MASTER

Knowledge Graphs for Improving Robot Operations in Logistics

Barenholz, Daniel

Award date:
2022

Eindhoven University of Technology
Department of Mathematics and Computer Science
Process Analytics

Knowledge Graphs for Improving Robot Operations in Logistics

Master Thesis

Daniël Barenholz

Supervisors:

Eindhoven University of Technologies	Dirk Fahland
Vanderlande Industries B.V.	Roel van den Berg
Vanderlande Industries B.V.	Jorn Bakker

Version 1.0

Utrecht, Eindhoven 2022

Abstract

Vanderlande is a company providing future-proof logistic process automation in, amongst others, warehousing for the food segment. This segment requires high availability and diversity of products with a limited workforce. To combat aforementioned problems within the food segment, Vanderlande has developed the *STOREPICK evolution*: a robotised, end-to-end Automated Case-Picking (ACP) warehousing solution, consisting of various modules, each with their own dataset(s). This thesis is the first data-driven approach to making Vanderlande's ACP solution more robust against errors. Part of the STOREPICK evolution is a palletizer cell, where a robot arm grabs and places cases on top of a pallet. We call this process (automated) palletisation. Notice how the palletisation process occurs in a physical setting. We implement a proof of concept data integration pipeline to construct a knowledge graph describing the physical palletisation process from the various available datasets, and evaluate which questions (about the palletisation process) can be answered reliably, either by querying it or using visual analytics. During this exploration on the usecase of knowledge graphs for modelling both a physical setting in tandem with its process for the purpose of detecting machine faults, we find that there is a critical data quality issue with respect to the recorded Z axis values of cases on pallets. We discuss the consequences of the data quality issue, and provide insights into other potential usecases of the graph as data model, comparing it to a more traditional tabular data format.

Keywords: proof of concept, reliability of machines, knowledge graphs, data integration pipeline, data quality issues

Preface

This work is the culmination of my master's programme, and concludes my studies of Data Science in Engineering (DSiE) at Eindhoven University of Technology (TU/e). It is the result of my graduation project, done in cooperation with Vanderlande Industries B.V. (Vanderlande) and the Process Analytics (PA) group of the Mathematics and Computer Science department at TU/e.

Words cannot express my gratitude to my supervisor and mentor throughout my entire master's programme, Dirk Fahland. Thank you for your invaluable guidance and input, and for giving me the opportunity to take upon myself an interesting research project at Vanderlande. My thanks also extend to my examination committee: Odysseas Papapetrou, Roel van den Berg, and Dirk Fahland.

This endeavor would not have been possible without Vanderlande trusting me with an important and brand-new project. There are too many people to name, but in particular I would like to thank Jorn Bakker and Roel van den Berg for their supervision during the project. My gratitude extends to all employees and interns alike whom I have sparred with, guiding me in which way to tackle the problems at hand.

Lastly, I would like to mention my coworkers at Utrecht University who decided hiring me as PhD Candidate whilst still having to finish a master's degree poses no issues. It is incredibly to me that I am given this amazing opportunity when other qualified candidates were available. Thanks should also go to my friends and family for their support (and more importantly patience) the past year.

Also: thank you, dear reader, for reading my master thesis! I hope you enjoy what you see. See you in five years for my PhD dissertation?

Daniël Barenholz

Eindhoven,

October 2022

Contents

1 Introduction

This section introduces the research problem of the thesis. Section 1.1 contextualises the problem. The specific research questions and hypotheses that we wish to investigate are elaborated on in Section 1.2. In said section we also explain the scope of the project. The approach and desired outcomes for the project are located in Section 1.3. Finally, in Section 1.4 we briefly sketch the contribution this thesis brings, from both a business and academic perspective.

1.1 Context and Motivation

Vanderlande is a company providing future-proof logistic process automation in various areas. Vanderlande is the world leader in the airport area, and one of the leaders in the parcel and warehousing areas. Their warehousing solutions are the first choice for major e-commerce players across the globe, helping them to fulfil their promise of same-day delivery for billions of orders. Furthermore, nine out of the fifteen largest global food retailers rely on Vanderlande's efficient and reliable solutions [1]. Reliability is one of the key important factors to Vanderlande's systems.

Within the warehousing area, Vanderlande distinguishes three *segments*: general merchandise, fashion, and *food*. The food segment in the warehousing area is the context of this work. The key challenges within this segment are (i) **High availability:** when shopping in your favourite supermarket, it is highly undesirable to see a notice that a particular product is no longer in stock. As such, it is important that products are always available at a store. Even if it is no longer present on a shelf, it should be easy to restock from the internal storage area. Even more, customers expect these items to be fresh. (ii) **Diversity:** one person may be lactose intolerant, where the other may have a gluten allergy. Some religions have certain requirements on how animals are slaughtered, or even which animals are allowed to be eaten. As a supermarket one would like to be able to provide products for all of these instances, resulting in a wide array of products, each with their own size, weight, and other properties. (iii) **Limited workforce:** with the increasing demand, there are more tasks to be completed, with fewer people to complete said tasks. To resolve this, *automation* is key, which comes with its own array of (technological) problems.

To combat the aforementioned problems within the food segment, Vanderlande has developed the *STOREPICK evolution*: a robotised, end-to-end Automated Case-Picking warehousing solution. STOREPICK focuses on, amongst others, scalability, flexibility, and an agreeable user experience, both for the end-user in a supermarket, as well as operators working behind the scenes. It consists of various components, ranging from a control room with CCTV, to software — Load Forming Logic (LFL) — that computes how items should be stacked on a pallet and the machinery to automatically palletise the computed recipes [2]. An info-graphic depicting STOREPICK, which is further explained in Section 3, is shown in Figure 1.

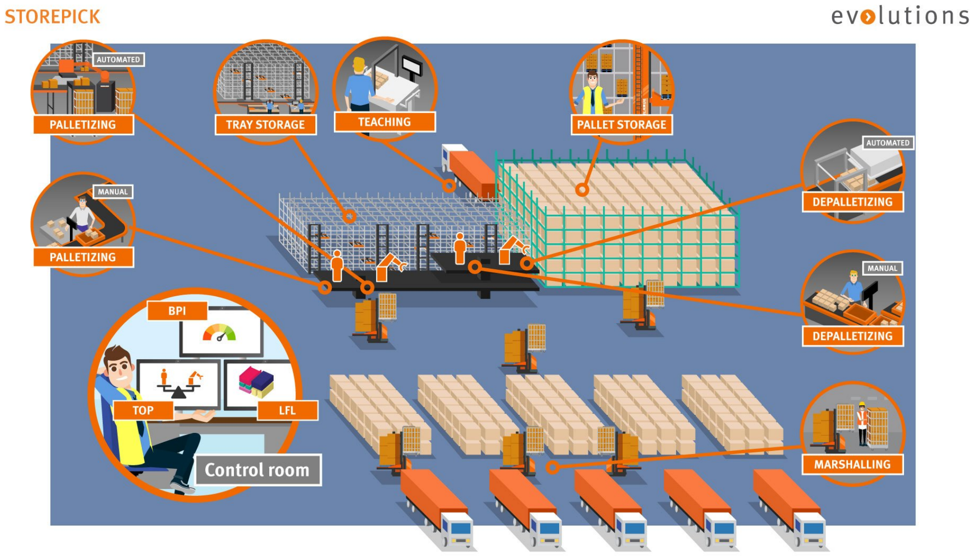

Figure 1: The Vanderlande STOREPICK evolution. The info-graphic shows the various components that make up STOREPICK.

Computing and consequently executing an order from a client is non-trivial. Realise that an order may consist of many pallets, with many different items, all which should arrive at their final location without issues. As with many automated systems, sometimes something might go wrong, resulting in undesired error states. In this work, we are using a data-driven approach to investigate why the ACP part of STOREPICK sometimes enters a specific error state during palletisation (the act of stacking items on a pallet using amongst others a robot arm), as Vanderlande wants to minimise the occurrence of this error state.

1.2 Research Questions and Scope

In particular, we are looking at a single type of error that can happen during palletization: the *STO* error. A part of the palletization system consists of a camera that hangs above the pallet on which items are to be stacked. This camera, commonly referred to as *STO* (see Section 3.2.1), takes pictures of every single related action that the palletizer performs: there is a picture when placing the pallet, a picture for the first case, a picture for the second case, ... When a machine vision component that is part of the camera believes that the picture taken shows something that is unexpected, an *STO error* is raised. In short: we are looking at errors due to unexpected behaviour exhibited during the placement of *cases* (the term cases is used to refer to products being stacked), and we want to do this using a *data-driven* approach.

There are many possible underlying reasons, and thus underlying avenues of research, as to why such STO errors might happen. These underlying reasons may both be hardware-related (e.g. faulty motors, wrongly calibrated robot arm, . . .) and software-related (e.g. incorrect commands to robot, incorrect corrections, . . .). After discussion with business experts and Vanderlande engineers, where each scored potential underlying reasons on a scale of one (1) to ten (10), we settled on looking into following 4 hypotheses, as these were scored highest amongst the potential underlying reasons.

HP 1 **Incorrect placements cause more STOs**, further explained in Section 3.3.1.

HP 2 **Building towers in the stack causes more STOs,** further explained in Section 3.3.2.

HP 3 **Height gaps cause more STOs**, further explained in Section 3.3.3.

HP 4 **Overhang causes more STOs**, further explained in Section 3.3.4.

1.3 Approach and Desired Outcomes

The first step of the approach is hidden, as it is already executed: identifying the main issues to analyse. The issue to analyse is the occurrence of the STO error, particularly in accordance with the hypotheses shown in Section 1.2. We want to use a *data-driven* approach to investigate this. Any data-driven approach requires data, hence the first non-hidden step of the approach is to acquire a suitable dataset. Data at Vanderlande is distributed over various locations, making data acquisition a non-trivial problem. There is no universal interface to all datasets either. As such, we need to acquire datasets from various sources, and then integrate them together. Since this research is the first data-driven approach done at Vanderlande, it is exploratory in nature: we desire only a proof of concept. As such, the data integration pipeline is to be made by coding it, as opposed to using existing integration tools (see Section 2.2 for background information).

Notice that the context of the process we investigate – picking and placing products in the right order according to some recipe – is set in the physical world. The physical setting itself influences the process: how the robot arm places case A influences how the next case B (and also any other cases C, D, ... after B) is placed. As such, to answer the hypotheses posed in Section 1.2 we must model the *physical setting* in tandem with the process itself. We propose to use a graph database for this (see 2.1 for background information), as they are visual by design (good visual analytical power for investigating the hypotheses) and we hypothesise they allow for easy modelling of a physical setting.

In summary, there are three (3) desired outcomes to this research project.

D1 The creation of a proof of concept data model for a graph of the palletisation process.

D2 An implementation of a proof of concept data integration and processing pipeline to construct such graph from available data sources.

D3 An evaluation of which questions about the palletisation process can be answered reliably on the graph.

1.4 Contribution and Findings

For Vanderlande, our contribution and findings are defined by delivering on the three desired outcomes: we find (Section 4) and integrate (Section 5) required datasets for modelling the process (**D2**). We propose a theoretical data model (Section 6.1) for the palletisation process (**D1**) and provide a proof of concept implementation in Section 6.2 (**D2**). We find that there is a critical issue in the recorded data for four specific fields, making the graph unusable as-is: no questions about the palletisation process can be answered reliably with the graph (**D3**).

In terms of academic contribution, this work can be seen as a case study on creating a knowledge graph for modelling the physical setting of a process in tandem with the process itself. We show how the graph has helped in finding the aforementioned critical issue in the recorded data (Section 7). For the four hypothesis from Section 1.2 we sketch the differences between using a graph as data model versus a tabular format specific to the physical setting (Section 8.2). We also (very briefly) sketch the powerful potential of the graph, should there not have been data issues (Section 8.4). This work also contributes to the field of fault detection by means of a novel approach: using a graph data model.

2 Background

This sections contains all relevant background information pertaining to four relevant topics: first, we explain what a *property graph* is, and how it relates to the *graph database model* in Section 2.1. Second, we explain the main idea behind *data integration* and why it is necessary in Section 2.2. Third, we show various related items to *reliability of machines* in Section 2.3, starting with machine failure (Section 2.3.1), through machine degradation (Section 2.3.2) to maintenance scheduling (Section 2.3.3). Finally, we briefly present related work done at Vanderlande in Section 2.4.

2.1 Property Graphs & (Graph) Database Models

Databases have been around for approximately a century [3], and they typically consist of *a data model*, *a query language*, and *integrity rules* [4]. The data model is a set of data structure types. It effectively explains *how* data is stored. The query language generally is a set of (query) operators or inference rules. When performing a query on the underlying data model, effectively one asks it a question to which an answer is desired. The query language thus is related to how to *use* the stored data: it is used to retrieve or derive the data that is stored in the data model. Finally, integrity rules ensure that any CRUD (create, remove, update, delete) operation done on the database produces again a valid database. The set of integrity rules is a collection of consistent database states, allowed changes of states, or both [5].

In theoretical computer science, a graph is commonly defined as a tuple $G = (V, E)$, where V denotes a set of *vertices* or *nodes* and E denotes a set of *edges*. This fails to capture, however, that there are many different types of graphs, and that these types stem from particular properties a graph might have. Examples of these properties are: directed vs undirected edges, weighted vs unweighted edges, labelled vs unlabelled, attributed vs unattributed, and potentially more. See [6] for an introduction on the consequences of a graph having a certain set of properties.

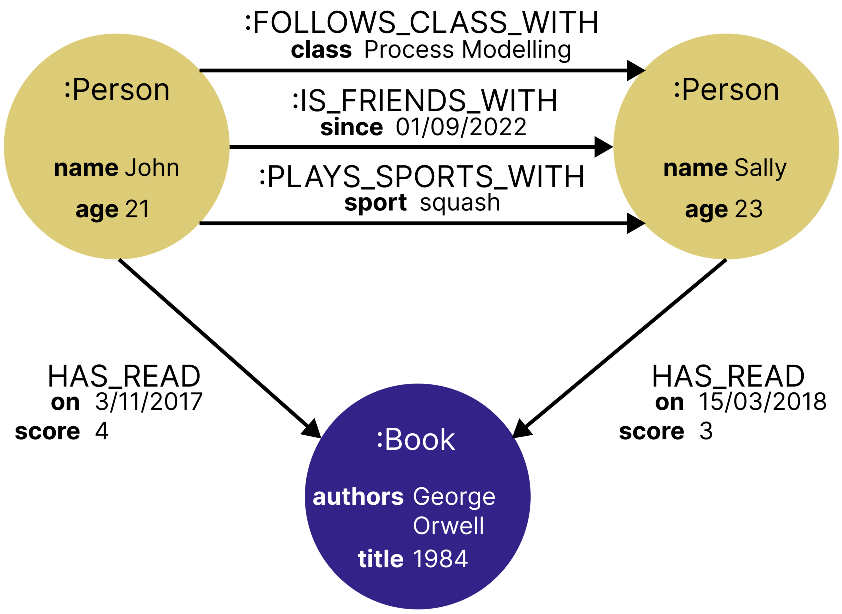

Figure 2: Example property graph, illustrating the various graph properties it has, adapted from [7].

In the context of this work, when we talk about a graph, we generally refer to a *(labelled) property graph*: this is a *directed*, *labelled*, and *attributed multigraph.* An example is shown in Figure 2. Directed here means that edges $e \in E$ have *direction*, illustrated by the arrow heads. Labelled means that both nodes and edges have *labels.* An edge might be used to denote a specific type of *relation* between its nodes, say *is_friends_with*, and nodes too may have a specific *type*, say *person.* With attributed we mean to say that nodes (and edges) have *attributes* (sometimes referred to as properties): a node of type *person* may store the persons name and age as attributes, and an edge with the *is_friends_with* relation may include metadata such as since when the two people were friends. Finally, multigraph means that between two nodes there may be *multiple* edges: in the illustration John is not only friends with Sally, but they also play squash together, and go to the same process modelling class.

A *graph* database is a database where the underlying data model typically is a *property graph* [6]. It is paired with a *graph query language* such as Cypher [8], which (naturally) allows for querying the graph. Besides querying for a single node-type, or single relation-type, Cypher (and most other graph query languages) allow for complex *pattern matching.* Using the same graph idea as shown in Figure 2, an example of pattern matching is: finding all books that John's friends have read between 2014 and 2016, where they rated it with a score higher than four. Other examples are finding pairs of two authors who have collaborated on multiple books, or finding authors who have no books in common. Besides the data model and query language, there are integrity constraints specific to a graph: the property graph schema. This schema might say that if a node p has type $person$, then the node must be unique (which makes sense when modelling: every person is unique). Other constraints that the schema might specify are mandatory property types (a person must have a name and date of birth) or cardinality constraints (a book is allowed only zero outgoing edges of type *has_read*, since books themselves cannot read). For a more formal explanation and definition of a graph database, see [6].

Property graphs and graph databases are used for various purposes. A non-exhaustive list of various usecases from a business perspective is provided by Neo4J, creators of Cypher, in [9]. Some fields/topics where graphs are used are supply chain management (to find potential weak links in the supply chain faster [10]), life sciences (map patient journeys to better understand disease progression [11]), and social network graphs (allows analysis directly on the domain model, as social networks are already graphs [12]). In academia, graphs are also used: in [13] a property graph is used to create an event knowledge graph, a type of knowledge graph to investigate inter-process relations within multi-process mining. In [14] graphs are used to create a model of the topology of a power grid network which is then used to do faster analysis than possible in relational databases. In [15] the authors use Twitter mentions to create a graph for analysing centrality measures. Even logging files from networks are put in a graph database in [16] to better analyse the relations between various files.

Although above paragraph is not, nor intended to be, a formal survey of graph database and property graph usages — we refer you to [17] — it is clear that (property) graphs are useful in many different scenarios. As such, in this work we investigate if they are also usable for modelling a physical process to find causes of errors in this process, directly using the available data. The cited sources are meant as an argument for the wide usability of graphs.

2.2 Data Integration

A property graph (Section 2.1) is a data model showing how various entities interact. Usually, these entities come from different sources, and must in some way be combined before they can be used (put into the graph). This is where data integration comes in: combining multiple datasets, from various sources, into one cohesive view of the underlying data. According to various toolmakers [18, 19, 20, 21, 22, 23], there are following paradigms when it comes to data integration.

Manual Data Integration As written in the name, manual data integration means to code your own integration pipeline, without using any specialised tooling. It is the most basic form of data integration, and is usable for quick prototypes. It is, however, far from scalable, and relatively error-prone. On the other hand, it allows for greater flexibility, as the user has total control over the integration.

Middleware Integration The term middleware is used to describe software that enables communication between various (legacy and new) applications. It acts as a bridge between various technologies. When applied to data integration, then this means that there is a piece of existing software (the middleware) that serves as a layer in-between applications who want to use data, and the data itself. Middleware is usually limited when it comes to usable data sources – all data sources need an implementation available in the middleware, and these may not always be present – and might not be the best for specialised needs.

Application-based Integration This idea is to let a smart application handle all data integration tasks. The application processes data from various sources to make them compatible with one another, and does so automatically, after a (very) complicated setup. Since this is mostly automated, analysts can work on doing analysis as opposed to finding and combining data together. Similar to middleware integration, the application must support the desired data sources.

Data Warehousing A data warehouse, in plain terms, is a (collection of) large server(s) with a lot of storage, that contains all data of the system. With data warehousing, all data is *copied* from source to a single centralised place, which can then be queried and analysed. This is sometimes also called *common storage integration*. The positive of data warehousing is that all data is available in one place (less time to find where data comes from), but since data is copied to a single place a lot of extra storage is required to facilitate this, which might be (very) expensive.

Data Virtualization Similar to a data warehouse, with data virtualization one provides a unified view of the data to the analysts and users. The big difference, however, is that this is a *virtual view*: no data is copied, and it stays on their source systems. Clearly, this requires less raw storage to achieve, but since data is stored on their source systems, those systems must be powerful enough to support the desired queries.

A common term that is used in the data integration world is **ETL**, which stands for **e**xtract-**t**ransform-**l**oad. Before data can be used, it must be *extracted* from various source systems. After extraction of the (raw) data, it usually must be *transformed* or otherwise preprocessed to fit the desired usecases, such as making a graph database. Then, it can be *loaded* (stored) into a database.

Both data integration and ETL have been extensively studied in academia. This paragraph provides a brief list of some relevant articles. First, a theoretical framework for semantic interoperability between heterogeneous data sources is coined in [24]. In [25], a theoretical perspective of data integration in its whole is provided. In [26] the authors provide some insight into previous work done in the data integration field, which has been rewritten into a complete book on the principles and ideas of data integration [27]. Another overview of problems and approaches for data integration is shown in [28]. For data integration specific to data warehousing, see [29], and for data integration specific to middleware integration (authors investigate "Data Federation" which means using a relational database as middleware), see [30]. For various ETL approaches, see [31], and a complete case study following the entire ETL process in [32].

Similar to Section 2.1, above overview is meant as background information on data integration. The datasets in this work do not have a common interface, and thus must all go through an ETL process to be used. In stead of integration tooling, for maximum flexibility and compatibility with existing systems we choose manual data integration to create an integration pipeline covering the entire ETL process.

2.3 Reliability of Machines

The overarching theme of this thesis is finding underlying reasons why machines do not do what we want them to do. This is what *reliability engineering* tries to do: making sure machines work reliably the way we want them to. It is an "art" that requires knowledge from various fields, such as *tribology* (application of the principles of friction, lubrication and wear [33], necessary to know how the mechanical components of machines behave), *mechanics* (stress mechanics to find how forces act on materials [34], fatigue mechanics to find how cracks behave in materials [35]), the broader field of *material science* [36], and more [37]. There are multiple books on reliability engineering as a whole [38], some focusing on practical aspects [39], and some focusing more on the theory [40]. For an attempt at a summary paper of the field and its challenges, see [41].

A single machine is already a complex interaction of various components, each component potentially having different materials and properties that should be accounted for when creating the machine. An entire system of machines, such as STOREPICK, is *even more complex* as it not only wants the individual machines to behave as expected, but also the entire system as a whole. Since reliability engineering is such a large field, we choose to zoom in on three (3) ideas from it, from "narrow" to "wide": *failure detection* (Section 2.3.1), required to investigate *machine degradation* (Section 2.3.2), which is required knowledge for *maintenance scheduling* (Section 2.3.3).

The reason we structure this section as such is to illustrate the depth of required necessary knowledge for only a single item pertaining to reliability of machines. These three topics are **not** representative of all related literature simply due to the sheer size of all related fields. There are more items to making a robot arm work well, ranging from proper software (controller engineering) to the scheduling of (hard) computational heuristic tasks, as well as finding an optimal allowed time-frame for those tasks to run. Even including those fields would not suffice: there is a mismatch between the computed recipe (see Section 3.1 for an introduction to recipes, and Section 4.5 for the related dataset) and reality.

Note that in most of the cited sources that create a model for reliability – either to detect failure, measure degradation, or schedule (preventative) maintenance – the authors assume that there is available data on how well machines perform. This data, generally, is assumed to be in tabular format. It can be interesting to think of different models and potential strategies when in stead of a tabular format, data is presented in a graph database, such as done in this work.

2.3.1 Failure Detection

One of the basic requirements to finding out whether or not a machine is reliable, is to find when there is a machine failure. There are various approaches one can take for machine failure detection. First, in [42] a hidden Markov model [43] is used for two scenarios of machine failure: indistinguishable (for instance, a box of manufactured nails) and distinguishable production units (anything with a unique identifier, such as a palletizer cell in STOREPICK). Second, in [44] authors relate tribology to machine failure and good maintenance practise. In plain terms, they investigate how wear and tear of materials influence reliability of machines.

Specific to robot arms, in [45] machine failure of industrial robots is investigated using various statistical techniques, as well as machine learning techniques. The paper evaluates the advantages and disadvantages of each of the used method, as well as a combined new method titled hybrid gradient boosting. They propose that *local joint information* – information on a specific joint of a robot arm – is the main driver for failure detection. And, finally, in [46] authors propose a data-driven approach to *anomaly detection* for early detection of machine failure. They perform this approach on a designed robot arm. Various semi-supervised techniques are evaluated and compared in terms of their classification (fault vs non fault) performance.

> Failure detection can be approached from a mechanical perspective [44], a statistical perspective [42, 45], through machine learning [45], and anomaly detection [46].

2.3.2 Machine Degradation

Machine failure (Section 2.3.1 is when the machine completely stops working. Sometimes, however, we are interested in the complete **machine degradation** process. This can be useful for various reasons, for instance to find a proper timeslot for preventative maintenance (Section 2.3.3). Machine degradation effectively says that in stead of investigating a binary state of a machine ("working" or "failing") one should consider a variety of steps in between.

In [47] authors make the connection between reliability of machines, human interaction with those machines, and machine degradation. They argue that machine degradation and human interaction is not mutually exclusive – a human operational fault gives a shock to the system, accelerating degradation – and model this using a Semi-Markov process [48]. They show usability of their model on the turret of a lathe. Deep convolutional neural networks [49] are used on low-cost sensor data in [50] to estimate degradation in bandsaw machines. The setting of the paper is that, usually, non-high-end manufacturers of bandsaws cannot justify the high cost associated with blade wear monitoring solutions. As such, authors create first a model using data from the monitoring solutions, and then attempt to approximate this with low-cost sensor data using a neural network. They show that the neural network, while using data that is more cost-effective, has higher performance in reporting on degradation.

Machine degradation is arguably synonymous to estimating remaining useful life: if there is only 50% of useful life remaining, the machine has degraded to 50%. In [51] authors state that estimating remaining useful life is achieved through data acquisition, pre-processing and prognostics modelling, and that expert knowledge needs to be available to define a failure threshold. They say that prognostics is hard if expert knowledge is missing, since there are many potential states a machine can be in during degradation. Two new algorithms (Summation Wavelet Extreme Learning Machine and Subtractive-Maximum Entropy Fuzzy Clustering) are proposed to automatically identify the states of degradation, and dynamically assign a failure threshold. A tool for machine operators that is supposed to help making decisions on the current stage of degradation is presented in [52]. They generate a Cox's proportional hazard model [53] to estimate the survival function of the system, and then use support vector machines and time-series techniques for *forecasting* the remaining useful life. Their method is validated on a methane compressor, and the authors argue that their tool is a reliable tool for machine prognostics.

