</td>
<td>653</td>
</tr>
<tr>
<td>005</td>
<td>0</td>
<td>step_1 - step_2 - step_3 - step_4</td>
<td>974</td>
</tr>
<tr>
<td>006</td>
<td>0</td>
<td>step_1 - step_2 - step_3 - step_4</td>
<td>974</td>
</tr>
</tbody>
</table>

Table 7.3: Mockup of the table bags per HL sequences

Table 7.3 provides the complete list of which bags followed each HL variant. In order to provide performance metrics of each bag along the HL process, and to aggregate them by process variant, we require the content of Table 7.4. The structure of the table traces HL sequences is very convenient to represent time metrics over each bag movement, being each bag movement an edge from a source node to a target node. We keep information about the bag identifier (ID1), the HL variant that the bag completed, and the load of the HL variant. We can either display time
performance by single bags or by HL variants.

<table>
<thead>
<tr>
<th>ID1</th>
<th>timestamp</th>
<th>process_step</th>
<th>duration</th>
<th>cumul...</th>
<th>HL_class</th>
<th>HL_sequence</th>
<th>number_of_bags</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>07:04:05</td>
<td>step1</td>
<td>00:00:00</td>
<td>00:00:00</td>
<td>0</td>
<td>step1 - step2 - step3 - step4</td>
<td>974</td>
</tr>
<tr>
<td>001</td>
<td>07:04:29</td>
<td>step1</td>
<td>00:00:24</td>
<td>00:00:24</td>
<td>0</td>
<td>step1 - step2 - step3 - step4</td>
<td>974</td>
</tr>
<tr>
<td>001</td>
<td>07:04:48</td>
<td>step1</td>
<td>00:00:43</td>
<td>00:00:43</td>
<td>0</td>
<td>step1 - step2 - step3 - step4</td>
<td>974</td>
</tr>
<tr>
<td>001</td>
<td>07:05:27</td>
<td>step1</td>
<td>00:01:22</td>
<td>00:01:22</td>
<td>0</td>
<td>step1 - step2 - step3 - step4</td>
<td>974</td>
</tr>
<tr>
<td>001</td>
<td>07:06:34</td>
<td>step2</td>
<td>00:02:29</td>
<td>00:02:29</td>
<td>0</td>
<td>step1 - step2 - step3 - step4</td>
<td>974</td>
</tr>
</tbody>
</table>

Table 7.4: Mockup of the table traces HL_sequences

7.2.2 Aggregate Bags over LL Variants

In this section, we describe how we obtain the tables that answer some analytics objectives from Section 3.6.5. We describe the data pipeline that is then the input of the visualizations in Section 7.3.2.

Figure 7.4: Data Pipeline of the Low-Level process classification

Figure 7.4 shows the Data Pipeline of the HL aggregation. For more details, see Algorithm 7 included in Appendix F.2. The output of this data pipeline are 3 tables. They fulfill analytics objectives presented in Section 3.6.5:

- The table routes (see Table 7.5) fulfills the objective to discover the routes taken in the LL process. This provides the user with insights on how are the bags being distributed in the system and how is their performance in time.
- The table bags per route (see Table 7.6) fulfills the objective to provide the user with data of the load of each LL route.
- The table traces routes (see Table 7.7) provides the input for Section 7.3.2, to provide performance-based visualizations to compare LL routes.

Data Pipeline

Having as input bag traces, we filter them by the subset of process steps that we want the LL routes to include. Then, we create LL process variants; we assign to each bag the LL_sequence it performed. Based on the number of bags that pursued each LL process variant, we assign an LL_class to each LL_sequence, being LL_class = 0 the class with the higher load. As the LL_class goes ascending, the number of bags per class goes descending. This results in the table routes (see Table 7.5). Joining this result with the previous intermediate result, we assign to each bag the
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

LL_sequence it pursued, but also the class of the route and the load of the route. This results in table bags_per_route (see Table 7.6).

From the reduced bag traces (traces), we calculate time metrics per process step. Joining these traces with the LL process variants, we obtain the table traces_routes (see Table 7.7).

Output

Below, we show the output tables of the Data Pipeline in Figure 7.4. They provide answers to some of the analytics objectives, and they are the input of the next visualizations in Section 7.3.2.

Table 7.5 provides the user with the load of each LL route. Discovering the LL routes answers the objective of providing the user with insights of how is the system being used; how is the load balancing system distributing the bags and how are the paths being used.

<table>
<thead>
<tr>
<th>LL_class</th>
<th>LL_sequence</th>
<th>number_of_bags</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>1</td>
<td>320102 - 320103 - 320104 - 320105</td>
<td>184</td>
</tr>
<tr>
<td>2</td>
<td>330101 - 330102 - 330103 - 330104 - 330105</td>
<td>163</td>
</tr>
<tr>
<td>3</td>
<td>310101 - 310102 - 310103 - 310104 - 310105</td>
<td>95</td>
</tr>
</tbody>
</table>

Table 7.5: Mockup of the table routes

Table 7.6 answers the objective that having the user identified default HL routes and deviating HL routes, we provide the user with access to the data of the load of each route.

<table>
<thead>
<tr>
<th>ID1</th>
<th>LL_class</th>
<th>LL_sequence</th>
<th>number_of_bags</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>002</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>003</td>
<td>1</td>
<td>320102 - 320103 - 320104 - 320105</td>
<td>184</td>
</tr>
<tr>
<td>004</td>
<td>1</td>
<td>320102 - 320103 - 320104 - 320105</td>
<td>184</td>
</tr>
<tr>
<td>005</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>006</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
</tbody>
</table>

Table 7.6: Mockup of the table bags_per_route

Table 7.7 answers the objective to provide performance metrics of each bag along the LL processes and to aggregate them by process variant. The structure of the table traces_routes is very convenient to represent time metrics over each bag movement, being each bag movement an edge from a source node to a target node. We keep information about the bag identifier (ID1), the LL variant that the bag completed, and the load of the LL variant. We can either display time performance by single bags or by LL variants.

<table>
<thead>
<tr>
<th>ID1</th>
<th>timestamp</th>
<th>process_step</th>
<th>duration</th>
<th>cumul...</th>
<th>LL_class</th>
<th>LL_sequence</th>
<th>number_of_bags</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>07:04:05</td>
<td>step_1</td>
<td>00:00:00</td>
<td>00:00:00</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>001</td>
<td>07:04:29</td>
<td>step_1</td>
<td>00:00:24</td>
<td>00:00:24</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>001</td>
<td>07:04:48</td>
<td>step_1</td>
<td>00:00:19</td>
<td>00:00:43</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>001</td>
<td>07:05:27</td>
<td>step_1</td>
<td>00:00:39</td>
<td>00:01:22</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>001</td>
<td>07:05:52</td>
<td>step_2</td>
<td>00:00:25</td>
<td>00:01:47</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
<tr>
<td>001</td>
<td>07:06:34</td>
<td>step_2</td>
<td>00:00:42</td>
<td>00:02:29</td>
<td>0</td>
<td>310103 - 310104 - 310105</td>
<td>271</td>
</tr>
</tbody>
</table>

Table 7.7: Mockup of the table traces_routes
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

7.3 Visualization of Process Variants

This section provides the visual analytics tools for domain-experts to perform performance-aware CC. We implemented the methodology with the data of a BHS use case, and the resulting visualizations displayed in this section are some examples of the output of this information visualization technique. These results of our implementation are then evaluated by our stakeholder in the next chapter, Chapter 8. Chapter 8 illustrates how can those tools be applied to answer several concrete analysis questions on a real-life BHS in a case study. The lessons learned from this case result in the methodology for PA-CC presented in Chapter 9.

Since we split our data pipeline in two parallel processes, we continue explaining the visualizations for HL processes (see Section 7.3.1) and LL processes (see Section 7.3.2) separately.

Rationale to split into HL visualizations and LL visualizations

In the visualization from Figure 7.7, we are comparing the time performance of a bag that recirculated in the Sorter 1 with two other bags that did not recirculate. This is possible because we maintained HL data (process step) and LL data (location_id). The insight the HL visualization provides is whether or not the recirculated package had a similar performance to the other packages. The insights obtained in the LL visualizations from Section 7.3.2 is which LL routes within a process step performed better, and how was the load distributed.

7.3.1 Visualizations of HL Variants

This section answers the objective to provide performance-based visualizations to compare bags along their HL process. Figure 7.5 shows the Data Pipeline to obtain the visualizations of High-Level processes described in this section. The input of this final data pipeline is described in Section 7.2.1.

![Figure 7.5: Data Pipeline for visualization of High Level processes](image)

Having bag traces structured like the table traces_HL_sequences (see Table 7.4) the user can first create a Line Chart to have a clear visualization of which were fast and slow bags along the HL process (see Section 7.3.1). In order to compare the time performance of bags along their HL process, we need to compare bags that followed the same HL process. Comparing bags that require different steps does not provide performance insights. Therefore, the user must select a HL
process to filter the data (the user has the table HL_sequences available to choose an HL_class). Also, in order to reduce the size of the analysis and view the data clearer, we suggest selecting a random sample of the bag traces available.

After that, we provide the user with graph visualizations of the bag traces on the Layout Graph (see Section 7.3.1). The user can select which time metric to observe, and can select a subset of the bags based on the previous observations.

Also, if a pair of bags had locations in common, we provide a visualization where the user can observe how does the time performance of one bag deviate from the other (see Section 7.3.1).

**Detect Variants according to Time**

We propose as a first visualization for the user, to plot the bag traces in a Line Chart, where x-axis are process steps, and the y-axis is a time metric. In this Line Chart, we are comparing the time performance of bags over a sequence of process steps, therefore, the user must choose one sequence of process steps to view (i.e. one HL_class).

**Objective**  This visualization answers the analysis objective to identify performance categories among bags of an HL variant. It is an intermediate visualization that allows comparing many bags with one another. The user can discern which bags to investigate further (by choosing a very clear outlier, or by selecting a subset of bags from different performance categories.

In the Algorithm 6, we show how we calculate for each bag the duration of each process_step, and the cumulative duration. Those are possible metrics to display in a line chart. We can also display the total time per bag or the remaining time per bag. See Appendix F.1 for further details.

**Interpretation** Figure 7.6 is a Line Chart of In System Time per bags over the High-Level Process (being the y-axis the time and the x-axis values the sorted process steps: Check-In, Sorter 1, Scanner, Sorter 2, Storage, Sorter 2, Sorter 3). The Line Chart answers the analytics objective of comparing the performance of bags along an HL process, to visually detect bags that belong to different categories. In the example of Figure 7.6 there are 10 bags displayed. We can see that bags with ID1 = 105243753 (light blue line) and ID1 = 105242996 (grey line) got delayed after being scanned, but apart from that, all the bags follow a similar pattern.