> Machine degradation can be investigated from various angles. It is related to machine failure and human interaction [47], is synonymous to estimating remaining useful life [51, 52], and consequently related to time series and forecasting [52]. Machine degradation, like many things, can also (quite successfully) be investigated using neural networks [50].

2.3.3 (Preventative) Maintenance Scheduling and Policies

When it is known what failure means for a machine (Section 2.3.1), and the way it degrades over time (Section 2.3.2), one can make schedules and policies for (preventative) maintenance. A machine maintenance policy is a document explaining when and how often what kind of maintenance is required. Preventative maintenance is maintenance done in order to prevent faults and issues. In [54] authors illustrate what machine maintenance policies should contain: performance of preventative maintenance measures, and reasoning about whether to repair or junk the machine having a fault. They present various "preventative and breakdown-repair" policies, containing reasoning according to a control-theoretic model on when which action(s) should be taken. In [55] authors assume a *Weibull* distribution (stemming from [56] where it was used to approximate the tensile strength of steel during fatigue testing) for machine failure and propose a schedule for maintenance using a genetic algorithm optimising both robustness and stability of the system. In [57] authors assume known "hazard rates" and use those to create a condition-based maintenance policy on a system-wide level, as opposed to looking at an individual machine as done in [55], minimising maintenance cost.

2.4 Previous Works in Vanderlande

As mentioned in Section 1, we are interested in specifically the `STO` errors that occur during palletization. Vanderlande's current approach to investigating these errors is *completely manual*: analysts must manually correlate an `STO` error to a particular LFL recipe, load the recipe into a visualisation tool, and compare the visualisation with the pictures and videos as taken by the STO camera (Section 3.2.1). Clearly, this is not particularly time efficient.

The first step to alleviate work from analysts is to understand the system. In [58] the complete warehousing system, we refer to it as STOREPICK (Section 3.1), is investigated by looking through a process mining lens. It illustrates the complexity of a system with *multiple case identifiers* relating to *multiple physical objects*. As the work is exploratory in nature, it does only help gain understanding of the system, but is not useful for finding `STO` causes.

Graph databases, and specifically Neo4J, have previously been used at Vanderlande. First, in [59] the physical layout of a baggage handling system (conveyor belts and more systems) in an airport, combined with process variants of how bags move through this system, is stored as a "routing database". The routing database can then be visualised using Bloom [60], vis.js [61], or similar graph visualisation tools, which is then used to see concretely unwanted process variants. Second, in [62] event knowledge graphs (from [13]) are used to model how tubs (containers carrying bags and suitcases) in a baggage handling system in airports move, to analyse their behaviour and performance. The tubs are part of the system, as opposed to bags that have a fixed entry and exit.

It is clear that Vanderlande has (sucessfully) used graph databases for different purposes [59, 62], and that they have investigated the STOREPICK system [58] before. In this work, we combine both ingredients, and attempt to analyse the STOREPICK system by using a graph database as data model.

3 Business Understanding

This section contains all information required to comprehend, in detail, the business problem we are trying to tackle. First, we further describe the context, that is, the STOREPICK system, in Section 3.1. Then, we zoom in on the palletizer cell and its relevant sub-components in Section 3.2. Finally, we state the main business question, and provide details for the four hypotheses (enumerated in Section 1.2) in Section 3.3.

3.1 STOREPICK Overview

As mentioned in Section 1.1, the context of the problem at hand is the STOREPICK system, which is depicted in Figure 1 in Section 1.1. The process that this system enables – note that here we describe a non-erroneous flow – is as follows:

1. When (new) products/items enter the STOREPICK system, they commonly arrive on pallets. These pallets are then stored in *pallet storage*.
2. If the product has not yet been entered into the database of the system, it gets sent to the *teaching station*, where various bits of information gets recorded (see Section 4.3 for the Teaching dataset).
3. A customer places an order. This order contains a list of desired products. The required products should be *depalletized*. Depalletization means taking the products from the pallet, and putting them onto trays with known sizes. The trays are used to move items throughout the entire system, and are stored in *tray storage*.
4. The LFL program attempts to find a solution for placing all required products onto the least possible number of pallets, while simultaneously maximizing efficiency and user-friendlyness when unpacking at location. It thus tries to heuristically solve a difficult optimisation problem (within a set duration), returning information on how all products are divided over pallets, and how each product should be moved to be placed at the desired position on the pallet, such that the end result is stable i.e. does not fall over during transportation. The result that LFL gives is called a *recipe* (see Section 4.5 for the LFL Recipes dataset).
5. When LFL has computed a recipe, possibly with a certain unknown error margin, it must be realised by a palletizer robot. To achieve this, trays with items are moved to a *palletizer* cell, where they are to be palletized according to a *recipe*. The palletizer robot follows the recipe, grabbing cases and placing them onto a pallet as it is instructed to do (see the StackInfo dataset in Section 4.4 for which information gets recorded during palletization).
6. After completing all pallets for a particular order, they get *marshalled* off to the customer, so the pallet finds its way to the customer.

Note that because LFL is heuristic, we cannot expect the solution as computed by LFL to be 100% “accurate” (we leave a potential definition of ‘accuracy’ out of the discussion as it is not relevant). Even if LFL uses internal *stability filters* to classify whether a computed recipe is stable enough, since it is unknown what a theoretically “correct” solution is, it is hard to argue how well this works. The “correct” solution that is given to the *palletizer cell* may thus not be “accurate” after all. On top of this likely existing error margin due to a heuristically computed recipe, there are compounding errors — for instance due to delayed maintenance of the arm it may have a deviation to the left — that can cause an item to be placed somewhere unexpected.

3.2 The palletizer cell

The part of the STOREPICK system that does palletization is its ACP module: the palletizer cell. The complete cell is shown in Figure 3. The process that the palletizer cell goes through is as described below.

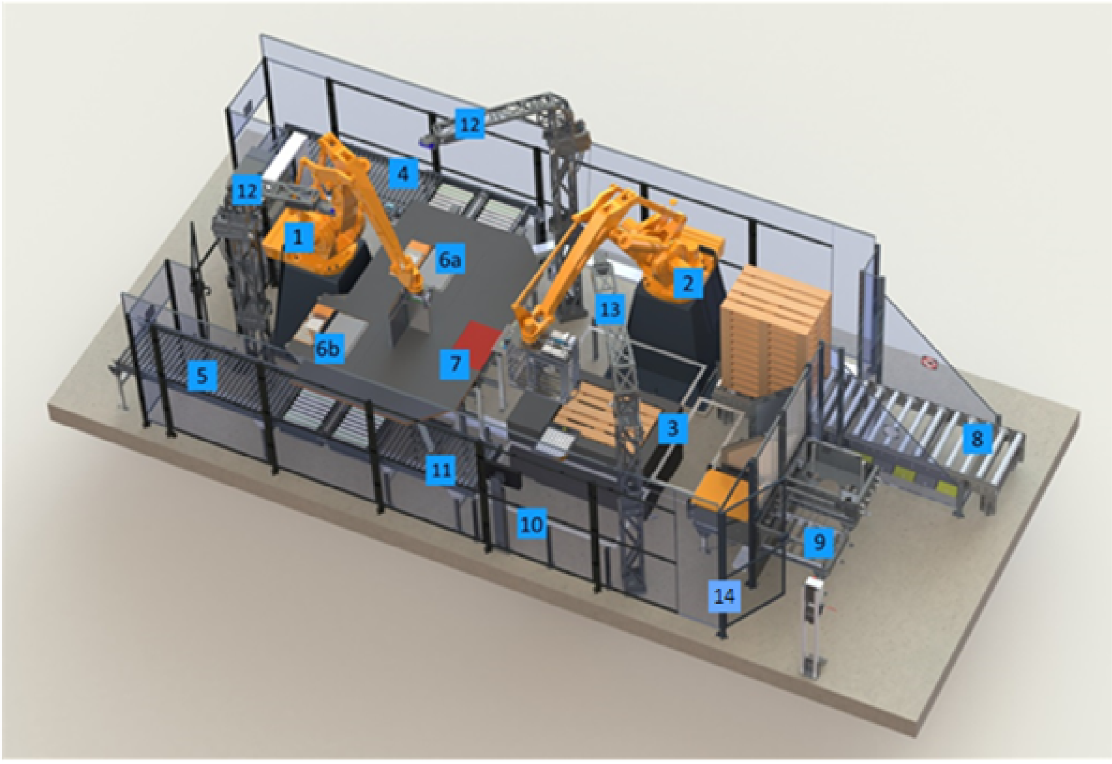

Figure 3: A single automatic palletizer cell, part of the STOREPICK evolution. The numbers refer to: 1. tray unload robot 2. **palletizer robot** 3. order load carrier lift (chimney) / pick-to position 4. tray infeed 5. tray outfeed 6. tray lifts 7. pick-from position 8. supply of stacks of empty pallets (in the pallet variant only) 9. supply of slip sheets (optional) 10. operator workplace (for exception handling) 11. manual unload position 12. tray pattern detection camera 13. **stack check camera** 14. access door with locks.

1. Trays with items enter the cell through the tray infeed (number 4).
2. The trays are moved towards the tray lifts (numbers 6a, 6b).
3. The lifts elevate the trays, so the tray unload robot (number 1) can grab the items from the tray, and push them towards the pick-from position (number 7, red area).
4. The empty trays leave the cell through the tray outfeed (number 5).
5. The palletizer robot (number 2) grabs the items on the pick-from position (number 7), and attempts to stack them onto a pallet at the pick-to position (number 3).
6. Every time the palletizer robot grabs an item, the stack check camera (number 13) takes a picture and evaluates whether or not this item was placed as desired.
7. When multiple items are placed onto the pallet, the lift lowers to make more space for new items.
8. At the end, the pallet is stacked, and removed from the cell.

We are interested in steps 5 and 6. In particular, we are interested in the **stack check camera** (Section 3.2.1) and the **palletizer robot** (Section 3.2.2).

3.2.1 Stack check camera

The *stack check camera* checks whether or not items are stacked on the pallet as desired, at every placed item. To decide whether or not a stack is as desired, it uses machine vision to locate where cases are, and compares its findings with the computed recipe from LFL. If there is a considerable difference between the placement as seen by the camera, and the placement from the recipe, then it is not as desired. In this case, the stack check camera raises an *STO error* (we refer to the stack check camera as the *STO camera* for this reason). **These STO errors are the errors we are interested in**. In particular, we are interested in *why* these errors occur.

3.2.2 Palletizer robot

The palletizer robot arm grabs cases and places them onto a pallet. It has grippers at the end of the arm, which are used to grap the case. Then, it moves the case according to the waypoints from the recipe, and releases the case at the release position. An illustration of (a simplified version of) how waypoints work is shown in Figure 4: first, the robot arm moves horizontally to waypoint 1 (red). Then, it moves downwards at angle to waypoint 2 (blue). Once there, it moves straight down until it reaches the release position (green), where it releases the case. The release position is always located at a fixed height under waypoint 2. The collection of all waypoints and positions is called the *flight path* of the robot arm.

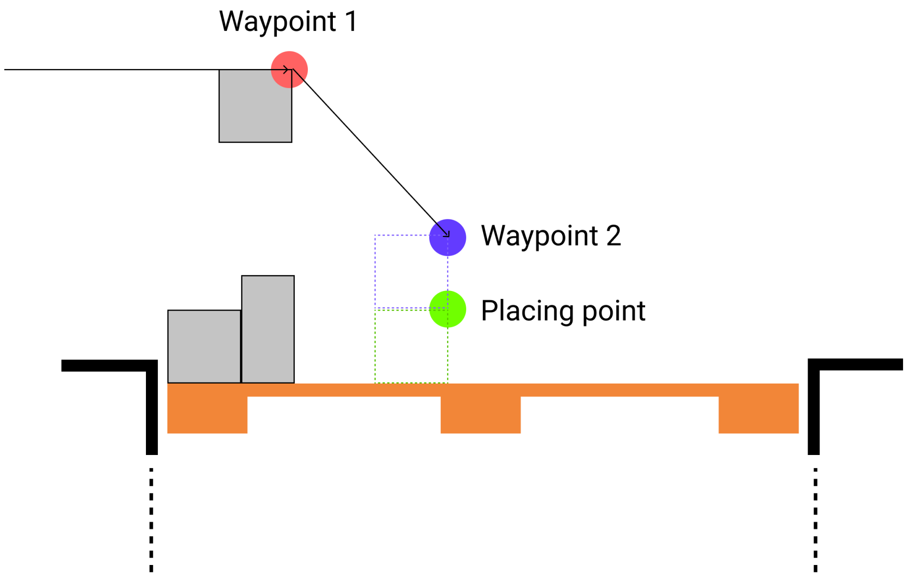

Figure 4: Illustration of robot arm and how it moves through waypoints.

3.2.3 Pick-to position: the pallet lift

The pallet itself rests upon a lift, as illustrated in Figure 5. During palletization the lift lowers the pallet so the cases can be stacked according to the recipe. At first, the pallet is supported by four "blocks" as shown on top of the illustration, but at a certain point there is a handover between the block-like support (blue) to a fork-like support (green), as indicated by the arrow. This handover between support structures is a potential cause for instability in the stack: the pallet itself and the cases stacked on it may shift in a direction, causing problems later in the stacking procedure.

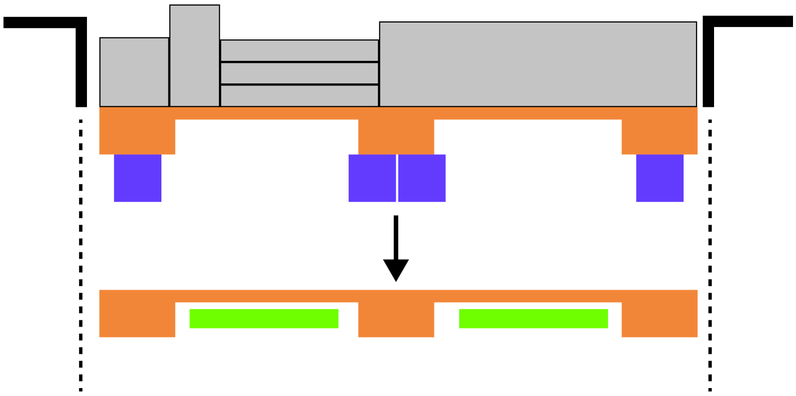

Figure 5: Illustration of the pick-to position: the pallet lift.

3.2.4 Palletizer Robot: Correction 1

The palletizer does not blindly follow the provided recipe. In order to combat potential mechanical issues, such as the handover between the support structure of the pallet as explained in Section 3.2.3, with help of the STO camera (Section 3.2.1) the palletizer may decide to shift waypoint 2 upwards if it decides that it is too low. This scenario is depicted in Figure 6: on the left, waypoint 2 and consequently the release position are too low for the case to be placed, lest it be crushed between the robot arm and pallet. To avoid this, the camera informs the system of a corrected waypoint 2 such that the case can be placed without issue.

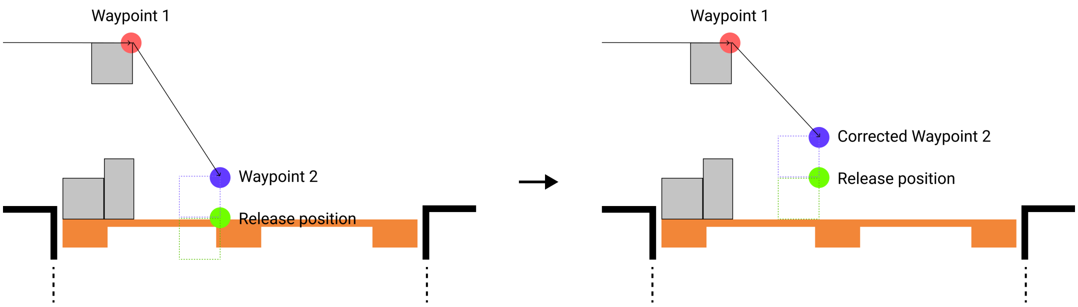

Figure 6: Illustration of the first type of local correction that the palletizer robot performs.

3.2.5 Palletizer Robot: Correction 2

Sometimes a case gets stuck on the edge of the lift shaft, blocking a next case from its normal route to be placed on the pallet. Similar to Correction 1 as shown in Section 3.2.4, with the help of the STO camera (Section 3.2.1) the palletizer may decide to shift waypoint 1 upwards if it sees that its flightpath is currently set in such a way that cases would hit each other. This scenario is depicted in Figure 7: on the left, there is a case stuck on the lift shaft which would prevent the orange case to be palletized according to its computed flight path, noted by the red cross. On the right of the illustration, one can see a corrected waypoint 1, resulting in a possible flightpath where the orange case can be moved and consequently palletized without issue, as indicated by the green checkmark.

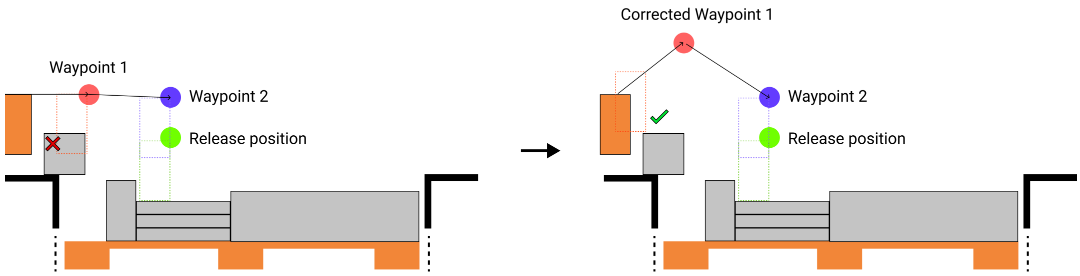

Figure 7: Illustration of the second type of local correction that the palletizer robot performs.

3.3 The Business Problem

In Sections 3.1 and 3.2 we explain the context of the problem. From this context, it is evident that palletization is a hard task to do well. Recall that a recipe is heuristically computed, and thus this recipe which is followed during palletization may have potential issues. There are possible mechanical problems, such as the lift handover as mentioned in Section 3.2.3, adding to the potential issues. Even more, we check "correctness" using machine vision, which might not be completely accurate. Another way to formulate this: there are **compounding errors**, which we believe is the underlying cause for STO issues. While the system already tries to correct for certain situations (Sections 3.2.4, 3.2.5), it is far from a solved problem.

In Section 1.2 we briefly mention the 4 hypotheses we have. These are further explained in Sections 3.3.1, 3.3.2, 3.3.3, and 3.3.4 for **HP** 1, **HP** 2, **HP** 3, and **HP** 4, respectively. Restating the business level problem that we try to answer: *"To what extent can we use a graph database to find underlying reasons for STO errors?"*

3.3.1 HP1: Incorrect placements cause more STOs.

We suspect that "incorrect placements" cause more STOs. In this phrase, incorrect placements are to be read as *a discrepancy between the placements as computed in the LFL recipe and the placements as recorded by the STO camera*. The reasoning as to why we decide to look into this possible underlying cause is due to the fact that it is the "logical" first thing to ask, given the context of the project: given some recipe that tells us how to stack, not doing so might be bad.

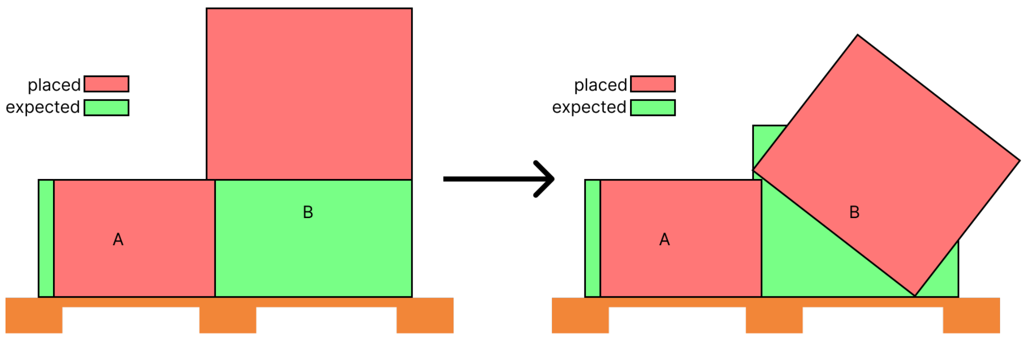

Figure 8: Illustration of incorrect placements: The left case (A) was placed in a different place than expected, causing the right case (B) to either crush (A) when placing, or being placed in a way that it falls down.

A possible result of incorrect placements is shown in Figure 8. On the left, one sees two cases (A) and (B) that are to be placed onto the load carrier according to the green boxes. If case (A) is, however, placed *incorrectly* a bit to the right, this means that case (B) can no longer be placed in its original space. If the robot were to attempt to place (B) on its original spot, either it will accidentally crush case (A) by pushing down onto the edge, or it will be placed onto this edge with no support anywhere else, and consequently fall down.

3.3.2 HP2: Building towers in the stack causes more STOs.

We suspect that "building towers" in a stack may be cause for STO errors. Here, building towers is to be interpreted as a stacking of various cases on top of one another, where the cases have different weights and sizes. The higher such a tower, intuitively, the less stable it may become and it may cause cases to fall over, which is undesirable.

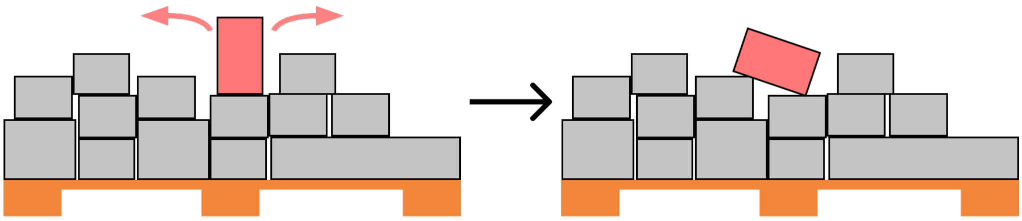

Figure 9: Illustration of a tower in the stack: The red case is built into a tower, without surrounding cases, causing it to possibly fall over to the left or right.

Figure 9 illustrates towers in a stack. On the left of the illustration, the red 'offender' case is placed onto two cases forming a tower. This possibly causes instability, allowing the case to, *during palletization*, tip over to the right or left, the latter shown on the right-hand side of the image.

3.3.3 HP3: Height gaps cause more STOs.

We suspect that height gaps cause more STOs. A height gap is created when some cases of various sizes fail to perfectly align with one another. These gaps may cause instability, causing cases to either move from their original position slightly, or in a worst case scenario fall down into the gap.

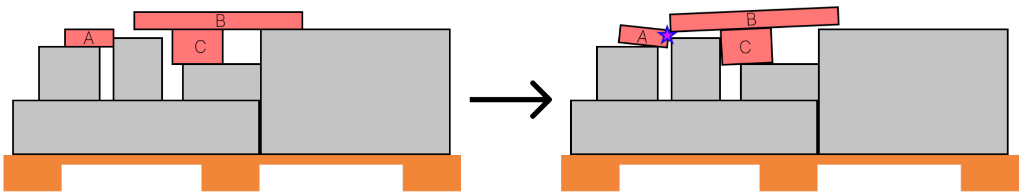

Figure 10: Illustration of height gaps in a stack: The red cases are placed in such a way that potentially they might shift due to height gaps in the stack. The purple-blue star indicates a possibility for cases (A) and (B) to fall into the gap completely.

An example of height gaps is shown in Figure 10. On the left is a hypothetical stacking of cases, where 'offender' cases are shown in red. These red cases are placed in such a way that there are (height) gaps in the stacking. On the right we show a potential cause: the cases move from their original position, ruining any future cases from reaching their desired position. Even more, as indicated by the purple-blue star, it could happen that due to the weight of case (B) both cases (A) and (B) fall down into the gap.

3.3.4 HP4: Overhang causes more STOs.

We suspect that "overhang" of cases cause more STOs. Since *overhang* may be interpreted to be between two cases, we clarify that here overhang is defined as *overhang of a particular case* *with respect to the load carrier*. If for this particular case its edges are protruding further than the boundaries of the load carrier, whether or not this was as computed beforehand, we say that there is overhang.

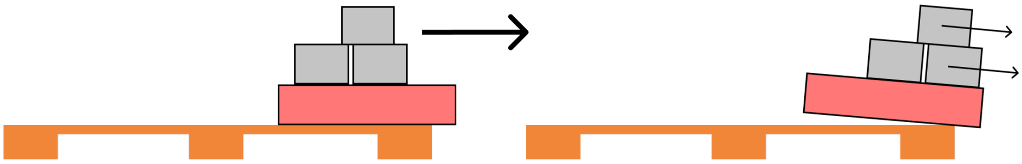

Figure 11: Illustration of overhang: Due to the red case being placed with considerable overhang, there is a chance that (during palletization) it and the cases on top of it shift off the pallet.

In Figure 11 we show a illustration of overhang. The right side of the red cases hangs over the load carrier: it has overhang. The illustration shows a possible cause of overhang, namely, that cases might shift due to the weight. Other possible consequences of overhang include that overhanging cases may break during transportation (the *marshalling* step from Figure 1) or that overhanging cases may cause the stack to get stuck when moving through tight areas.

4 Dataset Descriptions

This section explains the various datasets used during the thesis in Sections 4.1 through 4.5. Each of these subsections contains information on the particular *fields* or *columns* the dataset contains, noted using *cursive text*. Valuations are noted using the `typewriter font`. Besides information on what the dataset contains, we explain the extraction and transformation steps (if applicable - see Section 2.2 on ETL). For each dataset description we list data quality issues, and potential annoyances when it comes to the extraction/transformation of the data.

> ***Each dataset is retrieved for a 7 day long time-frame in December 2021, for*** 20 ***palletizer cells.***

In each section describing a dataset, we will briefly link to a carefully constructed toy example that will be used throughout the remainder of the thesis to explain Data Integration (Section 5) and the data model (Section 6). The toy example is for a hypothetical STOREPICK system using two palletizer cells (Section 3.2), where there are two orders. Order 1 desires only a single pallet, and order 2 desires two pallets. Figure 12 shows these pallets. Note that while the toy example has been carefully constructed, there may be some inconsistencies between explained text and the specific values - in such case the *explained text is always leading*.

4.1 SCADA

This dataset comes from the SCADA system — SCADA is a control system architecture consisting of amongst others graphical user interfaces for high-level supervision of machines and processes — in place at a specific STOREPICK installation. The SCADA system reports on *many* kinds of errors for all parts of STOREPICK, so not just for palletization. Figure 13 shows the important columns that the SCADA dataset contains on the left: each line in the dataset corresponds to some error, identified by its *error_type*. It has a starting time (*start_time*), textual error description (*error_id*), a reference to a part of the system where the error occurred (*error_part*), and potentially a *duration* and ending time (*end_time*).

There are also fields indicating how severe an error is, and whether or not the error is technical or operational in nature, but since we are only interested in a single *error_type*, we can safely ignore them, as they are identical per *error_type*. The single *error_type* we are interested in is `STO`. This dataset is the "starting point": it lists all STO errors (see Section 3.2.1) that occur.

Extract
In Table 8 (located in Appendix A.1) we show what (an anonymised version of) the SCADA dataset looks like after extraction. Some column names have been renamed, as indicated by the cursive font. Various values have been replaced with a representative placeholder. Numerical values have been rounded so the table can be displayed on a single page. For displaying reasons, the *technical*, *operational*, and *severity* columns are omitted. Notice how there are many entries (in this example *all of them*) that do not correspond to the particular error type we are interested in (`STO`).

> Before any filtering, there are a total of 112, 728 entries. The CSV file itself is approximately 16 MB in size. Out of all entries, only 2, 425 are related to the `STO` error we are interested in. This means that we investigate 2.15% of *all* errors in the STOREPICK system.

Transform
A transformed version of Table 8 (located in Appendix A.1) corresponding with the illustration in Figure 13 is shown in Table 1. We omit the *technical*, *operational*, *severity*, and *error_id* columns, as these are constant for the transformed dataset. For further displaying and referencing

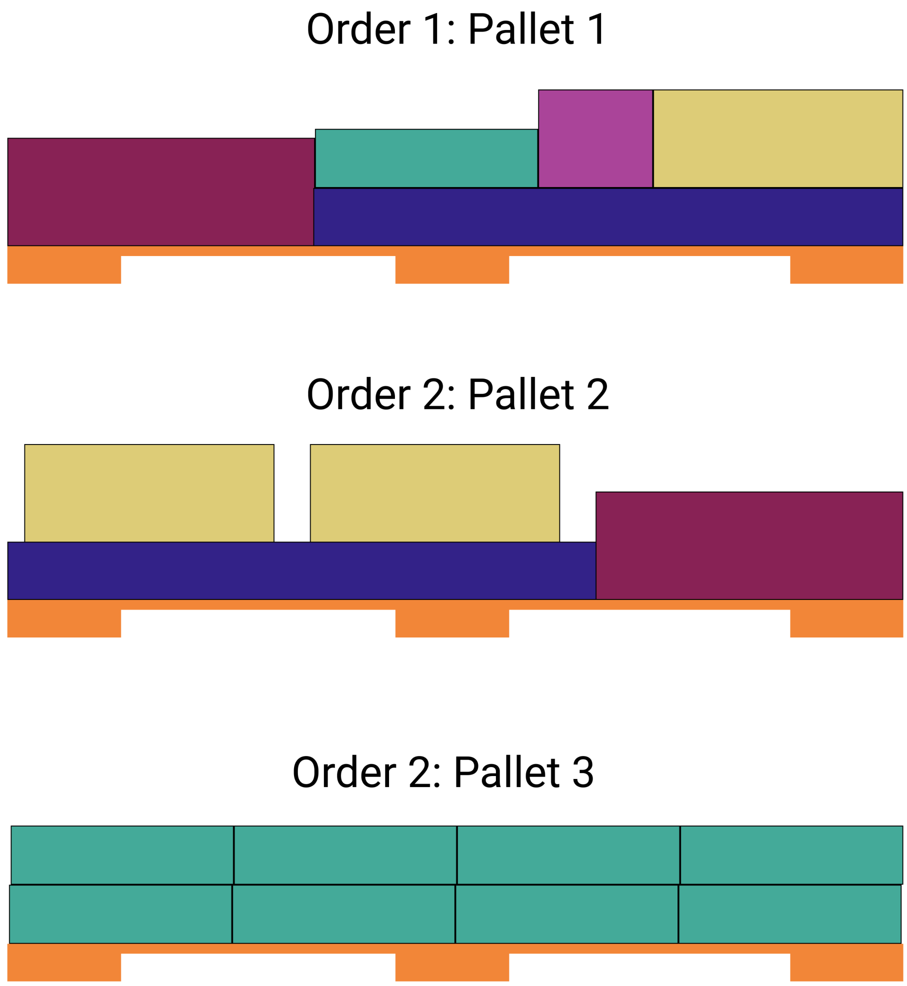

Figure 12: Pallets in the toy example.

convenience, we add a *row* indicator (not present in the dataset). Note that Table 1 is the toy example, and does not contain real data. Also note that timestamps after transformation are stored in *unix time*, but this would hinder understanding when displayed on paper, and as such timestamps are displayed according to their actual value.

row	**start_time**	**duration**	**end_time**	**error_id**	**error_part**
1	2021-12-10 11:58:12.890+1100			blocked_lift_shaft	1014.56.78
2	2021-12-10 11:56:52.816+1100	00:02:18.633	2021-12-10 11:59:11.449+1100	blocked_place_position	1014.56.78
3	2021-12-10 11:56:52.816+1100	00:01:45.567	2021-12-10 11:58:38.383+1100	missing_stack_surface	1024.56.78

Table 1: Toy example of the SCADA dataset after transformation.

Data Extraction Annoyances / Quality Issues
Exporting the SCADA dataset from the system to a CSV file is done by selecting a desired time-frame, and clicking an export button. An example of the "raw" data as exported is shown in Table 8 in Appendix A.1. In this raw data, the first three rows show that the *end_time* of some error is not contained in the selected time-frame, resulting in the related field to contain a textual message ("No end time within search window"), and the *duration* field to simply contain nothing. A workaround is to export a larger dataset, for a longer period of time, and then programmatically checking if the selected period of time is long enough to include all desired *end_times*. This step might need to be executed multiple times to find a correct period of time. Also, naturally, unwanted entries due to the longer time selection should be removed.

Another annoyance, is the fact that there is *no timezone information* in the data. When trying to combine this dataset with others, each potential timezone must be manually checked to find the required offset for a timestamp with a known timezone: it is quite a task to figure out which timezone the timestamps of each dataset is recorded in. As workaround, in an undisclosed online environment used by Vanderlande, one can change the timezone of the account to a known value (such as UTC), and re-export the dataset. Then, based on the differences in *start_time*, the original timezone can be rediscovered. In this specific case, this timezone is *UTC+11*, as indicated by the `+1100` in the timestamps in Table 1.

In terms of nice properties such as uniqueness and keys, we find that in the transformed toy dataset (Table 1) on row 1 there is no information for *duration* and *end_time*. This is due to the aforementioned reason that it is not contained in the search window. Another point to notice, perhaps most interesting, is that errors in rows 2 and 3 have *identical* starting times, but different reasons (*error_id*) for the error. Multiple *error_ids* may be set at the same time, and timestamps are not unique, nor is a combination of a timestamp and the part that caused the error (*error_part*).

4.2 Telegrams

The Telegram dataset comes from logging produced by programmable logic circuits (PLCs) within the STOREPICK system produce. A PLC, in this instance, is a tiny computer with sensors, that logs data every time a sensor sees something. These logging messages are called *Telegrams* within Vanderlande, hence the name of the dataset. There are *many* different types of Telegrams, but the dataset that is used in this thesis has been filtered down specifically to only `STO` related Telegrams during *extraction*. Figure 13 shows the important columns that the Telegram dataset contains on the right: *time* represents the timestamp of a single Telegram message, *error_part* is a reference to the related part of the system where the Telegram was generated, *pallet_id* represents the pallet which was being stacked at the time this Telegram was generated, and finally there are four Boolean variables indicating if there is a type of error related to the Telegram.

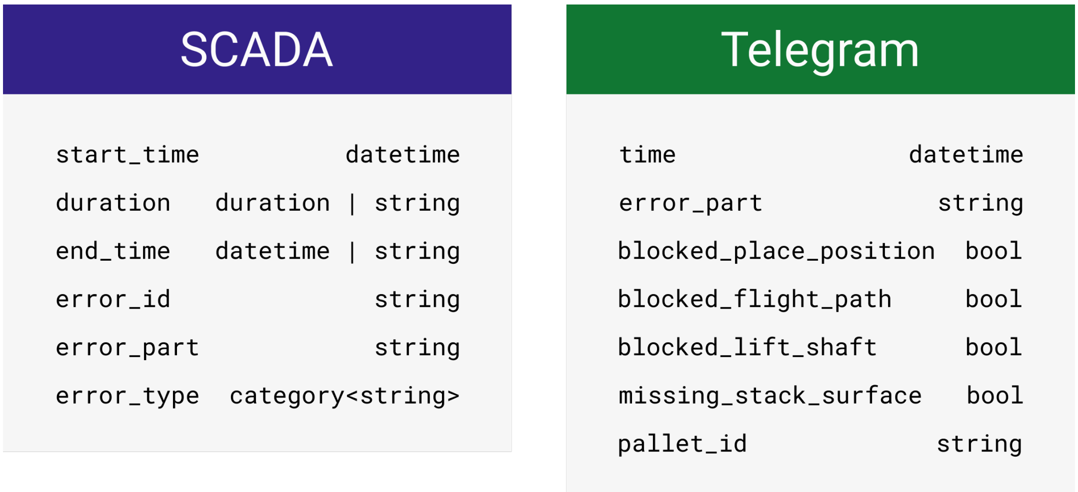

Figure 13: The SCADA (left, blue) and Telegram (right, green) datasets.

The possible error types are *blocked_place_position* (the original position where the current case is to be placed is blocked), *blocked_flight_path* (the flight path that the robot arm is instructed to follow to place the current case is blocked), *blocked_lift_shaft* (there are issues in the lift shaft causing the current case to be unable to be stacked) and *missing_stack_surface* (the case itself is missing, and thus cannot be stacked).

Extract

Extracting the dataset from the system with a query that filters solely on Telegram messages related to `STO`s gives us, besides a timestamp with timezone information and a correctly extracted *error_part* field, an unparsed singular field *data*. An example is shown below (manually added space after `Blocked_Place_Position=FALSE____________` for readability).

```
______Blocked_Flight_Path=FALSE________Blocked_Place_Position=FALSE____________
↪  Blocked_Lift_Shaft=FALSE_________Missing_Stack_Surface=FALSE___________Pallet_TSU_ID=  1
```

Transform

Clearly, an unparsed singular *data* field is not particularly useful. We transform it so we get the four Booleans (*blocked_place_position*, *blocked_flight_path*, *blocked_lift_shaft*, and *missing_stack_surface*) and the *pallet_id*. This is done by splitting on the equals sign (`=`), selecting an element after splitting (`[0]` for the first element, `[1]` for the second element, ...), and then splitting again on an underscore (`_`) to extract the `TRUE` or `FALSE` value. By testing if said extracted value is exactly equal to `TRUE`, we set the Boolean variables to `True` when it is `TRUE`, and to `False` when it is `FALSE`. For the *pallet_id* field we only cast the result after splitting on equality to an integer. Python code implementing this idea is shown below, where `df["data"]` selects the *data* field from a pandas DataFrame[1].

```
df["blocked_flight_path"] = df["data"].apply(
    lambda data: data.split("=")[1].split("_")[0] == "TRUE"
)
df["blocked_place_position"] = df["data"].apply(
    lambda data: data.split("=")[2].split("_")[0] == "TRUE"
```

[1] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

```
)
df["blocked_lift_shaft"] = df["data"].apply(
    lambda data: data.split("=")[3].split("_")[0] == "TRUE"
)
df["missing_stack_surface"] = df["data"].apply(
    lambda data: data.split("=")[4].split("_")[0] == "TRUE"
)
df["pallet_id"] = df["data"].apply(lambda data: int(data.split("=")[5]))
```

Data Extraction Annoyances / Quality Issues

As illustrated above, the Telegram messages are incorrectly parsed by the system itself. This means that if we export to CSV in a similar fashion to the SCADA dataset, the resulting CSV only contains a single *data* column, which requires parsing. Writing such parsing code is not particularly difficult due to the consistency of the *data* field, but it does add more potential points of failure.

> There are a total of 7 CSV files, with on average 221,865 messages per day. The complete dataset counts 1,553,053 Telegram messages.

Besides having to manually parse the field, it is important to realise that within the *data* column, the *pallet_id* field is stored with leading spaces, which should be taken into account. It should also be noted that exports are limited on a time-frame per day, since larger time-frames give a timeout.

time	**error_part**	**blocked _place _position**	**blocked _flight _path**	**blocked _lift _shaft**	**missing _stack _surface**	**pallet_id**
2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	TRUE	FALSE	pallet3
2021-12-10 00:56:52.816+0000	1014.56.82	FALSE	FALSE	TRUE	FALSE	pallet1
2021-12-10 00:56:52.816+0000	1024.56.82	TRUE	FALSE	FALSE	FALSE	pallet2

Table 2: Subset of 3 out of 21 lines of the toy example (Table 9) of the Telegram dataset after transformation.

Table 2 shows a subset of the Telegram data for the toy example (complete toy example in Table 9 in Appendix A.1). Lines where at least one Boolean variable is set to `TRUE` have been coloured green. Notice how in many of the rows in the full toy example (Table 9, Appendix A.1), none of the Boolean variables are set to `TRUE`: this is because a single Telegram message is generated *at least once for every case that is placed*. We say *at least once*, since there may be *retries* when the system decides the first message was not sent properly. These retries are a particularly annoying issue when integrating data, see Section 5.2.

4.3 Teaching

The teaching dataset corresponds to all data that is learned at the teaching station (see Figure 1). This dataset contains *many* fields, all related to physical properties of the case and particular metadata, such as a barcode number and a timestamp on when this item was first received in the system. Properties like *width*, *height*, *length*, and *weight* are some of the fields in this dataset. Each row of the teaching dataset corresponds to a single case, uniquely identified by *case_id*. For consistency, on the left of Figure 14 we shows (some of) these fields, similar to the other sections describing datasets.

Extract, Transform, Data Extraction Annoyances / Quality Issues
The Teaching dataset is extracted by Vanderlande operators on site. It is *not* available in any online environment as opposed to the SCADA and Telegram datasets. This makes getting a new dataset *very slow*, as it rests upon the operators to extract it for us. In an ideal situation, all teaching data should be provided in the same environment as other datasets.

case_id	weight (kg)	width (mm)	height (mm)	length (mm)
1	16	279	110	1000
2	3	128	110	1000
3	40	660	65	1000
4	24	250	66	1000
5	18	343	121	1000

Table 3: Toy example of the Telegram dataset after transformation.

Besides having no control over exporting, the dataset itself is simply *big* and *largely undocumented*. Considerable time has been spent understanding this dataset, and what all fields actually mean. A (very small) subset of the fields from the Teaching dataset corresponding to those in Figure 14 is shown in Table 3, the toy example, where each entry has been colour coded according to the colours used in Figure 12.

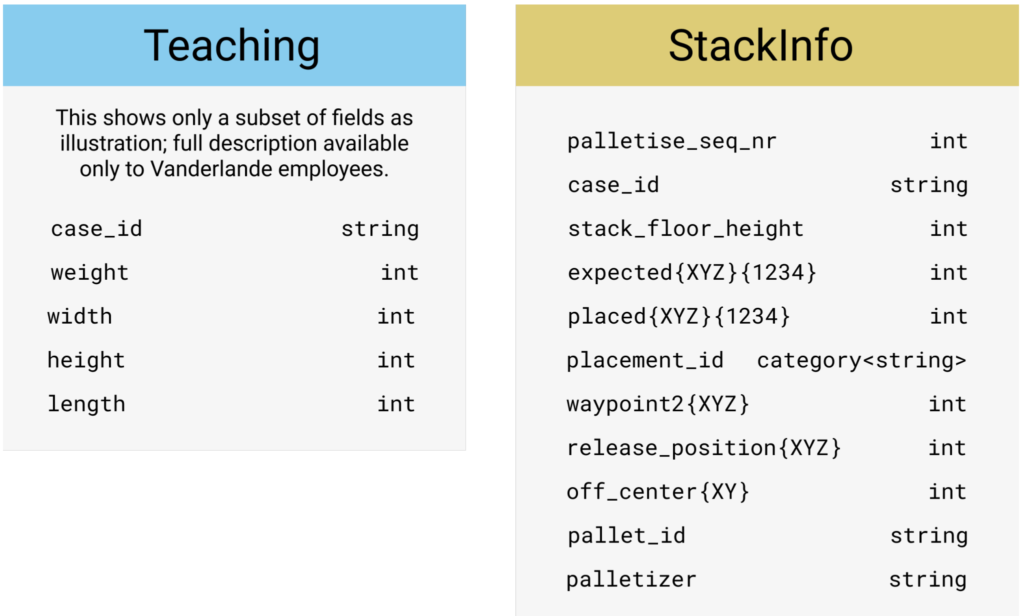

Figure 14: The Teaching (left, light blue) and StackInfo (right, yellow) datasets.

4.4 StackInfo

In this dataset one can find, per *palletizer*, per date, and per pallet (*pallet_id*), the order of cases that were stacked on a pallet as executed by the palletizer (indicated by *palletise_seq_nr*). Besides this order, for each placed case, it lists the expected placements (those as computed by LFL, indicated by *expected*{*XYZ*}{*1234*}), as well as the *actual* placements (indicated by *placed*{*XYZ*}{*1234*}). When we write *expected*{*XYZ*}{*1234*}, this should be expanded to all combinations: *expectedX1*, *expectedX2*, *expectedX3*, *expectedX4*, *expectedY1*, ..., *expectedZ4*. This expansion should be done for any variable where we use {} in its name. These placements (*expected*{*XYZ*}{*1234*} and *placed*{*XYZ*}{*1234*}) effectively are four three-dimensional points. It also has information on how the robot arm is supposed to move (indicated by *waypoint2*{*XYZ*}; see Section 3.2.2), and where it should release the case (indicated by *release_position*{*XYZ*}; see Section 3.2.2). Finally, there is available data on the difference between the expected and placed centre points of the case (*off_center*{*XY*}).

palletise _seq_nr	case _id	stack _floor _height	offCenterX	offCenterY	pallet_id	palletizer
0		0	−534	−555	pallet1	ACP1
1	5	0	10	9	pallet1	ACP1
3	3	0	−3	10	pallet1	ACP1
4	4	0	−4	−2	pallet1	ACP1
5	2	0	−8	6	pallet1	ACP1
6	1	0	10	10	pallet1	ACP1
0		0	770	776	pallet2	ACP2
1	3	0	−1	1	pallet2	ACP2
2	1	0	−3	4	pallet2	ACP2
3	1	0	−10	−1	pallet2	ACP2
3	5	0	1	−8	pallet2	ACP2
0		0	−460	826	pallet3	ACP1
1	4	0	−6	−9	pallet3	ACP1
2	4	0	7	10	pallet3	ACP1
2	4	0	3	−7	pallet3	ACP1
4	4	0	1	10	pallet3	ACP1
5	4	66	8	2	pallet3	ACP1
7	4	66	1	−5	pallet3	ACP1
8	4	66	3	2	pallet3	ACP1
9	4	66	2	−1	pallet3	ACP1

Table 4: Toy example of the StackInfo dataset after transformation, omitting various fields.

Extract

Similar to the Teaching dataset, the StackInfo dataset is *not* available for export. In fact, by default this data is not even recorded. It must be turned on (for instance during planned maintenance) on a per-pallet-cell basis. This means that for each pallet-cell an operator has to dive into the software running the cell, and enable a logging flag. Once turned on, the logs must manually be extracted (again, on a per-pallet-cell basis). In an ideal situation, the logs is available through the same environment as the SCADA and Telegram datasets are.

Transform

The extracted StackInfo dataset is deeply nested, and some information is only available in the path and filenames. An example is shown below.

```
DATA_ROOT/Stackinfo/ACP{XX}/{DDMMYYYY}/StackInfo/{YYYYMMDD}_{HHMMSS}___palletID.csv
```

To elaborate:

- For each palletizer cell, there is a separate folder (ACP07, ACP08, ...).
- In those folders, all stacks are separated into other folders, organised by date `DDMMYYYY` (02122021, 03122021, ...).

- The date folder contains a single subfolder named `StackInfo`, which contains all CSV files.
- A single CSV file contains a timestamp and a pallet ID in its filename.

Transforming to a usable data format is relatively straightforward (loop through folders, read CSV into a DataFrame, concatenate DataFrames), and is deemed not interesting to show. What we will show is the final result after transformation for the toy example in Table 4. We omit all points (*expected*{*XYZ*}{*1234*}, *placed*{*XYZ*}{*1234*}, *waypoint_2*{*XYZ*}, *release_position*{*XYZ*}) so we can fit the table in a page. Later, in Section 7, we will introduce some of these values when necessary. Note how entries are colour coded, similar to the toy example for the Teaching dataset (Table 3), to correspond with the colours in Figure 12.

Data Extraction Annoyances / Quality Issues
Important to note is that *expectedX1* does **NOT** correspond with *placedX1*. Instead, it follows following scheme:

- The *expectedX1* maps to the *placedX2* field: *expectedX1* → *placedX2*.
- The *expectedX2* maps to the *placedX3* field: *expectedX2* → *placedX3*.
- The *expectedX3* maps to the *placedX4* field: *expectedX3* → *placedX4*.
- The *expectedX4* maps to the *placedX1* field: *expectedX4* → *placedX1*.

This issue is identical for *expectedY*□ and *expectedZ*□ fields. Another point to note is that the *off_center* fields (computed as some difference between placed and expected locations) are also provided for load carriers (pallets). In such case, however, it remains unclear what this means precisely. Finally, note that the given *case_id* has a fixed length: there are leading spaces (not shown in toy example in Table 4 as all *case_id*s are of the same length), which should be taken into account when loading the dataset.

Another important note to make is that the *palletise_seq_nr* field is *not* a simple increasing count (which is what would be expected, as it denotes the sequence used to palletize cases). Sometimes numbers are skipped (in the toy example this happens for `pallet1`: the *palletise_seq_nr* skips from 1 to 3, (wrongly) indicating that the second case was missed), sometimes duplicated (in the toy example this happens for `pallet2`: the *palletise_seq_nr* field contains two rows where it is equal to 3, (wrongly) indicating that this case was placed twice), and sometimes both at the same time in the same pallet (in the toy example this happens for `pallet3`: number 2 was seemingly placed twice, and number 3 is seemingly missing). It would be interesting to investigate the underlying causes of this phenomenon in another project.

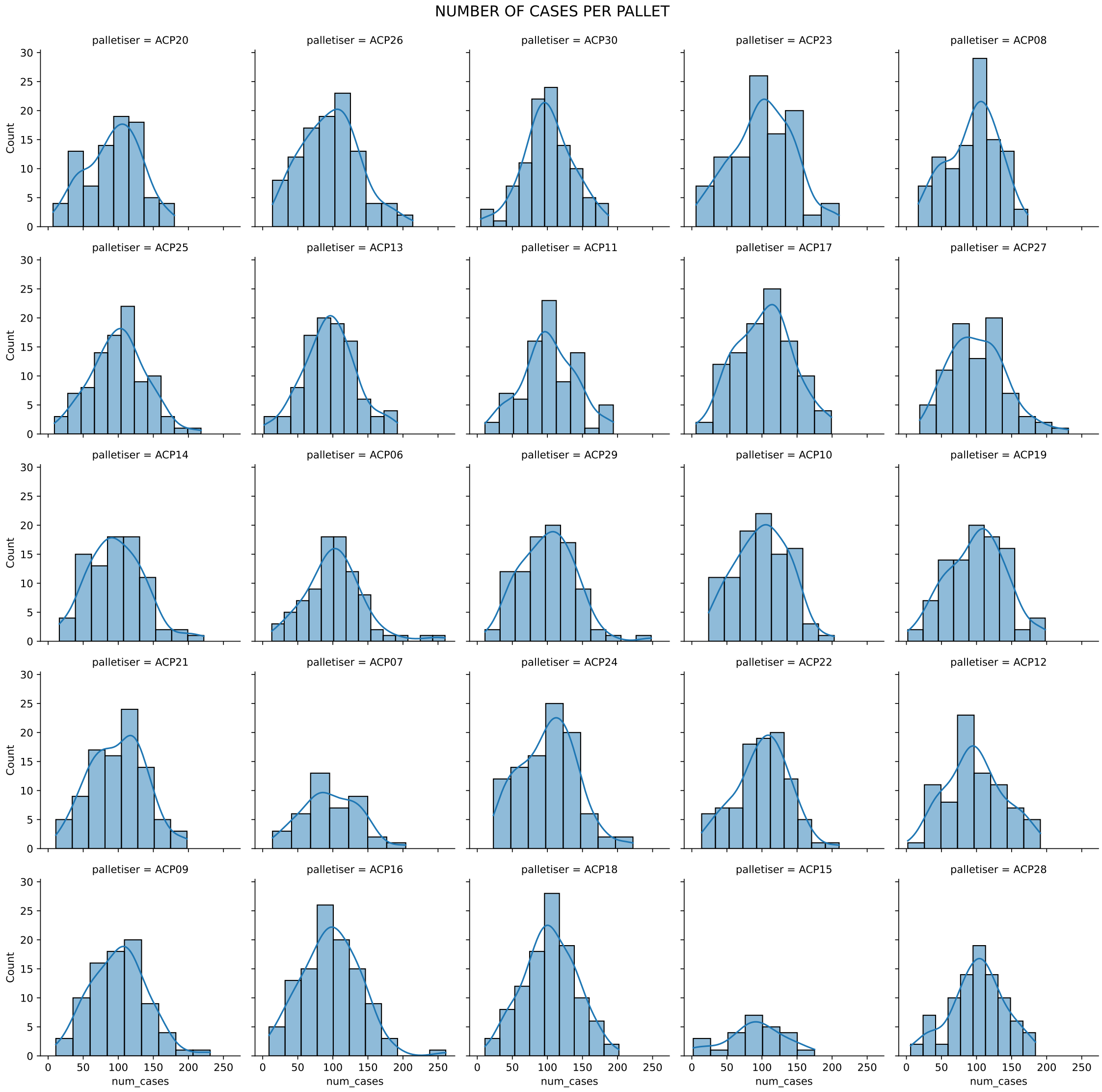

Figure 15: Number of cases per pallet, per palletizer cell.