**Graph Visualization of Time Performance per Trace**

**Objective**  These visualizations answer the analysis objective of displaying the routes taken in visualizations that are interpretable by the user. The visualizations shown in this section answer the objective of displaying the performance of bags along the system. The visualization of HL variants displays bags that belong to one HL variant since in order to compare the performance over time with one another, the bags must follow the same steps (in this case, HL steps, i.e. process steps).

**Interpretation** In the visualization from Figure 7.7, there are 3 bag traces displayed. The traces are plotted over the BHS layout, where each node is a node from the Layout Graph, i.e. a location in the airport. A bag trace is a sequence of arrows, i.e. edges, where each edge represents a movement.

The traces begin in the START node (the pink node). Two bags entered the system in the same Check-In line (purple nodes) and the other at the end of another Check-In line (orange node). The color of the arrows (i.e. edges) represents the cumulative time in the system per bag. In the
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

Figure 7.6: Line Chart of In System Time per bags over a High Level Process

Figure 7.7: Graph visualization of total time per bag (3 bags)

color legend, we can see which are the limit values of the color range. The HL process displayed
in this visualization is:

1. Check-In (bottom left corner)
2. Sorter 1 (there are 2 sorters in the middle of the layout, orange nodes, and green nodes)
3. Scanner (pink, orange, and purple nodes in between Sorter 1)
4. Sorter 1
5. Sorter 2 (red nodes and light green nodes, on top of Sorter 1)
6. Storage (the vertical line of dark blue nodes, the storage racks have multiple colors)
7. Sorter 2 (orange nodes on the top)
8. Sorter 3 (violet nodes in the bottom right corner)
9. Flight

**Fine-grained information** In Figure 7.8 we observe how the user has access to information about every node and edge by hovering on them. Since the edges are colored based on the cumulative time on the system, the user can easily detect where do the main delays take place. In this example, the visualization contains two bag traces that start in Check-In (bottom left) and finish in Sorter 3 (bottom right), and the user detected that one of the bags got delayed when entering Sorter 1 after being scanned (see the nodes that are being pointed at). When hovering over the edges, the user could see that the bag with ID1 = 30380049 left the Scanner with 8 minutes spent in the system, but when it reentered the Sorter 1, it had been in the system for 16 minutes. The user can then compare this 8 minutes delay with other bags and interpret it.

![Figure 7.8: Graph visualization of total time per bag (2 bags). Further information in the hovering notes](image-url)
Cluttered visualizations. Requisite of providing filters and aggregations  Figure 7.9 displays 10 bag traces along the system. With just 10 bags, the visualization is already cluttered: it is not obvious which bags recirculated, and it is not easy to follow a bag trace step by step. For this reason, it is important to provide the user with access to customize the visualizations, and apply the filters they require (e.g. display only the bags with ID1 = [A,B,C]). In Section 7.4 we explain the structure of the user interface created for the user to adapt the analysis to their needs.

![Figure 7.9: Graph visualization of total time per bag (10 bags)](image)

Figure 7.9 displays a closer look of Figure 7.9, were we zoom in one of the Sorter 3. We can see that 9 out of 10 bags entered the green Sorter 3. After zooming in, it is possible to discern the ID1 of each bag, and quickly realize which were the slow bags (the bags in the darker edges). After taking a closer look on the bigger picture, the user might then decide on which bags to focus next.

Detecting incidents Figure 7.11 is another closer look of Figure 7.9, this time focused on the exit of Sorter 1. In the bottom left side of the picture, we can see some bags leaving the Sorter 1 and moving to the Sorter 2 (notice that the Sorter 1 is the loop of pink and purple nodes in the bottom, the Sorter 2 is the upper part of the picture). In this picture, we focus on detecting the clearest delay in the visualization: one bag entering the Sorter 2 via the blue nodes got a much darker edge than before. This movement does not contain a scanner or any process that might cause the delay; it is a transport conveyor, and the delay might be caused by a traffic jam in the system.
Graph Visualization of Deviations

The goal of the visualization in Figure 7.12 is to compare the performance of two bag traces, where one bag trace is considered the baseline, and the other the alternative route. We observe the performance deviation of the alternative route over the baseline: when the alternative bag is faster than the baseline, it is displayed in blue, and when it is faster than the baseline, it is displayed in red. The baseline bag trace is displayed in white for clarity. This visualization is only possible for bag traces that have locations in common: we compare the time performance of the bags over locations they both passed through.

Objective This answers to the objective that, having the user determined by domain knowledge that a bag trace represents the default behavior and another deviates from it, provide the user with a visualization of the deviation between their performances.

Interpretation In Figure 7.12 we can see the baseline trace in white: it starts in the Check-In, passes through the Sorter 1 without recirculating, moves to Sorter 2 to be located in Storage, and then it is directed to a flight by the Sorter 2 and Sorter 3. The alternative trace follows a similar HL process, though it stays longer in the Sorter 1 due to recirculation, and it is scanned 3
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

times. The visualization allows us to see that the alternative trace was faster than the baseline in the Check-In area, but got delayed when it entered the Sorter 1, and it did not manage to catch up with the baseline trace. Hovering over an edge, we can see that when it exited Sorter 1, the alternative trace was deviating from the baseline by 6min 57s.

Figure 7.12: Comparison of performance between 2 bags - observe deviation

7.3.2 Visualizations of LL Variants

This section answers the objective to provide performance-based visualizations to compare the performance of bags along the LL routes. Figure 7.13 shows the Data Pipeline to obtain the Low-Level visualizations described in this section. The input of this final data pipeline is described in Section 7.2.2.

Having the bag traces structured like the table traces\_routes (see Table 7.7) the user can first create a Line Chart to have a clear visualization of which were fast and slow bags along the routes (see Section 7.3.2). Since analyzing the transport process of the bags along the entire airport would not provide clear insights about how some routing decision affects the bags’ performance, it is recommended to the user to filter the data by some HL events (e.g. select data from Checkin, or Sorter 1 + Scanner). Also, in order to reduce the size of the analysis and view the data clearer, we suggest selecting a random sample of the bag traces available.

After that, we provide the user with graph visualizations of the LL routes taken on the Layout Graph (see Section 7.3.2). The user can select which time metric to observe, and can select a subset of the routes based on the previous observations.
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

CHAPTER 7. VISUAL ANALYTICS OF ROUTES

Detect Variants according to Time

This visualization answers the objective to compare the time performance of bags through each bag movement, to visually detect which bags belong to different categories of performances.

The Line Chart in Figure 7.14 displays the cumulative In System Time of some bags over each bag movement in the Check-In step. Each line represents a bag, the y-axis represents the cumulative time in the system and the x-axis shows the number of movements of each bag (one movement is moving from one node to the next). The LL routes are represented by bag movements in the x-axis because to compare different LL routes, we could not compare the bags over concrete location_id values. In the legend, we can observe the hour of the day, the bag identifier, or the LL_class of the routes. This labels can help the user decide which routes to view in Section 7.3.2.

In the example from Figure 7.14, the color of the lines represents the hour of the day. We can observe that the hour of the day is related to the total time per bag in the Check-In step. The rate of arrival must be influenced by the hour of the day, and at peak hours there can be traffic jams in the main conveyor line, increasing the time of the bags.

Graph Visualization of Time Performance per Route.

These graph visualizations provide a solution to the analysis objective of observing the performance of physical routes, based on the load of bags they transported and the time performance of the bags. We are displaying the bags and comparing their performance in LL routes over three time metrics:

1. Cumulative In System Time
2. Remaining Time in the System
3. Total Time in the System

In Figure 7.15 we can follow each route along the system. Hovering on each edge, we see the information about the route and the time metric. This visualization is using the cumulative time in the system as a performance metric. Hence, as we follow the movements of each route, the cumulative time in the system increases. Following the route with LL_class = 0 (which is representing a load of 4 bags), we verify that at the start of the trace there is no accumulated
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

Figure 7.14: Line Chart of In System Time per bags over Low Level Processes in the Check In
time. After the bags have been registered in the Check-In desk and have been moved to the next
collector conveyor, the In System Time of that route is 1 minute 24 seconds. This metric is the
average of the bags in the route at this point. Moving further, the red edges represents a longer
conveyor that moves the bags to Sorter 1. We can see that the bags spend more time in this
movement due to the high variation in the color of the edge. The In System Time of route 0 at
that point was 4 minutes 38 seconds.

Likewise, we can observe the LL_routes in Figure 7.16 and understand the Remaining Time in
the System of the bags in each route, both by the color code and the metrics in the hovering notes.
At the start of the process, the routes show the maximum remaining time in the system, and as
they leave the system, we can see that one route (light green edge) has much less remaining time
left than the others.

Finally, having a visualization without so many color changes can provide an easier interpreta-
tion for the user about which LL routes had a better performance. Figure 7.17 displays the total
time in the system per route, and we verify it by seeing that route 0 shows the same measurement
in the different movements. From this example, we can observe that route 14 (the lighter blue)
was the fastest.
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

Figure 7.15: Graph visualization of total time per bag (10 bags) - Incidents

Figure 7.16: Graph visualization of total time per bag (10 bags) - Incidents
CHAPTER 7. VISUAL ANALYTICS OF ROUTES

Figure 7.17: Graph visualization of total time per bag (10 bags) - Incidents
7.4 User Interface

In this section, we present the user interface provided. (fulfillment of RQ6 in Section 1.2). The goal of providing a user interface to perform our method is to allow the user to decide over which data to analyze and which data to display in each visualization. The user can filter over which bags or routes to include, and can also decide over which time metrics to compute.

Also, we provide the option to execute the different pre-processing and visualization steps independently. Some steps have a longer execution but only need to be performed once (the 3 Data Preparation steps), while other steps are faster and the user might want to perform repeatedly (such as obtaining different visualizations). Figure 7.18 displays the pipeline of our application. For one set of data, the data preparation needs to be done only once, in the following order:

1. Generate Routing Graph: described in Section 6.1
2. Event Logs Pre-processing: described in Chapter 5
3. Pattern Matching: described in Section 6.2

Figure 7.18: Application Pipeline

In the Pattern Matching step, the user can also choose a batch size to store the completed traces into disk to release memory. After the Data Preparation is finished, the user obtains the intermediate files that are used in the Data Classification. Then the user can select which subsets of traces or process variants to include in which visualization, and select the parameters of the visualizations. In Appendix G we include screenshots of the running application.
Chapter 8

Evaluation

In this chapter we present a validation of the techniques introduced in Chapter 4 Solution proposal (whose results are displayed in 7 Visual Analytics of Routes). We analyzed the results obtained from visualizing data from a BHS use case. The analysis performed consisted of conducting a case study with domain experts, presenting to them the results obtained and extracting the list of benefits obtained and future improvements as lessons learned.

We start this chapter by reviewing in Section 8.1 the research questions proposed in Chapter 1. We continue in Section 8.2 presenting the case study validation. There we list the use cases that were conceived, and we consider to which extent do our results solve each use case.