In the dataset (271 MB in size) a total of 12,562 pallets are stacked. On average per day this equates to 1,795. Per palletizer, the average number of pallets is 94. Figure 15 shows the number of cases per pallet per palletizer. Based on this plot, it seems that most pallets have around 100 cases. Interestingly, palletizer cells `ACP07` and `ACP15` have stacked considerably less cases per pallets than other cells.

4.5 LFL Recipes

The LFL Recipes dataset contains all information that LFL used to compute how pallets are to be stacked. There is a lot of information available, so to keep things understandable, in Figure 16, we show only the *transformed* dataset. The *order_id* refers to the ID of the entire order, consisting of multiple suborders, each with their own *suborder_id*. A suborder relates directly with a single pallet. For each pallet, we have the height (*stack_height*), weight (*stack_weight*), used volume (*stack_volume*), the number of cases (*nr_cases_in_stack*), a *fillrate* indicating how well this pallet is filled, and coherence measures (*article_coherence*, *group_coherence*) that LFL uses when computing the recipe. For each case that is in the suborder, we have its id (*case_id*), the *sequence_id* indicating in which order cases are to be stacked, the *stacking_method* indicating what method is used to stack the cases (the *stacking_method* field describes if LFL used a particular heuristic to compute how to stack the case e.g. a *Tower* or *Layer* heuristic), and the *completed_height*.

LFL Recipe			
order_id	string	stack_height	int
suborder_id	string	stack_weight	int
recipe_id	string	stack_volume	int
nr_cases_in_stack	int	fillrate	float
article_coherence	float	group_coherence	float
case_id	string	completed_height	int
sequence_id	int	stacking_method	string

Figure 16: The LFL Recipes dataset.

Extract & Transform

The LFL dataset itself is extracted from the system by Vanderlande engineers. This information is not available in the online environment. The LFL recipes are delivered as a ZIP folder. In this ZIP, there are four (4) folders, each containing a single XML file. There is a lot of information in these files, ranging from a list of all required products for this order, metadata for said products, an ordering of movements of the robot arm to stack these products, and the entire computation process to get to its end result.

order _id	suborder _id	stack _height	stack _weight	nr_cases _in_stack	sequence _id	case _id	stacking _method
order1	suborder1	175	101	5	1	5	Idea1;MethodA
order1	suborder1	175	101	5	1	3	Idea1;MethodA
order1	suborder1	175	101	5	2	1	Idea1;MethodB
order1	suborder1	175	101	5	2	2	Idea1;MethodB
order1	suborder1	175	101	5	2	4	Idea1;MethodB
order2	suborder2	175	90	4	1	5	Idea2;MethodA; Rotated
order2	suborder2	175	90	4	1	3	Idea2;MethodB
order2	suborder2	175	90	4	2	1	Idea1;MethodA; Rotated
order2	suborder2	175	90	4	2	1	Idea1;MethodB; Rotated
order2	suborder3	132	192	8	1	4	MethodC
order2	suborder3	132	192	8	1	4	MethodC
order2	suborder3	132	192	8	1	4	MethodC
order2	suborder3	132	192	8	1	4	MethodC
order2	suborder3	132	192	8	2	4	MethodC
order2	suborder3	132	192	8	2	4	MethodC
order2	suborder3	132	192	8	2	4	MethodC
order2	suborder3	132	192	8	2	4	MethodC

Table 5: LFL Recipes dataset for the toy example.

We use a C# script (as opposed to python) to transform the delivered ZIP files into CSV fields with the fields as listed in Figure 16. The reason why C# is used, is because it can directly use available tooling made by Vanderlande engineers. This way, there is no longer a need to *manually* parse XML files, as this is done by the tooling. We simply have to select the desired pieces of information according to the way it is stored, and then export to CSV. This is done in batches of 50 recipes, since more than that takes too long.

> There are a total of 2,745 ZIP files taking up a total of 8.7 GB. After processing with the C# script this is reduced to 55 CSV files totalling 335 MB.

Data Extraction Annoyances / Quality Issues

As mentioned, the dataset is delivered as XML in a ZIP. This is quite annoying to manually parse, but we do not need to thanks to existing tooling available in C#. Table 5 shows a subset of the fields from Figure 16 for the toy example. Fields have been left out so the Table can be displayed on a single page. For confidentiality reasons, the *stacking_method* field contains placeholder names. One can see that there are a total of 2 orders for the dataset (`order1` and `order2` are the only values in the *order_id* column), with a total of 3 suborders. This corresponds to the three pallets shown in Figure 12. It should be noted that the *stacking _method* field contains a list of relevant information separated by a semicolon (`;`): there are various global ideas that LFL employs on how a case is meant to be placed, and within those ideas there are various methods to achieve them. Sometimes, no idea is given, and sometimes we have an indication that the case to be placed is *rotated* when placing it: cases have a default rotation when LFL computes the recipe, and this rotation means that for this particular recipe the case is no longer in this default rotation. Finally, the Table shows that some attributes are on the pallet-level, and some on case-level, precisely as mentioned in the introductory paragraph of this dataset.

5 Data Integration

This section shows how all datasets from Section 4 are integrated. The integration pipeline is manual, as we are focused first and foremost on the question if we can even integrate the data at all. As it turns out, there are *two missing links*. Figure 17 shows how all datasets are linked together. There, on the right side, there are two linking datasets not described in Section 4; these are the two missing links that were necessary for combining all datasets into one. The Figure also indicates which combination can be found in which section: the SCADA dataset (from Section 4.1) with the Telegram dataset (from Section 4.2) is linked in Section 5.1. The result of this combination is linked with the StackInfo dataset (from Section 4.4) in Section 5.2. The third join to compute is adding the LFL recipes (from Section 4.5), which requires the two linking datasets. This is explained in Section 5.3. The final link to be made is with the Teaching dataset (from Section 4.3), which is shown in Section 5.4. For each link to be made, we discuss data quality issues (if applicable).

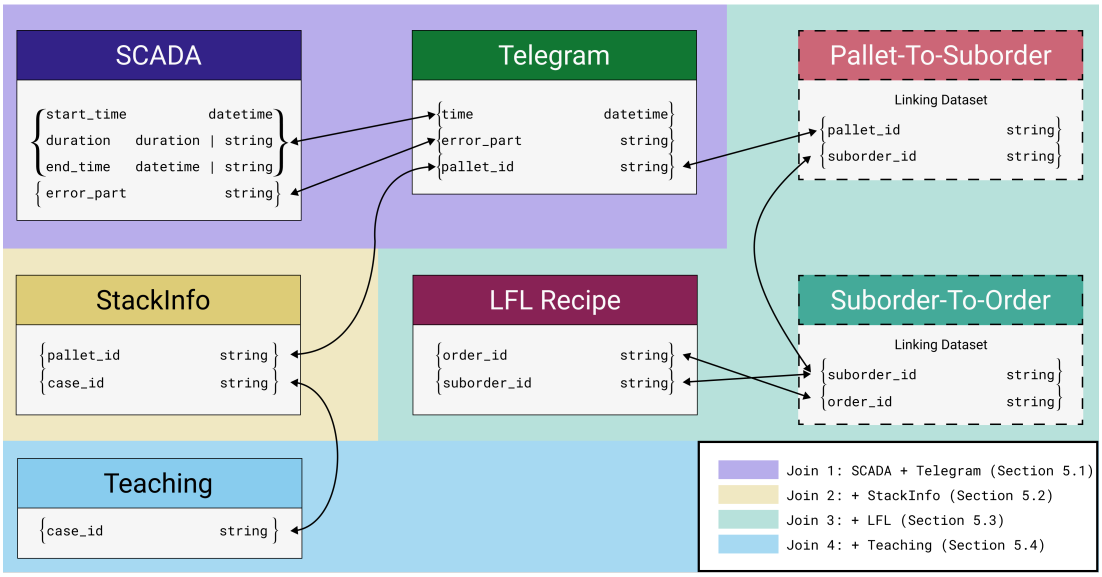

Figure 17: Data integration overview illustration.

5.1 Join 1: SCADA + Telegram

The SCADA dataset is the starting point of the integration pipeline, as this dataset contains information on the `STO` errors. Recall that it has *start_time* (s) and *end_time* (e) fields: for a single `STO` error, it gives an interval $[s, e]$. The Telegrams give a timestamp *time* (t). Both datasets also have a particular *error_part* (p_s for SCADA, p_t for Telegram). An entry from the SCADA dataset is related to an entry from the Telegram dataset if $(p_s = p_t) \wedge (t \in [s, e])$. The first part $(p_s = p_t)$ is straightforward, but the second part $(t \in [s, e])$ causes issues: each row in the SCADA dataset may correspond with n Telegram messages.

The only way to find out which ones are supposed to be linked together, is by looking at the timestamps. **Assuming identical timezones and no time-drift**, and to write somewhat efficient linking code, this means that one would have to *loop through time*, and match to SCADA and Telegrams respectively, after which a mapping is created between a SCADA entry → Telegram message. The lowest granularity of the *time* field in the Telegram dataset (as recorded - potentially this is incorrect) is lower than a nanosecond. Looping over each nanosecond in a 7 day time-frame is highly inefficient (for reference, there are $6.048 \cdot 10^{14}$ nanoseconds in 7 days), and thus we do not want to do this. One possibility is to round the time to the nearest millisecond, but this introduces issues such as two Telegrams having the same time. If the assumption that there is no time-drift does not hold, then rounding becomes even more involved to do accurately.

We would like to avoid implementing such time-based join. One way to get around this, is by making and then verifying following assumption: **Telegram messages have enough information to fully describe when `STO`s occur**. We check this assumption using the script in Codeblock 5 in Appendix A.3. Note that this *does not* implement the loop as described above, since it is precisely what we wisht to avoid. Instead, the script loops through (an arbitrarily chosen subset of) the loaded SCADA dataset, finds the reasons why STOs happen (encoded in the 4 Boolean variables), finds their start and end times with a bound of 10 seconds both ways (this bound was chosen to accommodate potential absolute differences between clocks of different systems), finds the noted palletizer, and then finally checks in the Telegram dataset if a corresponding datapoint can be found. The reason why it only uses a subset of the SCADA dataset is due to time constraints – it takes too long to check the entire dataset. Based on a subset of the dataset, it seems that we can safely make this assumption: Telegram messages have enough information to fully describe when `STO`s occur.

SCADA Dataset

row	start_time	duration	end_time	error_id	error_part
1	2021-12-10 11:58:12.890+1100			blocked_lift_shaft	1014.56.78
2	2021-12-10 11:56:52.816+1100	00:02:18.633	2021-12-10 11:59:11.449+1100	blocked_place_position	1014.56.78
3	2021-12-10 11:56:52.816+1100	00:01:45.567	2021-12-10 11:58:38.383+1100	missing_stack_surface	1024.56.78

Telegram Dataset

time	error_part	blocked _place _position	blocked _flight _path	blocked _lift _shaft	missing _stack _surface	pallet_id
2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	TRUE	FALSE	pallet3
2021-12-10 00:56:52.816+0000	1014.56.82	FALSE	FALSE	TRUE	FALSE	pallet1
2021-12-10 00:56:52.816+0000	1024.56.82	TRUE	FALSE	FALSE	FALSE	pallet2

Figure 18: Illustration showing that in the toy example SCADA and Telegram datasets nicely align.

The toy example we made also nicely correspond to the assumption that Telegram messages have enough information to fully describe when `STO`s occur. This is illustrated in Figure 18. Notice besides the fact that the assumption holds, that the timestamps are not in the same timezone. As mentioned in Section 4.1, the SCADA dataset did not initially contain timezone information, which made finding this link harder than it should have been due to the (incorrect) apparent time difference.

5.2 Join 2: + StackInfo

After combining the SCADA and Telegram datasets, we want to bring in the StackInfo dataset. Ideally, this link would be a trivial `join` operation. Sadly, it is not: there are mismatches between the number of messages (rows) in the Telegram dataset (per pallet), and the number of items placed (rows) in the StackInfo dataset (which is per pallet). This mismatch happens 2,272 times in the used dataset. Ignoring them seems bad, since most likely these are the pallets we are interested in. The toy example deliberately contains an instance of this mismatch, as illustrated in Figure 19: there are a total of eight (8) cases to be placed for `pallet3`, but there are ten (10) telegrams. Since each case is identical, there is no way to find which telegram messages correspond with which case(s).

Telegram Dataset

	time	error_part	blocked _place _position	blocked _flight _path	blocked _lift _shaft	missing _stack _surface	pallet_id
1	2021-12-10 00:58:47.696+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2	2021-12-10 00:58:40.404+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
3	2021-12-10 00:58:34.512+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
4	2021-12-10 00:58:31.824+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
5	2021-12-10 00:58:22.814+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
6	2021-12-10 00:58:18.890+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
7	2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	TRUE	FALSE	pallet3
8	2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
9	2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
10	2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3

StackInfo Dataset

	palletise _seq_nr	case _id	stack _floor _height	offCenterX	offCenterY	pallet_id	palletizer
	0		0	−460	826	pallet3	ACP1
1	1	4	0	−6	−9	pallet3	ACP1
2	2	4	0	7	10	pallet3	ACP1
3	2	4	0	3	−7	pallet3	ACP1
4	4	4	0	1	10	pallet3	ACP1
5	5	4	66	8	2	pallet3	ACP1
6	7	4	66	1	−5	pallet3	ACP1
7	8	4	66	3	2	pallet3	ACP1
8	9	4	66	2	−1	pallet3	ACP1

Figure 19: Data from the toy example illustrating mismatch between Telegrams and StackInfo.

Similar to the idea behind matching a SCADA entry to multiple Telegram entries, a first potential workaround is to group Telegrams into buckets that correspond to the case. The only field that would allow us to do this, is the *time* timestamp. As such, this workaround is infeasible, since there may be two Telegram messages for the same case with a relatively long time in between (e.g. an erroneous case where an operator had to intervene), or with a very short time in between (e.g. brief connection loss resulting in a retry).

A consequence of the inability to accurately match Telegram data to StackInfo data on a case-by-case basis is loss of data: we must consider only a pallet-level match. This has following consequences for the Telegram dataset:

1. The *error_part* field should be unique if grouped per pallet.
2. The *palletizer* field should be unique if grouped per pallet.
3. A *count* field – this field is added and counts the number of messages per pallet – should be set to its maximum value, as this indicates the number of messages for this pallet (using an average or other statistic is unreasonable, as the meaning becomes useless).
4. The Boolean indicator variables should be set to `TRUE` if any of the messages for this pallet are `TRUE`.
5. We add another indicator variable *maybe_STO*, which is the disjunction of the 4 Boolean indicator variables (so this is `TRUE` if for this particular row any other Boolean indicator variable was `TRUE`).

   Note: In the remainder of the data integration pipeline, we **assume that the** ***maybe_STO*** **field indicates that an STO occured**.

The above explanation to group Telegrams per pallet is realised in Codeblock 1, part of a Jupyter Notebook that implements the complete data loading and integration pipeline in python.

```python
# Group by pallet
groupedObject = telegram_df.groupby("pallet_id")

# Test: if we group by pallet, the error_part identifier is always unique
assert groupedObject["error_part"].nunique().nunique() == 1

# Test: if we group by pallet, the palletizer is always unique
assert groupedObject["palletizer"].nunique().nunique() == 1

# Count is set to max
group_count = groupedObject["count"].max()

# Time is transformed to start and end
group_time_start = groupedObject["time"].min()
group_time_end = groupedObject["time"].max()

# Indicator variables are set to True if any of the values are True
telegram_grouped_df = groupedObject[["blocked_flight_path", "blocked_place_position",
↪ "blocked_lift_shaft", "missing_stack_surface"]].any()

# Add extra column
telegram_grouped_df["maybe_STO"] = (telegram_grouped_df["blocked_flight_path"]) |
↪ (telegram_grouped_df["blocked_place_position"]) | (telegram_grouped_df["blocked_lift_shaft"]) |
↪ (telegram_grouped_df["missing_stack_surface"])

# Set previous computed series
telegram_grouped_df["number_of_telegram_messages"] = group_count
telegram_grouped_df["time_start"] = group_time_start
telegram_grouped_df["time_end"] = group_time_end
telegram_grouped_df["error_part"] = groupedObject["error_part"].first() # Add error_part
telegram_grouped_df["palletizer"] = groupedObject["palletizer"].first() # Add palletizer
```

Codeblock 1: Cell from Jupyter notebook implementing grouping of Telegrams.

Once the Telegram messages are grouped per pallet, they can be joined with the previous dataset using a `join` call. We use an *inner join*, since for the purpose of investigation we need both Telegram data (indicating when an STO occurs) and the StackInfo data (indicating how the pallet was stacked).

5.3 Join 3: + LFL Recipes

After combining SCADA with Telegrams (Section 5.1), and then consequently combining it with StackInfo (Section 5.2), we are left with only one potentially hard link to execute: the LFL Recipes dataset. From Figure 17 it is clear that there are two missing links when attempting to combine StackInfo/Telegram data (indicated by the dashed border, and the Linking Dataset text), which has a *pallet_id* field, to LFL data, which has *suborder_id* and *order_id* fields. In Section 5.3.1 we explain how both links (first from *pallet_id* to *suborder_id*, then from *suborder_id* to *order_id*) are acquired, followed by how to integrate them with the existing datasets in Section 5.3.2.

5.3.1 Acquiring the missing links

There are **two** missing links required to link all datasets together:

1. Link from *pallet_id* to *suborder_id*: Pallet-To-Suborder.
2. Link from *suborder_id* to *order_id*: Suborder-To-Order.

Both links can be found in the same online environment from which we can acquire the SCADA and Telegram datasets. Figure 20 shows both linking datasets side by side, the first one (Pallet-To-Suborder) on the left, and the second one (Suborder-To-Order) on the right.

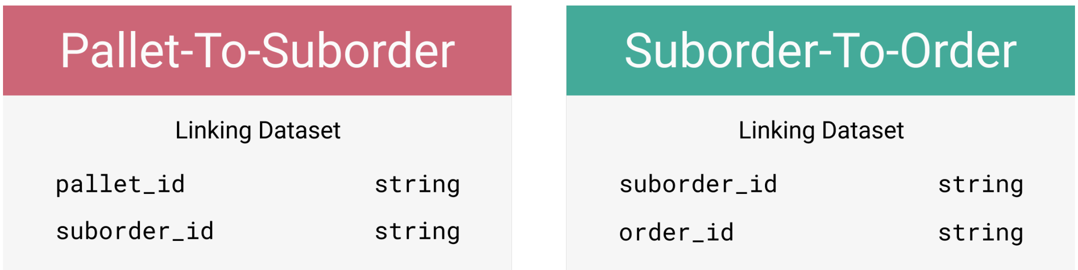

Figure 20: The Pallet-To-Suborder (left, pink) and Suborder-To-Order (right, cyan) linking datasets.

Extract & Transform

Extracting the first missing link, Pallet-To-Suborder, is very straightforward. We select the desired time-frame and click a button to export to CSV. This CSV has precisely the two columns we expect: *pallet_id* and *suborder_id*. The second missing link, Suborder-To-Order, when exported gives a malformed CSV file. This means that we need to manually parse it. An example of the raw CSV data for the toy example is shown in Codeblock 2.

```
suborder1,order1
"suborder2
suborder3",order2
```

Codeblock 2: Example of raw exported data for Pallet-To-Suborder linking dataset.

There are two main cases to parse. First, a normal line mapping the suborder to an order, such as line 1 in Codeblock 2. Second a list variant, mapping multiple suborders to a single order, such as lines 2 and 3 in Codeblock 2. We can extract both cases using regular expressions.

```
regex = re.compile("^(suborder\d*),(order\d*)$")
```

Codeblock 3: Regular expression for `suborder,order` pair: normal variant.

The regular expression for the normal variant (see Codeblock 3) is relatively straightforward. It works by matching *from the start of the line* (`^`) *select a suborder* (`(suborder\d*)`) directly followed by a comma (`,`), an order (`(order\d*)`), and the end of the line (`$`). The result is shown in on the left of Figure 21.

Match 0	0-15	suborder1,order1
Group 1	n/a	suborder1
Group 2	n/a	order1

Match 0	17-44	"suborder2 suborder3",order2
Group 1	n/a	suborder2
Group 2	n/a	suborder3
Group 3	n/a	order2

Figure 21: Results of regular expressions for retrieving `suborder,order` pairs. Left is the normal variant, right is the list variant.

The regular expression for the list variant (see Codeblock 4) is slightly more involved, as it requires *multiline* matching; we start by matching *from the start of the line* (`^`) a singular character `"`, followed by any number (`*` at the end) of *suborders followed by a new line* (`(suborder\d*\n)*`), and then a single suborder without a new line (`(suborder\d*)`), followed by the closing quote `"`, a comma (`,`), an order (`(order\d*)`), and the end of the line (`$`). The result is shown on the right in Figure 21.

```
regex = re.compile(
  '^"(suborder\d*\n)*(suborder\d*)",(order\d*)$',
  re.MULTILINE
)
```

Codeblock 4: Regular expression for `suborder,order` pair: list variant.

As shown in Figure 21, we retrieve groups containing suborders and orders, where the last group is always the order. This allows us to first extract the order, then loop over the groups (except the last element), and generative proper `suborder,order` pairs. All pairs can then be saved to a CSV file that is no longer malformed.

5.3.2 Join + Pallet-To-Suborder + Suborder-To-Order + LFL

Joining the missing links into the combination that has already been made (SCADA + Telegram + StackInfo) is very straightforward. The *pallet_id* field in the Pallet-To-Suborder dataset has a one-to-one relation with the already existing *pallet_id* field in the larger combined table. And, after parsing the Suborder-To-Order file we see an easy to use join emerge with a many-to-one relation between *suborder_id* and *order_id*. These two linking datasets behave nicely.

Joining with the LFL data is, however, not a plain `join` due to the fact that a case on the stack is *NOT* uniquely identified by its *case_id*; there may be multiple cases in a single stack. To resolve this issue, we need to loop over the sequence groups (identified by *sequence_id*) from LFL (cases in sequence group i MUST be placed before the cases in sequence group $i+1$; this induces a partial ordering), and then for each case we try to match it to the *best* case from the StackInfo dataset.

In short, we need to:

1. Loop over all recipes/suborders.
2. For a single recipe; loop over all sequence groups.
3. For a sequence group: loop over all cases.
4. For a case: loop over StackInfo information, and find the first case with identical *case_id* and with lowest *palletise_seq_nr*.
5. If such a case cannot be found: we cannot match for this pallet.

To do this (somewhat) efficiently (looping over rows in a `pd.DataFrame`, which is what we use for the implementation, is an anti-pattern and generally not recommended), we transform into *longform* tables, where a single row contains ALL information necessary for matching. The long-form table for the toy example is shown in Figure 22. Notice how a single row indeed contains all information for a single pallet. On these longform tables we execute a matching function implementing the steps listed above. After matching, we transform back to the original format of the table. The code for matching is available in Codeblock 7, in Appendix A.3.

	palletiseSeqNr	caseId	CaseId	SequenceId
suborder_id				
pallet1	[0, 1, 3, 4, 5, 6]	[<NA>, 5, 3, 4, 2, 1]	[<NA>, 5, 3, 4, 2, 1]	[<NA>, 1, 1, 2, 2, 2]
pallet2	[0, 1, 2, 3, 3]	[<NA>, 3, 1, 1, 5]	[<NA>, 3, 1, 1, 5]	[<NA>, 1, 1, 2, 2]
pallet3	[0, 1, 2, 2, 4, 5, 7, 8, 9]	[<NA>, 4, 4, 4, 4, 4, 4, 4, 4]	[<NA>, 4, 4, 4, 4, 4, 4, 4, 4]	[<NA>, 1, 1, 1, 1, 2, 2, 2, 2]

Figure 22: Illustration of longform tables.

5.4 Join 4: + Teaching

The final remaining dataset to integrate is the Teaching dataset (Section 4.3). This is a trivial integration step, as each row in the Teaching dataset is uniquely defined by its *case_id*. It is implemented as a single `merge()` call on a `pd.DataFrame`. When all datasets are joined together, we make sure that we write the result to disk. This avoids having to recompute these computationally expensive joins.

6 Data Model: Graph Database

This section explains all necessary steps to go from an integrated large table, to a graph database, which is the data model we use. In Section 6.1 we discuss precisely what the data model should contain to investigate the hypotheses **HP** 1-**HP** 4 explained in Section 3.3, which is summarised in Section A.2. In Section 6.2 we show how each node and relation from Section 6.1 can be implemented using Cypher.

6.1 Model Description

In this section we describe all required nodes, relations, and their properties the graph should have so we can investigate the four hypotheses from Section 3.3. We find these requirements by imagining first an empty graph, and then discovering the necessary data that should be present for a particular query used to answer a particular hypothesis. More concretely, the required nodes, relations, and properties to investigate hypothesis **HP** 1, **HP** 2, **HP** 3, **HP** 4 is described in Sections 6.1.1, 6.1.2, 6.1.3, and 6.1.4 respectively.

6.1.1 HP 1: Incorrect placements cause more STOs.

Investigating incorrect placements in the graph model is relatively straightforward. We need the placed (*placed*{*XYZ*}{*1234*}) and expected (*expected*{*XYZ*}{*1234*}) locations for a particular case, as well as the `offCenterX` and `offCenterY` properties. This means to be able to answer **HP** 1, we have to model at least a node in the graph corresponding to a single case. We call this node `Item`, since `case` is a Cypher keyword and cannot be used.

Nodes:

```
i:Item {
    placed{XYZ}{1234}: Integer,
    expected{XYZ}{1234}: Integer,
    offCenterX: Integer,
    offCenterY: Integer
}

p:Pallet {
  [pallet properties that are deemed interesting]
}
```

6.1.2 HP 2: Building towers in the stack cause more STOs.

Investigating towers in a stack requires us to add a property to the `Item` node stating which stacking method (*stacking_method*) LFL used for this particular case. Besides an extra property, we need a relation that allows us to travel downwards to the pallet itself (and recognise that we are indeed at the pallet). This means to be able to answer **HP** 2, in addition to Section 6.1.1 we have to model at least the following.

Nodes:

```
i:Item {
        stacking_method: List<String>,
        [... properties from Teaching that are deemed interesting]
}
```

Relations:

```
r:ON_TOP_OF {}
```

6.1.3 HP 3: Height gaps between cases cause more STOs.

To investigate height gaps, we need to somehow compute and store the gaps themselves. This can nicely be done by adding it as a property to the `ON_TOP_OF` relation. In an ideal theoretical situation, if the gap equals to 0, then this means that two cases (`Item` nodes) are directly on top of each other.

Since we are not solely interested in *height gaps in towers*, it seems reasonable to look at both the case raising an STO, as well as its *neighbours*: this means we need a `NEXT_TO` relation explaining which case(s) are next to the one raising an STO. Then, from the set of cases containing the one raising an STO and its neighbours, we can investigate whether or not height gaps are extremely prominent. This set of cases we call the **related cases**. This means to be able to answer **HP** 2, in addition to Sections 6.1.1, 6.1.2 we have to model at least the following.

Relations:

```
r:ON_TOP_OF {
        gap: Integer
}

r:NEXT_TO {}
```

6.1.4 HP 4: Overhang causes more STOs.

To investigate overhang, we need to look at the leftmost/rightmost/topmost/bottommost points of a case, given the XY-plane, and see whether or not these points exceed the pallet boundaries. To this end, we add the 4 properties to the item node, and a relation between node and pallet overhang with a property denoting the size. This means to be able to answer **HP** 2, in addition to Sections 6.1.1, 6.1.2, 6.1.3 we have to model at least the following.

Nodes:

```
i:Item {
        leftmost_point: Integer
        rightmost_point: Integer
        highest_point: Integer
        lowest_point: Integer
}
p:Pallet {
    width: Integer
    length: Integer
    placement: Point2D
}
```

Relations:

```
r:OVERHANG {
        amount: Integer,
        reason: String
}
```

6.2 Model Implementation

Section 6.1 describes a target graph data model. The end result of Section 5 is a large table containing data. In this section we present the method to take the large table, and create the desired graph from it. The method starts by pre-processing the large table to create `pallet.csv` and `cases.csv` files in Section 6.2.1. Then, for the remaining sections, we take one node or relation type from the theoretical model described in Section 6.1, and translate their requirements to a Cypher query. Running all the translated Cypher queries will give the desired data model.

6.2.1 Pre-processing

Before we can run any query, we need to pre-process the large table created by integrating all datasets together. Pre-processing is required due to two reasons, the first being that data is *highly skewed* (many pallets have no STO error). Any attempted statistical measure on the table as-is, is inconclusive due to the skewness. As such, pre-processing here means *extracting relevant information*. The relevant information means that for each pallet, we only keep data until the last STO. For pallets that have no STO in them, we discard them entirely (at a later point it will be interesting to also load these, but for now they are considered as unwanted datapoints). This preprocessing is done in python, and makes following **unchecked assumptions**.

1. **A single telegram message corresponds to a single placed case**: We already know that this does not necessarily need to be the case due to retries. However, it may hold, in which case we can proceed. If it does not hold, then perhaps this is a reason as to why there are gaps in the `palletise_seq_nr` field? In any case, we simply assume this to hold to continue with the analysis, and later come back on how it influences results if it does not hold.
2. **If the i^{th} telegram message states that there was an STO, then the i^{th} placed case is deemed as having raised the STO**: See previous unchecked assumption above, we know this does not need to hold.

For the remainder of Section 6.2 (more specifically, the referenced Cypher queries in each subsection) we assume that there are two files present in the import directory for Neo4J: `pallets.csv` and `cases.csv`. The first, `pallets.csv`, contains information related to a pallet or stack, whereas `cases.csv` contains information on cases (including on which pallet they are supposed to be stacked). Examples of the `pallets.csv` and `cases.csv` for the pallet of order 1 of the toy example can be found in Tables 7 and 6 respectively. Note that those tables do not show all fields, as this cannot be displayed easily on paper.

6.2.2 Node `Item`

To create the `Item` node – we call it `Item` since `case` is a reserved keyword in Cypher – we take the `cases.csv` file, and for each row we create a `Item` node. Since by default when loading from CSV all properties are strings, we manually set required data types where necessary. Cypher contains a `point` type which we can use for points in space, so fields that represent points such as *placed*{*XYZ*}{*1234*} can be cast to this type. Loading all data in batches as to not destroy memory is done as illustrated in Cypher Query 10.

6.2.3 Node `Pallet`

Creating the pallet node(s) can be done in two ways:

1. Given the `Item` nodes, match those where `placement_id` states it is a pallet, extract the `pallet_id` field, and create a node from that.
2. In python, group by `pallet_id`, export as CSV, and then load is similar to the `Item` node.

We choose the second option since this is computationally-wise faster, and its result is precisely the `pallets.csv` file, where each row describes metadata of a particular pallet, uniquely defined by a *pallet_id*. Similar to creating the `Item` node, we manually set required data types where necessary. Loading all data is again done in batches. The resulting query is shown in Cypher Query 11.

6.2.4 Relation ON

Given Item and Pallet nodes, we want to be able to say that a particular Item is *ON* a particular Pallet: this relation relates Items to Pallets. If an Item node and Pallet node have the same *pallet_id*, then an Item is considered to be placed on the Pallet. The related Cypher statements are shown in Cypher Query 12.

6.2.5 Relation PLACED_BEFORE

Given two Item nodes, say A and B, we want to be able to say that A was placed before B, similar to how the *palletise_seq_nr* indicates the order of cases stacked on a pallet. Since entries in cases.csv are ordered by time, and thus by *palletise_seq_nr*, we can use Neo4J's id() function, which is a function returning the internal identifier Neo4J uses for a particular node. These identifiers are assigned as an increasing count, in order of insertion in the database. So, due to the way data is imported, the check using id() is always correct. The query itself thus checks that both A and B have identical *pallet_id* fields, and that $id(A) = id(b) - 1$. The related Cypher statements are shown in Cypher Query 13.

6.2.6 Query set_STO_property

We want to be able to relate cases with STOs to other cases (among others), but this is impossible if there is no property indicating which case has an STO. Since we did not yet add this property during pre-processing (because the query we describe here was already written), we need to add the property with a Cypher query. It works by matching Item A, then optionally matching Item B such that (a)-[:PLACED_BEFORE]->(b) is null. In this case, A has no outgoing arrows for the PLACED_BEFORE, indicating that the node corresponds with the case placed last in cases.csv, and thus is assumed to be the case causing the STO (see the assumptions in Section 6.2.1). The related Cypher statements are shown in Cypher Query 14.

6.2.7 Relation ON_TOP

Deciding on a specific definition on when precisely some case A is considered to be on top of another case B is a **design decision** that should be made in accordance with experts. In this work, we choose the simplest possible definition: as long as there is more than 0 mm overlap between A and B (overlap is considered from a top-down view), then A is considered on top of B each other. Translating into Cypher, we have:

```
MATCH
  (a:Item),
  (b:Item)
```

We need to ensure that they are on the same pallet.

```
WHERE
  a.pallet_id = b.pallet_id
```

Then, we need to actually encode the relation into Cypher. Effectively, when looking at the stack from a top-down view, we need to find out if any corner of item A is contained within the coordinates of item B. Cypher can do this for us with point.withinBBox(point, lowerLeft, upperRight). If then the z coordinates of A are higher than B, then A is on top of B.

```
// Do this for a.placed1, a.placed2, a.placed3, a.placed4 with ORs in between
point.withinBBox(
  point( {x: a.placedX.x, y: a.placedX.y} ), // point to check
  point( {x: b.leftmost_point, y: b.bottommost_point} ), // lower left bounding box
  point( {x: b.rightmost_point, y: b.topmost_point} ), // upper right bounding box
)
```

The problem is that this is **extremely slow** to do. A better approach is to pre-compute the extreme points (leftmost, bottommost, rightmost, ...) of all cases, and “manually” write down all different situations where item A can be on top of B. Since doing this is tedious work, we use a *query generator* as shown in Script 8. The resulting query can be found as Query 15 in Appendix A.4. The generator roughly works as follows:

1. On top of the script, define whether or not to output queries for each individual situation. These may be useful to investigate, for instance, only situations where A is completely on top of B, with all edges of A contained in B.
2. Give names to possible situations in X and Y axis.
3. Create a dictionary, where each situation (dictionary keys are the names from step 2) is encoded.
4. Write boilerplate strings that will make up the final query.
5. Loop through the map, and generate the final query.
6. Loop a second time to create the *reason* property, which will be the (combined) name as decided in step 2.
7. Some more bookkeeping to keep the relation correct, and add the required *gap* property.

Realise that there are a total of 4 different situations for the X axis and the Y axis.

```
X = ["R_EDGE", "BOTH_X", "L_EDGE", "NONE_X"]  # 4 situations
Y = ["B_EDGE", "BOTH_Y", "T_EDGE", "NONE_Y"]  # 4 situations
```

The 4 situations for the X axis are illustrated in below figures.

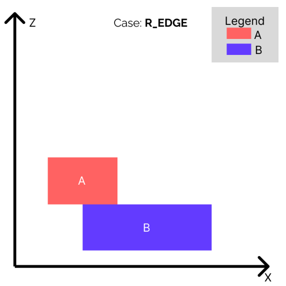

Figure 23: `R_EDGE`: Only the right edge is on top.

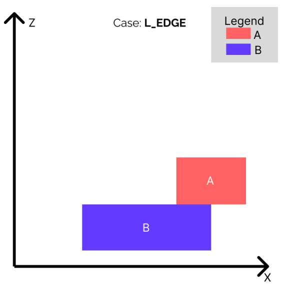

Figure 24: `L_EDGE`: Only the left edge is on top.

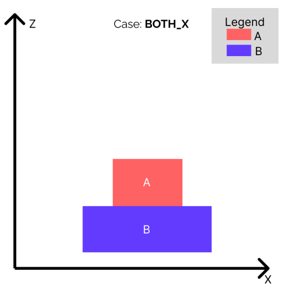

Figure 25: `BOTH_X`: Both edges are on top.

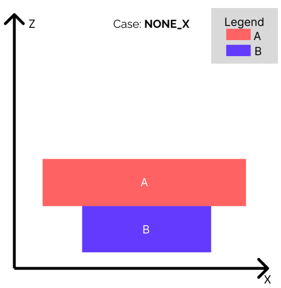

Figure 26: `NONE_X`: None of the edges are on top.

Encoding these situations is done in the dictionary created in step 3. For instance, to encode the `BOTH_X` situation (bottom left), we need that both edges of A are contained within B on the X-axis. This amounts to having the leftmost point of A being larger than the leftmost point of

B, and having the rightmost point of A being smaller than the rightmost point of B. With the precomputed properties, encoding this is relatively straightforward. Note that *equality* is added to not exclude very rare situations where items line up perfectly.

```
"BOTH_X": [
    "a.leftmost_point >= b.leftmost_point",  # equal to left
    "a.rightmost_point <= b.rightmost_point"  # equal to right
],
```

A complete illustrations of all 16 situations (4 for the X axis, times 4 for the Y axis) is shown in Figure 27. Note that all these situations are generated by the query generator in Script 8.

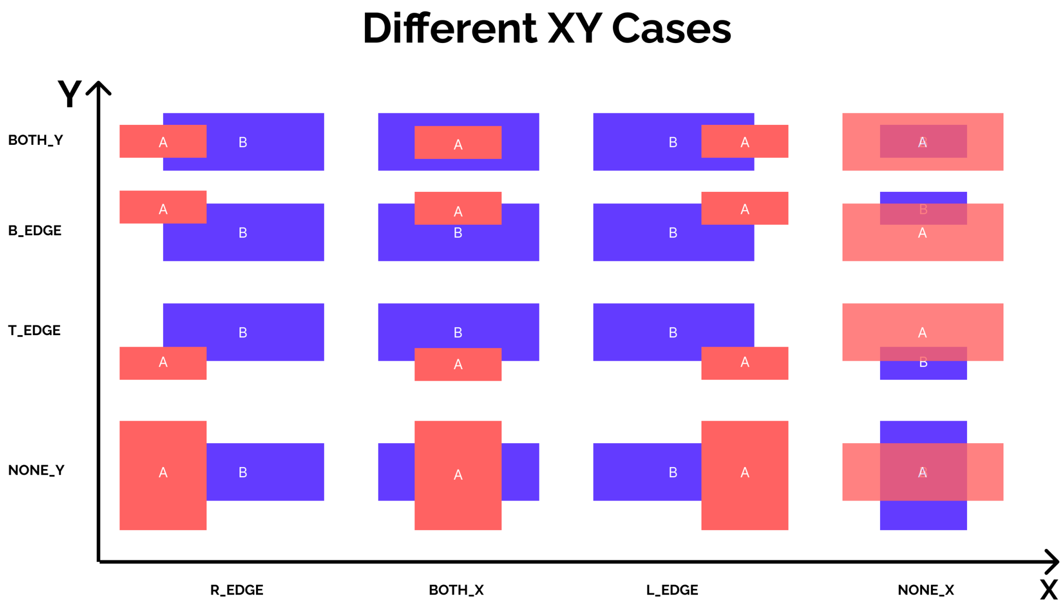

Figure 27: Overview of 16 possible situations for the `ON_TOP` relation.

6.2.8 Relation `NEXT_TO`

Similar to how we create the `ON_TOP` relation, we can create the `NEXT_TO` relation by emulating the same steps but with a different plane than the XY-plane. The problem is that in stead of 16 possible situations before, we now have 64 situations as the Z axis gets involved. Another way to reason about it, is that we have to compute `ON_TOP`, but for 4 different sides: the left (L), the front (F), the right (R), and the back (B). This is effectively what we do with the generator code shown in Script 9. The resulting query can be found as Query 16 in Appendix A.4. The generator roughly works as follows:

1. On top of the script, define whether or not to output queries for each individual situation. These may be useful to investigate, for instance, only situations where A is next to B, and where B is taller than A, and where A is contained in B.
2. Give names to possible situations in X, Y, **and Z axis**.
3. Create a dictionary, where each situation (dictionary keys are the names from step 2) is encoded.
4. Write boiler plate strings that will make up the final query.
5. Loop through the map, and generate the final query.

6. Loop a second time to create the *reason* property, which will be the (combined) name as decided in step 2.
7. Some more bookkeeping to keep the relation correct.

As mentioned earlier, we need to consider four different sides: the left, the front, the right, and the back. These are encoded in the dictionary of step 3 as follows.

```
"L": [
    "a.rightmost_point < b.leftmost_point",
],
"F": [
    "a.backmost_point < b.frontmost_point"
],
"R": [
    "a.leftmost_point > b.rightmost_point"
],
"B": [
    "a.frontmost_point > b.backmost_point"
],
```

Then, we need to look at the XZ-plane to find which case is bigger or smaller. The 4 possible situations here are encoded in the `Z` list.

```
Z = ["HIGH_Z", "SMALL_Z", "LOW_Z", "BIG_Z"]  # 4 situations
```

Finally, we have the XY-plane to consider, in which we have to distinguish between horizontal sides (front and back) and vertical sides (left and right). Combining all of these different possibilities together, we get a combination of the situations shown in Figure 28, where from each column one must be chosen (leading to, as explained before, a total of $4 \cdot 4 \cdot 4 = 64$ different situations).

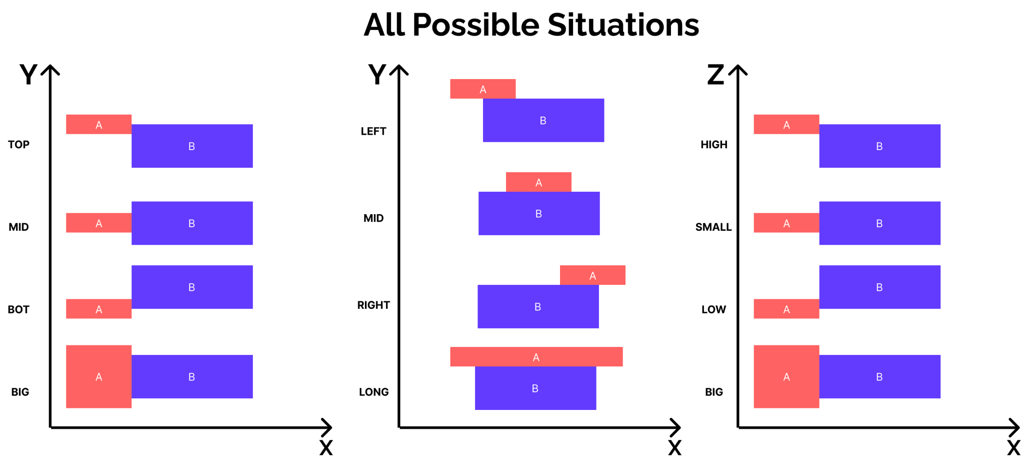

Figure 28: Illustration of "the three columns" where any combination is a single possible situation for the `NEXT_TO` relation.

To elaborate, we can choose *top* from the left column, *mid* from the middle column, and *small* from the right column. This combination gives us the situation where for two `Items` A and B the frontmost point of A is in-between the frontmost and backmost points of box B (top), the leftmost and rightmost points of A are contained within the leftmost and rightmost point of B (mid), and the lowest and highest points of A are contained within the lowest and highest points of B (small).

7 Results

In this section we describe, based on pallet 1 from order 1 of the toy example (Figure 12), what we expect the data model to look like. For convenience, pallet 1 is shown again in Figure 29. This Figure (29) shows the stack as computed by LFL.

Figure 29: Pallet 1 from Order 1 of the toy example (Figure 12).

After creating the entire data model (recall that this is created based on the *recorded* values, as opposed to the *computed* ones) we expect it to, for instance, show that the yellow-coloured case is on top of the blue-coloured case. Similarly, we expect the cyan-coloured case to be next to both the wine-coloured case, as well as the purple-coloured case. To provide a systematic method for evaluating if the created graphs shows what we “expect” it to, we consider each (node and relation) query described in Section 6.2 and give an illustration of its expected/desired/wanted output. This output is created by “manually” executing the related Cypher queries, using the toy example data *for only order 1*. Parts of the related `cases.csv` and `pallets.csv` files obtained after pre-processing the toy example are shown in Table 6 and Table 7 respectively.

pallet_id	**palletise _seq_nr**	**case_id**	**rightmost _point**	**leftmost _point**	**lowest _point**	**highest _point**
pallet1	0					
pallet1	1	5	343	0	0	121
pallet1	2	3	1004	344	66	131
pallet1	3	4	595	345	0	66
pallet1	4	2	724	596	131	241
pallet1	5	1	1003	724	111	221

Table 6: Part of the `cases.csv` file corresponding with the first pallet of the toy example.

pallet_id
pallet1

Table 7: Part of the `pallets.csv` file corresponding with the first pallet of the toy example.

If the expected and realised output correspond with one another, then this implies that the graph can be used for data analysis. If not, then either the corresponding query is incorrect, or the data that the query uses is inaccurate, potentially causing the graph to become unusable.

7.1 Node `Item` – see Section 6.2.2

The expected results after executing the query to create `Item` nodes (Query 10) is shown in Figure 30. The reason why Figure 30 illustrates the expected situation is as follows: the pre-processed `cases.csv` file lists precisely five cases, each with unique *case_id*, to be stacked for pallet with *pallet_id* being `pallet1`. The interested reader can verify correctness by looking at Table 6. For convenience, we have coloured the nodes according to their *case_id*, and it immediately becomes clear that the nodes correspond with the cases as shown in Figure 29. Verification is thus done by *looking at the relevant data*, paying specific attention to the *case_id* of cases (these need not be unique, but the number of cases placed should correspond with the number of nodes created).

Figure 30: Expected results after running Query 10.

For the 50 inspected pallets, all `Item` nodes are created as expected.

7.2 Node `Pallet` – see Section 6.2.3

The expected results after executing the query to create `Pallet` nodes (Query 11) is shown in Figure 31. The reason why Figure 31 illustrates the expected situation is as follows: the pre-processed `pallets.csv` file lists precisely one pallet with *pallet_id* being `pallet1`. The interested reader can verify correctness by looking at Table 7. There is only one node with type `Pallet` created, corresponding to the one pallet with *pallet_id* being `pallet1`. Verification is thus done by *looking at the relevant data*, paying specific attention to the *pallet_id* of the pallet (in this case, the *pallet_id* field uniquely defines a pallet).

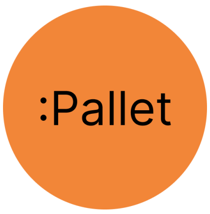

Figure 31: Expected results after running Query 11.

For the 50 inspected pallets, all `Pallet` nodes are created as expected.

7.3 Relation `ON` – see Section 6.2.4

The expected results after executing the query to create the `ON` relation (Query 12) is shown in Figure 32. The reason why Figure 32 illustrates the expected situation is as follows: the pre-processed `cases.csv` file lists precisely five cases, each with unique *case_id*, to be stacked for pallet with *pallet_id* being `pallet1`. The interested reader can verify correctness by looking at Table 6: for any pair of nodes A, B (with A having type `Item` and B having type `Pallet`) satisfying the path query `(a:Item)-[r:ON]->(b:Pallet)`, we require that the row in `cases.csv` corresponding to case A contains the `pallet_id` of B in its `pallet_id` field. Note that this is precisely how the corresponding Cyper query (Query 12) creates the relation. Similar to Section 7.1, for convenience we have coloured the nodes according to their *case_id*, and it immediately becomes clear that the nodes correspond with the cases as shown in Figure 29. Verification is thus done by *looking at the relevant data*, paying specific attention to the *case_id* of cases, and checking if their *pallet_id* corresponds to the *pallet_id* of the `Pallet` node.

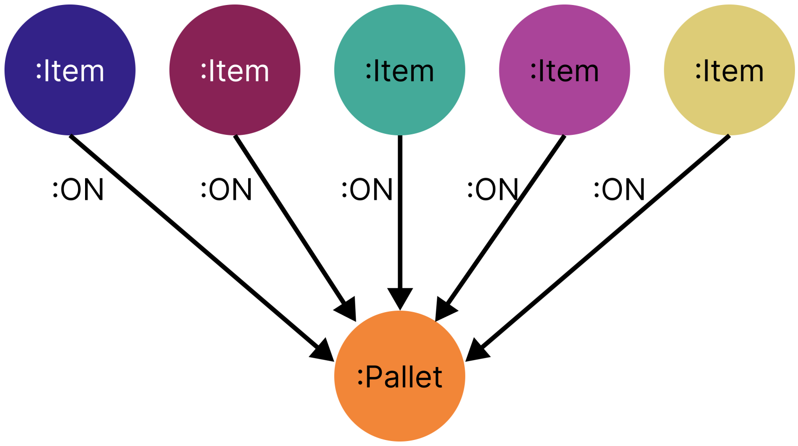

Figure 32: Expected results after running Query 12.

For the 50 inspected pallets, all `ON` relations are created as expected.

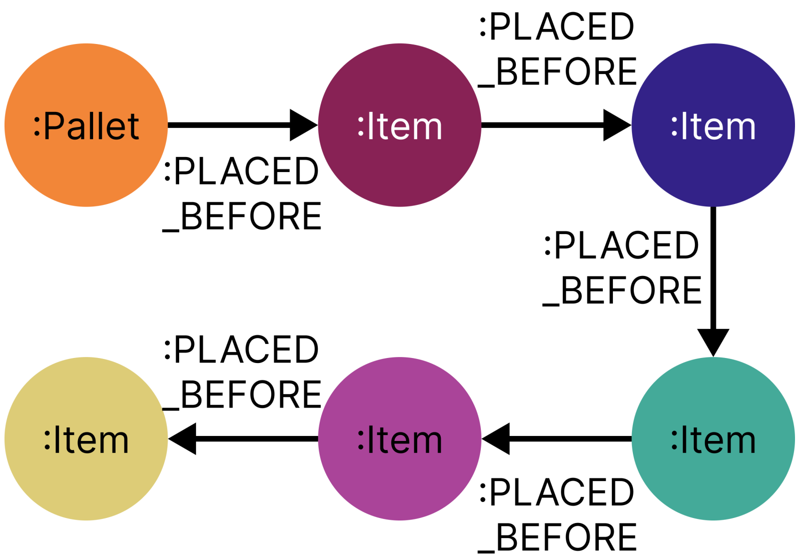

Figure 33: Expected results after running Query 13.

7.4 Relation `PLACED_BEFORE` - see Section 6.2.5

The expected results after executing the query to create the `PLACED_BEFORE` relation (Query 13) is shown in Figure 33. The reason why Figure 33 illustrates the expected situation is as follows: the pre-processed `cases.csv` file lists precisely six entries where the *pallet_id* field is equal to `pallet1`, and it lists those entries with a specific *palletise_seq_nr*. The interested reader can verify correctness by looking at Table 6: for any pair of nodes A, B (regardless of node type) satisfying the path query `(a)-[r:PLACED_BEFORE]->(b)`, we require that the row in `cases.csv` corresponding to node A contains identical `pallet_id` to the row corresponding to node B, and we require that the *palletise_seq_nr* field of A is less than or equal to that of B. Furthermore, if A

has type `Pallet`, then A is not allowed any incoming arrow. Note that this is precisely how the corresponding Cyper query (Query 13) creates the relation. Similar to Section 7.1, for convenience we have coloured the nodes according to their *case_id*, and it immediately becomes clear that the nodes correspond with the cases as shown in Figure 29. Verification is thus done by *looking at the relevant data*, paying specific attention to the *palletise_seq_nr* field for nodes as argued above.

For the 50 inspected pallets, all `PLACED_BEFORE` relations are created as expected.