In other words, we validate our results first via an internal test and next via an external evaluation. The research questions reviewed in Section 8.1 are analysis goals proposed by us that summarize the contributions of our method, whereas the use cases listed in Section 8.2 are validation goals from our target stakeholder, the Process Engineer of the BHS.

8.1 Validation of Research Questions

We start the evaluation by assessing if the research questions (RQ) introduced in Section 1.2 have been met and how.

RQ1. Given event traces with missing events and given ground truth information about the con-
sequentiality among events, i.e. given information about possible sequences of events, we aim
to complete the event traces by finding the missing sequences of events.

RQ1 has been completed by querying the sequence of missing events between each pair of
recorded events in a Graph Database, where we loaded all possible activities and the physical
connections among them. From one recorded event to the next, we queried Dijkstra’s shortest
path. See more details about the decisions taken and the implementation in Chapter 6.

RQ2. Given complete traces, we aim to group them by two kinds of process variants: sequences of
high-level events and sequences of low-level events. We aim to have access to information
about the load of each variant and the time performance of that route.

We detail the steps to complete RQ2 in Section 7.2 Aggregation of Bag Traces into HL and
LL Variants.

RQ3. Given bag traces and process variants, we aim to provide visualizations of their performance
that can be interpretable for the user.

In Section 7.3 we present multiple visualizations that complete the objective of RQ3. We
provide visualizations both for HL processes (see Section 7.3.1) and LL processes (see Section
7.3.2).
CHAPTER 8. EVALUATION

RQ4. Given bag traces, we want to visualize their routes over the Baggage Handling System.

In order to fulfill RQ4, we manage to plot the traces and the routes taken over the layout of the Baggage Handling System, by providing the nodes in the graph visualization with the same coordinates as the canvas that described the BHS layout. This visualization is very clear for the domain experts since they are used to work with this layout. Our system has over a thousand nodes, and distributing them in a well-known layout facilitates the user to instantly locate the process taking place in each node. This layout is being used in all the graph visualizations presented in Section 7.3:

- 7.3.1. Graph Visualization of Time Performance per Trace (see Figure 7.7)
- 7.3.1. Graph Visualization of Deviations (see Figure 7.12)
- 7.3.2. Graph Visualization of Time Performance per Route (see Figures 7.15, 7.16 and 7.17)

RQ5. Given bag traces, we want to visualize the time performance of the traces and the process variants along the entire process.

Two kinds of visualizations fulfill RQ5 for both HL processes and LL processes. Figure 7.6 provides a line chart of the time performance of traces over the sequence of HL events, and Figure 7.14 provides the equivalent line chart for LL processes. Also, the graph visualizations (Section 7.3.2 for HL processes and Section 7.3.2 for LL processes) display how the time measurements gradually variate over each trace or route from the beginning to the end of the process. The time measurements are available for each edge, but they are also displayed as a color within a color range, for the user to get an idea of its value in relation to the rest of edges.

RQ6. Given event data and structured data about the consequentiality of events, we aim to provide an interactive tool to obtain the performance of process variants and visualize it.

We fulfilled RQ6 by providing a UI for the user to execute the implementation of our methodology. We introduce this UI in Section 7.4, and some screenshots of the running application are included in Appendix G. We can conclude that the research questions, i.e. our internal goals have been met, and next, we validate our results with external input from our stakeholder.

8.2 Validation with the Stakeholder

This section describes the validation of our results with the stakeholder, a Process Engineer that analyzes the BHS we used to implement our methodology. The protocol we stuck to was the following. First, we had to familiarize the user with our methodology. We described the goal of our methodology to the stakeholder. We described which data we had available, and which results obtained. We listed all the questions that our results could provide answers to. Then, we met with the stakeholder to learn which use cases were more important to give an answer to in their work. In the meeting, we discussed with them our visualizations to see to which extent did they found our work useful if they considered it contributed to their current procedure and which limitations did they foresee. In the following lines, we present the use cases discussed, and the observations about how do our current results contribute to its analysis.

8.2.1 Stakeholder’s Use Cases

Case 1: Find bags affected by an incident

The system might detect an incident, e.g. such as a conveyor stops because there is a bag stuck in the system. The analyst then receives the information that at the time $T$, in location $L$, an incident was detected. To understand further what caused that event, the analyst wants to see which bags passed through location $L$ at that time. With our method, we propose to filter the
bags by timestamp $T$ and location $L$. We provide the complete bag traces to the analyst, where they can see all the locations each bag passed through their timestamps, thus this first goal is fulfilled.

Then, the analyst wants to observe which impact did the incident have on the system. For that, we propose to filter all the bag traces by timestamp $T$ and then use the graph visualization of LL routes within the HL event where location $L$ takes place (see Section 7.3.2 Graph Visualization of Time Performance per Route).

**Case 2: Find bags that do not take the default route**

The analyst wants to detect which bags do not follow a default route. The analyst defines a default route as an expected route along the airport. The process should not include unnecessary steps. The route taken depends on the entry and exit point of the bag in the system, the sequence of HL events the bag requires, and the load on the system. Here we show some examples of default routes and a non-default route (includes unnecessary steps):

- Default route 1: Check In - Sorter 1 - Scan - Sorter 3 - Flight
- Default route 2: Check In - Sorter 1 - Scan - Change Of Terminal
- Default route 3: Change Of Terminal - Sorter 1 - Scan - Sorter 3 - Flight
- Non-default route: Check In - Sorter 1 - Scan - Sorter 3 - Sorter 1 - Scan - Sorter 3 - Flight

Our method provides the list of HL routes to the analyst (as the examples shown above) and the user can reach conclusions about which HL routes are default and which are not. Then the analyst can decide which routes to examine further, and drill down the analysis to the bags that took each of the routes (this data is provided in the pipeline of HL variants, see Section 7.2.1). Then, having a target list of bags, the user can observe LL routes of those bags and their performance (both from the tables provided in 7.2.2 and the visualizations over the Layout Graph 7.3.2).

**Case 3: Observe the load of each route and the duration in the system of their bags**

The user does not have access to information about the complete races. About each bag, the analyst can know the total time they spend in the system (the time between the bag was registered in the system until it was boarded into the flight. However, that does not provide the analyst with the data about in which parts of the system are the bags moving slower. Our method to obtain this information is explained in Section 7.2.2 Aggregate Bags over LL Variants and the visualizations in 7.3.2 Graph Visualization of Time Performance per Route show the duration of each event in the routes and which was the load in each route.

**Case 4: Visualize the bag movements over the airport floorplan**

The analyst requires a user-friendly visualization that can be interpreted by anyone with a high-level understanding of the process. This use case is fulfilled with the graph visualizations that plot the routes over the Layout Graph (see Section 7.3.1, 7.3.1 and 7.3.2).

**Case 5: Detect the root causes of bags not meeting their deadline**

At the beginning of this thesis, we considered that enriching this thesis with domain knowledge was out of scope. However, our methodology can still help the analyst obtain this goal. If the user has the information about which bags' scheduling was affected and when, it is possible to filter our visualizations by those bags and those moments in time. That result provides the locations where the bags became time-critical. The next suggested visualization is to look at the state of
the system at those locations at those points in time.

We conclude that our methodology provides an answer to all the use cases presented by the stakeholder. In this project, enriching the method with domain knowledge was out of scope and we opted for providing enough flexibility to the user to filter the data and visualize specific use cases, assuming they have access to the domain knowledge. A future line of work will be to develop a method to perform root-cause analysis.

For our application to have the highest use for the user, it has to provide enough parameters for the user to perform different levels of filtering. Introducing domain knowledge in the methodology will not necessarily increase the usefulness of the application since that information is already accessible to the user. What the user requires is flexibility to perform analysis and access to all intermediate results. The methodology must be as transparent as possible, with a clear notation for the user to interpret the results correctly.
Chapter 9

The Methodology

After having detailed our approach to perform PA-CC on a BHS use case, in this chapter we summarize how we propose to tackle such a problem in a methodology that can be applied to more use cases. It consists of the Conceptualization of the Methodology described in Section 1.3, enclosing the Data Preparation and Modeling of CRISP-DM.

Our methodology includes two phases, Data Preparation and Modeling. We start by enumerating the context to implement the methodology, broken down into two sub-problems, and then we describe the methodology proposed for each phase (see Section 9.1 for Data Preparation and Section 9.2 for Modeling). The context of our methodology is split into the following sub-problems:

Sub-problem 1. Given traces in an event log, where multiple consecutive events are missing in each trace, and given structured ground truth information about which events can directly follow one another, we propose a method to enrich the traces to obtain the complete sorted list of events.

The methodology proposed to solve this sub-problem is explained in 9.1. Data Preparation.

Sub-problem 2. Given a Material Handling System without a process model, and having a set of complete traces, we propose a method to detect the performance deviation of the process variants.

The methodology proposed to solve this sub-problem is explained in 9.2. Modeling.

9.1 Data Preparation

In this section we explain the methodology we propose for the Data Preparation phase. It provides a solution to sub-problem 1 enunciated in the beginning of this chapter, and its output is used in the next phase Modeling. This methodology includes three stages: 9.1.1 Prepare a Graph Database, 9.1.2 Pre-process Event Data, and 9.1.3 Query the Graph. Below we list the sequence of steps to follow for each of them.

9.1.1 Prepare a Graph Database

Having structured ground truth information about the directly following events in a Material Handling System:

I. Extract ground-truth information

II. Insert in a Graph Database, where each possible activity is a node, and the edges between nodes show which activities can directly follow another.
CHAPTER 9. THE METHODOLOGY

III. The edges must contain a weight. Having a Material Handling System, the cost measure is the time spent between one location (node) and the next.

9.1.2 Pre-process Event Data
Having event data from a Material Handling System with HL and LL case identifiers:
I. Extract event data from multiple sources
II. Remove duplicate events
III. Correct inconsistencies of simultaneous events in a trace using domain knowledge.
IV. Enrich data with extra fields. Include the HL event that corresponds to each LL event.
V. Relate LL identifiers to HL identifiers to have one identifier per physical, individual case.

9.1.3 Query the Graph
Having a Graph Database as described in 9.1.1 Prepare a Graph Database, and having a set of incomplete sorted traces:
I. Establish connection to the Graph Database
II. For each pair of consecutive events in the incomplete trace (being start event the predecessor and end event the successor), do:
   i. Query the Dijkstra’s weighted shortest path in the graph from start event to end event, using the cost metric defined in 9.1.1. Prepare a Graph Database.
   ii. Store in the trace the sequence of activities obtained between start event and end event and assign the end event timestamp as their timestamp.

9.2 Modeling
In this section we explain the methodology we propose for the Modeling phase. It provides a solution to sub-problem 2 enunciated in the beginning of this chapter, and it can take place after having completed the previous phase Data Preparation. This methodology includes three stages: 9.2.1 Aggregation of Bag Traces and 9.2.2 Visualization of Process Variants. Below we list the sequence of steps to follow for each of them.