7.5 Relation `NEXT_TO` – see Section 6.2.8

An abstraction of the expected results after executing the query to create the `NEXT_TO` relation (Query 16, generated by Script 9) are shown in Figure 34. In particular, we abstract from the underlying *reason* property of the relation, to make the illustration easier to parse. The reason why Figure 34 illustrates the expected situation is as follows: the pre-processed `cases.csv` file contains precisely five cases, each with unique *case_id*, to be stacked for pallet with *pallet_id* being `pallet1`. For these cases, it contains information on the extreme points (*leftmost_point*, *rightmost_point*, ...). As example, based on the extreme points, one can see that for the case with *case_id* `4` its *leftmost_point* is 345 and the *rightmost_point* of case with *case_id* `5` is 343. Since there is also some overlap in height, indicated by the *highest_point* and *lowest_point* fields (the *highest_point* of the case with *case_id* `4` is 66, which is contained between the *lowest_point* (0) and *highest_point* (121) of case with *case_id* `5`), we expect the case with *case_id* `4` to be next to the case with *case_id* `5`. Note that Section 6.2.8 fully describes all necessary conditions and situations when we expect cases to be next to one another. The interested reader can verify correctness for all relations shown in Figure 34 by looking at Table 6. Note that for two cases A and B, if A is next to B then this implies B next to A (with opposite *reason* property): this is nicely shown in the abstraction of Figure 34 by means of bidirectional arrows. Similar to Section 7.1, for convenience we have coloured the nodes according to their *case_id*, and it immediately becomes clear that the nodes correspond with the cases as shown in Figure 29. Verification is done by *looking at the relevant data*, paying specific attention to the extreme points for `Item` nodes as argued above.

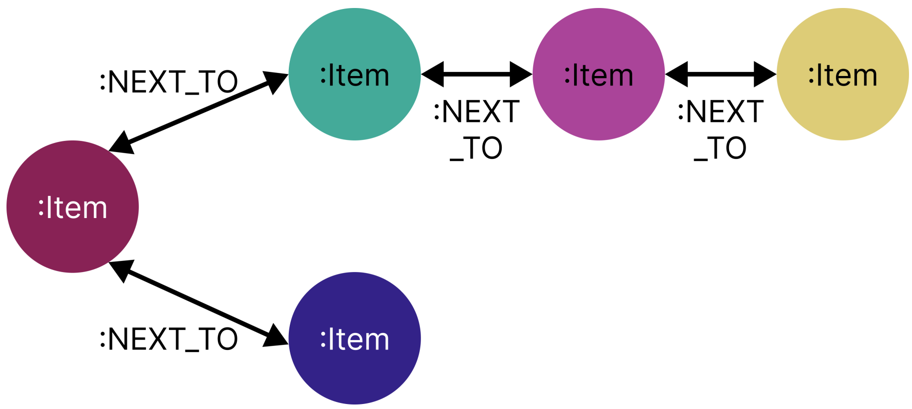

Figure 34: Expected results after running Query 16, generated by Script 9.

For the 50 inspected pallets, all `NEXT_TO` relations are created as expected. Furthermore, performing an identical abstraction as done in Figure 34 shows identical results for all 50 inspected pallets, implying that the `NEXT_TO` relation truly is bidirectional as it should be.

7.6 Issue: Relation `ON_TOP` – see Section 6.2.7

The expected results after executing the query to create the `ON_TOP` relation (Query 15, generated by Script 8) are shown in Figure 34. The reason why Figure 34 illustrates the expected situation is as follows: the pre-processed `cases.csv` file contains precisely five cases, each with unique *case_id*, to be stacked for pallet with *pallet_id* being `pallet1`. For these cases, it contains information on the extreme points (*leftmost_point*, *rightmost_point*, . . .). As example, based on the extreme points, one can see that for the case with *case_id* `3` its *highest_point* is 131 and the *lowest_point* of case with *case_id* `2` too is 131. Since there is also some overlap in width, indicated by the *leftmost_point* and *rightmost_point* fields (the *rightmost_point* of the case with *case_id* `4` is 724, which is contained between *leftmost_point* (344) and *rightmost_point* (1004) of case with *case_id* `2`), we expect the case with *case_id* `2` to be on top of the case with *case_id* `4`. Note that Section 6.2.7 fully describes all necessary conditions and situations when we expect cases to be on top of one another. The interested reader can verify correctness for all relations shown in Figure 35 by looking at Table 6. Note that for two cases A and B, if A is on top of B then surely B cannot be on top of A. Similar to Section 7.1, for convenience we have coloured the nodes according to their *case_id*, and it immediately becomes clear that the nodes correspond with the cases as shown in Figure 29. Verification is done by *looking at the relevant data*, paying specific attention to the extreme points for `Item` nodes as argued above, similar to the `NEXT_TO` relation (Section 7.5).

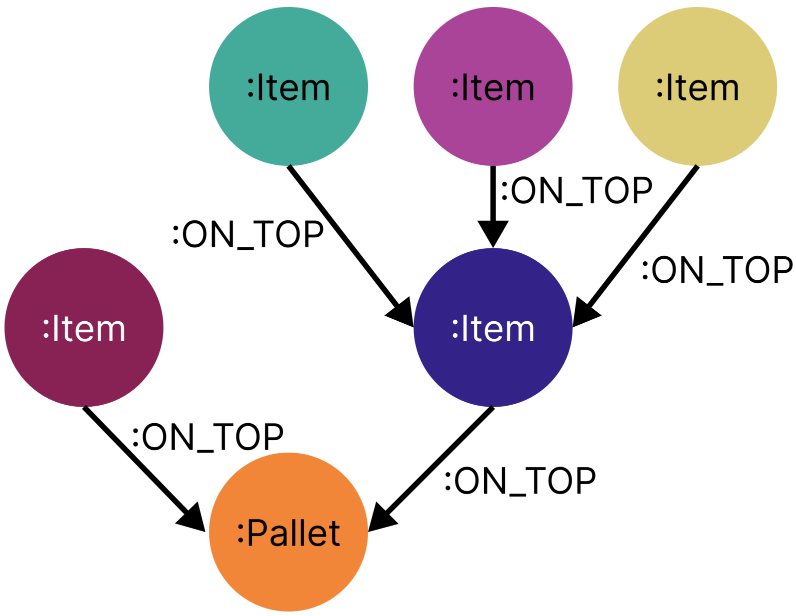

Figure 35: Expected results after running Query 15, generated by Script 8.

In stead of the expected model shown in Figure 35, we see results similar to Figure 36. For convenience, we extract the relevant situations and present them in Figure 37.

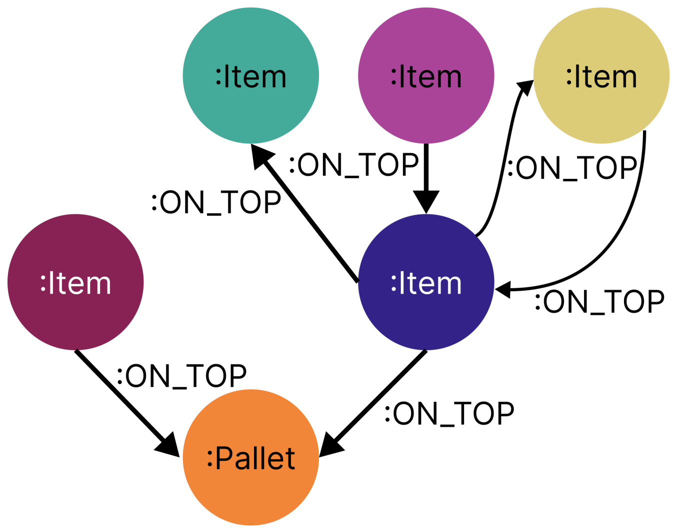

Figure 36: Actual results after running Query 15, generated by Script 8.

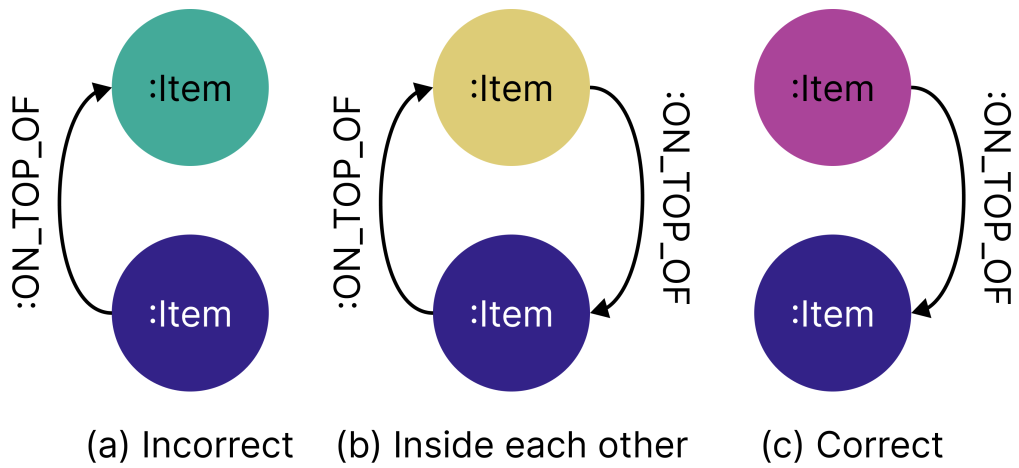

Figure 37: Illustration of the three situations visible in Bloom for the original dataset.

The situation on the left in Figure 37 (a) is incorrect, as the blue-coloured case is not supposed to be on top of the cyan-coloured case (see Figure 29: the cyan-coloured case is supposed to be on the blue-coloured case). The situation in the middle in Figure 37 (b) can never be true, since A on top of B implies B not on top of A. Nevertheless, we do observe this situation, and we suspect that it is (somehow) due to overlapping values for the Z-axis, since material phasing for now is still science-*fiction*. We expect only the yellow-coloured case to be on top of the blue-coloured case, but not the other way around (see Figure 29: the yellow-coloured case is on top of the blue-coloured case). Only the situation depicted on the right in Figure 37 (c) is correct, as it shows a single direct relation between the purple-coloured case and the blue-coloured case precisely as shown in Figure 29.

For the 50 inspected pallets, **NONE** of the relations are created as expected. This is particularly interesting, as the `NEXT_TO` relation was created *based on* the `ON_TOP` relation.

8 Discussion

In Section 7 we show that there is a **critical** data quality issue for the placements in the Z axis, making the data model as described in Section 6 for now unusable. We start this section by discussing the data quality issue in Section 8.1. Then, we proceed to discuss how the data model is to be used (should these data quality issues not exist) in Section 8.2, by showing how we want to answer the hypotheses from Section 3.3. Third, in Section 8.3, we (briefly) restate all assumptions and illustrate why they may potentially invalidate our results. We also provide other threats to validity of the thesis. We end the discussion in Section 8.4 where we communicate future avenues of research that might be of interest to Vanderlande and academia.

8.1 The Data Quality Issue

We explain the data quality issue in Section 8.1, but to further illustrate the differences between expected and realised behaviour, consider the illustration in Figure 38: on the left we show the pallet that is expected, corresponding one to one with Figure 29 and the expected graph in Figure 35, and on the right we show the pallet based on recorded placements, corresponding with the realised graph shown in Figure 36. Clearly, these stacks are not identical (which in an ideal world they should be).

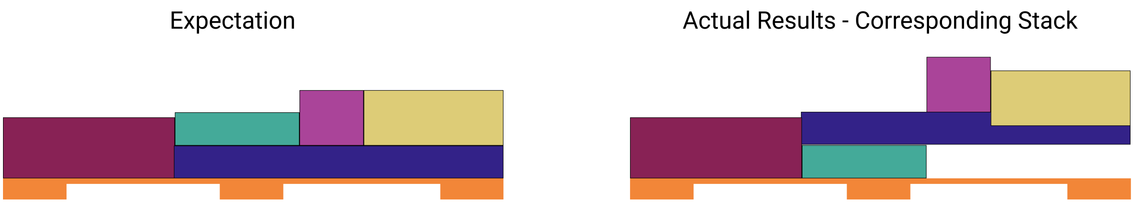

Figure 38: Illustration of the data quality issue.

We believe that the underlying cause for this critical data quality issue is specifically the recorded *placedZ{1234}* values: first realise that the `NEXT_TO` relation (Section 7.5 for results, Section 6.2.8 for implementation) is created *after* the `ON_TOP` relation (Section 7.6 for results, Section 6.2.7 for implementation). The `NEXT_TO` relation, which uses identical logic in the way the Cypher query is constructed to the `ON_TOP` relation, appears to have no issues at all based on the 50 pallets we have looked at. But the `ON_TOP` relation does have issues for all of these 50 pallets. Now, since the queries use identical logic, aside from human errors such as typos, we can exclude *the Cypher query is wrong* as reason for the observed behaviour. As such, we believe that the underlying reason is hidden in the data that the queries use. The `NEXT_TO` relation predominantly focuses on the *placedX{1234}* and *placedY{1234}* values through the computed *leftmost_point*, *rightmost_point*, *frontmost_point*, and *backmost_point* fields, whereas the `ON_TOP` relation predominantly focuses on the *placedZ{1234}* values through the computed *highest_point* and *lowest_point* fields. Since both queries creating the relations have identical logic, we conclude that the data quality issue is only present for *placedZ{1234}* values. Another way to reason is as follows: if the data issue is present for either *placedX{1234}* or *placedY{1234}* values, then we expect the `NEXT_TO` relation to produce similar situations to those described in Figure 37. These, however, do not occur in the subset of data we have looked at. As such, we exclude *placed{XY}{1234}* as having data quality issues.

While not desirable, it does to some extent make sense that we observe a data quality issue for *placedZ{1234}* values, but not for *placed{XY}{1234}* values. Recall that the STO camera (Section 3.2.1) uses computer vision to check if cases are placed where expected. While the internal workings on how the computer vision algorithm works is confidential, it is not too far-fetched to imagine that it is not perfect, and may introduce inaccuracies. These inaccuracies will be higher for the Z axis (and thus the *placedZ{1234}* values), as the camera has a top-down view, and intuitively differentiating between left and right (X axis) or front and back (Y axis) is (considerably) easier than perceiving depth (Z axis) – see for instance [63] where authors compare accuracy of various stereo cameras, or [64] for a lecture explaining why depth perception is hard. The STO camera seemingly works well enough for raising STOs, but the observed inaccuracies pose the question if there are missed STO errors. Another potential reason as to why we observe this issue for *placedZ{1234}* values, but not for *placed{XY}{1234}* values is due to the handover of support explained in Section 3.2.3. This handover occurs from top to bottom (Z axis). As such, it is intuitive that the values it influences most is those related with Z axis: the *placedZ{1234}* values.

We state that this data quality issue is **critical** since it invalidates any approach using the graph that we want to use to answer the hypotheses from Section 3.3. For **HP** 1 we want to investigate incorrect placements, which rest upon accurate enough measured placements for all axes, and thus also the Z axis and the *placedZ{1234}* values. For **HP** 2 we are interested in towers, but towers naturally cannot be investigated without the use of the `ON_TOP` relation (there is no way to look at towers if you do not know how cases relate in the Z axis). For **HP** 3 we are interested in height gaps, but height gaps too depend solely on the *placedZ{1234}* values, which are inaccurate. Finally, for **HP** 4 we want to investigate overhang, but as already argued earlier this depends on data that we do not have (Section 4.4). For a more elaborated version per hypothesis, see Section 8.2 where we show possible graph usage for hypothetical data without quality issues.

8.2 Graphs Usage

In this section, per hypothesis, we explain *how* the graph can be used if there was no data quality issue. There are two paradigms: either it comes down to *load the graph in Bloom, and add colours in a smart way to visually inspect the data*, or it is *use Cypher to query paths for new data that otherwise would be very hard to obtain.* We demonstrate these paradigms by means of examples, where all examples are based upon only the **related cases** to the STO error: if A is the case with the STO error, its related cases is the set containing A, all neighbours B of A that correspond to `(a:Item)-[r:NEXT_TO]->(b:Item)`, and then for A and all its neighbours the cases that support them, that is, all cases that are matched by repeatedly taking cases C corresponding to `(a:Item)-[r:ON_TOP]->(c:Item)`. To illustrate, for the pallet in Figure 29, if the purple-coloured case is the one with STO, then the set of related cases include the purple-coloured case (STO itself), cyan-coloured and yellow-coloured cases (neighbours), and the blue-coloured case (repeatedly go down from STO and its neighbours until pallet is reached).

> Important to note is that the discussion here only touches on the surface of all possibilities the graph brings in terms of analytical power, either through visualisation or direct querying. Also note that due to having no data to run on, provided Cypher queries in this discussion have not been verified for correctness.

8.2.1 HP 1: Incorrect placements cause more STOs.

To investigate this hypothesis, we first need to find incorrect placements. In the available data incorrect placements are found by looking at the *offCenter{XY}* fields, or by comparing the *expected{XYZ}{1234}* fields with the *placed{XYZ}{1234}* fields, keeping in mind how these related to one another (see Section 4.4). Questions that can be answered using a tabular format are limited mostly to statistics based on or over these fields, or need fancy custom-made visualisations (plots). Using the graph database and a graph visualisation tool such as Bloom one can answer various questions on incorrect placements by both filtering on relations (/properties/nodes), and colouring relations (/nodes) based on properties. For instance, one can colour `Item` nodes with a gradient that increases based on the value of the *offCenterX* or *offCenterY* field. By looking at the hierarchical view, if the colour consistently increases for the gradient it is evident that the more layers there are, the higher the *offCenterX* or *offCenterY* values. Effectively, the error propagation in the pallet is directly visible. By doing this for multiple pallets at the same time, one can potentially find patterns.

8.2.2 HP 2: Building towers in the stack causes more STOs.

To investigate this hypothesis, we first need to (define and) find towers in a stack. If we define towers as the physically built towers, then in the available data towers are found by looking at the *placed*{*XYZ*}{*1234*} fields. In tabular format, it is extremely hard to see such towers, unless visualisations are used (e.g. plotting the stack in 3D space). In a graph visualisation tool such as Bloom, which visualises directly the physical stack, one can immediately see towers by filtering on the `ON_TOP` relation. With Cypher one could write a query that counts the number of cases that are below the case with STO: `MATCH path = (a:Item {sto: True})-[r:ON_TOP*]->(b:Pallet) RETURN LENGTH(p) - 1` matches this path, then returns the length minus one to accommodate for the extra selected node of type `Pallet`, effectively counting cases. This can then be exported as CSV to analyse with statistical measures as desired.

8.2.3 HP 3: Height gaps cause more STOs.

To investigate this hypothesis, we first need to find height gaps. In the available data height gaps are found by looking at the *placed*{*XYZ*}{*1234*} fields. In tabular format, it is extremely hard to see height gaps, unless visualisations are used (e.g. plotting the stack in 3D space). In a graph visualisation tool such as Bloom, which visualises directly the physical stack, one can immediately see and investigate height gaps by colouring the `ON_TOP` relation based on the *gap* property, allowing for visual analysis as desired. Note that horizontal gaps can be investigated by doing the same for the `NEXT_TO` relation. We can also directly use Cypher to compute statistics over the gaps. For instance, we can compute the average, highest, lowest, or other statistic of the gap over a path from a case down to the pallet, and see if for cases with STOs these significantly differ from those without STO. For the average gap per path, the corresponding Cypher query is `MATCH path = (a:Item {sto: True})-[rels:ON_TOP*]->(b:Pallet) RETURN REDUCE(avgGap = 0, r IN rels | avgGap + (r.gap)/(LENGTH(path) - 1)) AS averageGap`.

8.2.4 HP 4: Overhang cause more STOs.

To investigate this, one needs to find overhang of cases with respect to the pallet. While not currently present in the data, for the sake of discussion we assume that from the *offCenter*{*XY*} fields we can accurately retrieve where pallets are placed, and consequently compute their *leftmost_point*, *rightmost_point*, *frontmost_point*, and *backmost_point* values to store as properties for `Pallet` nodes. Then, for each `Item` node A and `Pallet` node B corresponding with `(a:Item)-[r:ON]->(b:Pallet)` we can compute overhang directly based on relatively simple conditions (if *leftmost_point* of A is less than *leftmost_point* of B, then there is overhang on the left side - similar reasoning for front, back, and right sides) and store them either as properties in `Item` nodes, or possibly more conveniently directly in the `ON` relation, similar to how the `NEXT_TO` and `ON_TOP` relations have *gap* and *reason* properties.

If we assume the data is present in the `ON` relation, then investigating overhang can be done by looking at the average, highest, lowest, or other statistic of overhang over a path from a case down to the pallet, and see if for cases with STOs these significantly differ from those without STO. For the highest overhang for cases on a path down to the pallet, the corresponding Cypher query is similar to `MATCH (a:Item {sto: True})-[rels:ON_TOP*]->(b:Pallet) MATCH (a)-[s:ON]-(b) WITH MAX(s.overhang) as highestOverhang RETURN highestOverhang`.

8.3 Threats to Validity

At the start of the project, we had already decided on possibly using a graph database. By not keeping an open mind from the start, we may have influenced our thinking and prioritised certain research questions over others, causing us to potentially miss particular interesting questions that could also be worthwhile investigating. This is not necessary a threat to the validity of the presented work, but it does mean that using a graph database as data model for investigating STO errors is potentially a suboptimal approach.

That said, the current approach seems rather decent: throughout the work, we make only three assumptions. The first assumption, from Section 5.1, states that Telegram messages (described in Section 4.2) contain enough information to fully describe when STOs occur. The STO occurrences are present in the SCADA dataset (described in Section 4.1). We "verify" this assumption (using the script in Codeblock 5 in Appendix A.3) on an *arbitrarily* chosen subset of the SCADA dataset, and from that we conclude that it holds in general. **This does not need to be the case:** since the subset of the SCADA dataset used was arbitrarily chosen, there is little chance that it is representative of the entire dataset. This assumption can be mitigated by implementing the "hard" time-based join (explained in Section 5.1) in stead, which was not done due to time constraints.

The second and third assumptions are made in the introductory paragraph of Section 6, where it is explained how the combined dataset after integration (result from Section 5) must be *preprocessed* so it can be used for a Graph Database. The assumptions are explicitly mentioned in their respective section, but for completeness sake we show them here too.

1. A single telegram message corresponds to a single placed case.
2. If the i^{th} telegram message states that there was an STO, then the i^{th} placed case is deemed as having raised the STO.

We know that a single telegram *does not* correspond to a single placed case. This has been illustrated numerous times throughout the work. As a logical consequence of it not holding, we know that if the i^{th} telegram message states that there was an STO, then the i^{th} placed case *does not* need to be the case having raised the STO. The reason why we work with these assumptions, even if we already know they do not hold, is because they are necessary to continue with the analysis. If we cannot make these two assumptions, we cannot accurately pinpoint an STO case in a pallet. This in itself is an interesting avenue of research.

Besides the assumptions made, at the end of Section 5.2, we *aggregate* data to a higher level of abstraction. In particular, we refer to the grouping of Telegram messages (described in Section 4.2) per pallet, and redefining the Boolean indicator variables to be on a pallet level. This aggregation effectively loses crucial information (which specific Telegram related to the STO error). However, this crucial information can only be used *if we know how to relate a Telegram message to a case.*

8.4 Future Work

In Section 8.3 we hint at two potential questions that need an answer. The first is finding *how to accurately pinpoint for which case an STO error was raised.* Perhaps there are other datasets that include this information, which should then be integrated in the current prototypical integration pipeline explained in Section 5. If this is not recorded somewhere, then more thought is necessary on finding a way to record the information, so data-driven approaches like this work are feasible.

The second potential question from hinted at in Section 8.3 is *finding precisely how a Telegram message related to a placed case from the StackInfo dataset.* This most likely is related to previous paragraph, and it is not too far-fetched to think that by answering either question, both will be answered. Similar to previous paragraph, finding an answer to this question is interesting, as it better enables data-driven approaches like this work.

Besides the questions from Section 8.3, perhaps even more pressing, is finding *how to combat the data quality issue* discussed in Section 8.1. It seems very weird that cases are recorded as being stacked inside each other, while the STO camera works well enough to raise STO errors. Vanderlande might wish to investigate this phenomenon by looking at the STO camera's accuracy, as well as the code that produces the StackInfo dataset (described in Section 4.4). Even more, Vanderlande might wish to investigate if there is possible concern for missed STO errors due to inaccurate recorded placed Z values.

Yet another potentially interesting avenue for Vanderlande is hinted at in Section 4.4; sometimes, the *palletise_seq_nr* does not increase as a normal count. Perhaps this phenomenon is by itself an indicator for potential errors that are not yet recorded in the system.

Besides abnormalities about the data, in an ideal setting the proposed data model can be used to its full potential. It can be extended to an Event Knowledge Graph (proposed in [13]) and the techniques from said paper can then be applied. The graph can be further extended to the entire palletizer cell, as opposed to only pallets themselves, allowing Vanderlande to investigate all related errors to the cell in a new way. Other potential causes for STOs, besides the four from Section 3.3, can also be investigated using a Graph Database as data model (or even the integrated large table directly). Some examples of potential causes that have not been covered by this work, but might be interesting:

- Inaccurate (assumptions in) Teaching data.
- Inaccurate heuristics in LFL. *This is a particularly interesting idea, as the graph database models the physical structure of the stack. There may be interesting properties to be computed on the graph that can improve the heuristics.*
- Inaccurate tolerances for weight/size. *This too is a particularly interesting idea. The graph can be queried to find answers to ideas such as "the higher on the pallet, the higher the measurement errors, the more STOs".*
- Potential mechanical faults (one cell has proportionally more STO errors than others).
- Slanted palletizer lifts (STOs occur spatially only in a particular area, say the bottom right).

Besides future work for within Vanderlande, in academia this work opens many potentially interesting avenues of research. Recall the entire section on *Reliability of Machines* (Section 2.3), and that most (almost all) cited sources do not use a Graph Database as data model. It might be interesting to investigate to what extent Graph Databases can be used to improve existing methods for reliability, or even create entirely new methods.

9 Conclusion

In this work we have discovered which datasets are relevant to investigating underlying causes for STOs (see Figure 17 on how these datasets relate), and we give a proof of concept data integration pipeline combining these datasets in Section 5, delivering on half of outcome **D**2. After integrating all data, we have shown which properties should be present in a knowledge graph encoding the physical setting of the palletisation process in Section 6.1, fully delivering on desired outcome **D**1. We have given quite the explanation on how to implement this graph in Section 6.2, starting from the large table as retrieved at the end of Section 5, delivering on the other half of **D**2. Finally, we evaluated which questions about the palletisation process can currently be answered reliably on the graph, delivering on **D**3. We find that there is a **critical data quality issue** with respect to the recorded Z axis values of cases on pallets, causing the created graph to be unusable in its current state (and as such we did not answer hypothesis **HP** 1, **HP** 2, **HP** 3 or **HP** 4). We end the thesis with a discussion on the data quality issue (Section 8.1), and how we envision that the graph can be used if data was nice (Section 8.2). We strongly believe in the analytical power that graph databases bring, and as such recommend Vanderlande to look at the suggestions from Section 8.4 for making the graph usable.

List of Figures

List of Tables

List of Codeblocks, Scripts, and Queries

References

[1] "Company profile: About vanderlande - vanderlande industries," Jan 2022. [Online]. Available: https://www.vanderlande.com/about-vanderlande/company-profile/ 1

[2] "Warehousing evolutions: Storepick," Mar 2022. [Online]. Available: https://www.vanderlande.com/evolutions/storepick/ 1

[3] P. C. Kanellakis, "Elements of relational database theory," 1939. 4

[4] R. Angles, "The property graph database model," in *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018*, ser. CEUR Workshop Proceedings, D. Olteanu and B. Poblete, Eds., vol. 2100. CEUR-WS.org, 2018. [Online]. Available: http://ceur-ws.org/Vol-2100/paper26.pdf 4

[5] E. F. Codd, "Data models in database management," in *Proceedings of the 1980 workshop on Data abstraction, databases and conceptual modeling* -. ACM Press, 1980. [Online]. Available: https://doi.org/10.1145/800227.806891 4

[6] M. A. Rodriguez and P. Neubauer, "Constructions from dots and lines," 2010. [Online]. Available: https://arxiv.org/abs/1006.2361 4, 5

[7] neo4j, "Graph model - properties," 2022. [Online]. Available: https://neo4j.com/docs/getting-started/current/data-modeling/guide-data-modeling/ 4, 54

[8] "The neo4j cypher manual v4.4 - neo4j cypher manual," 2022. [Online]. Available: https://neo4j.com/docs/cypher-manual/4.4/ 5

[9] "Graph database use cases & solutions: Where to use a graph database," 2022. [Online]. Available: https://neo4j.com/use-cases/ 5

[10] "Supply chain graph database use cases: Data management & visualization," 2022. [Online]. Available: https://neo4j.com/use-cases/supply-chain-management/ 5

[11] "Graphs in life sciences — graph data science for life sciences — neo4j," 2022. [Online]. Available: https://neo4j.com/use-cases/life-sciences/ 5

[12] "Social network graph use cases — social media graph database — neo4j," 2022. [Online]. Available: https://neo4j.com/use-cases/social-network/ 5

[13] D. Fahland, "Process mining over multiple behavioral dimensions with event knowledge graphs," in *Lecture Notes in Business Information Processing.* Springer International Publishing, 2022, pp. 274–319. [Online]. Available: https://doi.org/10.1007/978-3-031-08848-3_9 5, 9, 52

[14] B. Kan, W. Zhu, G. Liu, X. Chen, D. Shi, and W. Yu, "Topology Modeling and Analysis of a Power Grid Network Using a Graph Database," *International Journal of Computational Intelligence Systems*, vol. 10, no. 1, pp. 1355–1363, Sep. 2017. [Online]. Available: https://www.atlantis-press.com/journals/ijcis/25883598 5

[15] P. E. Nalwoga Lutu, "Using Twitter Mentions and a Graph Database to Analyse Social Network Centrality," in *2019 6th International Conference on Soft Computing & Machine Intelligence (ISCMI).* Johannesburg, South Africa: IEEE, Nov. 2019, pp. 155–159. [Online]. Available: https://ieeexplore.ieee.org/document/9004313/ 5

[16] L. Diederichsen, K.-K. R. Choo, and N.-A. Le-Khac, "A graph database-based approach to analyze network log files," in *Network and System Security*, J. K. Liu and X. Huang, Eds. Cham: Springer International Publishing, 2019, pp. 53–73. 5

[17] R. kumar Kaliyar, “Graph databases: A survey,” in *International Conference on Computing, Communication & Automation.* Greater Noida, India: IEEE, May 2015, pp. 785–790. [Online]. Available: http://ieeexplore.ieee.org/document/7148480/ 5

[18] “The Best Techniques for Data Integration in 2021.” [Online]. Available: https://www.matillion.com/resources/blog/the-best-techniques-for-data-integration-in-2021 5

[19] “What is Data Integration? The Ultimate Guide.” [Online]. Available: https://www.matillion.com/what-is-data-integration-the-ultimate-guide/ 5

[20] “5 Data Integration Methods and Strategies.” [Online]. Available: https://www.talend.com/resources/data-integration-methods/ 5

[21] H. Lund, “Most common types of data integration methods.” [Online]. Available: https://www.rapidionline.com/blog/most-common-types-of-data-integration-methods 5

[22] N. Fatima, “Common Data Integration Techniques and Technologies Explained,” Sep. 2019. [Online]. Available: https://www.astera.com/type/blog/data-integration-techniques/ 5

[23] “What is Data Integration: Popular Methods And Applications,” May 2021. [Online]. Available: https://www.simplilearn.com/what-is-data-integration-article 5

[24] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel, “Context interchange,” *ACM Transactions on Information Systems*, vol. 17, no. 3, pp. 270–293, Jul. 1999. [Online]. Available: https://doi.org/10.1145/314516.314520 6

[25] M. Lenzerini, “Data integration: a theoretical perspective,” in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '02.* ACM Press, 2002. [Online]. Available: https://doi.org/10.1145/543613.543644 6

[26] A. Halevy, A. Rajaraman, and J. Ordille, “Data integration: The teenage years.” 01 2006, pp. 9–16. 6

[27] A. Doan, A. Halevy, and Z. Ives, *Principles of data integration.* Elsevier, 2012. 6

[28] P. Ziegler and K. R. Dittrich, “Data integration — problems, approaches, and perspectives,” in *Conceptual Modelling in Information Systems Engineering.* Springer Berlin Heidelberg, pp. 39–58. [Online]. Available: https://doi.org/10.1007/978-3-540-72677-7_3 6

[29] D. Calvanese, G. de Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, “Data Integration in Data Warehousing,” *International Journal of Cooperative Information Systems*, vol. 10, no. 03, pp. 237–271, Sep. 2001. [Online]. Available: https://doi.org/10.1142/s0218843001000345 6

[30] L. M. Haas, E. T. Lin, and M. A. Roth, “Data integration through database federation,” *IBM Systems Journal*, vol. 41, no. 4, pp. 578–596, 2002. [Online]. Available: https://doi.org/10.1147/sj.414.0578 6

[31] S. Sharma and R. Jain, “Modeling ETL process for data warehouse: An exploratory study,” in *2014 Fourth International Conference on Advanced Computing & Communication Technologies.* IEEE, Feb. 2014. [Online]. Available: https://doi.org/10.1109/acct.2014.100 6

[32] R. Wijaya and B. Pudjoatmodjo, “An overview and implementation of extraction-transformation-loading (ETL) process in data warehouse (case study: Department of agriculture),” in *2015 3rd International Conference on Information and Communication Technology (ICoICT).* IEEE, May 2015. [Online]. Available: https://doi.org/10.1109/icoict.2015.7231399 6

[33] “Definition of TRIBOLOGY.” [Online]. Available: https://www.merriam-webster.com/dictionary/tribology 7

[34] "Introduction to Stress and Equations of Motion." [Online]. Available: https://www.comsol.com/multiphysics/stress-and-equations-of-motion?parent=structural-mechanics-0182-202 7

[35] "Material Fatigue Definition." [Online]. Available: https://www.comsol.com/multiphysics/material-fatigue 7

[36] S. Kakani, *Material science*. New Age International (P) Ltd., Publishers, 2004. 7

[37] K. M. Blache, "Where do reliability engineers come from?" Nov 2013. [Online]. Available: https://reliabilityweb.com/articles/entry/where_do_reliability_engineers_come_from 7

[38] K. K. Aggarwal, *Reliability Engineering*. Springer Netherlands, 1993. [Online]. Available: https://doi.org/10.1007/978-94-011-1928-3 7

[39] P. D. T. O'Connor and A. Kleyner, *Practical Reliability Engineering*. Wiley, Nov. 2011. [Online]. Available: https://doi.org/10.1002/9781119961260 7

[40] A. Birolini, *Reliability Engineering*. Springer Berlin Heidelberg, 2004. [Online]. Available: https://doi.org/10.1007/978-3-662-05409-3 7

[41] E. Zio, "Reliability engineering: Old problems and new challenges," *Reliability Engineering & System Safety*, vol. 94, no. 2, pp. 125–141, Feb. 2009. [Online]. Available: https://doi.org/10.1016/j.ress.2008.06.002 7

[42] A. H. Tai, W.-K. Ching, and L. Chan, "Detection of machine failure: Hidden markov model approach," *Computers & Industrial Engineering*, vol. 57, no. 2, pp. 608–619, Sep. 2009. [Online]. Available: https://doi.org/10.1016/j.cie.2008.09.028 7

[43] S. R. Eddy, "What is a hidden markov model?" *Nature biotechnology*, vol. 22, no. 10, pp. 1315–1316, 2004. 7

[44] B. Roylance, "Machine failure and its avoidance - tribology's contribution to effective maintenance of critical machinery," in *Wear - Materials, Mechanisms and Practice*. John Wiley & Sons Ltd, Sep. 2014, pp. 425–452. [Online]. Available: https://doi.org/10.1002/9780470017029.ch16 7

[45] M. A. Costa, B. Wullt, M. Norrlöf, and S. Gunnarsson, "Failure detection in robotic arms using statistical modeling, machine learning and hybrid gradient boosting," *Measurement*, vol. 146, pp. 425–436, Nov. 2019. [Online]. Available: https://doi.org/10.1016/j.measurement.2019.06.039 7

[46] M. Riazi, O. Zaiane, T. Takeuchi, A. Maltais, J. Günther, and M. Lipsett, "Detecting the onset of machine failure using anomaly detection methods," in *Big Data Analytics and Knowledge Discovery*. Springer International Publishing, 2019, pp. 3–12. [Online]. Available: https://doi.org/10.1007/978-3-030-27520-4_1 7

[47] H. Che, S. Zeng, and J. Guo, "Reliability assessment of man-machine systems subject to mutually dependent machine degradation and human errors," *Reliability Engineering & System Safety*, vol. 190, p. 106504, Oct. 2019. [Online]. Available: https://doi.org/10.1016/j.ress.2019.106504 8

[48] S.-Z. Yu, "Hidden semi-markov models," *Artificial intelligence*, vol. 174, no. 2, pp. 215–243, 2010. 8

[49] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017. 8

[50] P. Li, X. Jia, J. Feng, F. Zhu, M. Miller, L.-Y. Chen, and J. Lee, "A novel scalable method for machine degradation assessment using deep convolutional neural network," *Measurement*, vol. 151, p. 107106, Feb. 2020. [Online]. Available: https://doi.org/10.1016/j.measurement.2019.107106 8

[51] K. Javed, R. Gouriveau, and N. Zerhouni, "Novel failure prognostics approach with dynamic thresholds for machine degradation," in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Nov. 2013. [Online]. Available: https://doi.org/10.1109/iecon.2013.6699844 8

[52] V. T. Tran, H. T. Pham, B.-S. Yang, and T. T. Nguyen, "Machine performance degradation assessment and remaining useful life prediction using proportional hazard model and support vector machine," *Mechanical Systems and Signal Processing*, vol. 32, pp. 320–330, Oct. 2012. [Online]. Available: https://doi.org/10.1016/j.ymssp.2012.02.015 8

[53] D. R. Cox, "Regression models and life-tables," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 34, no. 2, pp. 187–202, 1972. 8

[54] C. Gaimon and G. L. Thompson, "Optimal preventive and repair maintenance of a machine subject to failure," *Optimal Control Applications and Methods*, vol. 5, no. 1, pp. 57–67, Jan. 1984. [Online]. Available: https://doi.org/10.1002/oca.4660050105 8

[55] Z. Lu, W. Cui, and X. Han, "Integrated production and preventive maintenance scheduling for a single machine with failure uncertainty," *Computers & Industrial Engineering*, vol. 80, pp. 236–244, Feb. 2015. [Online]. Available: https://doi.org/10.1016/j.cie.2014.12.017 8, 9

[56] W. Weibull, "A statistical theory of the strength of materials," *Proc. Royal 4cademy Engrg Science*, vol. 15, 1939. 8

[57] T. Xia, L. Xi, X. Zhou, and J. Lee, "Condition-based maintenance for intelligent monitored series system with independent machine failure modes," *International Journal of Production Research*, vol. 51, no. 15, pp. 4585–4596, Aug. 2013. [Online]. Available: https://doi.org/10.1080/00207543.2013.775524 8

[58] K. W. Verhaegh, "Process mining for systems with automated batching," Master's thesis, Eindhoven University of Technology, 5612 AZ Eindhoven, August 2018. 9

[59] A. T. Pi, "Performance-aware conformance checking on material handling systems," Master's thesis, Eindhoven University of Technology, 5612 AZ Eindhoven, July 2019. 9

[60] neo4j, "Bloom - neo4j graph data platform," 2022. [Online]. Available: https://neo4j.com/product/bloom/ 9

[61] vis.js, "vis.js network documentation," 2022. [Online]. Available: https://visjs.github.io/vis-network/docs/network/ 9

[62] V. Chu, "Using event knowledge graphs to model multi-dimensional dynamics in a baggage handling systme," Master's thesis, Eindhoven University of Technology, 5612 AZ Eindhoven, April 2022. 9

[63] G. Halmetschlager-Funek, M. Suchi, M. Kampel, and M. Vincze, "An empirical evaluation of ten depth cameras: Bias, precision, lateral noise, different lighting conditions and materials, and multiple sensor setups in indoor environments," *IEEE Robotics & Automation Magazine*, vol. 26, no. 1, pp. 67–77, Mar. 2019. [Online]. Available: https://doi.org/10.1109/mra.2018.2852795 49

[64] R. Rao, "Stereo and 3d vision," University Lecture, 2009. [Online]. Available: https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf 49

A Appendices

A.1 Dataset Tables

Start time	**Error ID**	*Error Part*	**Duration (within window)**	**End time (within window)**	*Error Type*
2021-12-10 11:59:54.677	88.8-Some Error Message Here	8888.88.88		No end time within search window	ABC
2021-12-10 11:59:51.853	88.8-Some Error Message Here	8888.88.88		No end time within search window	DEF
2021-12-10 11:57:04.373	88.8-Some Error Message Here	8888.88.88		No end time within search window	GHI
2021-12-10 11:54:57.767	88.8-Some Error Message Here	8888.88.88	00:02:36.813	2021-12-10 11:57:34.580	JK
2021-12-10 11:54:57.703	88.8-Some Error Message Here	8888.88.88	00:02:42.874	2021-12-10 11:57:40.577	LM
2021-12-10 11:54:55.063	88.8-Some Error Message Here	8888.88.88	00:00:01.004	2021-12-10 11:54:56.067	NOP

Table 8: The SCADA dataset as extracted from the system - data has been anonymised.

time	error_part	blocked _place _position	blocked _flight _path	blocked _lift _shaft	missing _stack _surface	pallet_id
2021-12-10 00:58:47.696+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:40.404+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:34.512+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:31.824+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:22.814+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:18.890+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	TRUE	FALSE	pallet3
2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:12.890+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet3
2021-12-10 00:58:05.452+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet1
2021-12-10 00:57:58.937+0000	1024.56.82	FALSE	FALSE	FALSE	FALSE	pallet2
2021-12-10 00:57:51.144+0000	1024.56.82	FALSE	FALSE	FALSE	FALSE	pallet2
2021-12-10 00:57:47.254+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet1
2021-12-10 00:57:40.698+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet1
2021-12-10 00:57:35.006+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet1
2021-12-10 00:57:30.036+0000	1024.56.82	FALSE	FALSE	FALSE	FALSE	pallet2
2021-12-10 00:57:24.423+0000	1024.56.82	FALSE	FALSE	FALSE	FALSE	pallet2
2021-12-10 00:57:18.057+0000	1014.56.82	FALSE	FALSE	FALSE	FALSE	pallet1
2021-12-10 00:56:52.816+0000	1014.56.82	FALSE	FALSE	TRUE	FALSE	pallet1
2021-12-10 00:56:52.816+0000	1024.56.82	TRUE	FALSE	FALSE	FALSE	pallet2

Table 9: Toy example of the Telegram dataset after transformation.

A.2 Data Model

Based on the four hypotheses, and on further implementation details listed in Section 6.2, we need following nodes and relations in the graph.

Nodes:

```
i:Item {
    placed{XYZ}{1234}: Integer,
    expected{XYZ}{1234}: Integer,
    offCenterX: Integer,
    offCenterY: Integer,
    stacking_method: List<String>,
    rightmost_point: Integer,
    leftmost_point: Integer,
    lowest_point: Integer,
    highest_point: Integer,
    frontmost_point: Integer,
    backmost_point: Integer,
    [... properties from Teaching that are deemed interesting]
}

p:Pallet {
    width: Integer
    length: Integer
    placement: Point2D
    [pallet properties that are deemed interesting]
}
```

Relations:

```
r:ON_TOP_OF {
    reason: String
        gap: Integer
}

r:NEXT_TO {
    reason: String
}

r:OVERHANG {
        amount: Integer,
        reason: String
}
```

A.3 Scripts

```python
from math import floor
from math import ceil

# Ensure they're sorted by time
sto_scada_df = sto_scada_df.sort_values(by="start_time", ignore_index=True)
telegram_df = telegram_df.sort_values(by="time", ignore_index=True)

assumptions_hold = []
count = 0
max_count = 100
for idx, entry in sto_scada_df.iterrows():
  count = count + 1
  # Grab the bounds on time, with 10 seconds both way
  l = floor(entry["start_time"]) - (60)
```

```

  if type(entry["end_time"]) == type(0.0):
    u = ceil(entry["end_time"]) + (60)
  else:
    u = l + (10 * 24 * 60 * 60)


  # Scada mentions that the palletizer is this one
  wanted = entry["palletizer"]

  # Reasons why there's an STO
  reasons = []
  if entry["blocked_lift_shaft"]:
    reasons.append("blocked_lift_shaft")

  if entry["missing_stack_surface"]:
    reasons.append("missing_stack_surface")

  if entry["blocked_place_position"]:
    reasons.append("blocked_place_position")

  if entry["blocked_flight_path"]:
    reasons.append("blocked_flight_path")

  # Check if said palletizer is in the telegrams filtered on times
  test_me = telegram_df[telegram_df["time"] > l]
  test_me = test_me[test_me["time"] < u]

  # Filter on reasons
  for reason in reasons:
    if reason:
      test_me = test_me[test_me[reason]]

  assumptions_hold.insert(idx, wanted in test_me['palletizer'].values)
  if count > max_count:
    break


print(False in assumptions_hold)
```

Script 5: Script to check assumption that Telegrams contain enough information for STO errors.

```
<Query Kind="Program">
  // [ relative dependencies omitted ]
  <NuGetReference>CsvHelper</NuGetReference>
  <Namespace>CsvHelper</Namespace>
  <Namespace>CsvHelper.Configuration</Namespace>
  <Namespace>CsvHelper.Configuration.Attributes</Namespace>
  <Namespace>CsvHelper.Expressions</Namespace>
  <Namespace>CsvHelper.TypeConversion</Namespace>
  <Namespace>System.Globalization</Namespace>
  // [ Vanderlande namespaces omitted ]
  <RuntimeVersion>3.1</RuntimeVersion>
</Query>

```

```
/// <summary>
/// Main processing function.
/// Grabs relevant information from LFL files, and then creates a CSV from it.
/// </summary>
public static void Main() {
  // Keep track of running time.
  System.Diagnostics.Stopwatch timer = System.Diagnostics.Stopwatch.StartNew();

  // Path to LFL ZIP files.
  string BASE_PATH = @"C:\Users\daniel\Documents\_Project\_datasets\LFL";

  // Loop through all folders, each containing at most 50 ZIP files
  int NUMBER_OF_FOLDERS = 55;
  for (int i = 1; i < NUMBER_OF_FOLDERS + 1; i++) {
    // Time Reading the files
    var T = System.Diagnostics.Stopwatch.StartNew();
    string folder = Path.Join(BASE_PATH, i.ToString());
    List<CSVItem> items = ProcessFolder(folder);
    T.Stop();
    Console.WriteLine("Reading folder {0} took {1} ms", i, T.ElapsedMilliseconds);

    // Time writing CSV
    T = System.Diagnostics.Stopwatch.StartNew();
    string outFile = Path.Join(BASE_PATH, "_processed", i.ToString() + ".csv");
    toCSV(outFile, items); // assumes '_processed' folder already exists
    T.Stop();
    Console.WriteLine("Writing {0} took {1} ms", outFile, T.ElapsedMilliseconds);
  }
  // Write single line containing all time.
  timer.Stop();
  Console.WriteLine("Everything took {0} ms!", timer.ElapsedMilliseconds);
}

/// <summary>
/// This is the data that we extract from the recipes.
/// Most fields are straightforward. Comments added for clarity.
/// </summary>
class CSVItem {
  // The order itself
  public string OrderId { get; set; }
  public string RecipeId { get; set; } // Linking with stackinfo is done on RecipeId.
  // Stack KPIs
  public int StackHeight { get; set; }
  public long StackWeight { get; set; }
  public long StackVolume { get; set; }
  public int NrCasesInStack { get; set; }
  public double StackGroupCoherence { get; set; }
  public double StackArticleCoherence { get; set; }
  public double StackFillrate { get; set; }
  // Per Case fields
  public string CaseId { get; set; }
  public int SequenceId { get; set; }
  public int CompletedHeigth { get; set; }
  public string StackingMethod { get; set; }
}
```

```
/// <summary>
/// Processes a folder with LFL zip files.
/// </summary>
static List<CSVItem> ProcessFolder(string RECIPES) {
  // Read ZIP files in parallel
  // There is NO GUARANTEE on the order!
  List<OrderData> orders = MessageFiles.ListFiles(RECIPES)
                    .AsParallel()
                    .Select(f => f.Read())
                    .ToList();

  // Sort by name: We do this for consistency.
  // If we don't sort, runs are subject to race conditions due to the parallel loading.
  // We DO NOT want to do this for the large dataset,
  // since it'll take a lot of time (sorting is O(n log (n))).

  /*
  orders.Sort((order1, order2) => {
    return order1.Name.CompareTo(order2.Name);
  });
  */

  // Create a collection of items that we want to write to CSV
  List<CSVItem> data = new List<CSVItem>();

  // Loop through all orders
  foreach (OrderData order in orders) {
    // This should be the order ID.
    string orderId = order.Order.OrderId;

    // Create mapping from RecipeId to its KPIs.
    Dictionary<string, IStackKpi> recipeToKPI = getMapFromRecipeToKPI(order);

    // Loop through all recipes in the order
    foreach (StackingRecipe recipe in order.Recipe.StackingRecipes) {
      // Grap recipe ID: This is what we need to match with stackinfo CSVs.
      string recipeId = recipe.IdString;
      // Grab KPIs from mapping
      IStackKpi kpi = recipeToKPI[recipeId];

      int StackHeight = kpi.Height;
      long StackWeight = kpi.Weight;
      long StackVolume = kpi.Volume;
      int NrCases = kpi.NrCases;
      double GroupCoherence = kpi.GroupCoherence;
      double ArticleCoherence = kpi.ArticleCoherence;
      double Fillrate = kpi.Fillrate;

      // Map each caseID to its related stacking method
      Dictionary<Tuple<string, Vector3D>, string> toAction = convertDictionary(
        RecipeActions.GetStackerPlacementDetails(order.OrderResult, recipe)
      );

      // Loop over all stacking sequence groups
      foreach (var sequenceGroup in recipe.SequenceGroups) {
        // Extract the completed height; this is 'stackFloorHeightMm' in stackinfo CSVs.
```

```
        int completedHeight = sequenceGroup.CompletedHeight;
        int sequenceGroupId = sequenceGroup.Id;

        // Loop through each step, and grab the case IDs and related stacking method.
        foreach (var stackingStep in sequenceGroup.StackingSteps) {
          // Define the variables here
          string caseId;
          string action;

          // Set variables accordingly
          if (stackingStep.IsSlipsheetStep) {
            // [REDACTED COMMENT]
            caseId = stackingStep.Position.ItemId;
            action = null;
          } else if (stackingStep.IsProductStep) {
            // Happy flow
            caseId = stackingStep.Position.ItemId;
            action = toAction[new(caseId, stackingStep.Position.Position)];
          } else {
            // This should never happen
            caseId = null;
            action = null;
          }

          // Add a new item to the collections of items to write to CSV
          data.Add(new CSVItem {
            OrderId = orderId,
            RecipeId = recipeId,
            StackHeight = StackHeight,
            StackWeight = StackWeight,
            StackVolume = StackVolume,
            NrCasesInStack = NrCases,
            StackGroupCoherence = GroupCoherence,
            StackArticleCoherence = ArticleCoherence,
            StackFillrate = Fillrate,
            CaseId = caseId,
            SequenceId = sequenceGroupId,
            CompletedHeigth = completedHeight,
            StackingMethod = action,
          });
        }
      }
    }
  }

  // Return the list of items
  return data;
}

/// <summary>
/// Writes a list of CSVItems to a CSV file.
/// </summary>
static void toCSV(string path, List<CSVItem> data) {
  using (var writer = new StreamWriter(path))
  using (var csv = new CsvWriter(writer, CultureInfo.InvariantCulture)) {
    csv.WriteRecords(data);
```

```
  }
}

/// <summary>
/// Creates a dictionary from a guid (~unique case ID in LFL) to the action it was packed with.
/// </summary>
static Dictionary<Tuple<string, Vector3D>, string> convertDictionary(
  IDictionary<ProductPosition, StackerPlacementDetails> mapping
) {
  // Create empty dictionary
  Dictionary<Tuple<string, Vector3D>, string> dict
      = new Dictionary<Tuple<string, Vector3D>, string>();

  // Loop through each item in the mapping as returned by LFL, retrieve wanted items,
  // and add to the mapping.
  foreach(var item in mapping) {
    string caseId = item.Key.ItemId;
    Vector3D pos = item.Key.Position;
    string action = item.Value.Action;
    dict.Add(new (caseId, pos), action);
  }

  return dict;
}

/// <summary>
/// Creates a dictionary from a recipe ID to its KPIs and returns it.
/// </summary>
static Dictionary<string, IStackKpi> getMapFromRecipeToKPI(OrderData order) {
  // Create empty dictionary
  Dictionary<string, IStackKpi> recipeToKPI = new Dictionary<string, IStackKpi>();

  // Loop through KPIs, and add dictionary entry
  foreach (IStackKpi kpi in order.OrderResult.OrderKpi.StackKpis) {
    recipeToKPI.Add(kpi.RecipeId, kpi);
  }

  return recipeToKPI;
}
```