9.2.1 Aggregation of Bag Traces
Having a set of complete bag traces:
I. Make process variants:
   i. For each trace, create the sequence of HL events that takes place
   ii. Count how many cases took each sequence of HL events
   iii. Assign a class to each sequence of HL events, being the first class the most used sequence and the last class the last used sequence
II. Compute time metrics per variant: for each trace, compute the time metrics for each HL event using their timestamps. Those time metrics are:
   i. Event duration
ii. Cumulative time in the system  
iii. Remaining time in the system  
iv. Total time in the system

This is how we obtain HL process variants and their time metrics. For LL variants, repeat the same steps for LL events instead of HL events. The traces can be shortened (e.g. include only LL events from the HL event ‘CHECK IN’) to observe shorter process variants.

9.2.2 Visualization of Process Variants

Having a set of HL and LL process variants, and the complete bag traces contained in them:

I. Using domain knowledge, decide which HL variants are intended routes and which are deviating routes.

II. Visualize their time performance in one of the following:

   i. Line chart of cumulative time in the system (y-axis) per HL event (x-axis) per trace or HL variants. The same visualization can be used to observe LL variants displaying the number of movements per route in the x-axis.

   ii. Graph visualization of traces where each activity is a node and each movement from one event to the next is an edge between activities. The edges include the case id and a time measurement. The color of the edge represents the time measurement in the range if time measurements present. Advanced visualization. The nodes can be distributed in the canvas as the Material Handling System layout for a clearer understanding of the routes in the system.

   iii. Graph visualization of LL routes where each activity is a node and each movement from one event to the next is an edge between activities. The edges include the class of the LL route variant and its time measurement. The color of the edge represents the time measurement in the range if time measurements present. The width of the edge represents the load of the route. Advanced visualization. The nodes can be distributed in the canvas as the Material Handling System layout for a clearer understanding of the routes in the system.

III. Interact in the visualizations by filtering in/out traces or process variants to obtain specific visualizations. Select time metrics to observe.
Chapter 10

Conclusions

In this chapter, we summarize what was observed from the outcome of this master thesis. In Section 10.1 we conclude the contributions this project provides both to the business and the scientific community. We finalize by presenting some potential lines of future work in Section 10.2.

10.1 Contributions

10.1.1 Contributions to the Business

Having a Baggage Handling System use case where no process model was defined and the traces in the log were incomplete, we proposed a methodology to obtain the performance of the process variants. After testing our implementation, we provided a Proof of Concept application that can perform the following tasks:

- Observe the bag movements along the airport. The initial traces were not complete, and now we can see every location the bag went through.

- Compare performance of bags along the airport.

- Observe the load each route in the airport had, and how well did the bags perform in each route (i.e. Time-Lapse Since Start, Total Time Per Bag, and Remaining Time In The System).

- Observe the bags movements along the airport.

- Observe the routes taken along the airport.

- Re-sample the data, choosing a subset of items, or specific bags or routes.

- Detect unexpected bag movements. If a bag trace follows an impossible routing according to the Routing Configuration, the program will launch an error.

10.1.2 Scientific Contributions

In this section, we assess how did this master thesis contribute to the scientific community by observing which of our contributions were already covered in related work, and which were not possible given the initial conditions. For that, we briefly review again the Deviation Detection techniques commented in Chapter 2.

Xixi Lu reviewed four existing Deviation Detection techniques and proposed a new Deviation Detection\[21\]. Her method managed to detect deviations from event logs only, without requiring a precise normative model of the process. Her method was based on finding patterns and trusting
CHAPTER 10. CONCLUSIONS

the relations among events [21].

In our case, we also wanted to detect deviations in the process, and we did not have a model defined. However, our event logs are not trustworthy because they are incomplete. The ground truth information we have available is the routing information; it contains which sequences of events are physically possible. Thus, after generating the shortest paths between events in the log, we have the complete sequences of events. The complete log is then trustworthy because the freedom of the bag to execute events is limited by the physical layout. From the logs then we create the model of the best routes, to detect the traces that deviate.

In Chapter 2, we list the four techniques and point out why do they not fit our problem. DDT1 does not apply in our use case since there is no normative model available. DDT2 obtains a model of most frequent behavior, which is not equivalent to the best processes. Our goal is a model of the happy routes. DDT3 could classify the traces by the routes that they follow, but without finding the missing events in the traces, the traces would not represent real routes. Lastly, DDT4 would require to use domain knowledge to detect as deviating traces the sub-optimal routes. After comparing those four Deviation Detection techniques with our method, we conclude that our use case has some requirements that are not met in other techniques. Those are:

- It is required to complete the traces using routing information, otherwise the logs can not be trusted as ground truth information.
- The discovered models from other techniques don’t necessarily include the ideal routes. To decide which traces deviate from the process and which are the intended process, we require the domain knowledge to decide which traces follow happy HL processes.

We emphasize that the ground truth in the BHS is the physical layout itself, and which movements are possible along the system (routing information). This information permits completing the traces in the log, and then the traces become the ground truth.

10.2 Future Work

In this master thesis, we managed to obtain visualizations of performance deviation among process variant. We manage to observe distance deviation only visually, looking at the steps each route took and the distances in the Layout Graph. We provided time deviation both visually and as data measurements, based on the timestamps of the events.

A first limitation that was out of scope in this project was to include a root-cause analysis of the deviations. For this project, we considered that domain knowledge was required to understand the cause of each deviation. A future line of work could be to enrich the data with domain knowledge to understand further the causes of the deviations in the process.

Another continuation line would be to enrich the bags data with system information. This would contribute to directly relate the trace with how does the physical route looks like, so its distance performance could be automatically assessed. In this project, we have only managed to assess the performance over time, and the distance performance could only be visually estimated by how the route looked over the Layout Graph. This line of work could be used to automate the benchmarking of process variants without requiring to visualize the data.

The ground truth information about the system we had available was the configuration of the system that was defined when the system was built. This is the information used to decide the real-time routing of the bags along the airport. A future goal is to create a model to improve the ground truth of the system (e.g. which route is faster). That model would be fed with the
output of this project, i.e. the complete event data of the bag traces and the routes. It could be a Machine Learning method continuously fed with event data from the BHS. This could lead to a prediction method of the performance of the routes in the system.

So far, our methodology allows comparing real-life process variants (from the event data) to each other. Another next line of work would be to obtain the benchmark process variants, i.e. obtain the best routes from a model. As previously described, by feeding a Machine Learning model with the complete event data from our processes, the model could learn which are the ideal routes, and this information would be updated based on the performance of each route in the newest bag traces.

In this project, we developed a Performance-aware Conformance Checking Method based on the Use Case of a Baggage Handling System. The method has been tested and evaluated over this use case. The goal has been to provide process analysts with the data and information visualization that answers their questions about the performance of the process variants. We are positive that this methodology will be successfully applied and implemented on other systems and we hope that it will contribute to the development of Process Mining techniques on other complex systems.
Bibliography


[14] Li J. Ma S. Tang N. Wu Y. Wu Y. Fan, W. Graph pattern matching: From intractable to polynomial time. PVLDB, 3:264–275, 09 2010. 10


BIBLIOGRAPHY

[18] Yu J. Yu P. Jiefeng, C. Graph pattern matching: A join/semijoin approach. In IEEE Transactions on Knowledge and Data Engineering, 2011. 10


90 Performance-aware Conformance Checking on Material Handling Systems
Appendix A

Complete Graph of Routing Database

A.1 Cypher queries to build the complete routing graph

In Figure A.1 we display the inner joins performed among the tables form the Routing Database. We split the columns in different files to reduce the size of the files to load. In the file segments.csv we store the result of joining the table Segments and two instances of the table Nodes. The file usages.csv contains the same tables as segments, plus Usages, Destinations, and Preferences. The file rules.csv contains the tables Usages, Destinations, Preferences, Blocking Rules, and two instances of Error Status.

Figure A.1: Inner joins among the routing tables
APPENDIX A. COMPLETE GRAPH OF ROUTING DATABASE

```sql
// CREATE INDEXES
CREATE INDEX ON :LOCATION(location_id)
CREATE INDEX ON :USAGE(destination_id)

// Create nodes LOCATION
LOAD CSV WITH HEADERS FROM 'file:///segments.csv' AS line
FIELDTERMINATOR ";
MERGE (id_start:LOCATION {location_id: line.LOCATIONID_START})
MERGE (id_end:LOCATION {location_id: line.LOCATIONID_END})
MERGE (id_start)-[:SEGMENT_TO]->(id_end);

// Load 2. Add properties in LOCATIONS
LOAD CSV WITH HEADERS FROM 'file:///segments.csv' AS line
FIELDTERMINATOR ";
MATCH (id_start:LOCATION {location_id: line.LOCATIONID_START})-[r:SEGMENT_TO]->(id_end:LOCATION {location_id: line.LOCATIONID_END})
SET r.avg_duration = line.avg_duration
SET r.transport_default = line.transport_default;

// Load 3. Create ROUTE nodes
MATCH (id_start:LOCATION)-[r:SEGMENT_TO]->(id_end:LOCATION)
CREATE (route:ROUTE)-[:FROM]->(id_start)
MERGE (route)-[:TO]->(id_end);

// Load 4. Create USAGE nodes
LOAD CSV WITH HEADERS FROM 'file:///usages.csv' AS line
FIELDTERMINATOR ";
MERGE (u:USAGE {id: line.usage_id, destination_id: line.destination_id, destination: line.destination_description, route_preference: line.preference_description, id_route_preference: line.preference_level})
WITH line, u
MATCH (id_start:LOCATION {location_id: line.LOCATIONID_START})-[:FROM]-(route:ROUTE)-[:TO]-(id_end:LOCATION {location_id: line.LOCATIONID_END})
MERGE (route)-[:WITH_DESTINATION]->(u)
RETURN count(line);

// Load 5. Create RULES nodes
LOAD CSV WITH HEADERS FROM 'file:///rules.csv' AS line
FIELDTERMINATOR ";
MERGE (ss_min:STATUS {description: line.status_description, severity: line.severity_level})
MERGE (ss_max:STATUS {description: line.status_description, severity: line.severity_level})
MERGE (rule:RULE {id: line.rule_id, description: line.rule_description, unblock_delay: line.unblock_delay, block_delay: line.block_delay})
MERGE (rule)-[:MIN_STATUS]->(ss_min)
MERGE (rule)-[:MAX_STATUS]->(ss_max)
WITH line, rule
MATCH (u:USAGE {id: line.usage_id, destination: line.destination_description, route_preference: line.preference_description, id_route_preference: line.preference_level})
MERGE (rule)-[:APPLIES_TO]->(u)
RETURN count(line);
```

Listing A.1: Cypher queries to load the routing tables in the graph
A.2 Querying Real-time Routing

To route one bag from one node to the next, we require the data model from section 3.4.2, or the property graph in section 3.4.3.

The query that would route a bag in real-time answers the following question\(^1\): Being start\(_{\text{node}}\) the node where the bag is located, and being end\(_{\text{node}}\) the next node the bag will be transported to, find next\(_{\text{node}}\), where next\(_{\text{node}}\) is directly connected to start\(_{\text{node}}\) that fits one of those cases:

- It is the only outgoing segment
- It is the outgoing segment that has transport\(_{\text{default}}\) = True
- If there is more than one outgoing segment and both have the property transport\(_{\text{default}}\) = False, then the log provides a destination. Do:
  - Find the nodes USAGES connected to a node ROUTE, connected to the start\(_{\text{node}}\) via a relationship FROM, where the usage has the destination as a property.
  - From those possible usages, select the one with the highest preference (closer to 1).
  - Find the node LOCATION connected via a relationship TO to the route connected to that USAGE.

The above is expressed in the following cypher query.

```
// Case where no destination is assigned. Find the default route. Either
// transport_default = True, or there is only one segment (hence, limit
// the results to 1).
MATCH path = (start_node:LOCATION {location_id:'30501101'})-[rel:SEGMENT_TO]->(end_node:LOCATION)
WITH path, rel
ORDER BY rel.transport_default DESC LIMIT 1
RETURN path
```

Listing A.2: Real-time routing query, where the destination is not provided

![Patterns searched in real-time routing queries](image)

(a) Pattern with destination  (b) Pattern without destination  (c) Pattern for all routes

Figure A.2: Patterns searched in real-time routing queries

\(^1\)Blocking Rules and Error Status have not been included in the query, since the error status was not available in the logs. We follow the assumption that all routes were available.
APPENDIX A. COMPLETE GRAPH OF ROUTING DATABASE

Listing A.3: Real-time routing query, where the destination is provided

In terms of complexity, the routing of a bag from one node to the next is decided by a regular
simple path where regular expressions are used as restrictions to the query. In this case, the queries
are content-based reachability, hence pattern matching.

In Listing A.3, we describe the case when the event log provides a destination, and the bag
must not follow the default route. For better readability, we show in Figure A.2a the pattern
described in the MATCH clause. In Listing A.2, a destination is not provided, and we assume the
default route. In that case, the MATCH clause is displayed in Figure A.2b. To be able to cover
both scenarios, we’d use a disjunctive query composed of both subqueries, obtaining the pattern
from Figure A.2c.

A.3 Complexity Comparison

Here we compare which would be the computational complexity of executing the real-time routing
query in a relational database versus in a Graph Database.

Data Cardinality

- $|\text{Segments}| = 1,819$
- $|\text{Nodes}| = 1,590$
- $|\text{Usages}| = 35,818$
- $|\text{Destinations}| = 475$
- $|\text{Preferences}| = 3$
- $|\text{BlockingRules}| = 54,360$
- $|\text{ErrorStatus}| = 9$

Computational complexity of the pattern matching query in a relational Database:
The complexity of performing such a query in a relational database is equivalent to performing
natural joins between all the tables used.

\[
\text{cost} = |\text{Nodes}| \times |\text{Segments}| \times |\text{Usages}| \times |\text{Destinations}| \times |\text{Preferences}|
\]

Knowing the data cardinality of our BHS, then

\[
\text{cost} = 1,590 \times 1,819 \times 35,818 \times 475 \times 3 = 147,620,278,336,500
\]
Computational complexity of the pattern matching query in a Graph Database: The execution plan of this query follows 3 steps: lookup the start_node among all the nodes, (\(|\text{Nodes}|\)), then reach to its outgoing edges (maximum 2 db hits), and move to their usages (maximum \(|\text{Usages}|\)).

\[
\text{cost} = |\text{Nodes}| + 2 + |\text{Usages}|
\]

Knowing the data cardinality of our BHS, then

\[
\text{cost} = 1,590 + 2 + 35,818 = 37,410
\]

We confirm that given our data model and our data cardinality, executing this pattern matching query in a Graph Database is significantly less complex than in a relational database.

### A.4 Shortest Path Queries

In real-time routing, it was necessary to have the schema of the relational model in Figure 3.22, or the entities in the graph in Figure 3.23. However, our goal is to find the missing locations between every pair of nodes in the log. For that, we can reduce our data model to the information represented in Figure 6.1. The property graph we create only has one label, LOCATION, which contains the entities from the table NODES, and the nodes are connected to each other with edges of type SEGMENT_TO. Each record from the table SEGMENTS becomes a triplet (start_node, segment_to, end_node), where start_node is the source and end_node is the sink of the directed edge segment_to (see the property graph in Figure 6.2).

```sql
// Recursive query that returns all possible paths from start_node to end_node
MATCH path = ( start_node : LOCATION { location_id : '30501101' }) - [ rel : SEGMENT_TO* ] -> ( end_node : LOCATION { location_id : '30501301' })
RETURN path
```

Listing A.4: All the path between a pair of consecutive nodes in the log

In Listing A.4 we show Neo4j functionality to perform recursive paths. The result of that query would return all possible paths between the pair of nodes, which is not our goal. The query we use to find the unique path between each pair of locations in the trace is the one in Listing A.5.

```sql
// Shortest path between from start_node to end_node
MATCH ( start_node : LOCATION { locationid : '30501101' } ), ( end_node : LOCATION { locationid : '30501301' } )
WITH start_node, end_node
CALL apoc.algo.dijkstra( start_node, end_node, 'SEGMENT_TO>', 'avg_duration', 1000, 1)
YIELD path, weight
RETURN path
```

Listing A.5: Shortest path between a pair of consecutive nodes in the log using Dijkstra's shortest path algorithm

The query in Listing A.5 will find the shortest path between two locations connected by conveyors, but it can still throw an error in some cases. If a bag trace contains a manual movement, i.e. a worker manually moved a bag from one location to another and the pair of locations were not connected by conveyors, the query will return an empty result. If the insertion query described in Listing D.2 is executed in the graph, then the shortest path query in Listing A.5 will also consider the manual movement segments.
Appendix B

Tables Definition of the Routing Database

Segments  The table segments contains all the segments in the Layout Graph. A segment connects a pair of nodes, the start_node and the end_node. The segment is directed to the end_node.

- segment_id: unique, primary key
- start_node: foreign key to node_id
- end_node: foreign key to node_id
- transport_default: Boolean
- avg_duration: (seconds, s.SSS)

Nodes  The table nodes contains all the nodes in the Layout Graph.

- node_id: unique, primary key
- description: string
- area_id: integer
- zone_id: integer
- equipment_id: integer
- node_type: MERGE, DIVERT, LINEAR, EXIT, ENTRY

![Figure B.1: Node types: configuration of incoming and outgoing edges per node](image)

96 Performance-aware Conformance Checking on Material Handling Systems
APPENDIX B. TABLES DEFINITION OF THE ROUTING DATABASE

Usages The table usages shows which segments connect to which destinations. It is possible to arrive to a destination via many segments, but some will have more preference than others. E.g. both segments $s_1$ and $s_2$ connect to destination $d_{10}$, but segment $s_1$ has more preference because it connects to $d_{10}$ via a shorter route. However, if segment $s_1$ is unavailable, then segment $s_2$ will be chosen, because it also connects to $d_{10}$.

- usage_id: unique, primary key
- destination_id: foreign key to destination_id
- preference_level: foreign key to preference_level
- segment_id: foreign key to segment_id

Destinations The table destinations contains some nodes in the Layout Graph. Those are nodes that won’t be reached by default unless they are specified in the logs.

- destination_id: unique, primary key
- description: string

Preferences The table preference contains the possible levels of preference that can be assigned to a usage. The highest level of preference is 1, and the higher the number, the lower the preference.

- preference_level: integer, unique, primary key
- description: string

Blocking Rules The table blocking_rules contains the triggers that make a segment unavailable when it shows an error between a severity range.

- rule_id: unique, primary key
- block_delay: time
- unblock_delay: time
- min_error_status: foreign key to status_id
- min_error_status: foreign key to status_id
- usage_id: foreign key to usage_id

Error_status The table segments contains all the segments in the Layout Graph.

- status_id: unique, primary key
- description: string
- severity_level: integer
Appendix C

Event Logs Pre-processing Algorithms

C.1 Algorithm to Reduce Bag Identifiers

input: Bag Information Report.csv // many extra fields: [ID1, ID2, timestamp, ...]
output: final_bagtags // fields: [ID1, ID2]

1 begin
2  Load data;
3  bagtag_changes ← Read (Bag Information Report.csv);
4  Keep a projection: keep only the attributes [ID1, ID2, timestamp];
5  bagtag_changes ← Π[ID1,ID2,timestamp] bagtag_changes;
6  Sort the records chronologically;
7  bagtag_changes ← bagtag_changes SET INDEX(timestamp);
8  bagtag_changes ← bagtag_changes SORT BY INDEX;
9  bagtag_changes ← bagtag_changes RESET INDEX;
10 Remove duplicate pairs (ID1,ID2). Keep the first occurrence;
11 bagtag_changes ← DropDuplicates(bagtag_changes,subset=[ID1, ID2],keep='first');
12 Create list of all the ID2 values. Each will be assigned the final ID1 of the bag;
13 final_bagtags ← bagtag_changes.ID2 UNIQUE;
14 Add an empty ID1 column to final_bagtags;
15 final_bagtags.ID1 ← ∅;
16 Iterate through the rows of bagtag_changes;
17 for row ∈ bagtag_changes do
18  Find row in final_bagtags with ID2 = row.ID2. Get ID1 value as old.ID1;
19  old.ID1 ← line.ID1 where line ∈ final_bagtags ∧ final_bagtags.ID2 = row.ID2;
20  If old.ID1 is an empty value, we insert row.ID1 in line.ID1;
21  if old.ID1 = ∅ then // new ID2
22    line.ID1 ← row.ID1 where line ∈ final_bagtags ∧ final_bagtags.ID2 = row.ID2;
23  else
24    Update all the final_bagtags.ID1 values where final_bagtags.ID1 = old.ID1 for row.ID1;
25    final_bagtags.ID1 ← row.ID1 where final_bagtags.ID1 = old.ID1;
26  end
27 end
28 end

Algorithm 1: Reduce bag identifiers
In Table C.1 we display an example where a log of 6 pairs (ID1, ID2) identify 2 different bags. The table Bag Tags’ Changes shows the initial log, and the table Final Bag Tags shows how we relate all the ID2 values with a unique bag identifier, i.e. the last ID1 value for each bag.

Table C.1 shows iterations of the Algorithm 1: Algorithm to Reduce Bag Identifiers. In the 1st and 2nd iterations, the ID2 value still does not exist in the table Final Bag Tags, and it is introduced as a new record. In the 3rd iteration, we encounter an ID2 value that already exists in the Final Bag Tags table. The value to be updated in the Final Bag Tags table is ID1=1. Iterations 4th and 5th encounter new ID2 values, and the pairs are inserted in the Final Bag Tags table. In the 6th iteration, we find the value ID2=4 in the Final Bag Tags table, therefore all the rows with ID1=2 must be updated. When we finish iterating over the Bag Tags’ Changes, we can see that we have two different bags, identified by ID1=4 and ID1=3, and every ID2 value can be linked to one of those bag identifiers. The events in the Tracking Reports are identified by ID2 values, and now we can link every event to a unique bag, to recreate each trace.
### APPENDIX C. EVENT LOGS PRE-PROCESSING ALGORITHMS

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>00:00:13</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>00:01:04</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00:01:17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>00:01:34</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>00:01:52</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>00:02:11</td>
</tr>
</tbody>
</table>

(a) Bag Tags’ Changes - 1st iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>00:00:13</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>00:01:04</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00:01:17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>00:01:34</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>00:01:52</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>00:02:11</td>
</tr>
</tbody>
</table>

(b) Final Bag Tags - 1st iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

(d) Final Bag Tags - 2nd iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>00:00:13</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>00:01:04</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00:01:17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>00:01:34</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>00:01:52</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>00:02:11</td>
</tr>
</tbody>
</table>

(c) Bag Tags’ Changes - 2nd iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

(f) Final Bag Tags - 3rd iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>00:00:13</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00:01:04</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>00:01:34</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>00:01:52</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>00:02:11</td>
</tr>
</tbody>
</table>

(e) Bag Tags’ Changes - 3rd iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

(h) Final Bag Tags - 4th iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>00:00:13</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00:01:04</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>00:01:34</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>00:01:52</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>00:02:11</td>
</tr>
</tbody>
</table>

(g) Bag Tags’ Changes - 4th iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

(j) Final Bag Tags - 5th iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>00:00:13</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>00:01:04</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>00:01:34</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>00:01:52</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>00:02:11</td>
</tr>
</tbody>
</table>

(i) Bag Tags’ Changes - 5th iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

(l) Final Bag Tags - 6th iteration

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

(k) Bag Tags’ Changes - 6th iteration

Table C.1: Iterations of the algorithm “Reduce bag identifiers”
Algorithm 2: Remove duplicate events

**input**: Tracking_Report_1.csv // fields: [ID2, timestamp, AreaID, ZoneID, EquipmentID, destination]
**input**: Tracking_Report_2.csv // fields: [ID2, timestamp, AreaID, ZoneID, EquipmentID, destination]
**output**: logs // fields: [ID2, timestamp, activity]

1. Load data;
2. TR1 ← Read (Tracking_Report_1.csv);
3. TR2 ← Read (Tracking_Report_2.csv);
4. Remove rows with missing values;
5. TR1 ← DropEmptyValues(TR1, subset=[ID2]);
6. TR2 ← DropEmptyValues(TR2, subset=[ID2]);
7. Union the two data sources;
8. logs ← TR1 ∪ TR2;
9. Remove duplicate events. Keep the first occurrence;
10. logs ← DropDuplicates(logs, keep='first');
11. Create attribute 'activity': concatenate AreaID, ZoneID and EquipmentID values;
12. logs.activity ← logs.AreaID || logs.ZoneID || logs.EquipmentID;
13. Projection of logs: keep only the attributes (ID2, timestamp, activity);
14. logs ← \( \Pi_{ID2,\text{timestamp},\text{activity}} \)logs;
15. Sort each trace chronologically;
16. logs ← logs SET INDEX(ID2,timestamp);
17. logs ← logs SORT BY INDEX;
18. Remove duplicate events. When encounter duplicate rows, just keep the first occurrence;
19. logs ← DropDuplicates(logs, keep='first');
20. Apply the Find Duplicates Algorithm. Split the logs based on if there are repeated timestamps in a trace;
21. logs_no_duplicates ← DropDuplicates(logs, subset=[ID2, timestamp], keep=False);
22. logs_duplicates ← logs - logs_no_duplicates // fields: [location_id, node_type];
23. By visual inspection, check physical order of duplicate events. Create list of preceding events.
24. Modify timestamps of preceding events to 1 ms earlier;
25. for row ∈ logs_duplicates do
26.  if row.activity ∈ list_preceding_events then
27.    Modify timestamps of preceding events to 1 ms earlier;
28.    row.timestamp ← row.timestamp - 1 ms;
29.  end
30. end
31. Union logs_no_duplicates and logs_duplicates again;
32. logs ← logs_duplicates ∪ logs_no_duplicates;
33. end
APPENDIX C. EVENT LOGS PRE-PROCESSING ALGORITHMS

C.3 Algorithm to Parse Process Steps

```
<nodes>
  <node isc_id_reporting = "7740.01.01" process_step = "PRE_SORT_TO_FINAL_SORT" />
</nodes>
```

Listing C.1: Snippet of the XML file with the definition of each node

**input**: Project Definition.xml
**input**: logs // fields: [ID2, timestamp, activity]
**output**: logs // fields: [ID2, timestamp, activity, process_step]

```
begin
  Load data;
  xml_file ← Read(Project Definition.xml);
  Initialize empty node_list // fields: [activity, process_step ];
  node_list ← ∅;
  Parse the XML file;
  for node ∈ xml_file do
    Retrieve values;
    process_step ← node.get(process_step);
    activity ← node.get(location_id);
    Insert in results;
    node_list ← node_list.insert(activity, process_step);
  end
  Join node_list with logs;
  logs ← InnerJoin(logs, node_list)
end
```

Algorithm 3: Parse process steps

```
import xml.etree.ElementTree as ET

# Parse the XML file
root = ET.parse(HL_system_path).getroot()
set_location_id = set()

for node in root.find('nodes'):
    process_step = node.get('process_step')
    isc_id_reporting = node.get('isc_id_reporting')
    if isc_id_reporting != None:
        set_location_id.update(((process_step, location_id)))

# Parsed content in df_location_id
df_location_id = pd.DataFrame(list(set_location_id), columns=['process_step', 'activity'])

# Join df_location_id with logs
logs = pd.merge(logs, df_location_id, on='activity', how='inner')
```

Listing C.2: Parse the BHS Definition file to obtain the process steps of the nodes in the LG
C.4 Algorithm to Parse LG Coordinates

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg >
<svg xmlns="" xmlns:xlink="" xmlns:ev="" xmlns:v="" >
<title>LG</title>
<style type="text/css"></style>
<g v:mID="0" v:index="1" v:groupContext="backgroundPage"></g>
<g v:mID="4" v:index="2" v:groupContext="foregroundPage">
  <g id="group13-494" transform="translate(623.622,-1105.51)" v:mID="13"
    v:groupContext="group">
    <v:custProps>
      <v:cp v:nameU="location_id" v:val="VT4(7706.04.99)"/>
      <v:cp v:nameU="process_step" v:val="VT4(LINK_BRIDGE)"/>
      <v:cp v:nameU="node_id" v:val="VT4(872)"/>
      <v:cp v:nameU="node_type" v:val="VT4(WSN)"/>
      <v:cp v:nameU="destination_id" v:val="VT4(356)"/>
    </v:custProps>
  </g>
</g>
</svg>
```

Listing C.3: Snippet of the SVG file with the coordinates of each node

```plaintext
input: physical_layout.svg
input: logs // fields: [ID2, timestamp, activity]
output: nodes_coordinates.csv // fields: [location_id, x, y]
1 begin
2 Load data;
3 svg_file ← Read(physical_layout.svg);
4 Initialize empty node_list // fields: [location_id, x, y];
5 node_list ← ∅;
6 Parse the SVG file;
7 for group ∈ svg_file do
8     for node ∈ group do
9         Retrieve values;
10        location_id ← node.get(value);
11        transform ← group.get(transform) // Get the CSS property transform;
12        Apply a regular expression to the translate function to obtain the relative coordinates of the node on the canvas;
13        translate ← FindAll(transform.regex);
14        Insert in results;
15        node_list ← insert(location_id, translate.x, translate.y);
16    end
17 end
18 Write results in a file;
19 nodes_coordinates.csv ← Write(node_list)
20 end
```

Algorithm 4: Parse nodes coordinates
def parse_nodes_coordinates(LG_path, output_path):
    root1 = ET.parse(LG_path).getroot()
    set_coordinates = set()

    for g in root1.findall("_:_g[@v:groupContext='foregroundPage']/_:_g[@v:groupContext='group']"):
        for isc_id_reporting in g.findall("./v:custProps/v:cp[@v:nameU='isc_id_reporting'][@v:lbl='']"):
            value = isc_id_reporting.get("val")
            transform = g.get("transform")
            translate = re.findall(r'translate\((-?\d+(?:\.&\d+|)),(-?\d+(?:\.&\d+|))\)',
                                   transform)
            set_coordinates.update([(location_id, translate[0][0], translate[0][1])])

    df_coordinates = pd.DataFrame(list(set_coordinates), columns=['location_id', 'x', 'y'])
    df_coordinates.to_csv(output_path + '\' + 'nodes_coordinates.csv', index=False)
    return

Listing C.4: Parse the SVG file to obtain the coordinates of the nodes in the LG
Appendix D

Cypher Queries and Algorithms to Complete the Bag Traces in the Graph Database

D.1 Cypher Queries to Build the Routing Graph

```cypher
// Create nodes LOCATION
LOAD CSV WITH HEADERS FROM 'file:///nodes.CSV' AS line
FIELDTERMINATOR ','
MERGE (location:LOCATION {location_id: trim(line.location_id), node_type: line.node_type})

// Create segments
LOAD CSV WITH HEADERS FROM 'file:///segments.CSV' AS line
FIELDTERMINATOR ','
MATCH (id_start:LOCATION {location_id: trim(line.locationid_start)}), (id_end:LOCATION {location_id: trim(line.locationid_end)})
MERGE (id_start) -[r:SEGMENT_TO]->(id_end)
SET r.transport_default = line.transport_default
SET r.avg_duration = ToFloat(line.avg_duration)
```

Listing D.1: Cypher queries to load the routing graph

D.2 Cypher query to include manual movements in the routing graph

```cypher
// Create SEGMENT_TO between nodes (EXT)-[]->(ETE) with avg_duration = 999
MATCH (n:LOCATION {node_type: 'EXT'}), (m:LOCATION {node_type: 'ETE'})
MERGE (n)-[r:SEGMENT_TO {avg_duration: 999, description: 'Manual_Movement'}]->(m)
```

Listing D.2: Cypher query to create manual movements
APPENDIX D. CYPHER QUERIES AND ALGORITHMS TO COMPLETE THE BAG TRACES IN THE GRAPH DATABASE

D.3 Pattern Matching Algorithm to Complete the Bag Traces

```cypher
MATCH (start_node:LOCATION {location_id: 'START'}) WITH start_node
MATCH (s:LOCATION {location_id: 'activity'}) WITH s, start_node
RETURN id(start_node) AS source_id, left(trim(start_node.location_id), 4) AS source_label, start_node.location_id AS source_name, properties(start_node) AS source_keys, 'bag_id' AS bag_id, 'process_step' AS process_step, 'timestamp' AS timestamp, id(s) AS target_id, left(trim(s.location_id), 4) AS target_label, s.location_id AS target_name, properties(s) AS target_keys
```

Listing D.3: Cypher Pattern Matching query for the first event of a trace

```cypher
MATCH (s) WHERE ID(s)='last_node' WITH s
MATCH (e:LOCATION {location_id: 'activity'}) WITH s,e
CALL apoc.algo.dijkstra(s, e, 'SEGMENT_TO>', 'avg_duration', 1000, 1)
YIELD path, weight WITH path
UNWIND relationships(path) AS r WITH startNode(r) AS _s_, endNode(r) AS _e_
RETURN id(_s_) AS source_id, left(trim(_s_.location_id), 4) AS source_label, _s_.location_id AS source_name, properties(_s_) AS source_keys, 'bag_id' AS bag_id, 'process_step' AS process_step, 'timestamp' AS timestamp, id(_e_) AS target_id, left(trim(_e_.location_id), 4) AS target_label, _e_.location_id AS target_name, properties(_e_) AS target_keys
```

Listing D.4: Cypher Pattern Matching query for every event of a trace except the first
APPENDIX D. CYpher QUERIES AND ALGORITHMS TO COMPLETE THE BAG TRACES IN THE GRAPH DATABASE

Figure D.2: Interpretation of the MATCH and the RESULT clause of Listing D.4

1 MATCH (s) WHERE ID(s)='last_node' WITH s
2 MATCH (end_node : LOCATION {location_id: 'END'}) WITH end_node, s
3 RETURN id(s) AS source_id, left(trim(s.location_id), 4) AS source_label, s.location_id AS source_name, properties(s) AS source_keys, 'bag_id' AS bag_id, 'process_step' AS process_step, 'timestamp' AS timestamp, id(end_node) AS target_id, left(trim(end_node.location_id), 4) AS target_label, end_node.location_id AS target_name, properties(end_node) AS target_keys

Listing D.5: Cypher Pattern Matching query to end a trace

Figure D.3: Interpretation of the MATCH and the RESULT clause of Listing D.5
APPENDIX D. CYPHER QUERIES AND ALGORITHMS TO COMPLETE THE BAG TRACES IN THE GRAPH DATABASE

```python
def query_START(bag_id, timestamp, activity, process_step):
    query = ""
    MATCH (start_node:LOCATION {{location_id: 'START'}}) WITH start_node
    MATCH (s:LOCATION {{location_id: '{activity} '}}) WITH s, start_node 
    RETURN id(start_node) AS source_id, left(trim(start_node.location_id), 4) AS source_label, start_node.location_id AS source_name, properties(start_node) AS source_keys, {bag_id} AS bag_id, '{process_step}' AS process_step, '{timestamp}' AS timestamp, id(s) AS target_id, left(trim(s.location_id), 4) AS target_label, s.location_id AS target_name, properties(s) AS target_keys
    ""
    query = query.format(bag_id=str(bag_id), timestamp=timestamp, activity=str(activity), process_step=process_step)
    return query

def query_ELSE(bag_id, timestamp, activity, process_step, last_node):
    query = ""
    MATCH (s) WHERE ID(s)={last_node} WITH s 
    MATCH (e:LOCATION {{location_id:'{activity}'}}) WITH s,e CALL apoc.algo.dijkstra(s, e, 'SEGMENT_TO>', 'avg_duration', 1000, 1) YIELD path, weight WITH path 
    UNWIND relationships(path) AS r WITH startNode(r) AS _s_, endNode(r) AS _e_ 
    RETURN id(_s_) AS source_id, left(trim(_s_.location_id), 4) AS source_label, _s_.location_id AS source_name, properties(_s_) AS source_keys, {bag_id} AS bag_id, '{process_step}' AS process_step, '{timestamp}' AS timestamp, id(_e_) AS target_id, left(trim(_e_.location_id), 4) AS target_label, _e_.location_id AS target_name, properties(_e_) AS target_keys
    ""
    query = query.format(bag_id=str(bag_id), timestamp=timestamp, activity=str(activity), process_step=process_step, last_node=last_node)
    return query

def query_END(bag_id, timestamp, process_step, last_node):
    query = ""
    MATCH (s) WHERE ID(s)={last_node} WITH s 
    MATCH (end_node:LOCATION {{location_id: 'END'}}) WITH end_node, s 
    RETURN id(s) AS source_id, left(trim(s.location_id), 4) AS source_label, s.location_id AS source_name, properties(s) AS source_keys, {bag_id} AS bag_id, '{process_step}' AS process_step, '{timestamp}' AS timestamp, id(end_node) AS target_id, left(trim(end_node.location_id), 4) AS target_label, end_node.location_id AS target_name, properties(end_node) AS target_keys
    ""
    query = query.format(bag_id=str(bag_id), timestamp=timestamp, process_step=process_step, last_node=last_node)
    return query
```

Listing D.6: Python functions that return the cypher queries with parameters, to be executed
APPENDIX D. CYPHER QUERIES AND ALGORITHMS TO COMPLETE THE BAG TRACES IN THE GRAPH DATABASE

input: bolt_connection
input: preprocessed_logs.csv
// fields: [ID1, timestamp, activity, process_step]
input: nodes.csv // fields: [location_id, node_type]
output: complete_traces.csv

begin
    bolt ← bolt connection;
    graph ← Graph(bolt);
    Create START and END nodes: the virtual start and end of each trace;
    graph ← Run(query);
    Load data;
    logs ← Read(preprocessed_logs.csv);
    Remove nodes not in the database and add node_type information;
    nodes ← Read(nodes.csv);
    logs ← InnerJoin(logs, nodes) // inner join ON (logs.activity, nodes.location_id);
    Make a list of ID1 values to iterate through the bag traces;
    bag_list ← logs.activity UNIQUE;
    Sort each trace chronologically;
    logs ← logs SET INDEX(ID1, timestamp);
    logs ← logs SORT BY INDEX;
    Remove consecutive duplicate events;
    logs ← logs.RemoveConsecutiveDuplicates(activity);
    Initialize output;
    complete_traces ← empty Dataframe;
    Iterate through bag traces;
    for ID1, trace ← logs.groupby(level=0) do
        for timestamp, row ← logs.groupby(level=1) do
            Special treatment of the first element of each trace;
            if row is the first element then
                query ← query.START (ID1, timestamp, row.activity, row.step)
            else
                query ← query.ELSE (ID1, timestamp, row.activity, row.step, last_node)
            end
            Execute query in Neo4j. Store result:
            result ← graph.Run(query);
            Union result from query with the final output;
            complete_traces ← complete_traces ∪ result
            Return id of last node, will be used to connect to END_node;
            If this row is the last in the trace;
            if row is the last element then
                query ← query.END (ID1, timestamp, row.step, last_node);
                Execute query in Neo4j. Store result:
                result ← graph.Run(query);
                Union result from query with the final output;
                complete_traces ← complete_traces ∪ result;
            end
        end
    end
    logs ← logs.RESET INDEX;
    complete_traces.csv ← Write(complete_traces);
end

Algorithm 5: Pattern Matching
D.4 Graph Data Model having Bag Traces Stored in the GraphDB

Our first approach to store the complete bag traces was to store them in the graph. Each bag movement would be an edge with the label MOVED_TO, and it would connect the locations the bag passed by. The edge would contain the information about the bag identifier, and the time of the movement. In Figure D.4 we show how would the data model of the property graph change, and in Figure D.5, we see an example of a trace, as it moved from one node to the next.

Figure D.4: Data Model of the property graph. The information about the bag traces is stored in the MOVED_TO edges

Figure D.5: Example of a trace in the graph
Appendix E

Code to Obtain Information Visualizations

E.1 Assign Performance Metric as Edge Color

```python
max_rel_keys = df[column_to_color].max()
color_list = []
cmap = plt.get_cmap(palette, int(max_rel_keys/10**10))
for i in range(cmap.N):
    rgb = cmap(i)[:3] # will return rgba, we take only first 3 so we get rgb
color_list.append(matplotlib.colors.rgb2hex(rgb))

for index, row in df.iterrows():
    color = color_list[int(row[column_to_color]/10**10)-1] if int(row[column_to_color]/10**10)-1 > -1 else color_list[int(row[column_to_color]/10**10)]

    edges.append({"from": source_info["id"], "to": target_info["id"], "label": str(rel_id), "title": "route: " + str(rel_id) + ", " + column_to_color + " + repr(pd.Timedelta(rel_keys, unit = 'ns')) + " load: " + str(row[column_to_weight]) + " color: " + color, "smooth": {"type": 'curvedCW', "roundness": edge_roundness }})
```

Listing E.1: Assign color to edges. choose a palette. Assign the minimum and maximum values to the minimum and maximum performance. Each possible performance is a color in-between
E.2 Pass Nodes and Edges to the JS vis.Network

```python
for index, row in df.iterrows():
    # read values
    source_info = {
        "id": source_id, "label": source_name, "group": source_label, "title": repr(source_keys), "x":source_xy.iat[0,1], "y":source_xy.iat[0,2]}
    if source_info not in nodes:
        nodes.append(source_info)

    target_info = {
        "id": target_id, "label": target_name, "group": target_label, "title": repr(target_keys), "x":target_xy.iat[0,1], "y":target_xy.iat[0,2]}
    if target_info not in nodes:
        nodes.append(target_info)

    edges.append({
        "from": source_info["id"], "to": target_info["id"], "label": str(rel_id), "title": "route: "+str(rel_id)+", "+column_to_color": "+repr(pd.Timedelta(rel_keys, unit='ns'))+, load: "+str(row[column_to_weight]), "value":row[column_to_weight], "color": color, "smooth": {"type": 'curvedCW', "roundness": edge_roundness }
    })
```

Listing E.2: Pass nodes and edges to the JS vis.Network
E.3 Nodes and Edges Objects in JS vis.Network

Properties of nodes on JS vis.Network:
- *id*: unique identifier of a node
- *label*: text on the node
- *group*: category to color the node
- *title*: extra information about the node. Floating note appears when we hover on the node
- *x*: location of the node on the canvas on the x-axis
- *y*: location of the node on the canvas on the y-axis

Properties of edges on JS vis.Network:
- *from*: node ID where the edge starts
- *to*: node ID where the edge ends (with the arrow pointer)
- *label*: text on the edge
- *title*: extra information about the edge. Floating note appears when we hover on the edge
- *value*: thickness of the edge
- *color*: color of the edge
- *smooth*: shape of the edge. We assign curved shape to the edges and a different level of roundness to each edge between each pair of nodes in order for them not to overlap.
  - *type*: 'curvedCW'
  - *roundness*: a different number for every edge for a pair of nodes
Appendix F

Variants Generation Algorithms
F.1 Algorithm to Aggregate Bag Traces into HL Variants

\[\text{input}: \text{complete\_traces.csv} \]
\[\text{output}: \text{HL\_sequences.csv, bags\_per\_HL\_sequences.csv, traces\_HL\_sequences.csv} \]

begin
  Load data;
  traces ← Read(complete\_traces.csv);
  Projection of traces with attributes [ID1, timestamp, process\_step, source\_name, target\_name];
  HL\_sequences ← \Pi_{(ID1,timestamp,process\_step,source\_name,target\_name)}\text{traces};
  Having bag traces, make sequences of process\_steps;
  Sort each trace chronologically;
  HL\_sequences ← HL\_sequences \text{SET INDEX}(ID1,timestamp);
  HL\_sequences ← HL\_sequences \text{SORT BY INDEX};
  Remove records with consecutive duplicate process\_step;
  HL\_sequences ← HL\_sequences.RemoveConsecutiveDuplicates(process\_step);
  // HL\_sequences ← HL\_sequences.loc[HL\_sequences[process\_step].shift() != logs[process\_step]];
  HL\_sequences ← HL\_sequences \text{RESET INDEX};
  // Calculate time metrics;
  Calculate duration of each process\_step per bag;
  HL\_sequences.duration ← HL\_sequences.ConsecutiveDifference(timestamp);
  // HL\_sequences[']duration'] ← HL\_sequences['timestamp'] - HL\_sequences['timestamp'].shift().fillna(pd.to\_timedelta("00:00:00"));
  Remove the process\_step = Storage. Its duration is not part of the process;
  HL\_sequences ← HL\_sequences[process\_step != Storage];
  Calculate the cumulative duration per trace;
  groupby\_cumsum ← \gamma(ID1,timestamp)(duration);HL\_sequences;
  // groupby\_cumsum ← HL\_sequences.groupby(['ID1']).apply(lambda x: x['duration'].cumsum());
  HL\_sequences ← ConcatenateColumns(HL\_sequences, groupby\_cumsum, on=['ID1', index]);
  // Create sequences of process\_steps;
  For each trace, concatenate the process\_step values. Obtain an array of tuples (ID1, sequence);
  bags\_per\_HL\_sequences ← 0; \text{fields: (ID1, HL\_sequence)};
  foreach trace ∈ HL\_sequences do
    HL\_sequence ← trace.ConcatenateValues(process\_step);
    bags\_per\_HL\_sequences ← bags\_per\_HL\_sequences.insert(trace.ID1, HL\_sequence);
  end
  // bags\_per\_HL\_sequences ← [(x,'*'.join([row for row in HL\_sequences.loc[[HL\_sequences['ID1'] == x, 'process\_step']]]) for x in set(HL\_sequences['ID1'])];
  Assign a class to each sequence;
  Calculate the number of bags for each sequence of process\_steps;
  HL\_sequences ← \gamma(HL\_sequences.COUNT(ID1)) bags\_per\_HL\_sequences;
  Assign a class to each sequence, based on the descending number of bags per sequence;
  HL\_sequences ← HL\_sequences \text{SORT BY number\_of\_bags};
  HL\_sequences.HL\_class ← HL\_sequences.index;
  bags\_per\_HL\_sequences ← InnerJoin(bags\_per\_HL\_sequences, HL\_sequences) // inner join ON HL\_sequence;
  HL\_sequences ← InnerJoin(HL\_sequences, bags\_per\_HL\_sequences) // inner join ON ID1;
  Export results;
  HL\_sequences.csv ← Write(HL\_sequences) // fields: [HL\_class, HL\_sequence, number\_of\_bags];
  bags\_per\_HL\_sequences.csv ← Write(bags\_per\_HL\_sequences) // fields: [ID1, HL\_class, HL\_sequence, number\_of\_bags];
  traces\_HL\_sequences.csv ← Write(HL\_sequences) // fields: complete\_traces + [HL\_class, HL\_sequence, number\_of\_bags];
end

Algorithm 6: Classify sequences of process steps
APPENDIX F. VARIANTS GENERATION ALGORITHMS

F.2 Algorithm to Aggregate Bag Traces into LL Variants

input : `complete_traces.csv`
// fields: [ID1, timestamp, process_step, source_name, target_name, ...]
input : `HL_list`
// list of process_steps to include
output: `HL_sequences.csv` // fields: [HL_class, HL_sequence, number_of_bags]
output: `bags_per_route.csv` // fields: [ID1, HL_class, HL_sequence, number_of_bags]
output: `traces_HL_sequences.csv` // fields: `complete_traces + [HL_class, HL_sequence, number_of_bags]`

begin

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Load data;</td>
</tr>
<tr>
<td>2</td>
<td><code>traces ← Read(complete_traces.csv)</code>;</td>
</tr>
<tr>
<td>3</td>
<td>Slice traces to a subset of High-Level events;</td>
</tr>
<tr>
<td>4</td>
<td><code>traces ← σ(process_step ∈ HL_list) traces</code>;</td>
</tr>
<tr>
<td>5</td>
<td>Create sequences of low-level events (routes);</td>
</tr>
<tr>
<td>6</td>
<td>For each trace, concatenate the location_id values. Obtain an array of tuples (ID1, LL_sequence);</td>
</tr>
<tr>
<td>7</td>
<td><code>bags_per_route ← ∅</code> // fields: [ID1, LL_sequence];</td>
</tr>
<tr>
<td>8</td>
<td>foreach <code>trace ∈ traces</code> do</td>
</tr>
<tr>
<td>9</td>
<td>LL_sequence ← `trace.ConcatenateValues(source_name)</td>
</tr>
<tr>
<td>10</td>
<td>bags_per_route ← bags_per_route.insert(trace.ID1, LL_sequence);</td>
</tr>
<tr>
<td>11</td>
<td>end</td>
</tr>
<tr>
<td>12</td>
<td>// bags_per_route ← [x,&quot;&quot;:join([row for row in traces.loc[(traces['ID1'] == x), 'source_name']])&quot;&quot;:&quot;&quot;.join([row for row in traces.loc[(traces['ID1'] == x) &amp; (traces['ID1'] != traces['ID1'].shift()), 'target_name']]) for x in set(traces['ID1'])];</td>
</tr>
<tr>
<td>13</td>
<td>Assign a class to each route;</td>
</tr>
<tr>
<td>14</td>
<td>Calculate the number of bags for each route;</td>
</tr>
<tr>
<td>15</td>
<td><code>routes ← γ([LL_sequence, COUNT(ID1)]) bags_per_route;</code></td>
</tr>
<tr>
<td>16</td>
<td>Assign a class to each route, based on the descending number of bags per route;</td>
</tr>
<tr>
<td>17</td>
<td><code>routes ← routes SORT BY number_of_bags;</code></td>
</tr>
<tr>
<td>18</td>
<td>routes.LL_class ← routes.index;</td>
</tr>
<tr>
<td>19</td>
<td>bags_per_route: list of bags and route they belong to;</td>
</tr>
<tr>
<td>20</td>
<td>bags_per_route ← InnerJoin(bags_per_route, routes) // inner join ON LL_sequence;</td>
</tr>
<tr>
<td>21</td>
<td><code>traces_routes: bag traces and route they belong to;</code></td>
</tr>
<tr>
<td>22</td>
<td><code>traces ← InnerJoin(traces, bags_per_route) // inner join ON ID1;</code></td>
</tr>
<tr>
<td>23</td>
<td>Calculate time metrics;</td>
</tr>
<tr>
<td>24</td>
<td>Calculate duration of each low-level event per bag;</td>
</tr>
<tr>
<td>25</td>
<td><code>traces.duration ← traces.ConsecutiveDifference(timestamp);</code></td>
</tr>
<tr>
<td>26</td>
<td>// <code>traces['duration'] ← traces['timestamp']−traces['timestamp'].shift().fillna(pd.to_timedelta(&quot;00:00:00&quot;));</code></td>
</tr>
<tr>
<td>27</td>
<td>Calculate the cumulative duration per trace;</td>
</tr>
<tr>
<td>28</td>
<td>groupby_cumsum ← <code>γ([ID1].Cumsum(duration))</code> traces;</td>
</tr>
<tr>
<td>29</td>
<td>// groupby_cumsum ← traces.groupby('ID1').apply(lambda x: x['duration'].cumsum());</td>
</tr>
<tr>
<td>30</td>
<td><code>traces ← ConcatenateColumns(traces, groupby_cumsum, on=[ID1,index]);</code></td>
</tr>
<tr>
<td>31</td>
<td>Calculate the total time in the system for each bag;</td>
</tr>
<tr>
<td>32</td>
<td><code>traces.total_time ← Last(traces.cumsum);</code></td>
</tr>
<tr>
<td>33</td>
<td>Calculate remaining time in the system for each bag;</td>
</tr>
<tr>
<td>34</td>
<td><code>traces.remaining_time ← traces.total_time − traces.cumsum;</code></td>
</tr>
<tr>
<td>35</td>
<td>Export results;</td>
</tr>
<tr>
<td>36</td>
<td><code>routes.csv ← Write(routes) // fields: [LL_class, LL_sequence, number_of_bags];</code></td>
</tr>
<tr>
<td>37</td>
<td><code>bags_per_route.csv ← Write(bags_per_route) // fields: [ID1, LL_class, LL_sequence, number_of_bags];</code></td>
</tr>
<tr>
<td>38</td>
<td><code>traces_routes.csv ← Write(traces) // fields: complete_traces + [LL_class, LL_sequence, number_of_bags];</code></td>
</tr>
</tbody>
</table>

Algorithm 7: Classify low-level routes
Appendix G

User Interface Screenshots

We provide a Proof of Concept UI for the user to execute our implementation. The user can create a workspace to load the input data, and execute the steps of the Data Preparation individually. In the Modeling/Data classification phase, the user can customize the visualizations available, by selecting a subset of traces or process variants to visualize, and selecting the parameters to include in the visualizations. Here we display some of the menus of the application:

- Figure G.1 Menu 1 - Settings
- Figure G.2 - Menu 2 - Create Workspace
- Figure G.3 - Menu 2 - Select a date to evaluate
- Figure G.4 - Menu 3 - Main Menu
- Figure G.5 - Menu 4 - Steps in Data Preparation
- Figure G.6 - Menu 4 - Requisites in Data Classification
- Figure G.7 - Menu 5 - Steps in Data Classification
- Figure G.8 - Menu 6 - Steps in High-level Classification
- Figure G.9 - Menu 6 - Steps in Low-level Classification
APPENDIX G. USER INTERFACE SCREENSHOTS

Figure G.1: Menu 1 - Settings

Figure G.2: Menu 2 - Create Workspace

Figure G.3: Menu 2 - Select a date to evaluate

Figure G.4: Menu 3 - Main Menu

Figure G.5: Menu 4 - Steps in Data Preparation

Figure G.6: Menu 4 - Requisites in Data Classification
APPENDIX G. USER INTERFACE SCREENSHOTS

Figure G.7: Menu 5 - Steps in Data Classification

Figure G.8: Menu 6 - Steps in High-level Classification

Figure G.9: Menu 6 - Steps in Low-level Classification