Script 6: Anonymised version of the C# script to preprocess LFL `.zip` files, using Vanderlande tooling.

```
    def lfl_match(recipe):
  """
  Function that matches an LFL recipe to a StackInfo CSV, which takes as input
  "longform" dataframes as explained in the notebook. It should only be run after
  two sanity checks, which are automatically done.

  Assumes that the set of cases and number of items is identical!
  """
  # Grab the ID from this row
  recipe_id = recipe.index

  # Keep track of the matches as list
```

```
    matches = []
    matches_okay = []

    # Initialise dictionary with False.
    # This will keep track of all matched 'palletiseSeqNr' on a recipe basis.
    matched = defaultdict(lambda: False)

    # Loop through LFL sequence groups
    for (lflSequenceGroup, lflCase) in zip(recipe["SequenceId"], recipe["CaseId"]):
      # Boolean for verifying single case
      foundMatch = False

      # Loop through stackinfo
      for (siSequenceNumber, siCase) in zip(recipe["palletiseSeqNr"], recipe["caseId"]):
        # Skip load carrier
        if pd.isnull(siCase):
          continue
        # Skip already used cases
        elif matched[siSequenceNumber]:
          continue
        # Skip siCase if we can't match
        elif lflCase != siCase:
          continue
        # Same case for an unmatched item: MATCH!
        else:
          foundMatch = True
          matched[siSequenceNumber] = True
          break

      # We can match; add tuple
      if foundMatch:
        # Tuple: lflSequenceGroup, siSequenceNumber, caseID
        matches.append((lflSequenceGroup, siSequenceNumber, lflCase))
        matches_okay.append(True)
      else:
        # Not possible to match this particular one, so set to False
        matches.append((None, None, None))
        matches_okay.append(False)
```

Script 7: Script to match LFL data to StackInfo data. Used for joining all datasets together.

```
import os

# Script that produces all queries for NEXT_TO.
if __name__ == "__main__":
    # Where to output files
    RELATION_PATH = r"/mnt/c/Users/daniel/Documents/_Project/neo4j/queries/generated/"
    # Whether or not to output queries for each situation
    SINGLE_FILE = True

    X = ["R_EDGE", "BOTH_X", "L_EDGE", "NONE_X"]  # 4 situations
    Y = ["B_EDGE", "BOTH_Y", "T_EDGE", "NONE_Y"]  # 4 situations

    # Encodes when (a:Item) is KEY w.r.t (b:Item) in a check to be executed in Cypher
    mp = {
```

```
        # X
        "R_EDGE": [
            "a.leftmost_point < b.leftmost_point",
            "a.rightmost_point <= b.rightmost_point",  # equal to right
            "a.rightmost_point > b.leftmost_point"
        ],
        "BOTH_X": [
            "a.leftmost_point >= b.leftmost_point",  # equal to left
            "a.rightmost_point <= b.rightmost_point"  # equal to right
        ],
        "L_EDGE": [
            "a.leftmost_point >= b.leftmost_point",  # equal to left
            "a.rightmost_point > b.rightmost_point",
            "a.leftmost_point < b.rightmost_point"
        ],
        "NONE_X": [
            "a.leftmost_point < b.leftmost_point",
            "a.rightmost_point > b.rightmost_point"
        ],
        # Y
        "B_EDGE": [
            # equal to front (bottom)
            "a.frontmost_point >= b.frontmost_point",
            "a.backmost_point > b.backmost_point",
            "a.frontmost_point < b.backmost_point"
        ],
        "BOTH_Y": [
            # equal to front (bottom)
            "a.frontmost_point >= b.frontmost_point",
            "a.backmost_point < b.backmost_point"  # equal to back (top)
        ],
        "T_EDGE": [
            "a.frontmost_point < b.frontmost_point",
            "a.backmost_point <= b.backmost_point",  # equal to back (top)
            "a.backmost_point > b.frontmost_point"
        ],
        "NONE_Y": [
            "a.frontmost_point < b.frontmost_point",
            "a.backmost_point > b.backmost_point"
        ]
    }

    # Generate cypher queries
    all_lines = [
        "// Relation: ON_TOP.",
        "// Contains 16 sub-queries to create.",
        "\n\n\n"
    ]
    all_rels = []
    all_filename = "relation_ON_TOP_COMPLETE.cypher"
    for x in X:
        for y in Y:
            rel = f"{x}_{y}_ABOVE"
            all_rels.append(rel)

            filename = f"relation_{rel}.cypher"
```

```
            # Relation: creation
            lines = [
                f"// Relation: {rel}",
                "MATCH",
                "    (a:Item),",
                "    (b:Item)",
                "WHERE",
                "    id(a) <> id(b)",
                "    AND a.pallet_id = b.pallet_id",
                # "    AND a.lowest_point >= b.highest_point"  # Z-axis check
                "    AND a.lowest_point + 59 > b.highest_point"  # Z-axis check + constant variable
                ↪ for max case height (60)
            ]
            for reason in mp[x]:
                lines.append(f"    AND {reason}")
            for reason in mp[y]:
                lines.append(f"    AND {reason}")
            lines += [
                "MERGE",
                f"    (a)-[r:{rel}]->(b);"
            ]

            # Whitelines
            lines.append("\n")

            # Relation: deletion
            lines += [
                f"// Relation: {rel} remove extra",
                "MATCH",
                f"    (a:Item)-[r:{rel}]->(b:Item)-[t:{rel}]->(c:Item),",
                f"    (a:Item)-[q:{rel}]->(c:Item)",
                "WHERE",
                "    id(a) <> id(b)",
                "    AND id(a) <> id(c)",
                "    AND id(b) <> id(c)",
                "DELETE",
                "    q;"
            ]

            # Write to single file
            if SINGLE_FILE:
                with open(os.path.join(RELATION_PATH, filename), 'w') as f:
                    print(f"Writing file: '{filename}'...")
                    f.write('\n'.join(lines))

            # Append to big list of lines
            all_lines += lines

    # Create ON TOP relation with reason property
    create_on_top = ["// Relation: ON_TOP { reason }"]
    for rel in all_rels:
        create_on_top += [
            "MATCH",
            f"    (a:Item) -[:{rel}]->(b:Item)",
            "WHERE",
            "    id(a) <> id(b)",
```

```
            "CREATE",
            "    (a)-[:ON_TOP {reason: " + '"' + rel + '"' + "}]->(b);\n"
        ]

    # Write to single file
    if SINGLE_FILE:
        with open(os.path.join(RELATION_PATH, r"relation_ON_TOP-reason.cypher"), 'w') as f:
            print(f"Writing file: 'relation_ON_TOP-reason.cypher'...")
            f.write('\n'.join(create_on_top))

    # Add gap property
    add_gap = [
        "// Relation: ON_TOP { gap }",
        "MATCH",
        "    (a:Item) -[r:ON_TOP]-> (b:Item)",
        "WITH",
        "    r,"
        "    a.lowest_point - b.highest_point AS gap",
        "SET",
        "    r.gap = gap;"
    ]

    # Write to single file
    if SINGLE_FILE:
        with open(os.path.join(RELATION_PATH, r"relation_ON_TOP-gap.cypher"), 'w') as f:
            print(f"Writing file: 'relation_ON_TOP-gap.cypher'...")
            f.write('\n'.join(add_gap))

    # Remove double created relations
    remove = [
        "// Relation: ON_TOP remove extra",
        "MATCH",
        "    (a:Item)-[r:ON_TOP]->(b:Item)-[t:ON_TOP]->(c:Item),",
        "    (a:Item)-[q:ON_TOP]->(c:Item)",
        "WHERE",
        "    id(a) <> id(b)",
        "    AND id(a) <> id(c)",
        "    AND id(b) <> id(c)",
        "DELETE",
        "    q;"
    ]

    # Write to single file
    if SINGLE_FILE:
        with open(os.path.join(RELATION_PATH, r"relation_ON_TOP-remove.cypher"), 'w') as f:
            print(f"Writing file: 'relation_ON_TOP-remove.cypher'...")
            f.write('\n'.join(remove))

    # Append to big list of lines
    all_lines += create_on_top
    all_lines += add_gap
    all_lines += remove

    # Write to big file
    with open(os.path.join(RELATION_PATH, all_filename), 'w') as f:
        print(f"Writing file: '{all_filename}'...")
```

```
        f.write('\n'.join(all_lines))
```

Script 8: Query generator for all possible situations encoding the `ON_TOP` relation.

```
import os

# Script that produces all queries for NEXT_TO.
if __name__ == "__main__":
    RELATION_PATH = r"/mnt/c/Users/daniel/Documents/_Project/neo4j/queries/generated/"
    # Whether or not to output queries for each situation
    SINGLE_FILE = True

    sides = ["L", "F", "R", "B"]  # 4 edges to consider

    Z = ["HIGH_Z", "LOW_Z", "BIG_Z", "SMALL_Z"]  # 4 situations
    Y = ["TOP_Y", "MID_Y", "BOT_Y", "BIG_Y"]  # sides L+R
    X = ["LEFT_X", "MID_X", "RIGHT_X", "LONG_X"]  # sides F+B

    # Encodes when (a:Item) is KEY w.r.t (b:Item) in a check to be executed in Cypher
    mp = {
        # sides
        "L": [
            "a.rightmost_point < b.leftmost_point",
        ],
        "F": [
            "a.backmost_point < b.frontmost_point"
        ],
        "R": [
            "a.leftmost_point > b.rightmost_point"
        ],
        "B": [
            "a.frontmost_point > b.backmost_point"
        ],

        # XZ-plane
        "HIGH_Z": [
            "a.lowest_point < b.highest_point",
            "a.lowest_point >= b.lowest_point",  # equality with bot
            "a.highest_point > b.highest_point"
        ],
        "LOW_Z": [
            "a.lowest_point < b.lowest_point",
            "a.highest_point > b.lowest_point",
            "a.highest_point <= b.highest_point"  # equality with top
        ],
        "BIG_Z": [
            "a.highest_point > b.highest_point",
            "a.lowest_point < b.lowest_point"
        ],
        "SMALL_Z": [
            "a.highest_point <= b.highest_point",  # equality with top
            "a.lowest_point >= b.lowest_point"     # equality with bot
        ],

        # XY-plane, L+R sides
```

```
        "TOP_Y": [
            "a.frontmost_point < b.backmost_point",
            "a.frontmost_point >= b.frontmost_point",  # equality with front
            "a.backmost_point > b.backmost_point"
        ],
        "MID_Y": [
            "a.frontmost_point < b.frontmost_point",
            "a.backmost_point > b.frontmost_point",
            "a.backmost_point <= b.backmost_point"  # equality with back
        ],
        "BOT_Y": [
            "a.backmost_point > b.backmost_point",
            "a.frontmost_point < b.frontmost_point"
        ],
        "BIG_Y": [
            "a.backmost_point <= b.backmost_point",  # equality with back
            "a.frontmost_point >= b.frontmost_point"    # equality with front
        ],

        # XY-plane, F+B sides
        "LEFT_X": [
            "a.leftmost_point < b.rightmost_point",
            "a.leftmost_point >= b.leftmost_point",  # equality with left
            "a.rightmost_point > b.rightmost_point"
        ],
        "MID_X": [
            "a.leftmost_point < b.leftmost_point",
            "a.rightmost_point > b.leftmost_point",
            "a.rightmost_point <= b.rightmost_point"  # equality with right
        ],
        "RIGHT_X": [
            "a.rightmost_point > b.rightmost_point",
            "a.leftmost_point < b.leftmost_point"
        ],
        "LONG_X": [
            "a.rightmost_point <= b.rightmost_point",  # equality with right
            "a.leftmost_point >= b.leftmost_point"    # equality with left
        ]
    }

    # Generate cypher queries
    all_lines = [
        "// Relation: NEXT_TO.",
        "// Contains 64 sub-queries to create.",
        " \n \n \n"
    ]
    all_rels = []
    all_filename = "relation_NEXT_TO_COMPLETE.cypher"

    for side in sides:
        for z in Z:
            # Decide which names to use based on L+R / F+B
            O = Y if side == "L" or side == "R" else X
            for o in O:
                rel = f"{side}_{z}_{o}"
                all_rels.append(rel)
```

```
                filename = f"relation_{rel}.cypher"
                # Relation: creation
                lines = [
                    f"// Relation: {rel}",
                    "MATCH",
                    "    (a:Item),",
                    "    (b:Item)",
                    "WHERE",
                    "    id(a) <> id(b)",
                    "    AND a.pallet_id = b.pallet_id",
                ]
                for reason in mp[side]:
                    lines.append(f"    AND {reason}")
                for reason in mp[z]:
                    lines.append(f"    AND {reason}")
                for reason in mp[o]:
                    lines.append(f"    AND {reason}")
                lines += [
                    "MERGE",
                    f"    (a)-[r:{rel}]->(b);"
                ]

                # Whitelines
                lines.append("\n")

                # Relation: deletion
                lines += [
                    f"// Relation: {rel} remove extra",
                    "MATCH",
                    f"    (a:Item)-[r:{rel}]->(b:Item)-[t:{rel}]->(c:Item),",
                    f"    (a:Item)-[q:{rel}]->(c:Item)",
                    "WHERE",
                    "    id(a) <> id(b)",
                    "    AND id(a) <> id(c)",
                    "    AND id(b) <> id(c)",
                    "DELETE",
                    "    q;"
                ]

                if SINGLE_FILE:
                    # Write to single file
                    with open(os.path.join(RELATION_PATH, filename), 'w') as f:
                        print(f"Writing file: '{filename}'...")
                        f.write('\n'.join(lines))

                # Append to big list of lines
                all_lines += lines

    # Create NEXT_TO relation with reason property
    create_next_to = ["// Relation: NEXT_TO { reason }"]
    for rel in all_rels:
        create_next_to += [
            "MATCH",
            f"    (a:Item) -[:{rel}]->(b:Item)",
            "WHERE",
```

```
                "    id(a) <> id(b)",
                "CREATE",
                "    (a)-[:NEXT_TO {reason: " + '"' + rel + '"' + "}]->(b);",
            ]

    # Write to single file
    if SINGLE_FILE:
        with open(os.path.join(RELATION_PATH, r"relation_NEXT_TO-reason.cypher"), 'w') as f:
            print(f"Writing file: 'relation_NEXT_TO-reason.cypher'...")
            f.write('\n'.join(create_next_to))

    """
    This can not be done so naively. There will be (probably many) cases where this matches something
↪   we want to keep. For now, it's a comment.

    # Remove double created relations
    remove = [
        "// Relation: NEXT_TO remove extra",
        "MATCH",
        "    (a:Item)-[r:NEXT_TO]->(b:Item)-[t:NEXT_TO]->(c:Item),",
        "    (a:Item)-[q:NEXT_TO]->(c:Item)",
        "WHERE",
        "    id(a) <> id(b)",
        "    AND id(a) <> id(c)",
        "    AND id(b) <> id(c)"
        "DELETE",
        "    q;"
    ]

    # Write to single file
    with open(os.path.join(RELATION_PATH, r"relation_NEXT_TO-remove.cypher"), 'w') as f:
        print(f"Writing file: 'relation_NEXT_TO-remove.cypher'...")
        f.write('\n'.join(remove))
    """


    # Append to big list of lines
    all_lines += create_next_to
    # all_lines += remove

    # Write to big file
    with open(os.path.join(RELATION_PATH, all_filename), 'w') as f:
        print(f"Writing file: '{all_filename}'...")
        f.write('\n'.join(all_lines))
```

Script 9: Query generator for all possible situations encoding the NEXT_TO relation.

A.4 Queries

```
// Node: Item
:auto LOAD CSV WITH HEADERS FROM 'file:///cases.csv' AS row
CALL {
    WITH row
    CREATE (i:Item {
```

```
        pallet_id: row.pallet_id,
        palletiseSeqNr: toIntegerOrNull(row.palletiseSeqNr),
        caseId: toIntegerOrNull(row.caseId),
        stackFloorHeightMm: toIntegerOrNull(row.stackFloorHeightMm),
        expected1: point({
            x: toIntegerOrNull(row.expectedX1),
            y: toIntegerOrNull(row.expectedY1),
            z: toIntegerOrNull(row.expectedZ1)
        }),
        expected2: point({
            x: toIntegerOrNull(row.expectedX2),
            y: toIntegerOrNull(row.expectedY2),
            z: toIntegerOrNull(row.expectedZ2)
        }),
        expected3: point({
            x: toIntegerOrNull(row.expectedX3),
            y: toIntegerOrNull(row.expectedY3),
            z: toIntegerOrNull(row.expectedZ3)
        }),
        expected4: point({
            x: toIntegerOrNull(row.expectedX4),
            y: toIntegerOrNull(row.expectedY4),
            z: toIntegerOrNull(row.expectedZ4)
        }),
        placed1: point({
            x: toIntegerOrNull(row.placedX1),
            y: toIntegerOrNull(row.placedY1),
            z: toIntegerOrNull(row.placedZ1)
        }),
        placed2: point({
            x: toIntegerOrNull(row.placedX2),
            y: toIntegerOrNull(row.placedY2),
            z: toIntegerOrNull(row.placedZ2)
        }),
        placed3: point({
            x: toIntegerOrNull(row.placedX3),
            y: toIntegerOrNull(row.placedY3),
            z: toIntegerOrNull(row.placedZ3)
        }),
        placed4: point({
            x: toIntegerOrNull(row.placedX4),
            y: toIntegerOrNull(row.placedY4),
            z: toIntegerOrNull(row.placedZ4)
        }),
        placementId: row.placementId,
        waypoint2: point ({
            x: toIntegerOrNull(row.waypoint2X),
            y: toIntegerOrNull(row.waypoint2Y),
            z: toIntegerOrNull(row.waypoint2Z)
        }),
        releasePosition: point({
            x: toIntegerOrNull(row.releasePositionX),
            y: toIntegerOrNull(row.releasePositionY),
            z: toIntegerOrNull(row.releasePositionZ)
        }),
        offCenterX: toIntegerOrNull(row.offCenterX),
```

```
        offCenterY: toIntegerOrNull(row.offCenterY),
        CompletedHeigth: toIntegerOrNull(row.CompletedHeigth),
        StackingMethod: row.StackingMethod,
        blocked_flight_path: toBooleanOrNull(row.blocked_flight_path),
        blocked_place_position: toBooleanOrNull(row.blocked_place_position),
        blocked_lift_shaft: toBooleanOrNull(row.blocked_lift_shaft),
        missing_stack_surface: toBooleanOrNull(row.missing_stack_surface),
        leftmost_point: toIntegerOrNull(row.leftmost_point),
        rightmost_point: toIntegerOrNull(row.rightmost_point),
        frontmost_point: toIntegerOrNull(row.frontmost_point),
        backmost_point: toIntegerOrNull(row.backmost_point),
        lowest_point: toIntegerOrNull(row.lowest_point),
        highest_point: toIntegerOrNull(row.highest_point),
        TEACHING_LENGTH: toIntegerOrNull(row.TEACHING_LENGTH),
        TEACHING_WIDTH: toIntegerOrNull(row.TEACHING_WIDTH),
        TEACHING_HEIGHT: toIntegerOrNull(row.TEACHING_HEIGHT),
    })
} IN TRANSACTIONS OF 250 ROWS
```

Cypher Query 10: Cypher query for `Item` node.

```
// Node: Pallet
:auto LOAD CSV WITH HEADERS FROM 'file:///pallets.csv' AS row
CALL {
        WITH row
        CREATE (p:Pallet {
                pallet_id: row.pallet_id,
                palletizer: row.palletizer,
                suborder_id: row.suborder_id,
                order_id: row.order_id,
                StackHeight: toFloatOrNull(row.StackHeight),
                StackWeight: toFloatOrNull(row.StackWeight),
                StackVolume: toFloatOrNull(row.StackVolume),
                NrCasesInStack: toIntegerOrNull(row.NrCasesInStack),
                StackGroupCoherence: toFloatOrNull(row.StackGroupCoherence),
                StackArticleCoherence: toFloatOrNull(row.StackArticleCoherence),
                StackFillrate: toFloatOrNull(row.StackFillrate),
        })
} IN TRANSACTIONS OF 250 ROWS
```

Cypher Query 11: Cypher query for `Pallet` node.

```
// Relation: ON
MATCH
    (i:Item),
    (p:Pallet)
WHERE
    i.pallet_id = p.pallet_id
CREATE
    (i)-[r:ON]->(p)
```

Cypher Query 12: Cypher query for `ON` relation.

```
// Relation: PLACED_BEFORE
MATCH
    (a:Item),
    (b:Item)
WHERE
    a.pallet_id = b.pallet_id
    AND id(a) = id(b) - 1
CREATE
    (a)-[r:PLACED_BEFORE]->(b)
```

Cypher Query 13: Cypher query for PLACED_BEFORE relation.

```
// Set STO property on last node in path
MATCH
        (a:Item)
OPTIONAL MATCH
        (a)-[:PLACED_BEFORE]->(b)
WITH
        a, b
WHERE
        b IS NULL
SET
        a.STO = True
```

Cypher Query 14: Cypher query for set_STO_property.

```
// Relation: ON_TOP.
// Contains 16 sub-queries to create.




// Relation: R_EDGE_B_EDGE_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.backmost_point
MERGE
    (a)-[r:R_EDGE_B_EDGE_ABOVE]->(b);


// Relation: R_EDGE_B_EDGE_ABOVE remove extra
MATCH
```

```
    (a:Item)-[r:R_EDGE_B_EDGE_ABOVE]->(b:Item)-[t:R_EDGE_B_EDGE_ABOVE]->(c:Item),
    (a:Item)-[q:R_EDGE_B_EDGE_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_EDGE_BOTH_Y_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point < b.backmost_point
MERGE
    (a)-[r:R_EDGE_BOTH_Y_ABOVE]->(b);


// Relation: R_EDGE_BOTH_Y_ABOVE remove extra
MATCH
    (a:Item)-[r:R_EDGE_BOTH_Y_ABOVE]->(b:Item)-[t:R_EDGE_BOTH_Y_ABOVE]->(c:Item),
    (a:Item)-[q:R_EDGE_BOTH_Y_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_EDGE_T_EDGE_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point <= b.backmost_point
    AND a.backmost_point > b.frontmost_point
MERGE
    (a)-[r:R_EDGE_T_EDGE_ABOVE]->(b);


// Relation: R_EDGE_T_EDGE_ABOVE remove extra
MATCH
    (a:Item)-[r:R_EDGE_T_EDGE_ABOVE]->(b:Item)-[t:R_EDGE_T_EDGE_ABOVE]->(c:Item),
```

```
    (a:Item)-[q:R_EDGE_T_EDGE_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_EDGE_NONE_Y_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:R_EDGE_NONE_Y_ABOVE]->(b);


// Relation: R_EDGE_NONE_Y_ABOVE remove extra
MATCH
    (a:Item)-[r:R_EDGE_NONE_Y_ABOVE]->(b:Item)-[t:R_EDGE_NONE_Y_ABOVE]->(c:Item),
    (a:Item)-[q:R_EDGE_NONE_Y_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: BOTH_X_B_EDGE_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.backmost_point
MERGE
    (a)-[r:BOTH_X_B_EDGE_ABOVE]->(b);


// Relation: BOTH_X_B_EDGE_ABOVE remove extra
MATCH
    (a:Item)-[r:BOTH_X_B_EDGE_ABOVE]->(b:Item)-[t:BOTH_X_B_EDGE_ABOVE]->(c:Item),
    (a:Item)-[q:BOTH_X_B_EDGE_ABOVE]->(c:Item)
WHERE
```

```
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: BOTH_X_BOTH_Y_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point < b.backmost_point
MERGE
    (a)-[r:BOTH_X_BOTH_Y_ABOVE]->(b);


// Relation: BOTH_X_BOTH_Y_ABOVE remove extra
MATCH
    (a:Item)-[r:BOTH_X_BOTH_Y_ABOVE]->(b:Item)-[t:BOTH_X_BOTH_Y_ABOVE]->(c:Item),
    (a:Item)-[q:BOTH_X_BOTH_Y_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: BOTH_X_T_EDGE_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point <= b.backmost_point
    AND a.backmost_point > b.frontmost_point
MERGE
    (a)-[r:BOTH_X_T_EDGE_ABOVE]->(b);


// Relation: BOTH_X_T_EDGE_ABOVE remove extra
MATCH
    (a:Item)-[r:BOTH_X_T_EDGE_ABOVE]->(b:Item)-[t:BOTH_X_T_EDGE_ABOVE]->(c:Item),
    (a:Item)-[q:BOTH_X_T_EDGE_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
```

```
DELETE
    q;
// Relation: BOTH_X_NONE_Y_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:BOTH_X_NONE_Y_ABOVE]->(b);


// Relation: BOTH_X_NONE_Y_ABOVE remove extra
MATCH
    (a:Item)-[r:BOTH_X_NONE_Y_ABOVE]->(b:Item)-[t:BOTH_X_NONE_Y_ABOVE]->(c:Item),
    (a:Item)-[q:BOTH_X_NONE_Y_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_EDGE_B_EDGE_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.rightmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.backmost_point
MERGE
    (a)-[r:L_EDGE_B_EDGE_ABOVE]->(b);


// Relation: L_EDGE_B_EDGE_ABOVE remove extra
MATCH
    (a:Item)-[r:L_EDGE_B_EDGE_ABOVE]->(b:Item)-[t:L_EDGE_B_EDGE_ABOVE]->(c:Item),
    (a:Item)-[q:L_EDGE_B_EDGE_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
```

```
// Relation: L_EDGE_BOTH_Y_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.rightmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point < b.backmost_point
MERGE
    (a)-[r:L_EDGE_BOTH_Y_ABOVE]->(b);


// Relation: L_EDGE_BOTH_Y_ABOVE remove extra
MATCH
    (a:Item)-[r:L_EDGE_BOTH_Y_ABOVE]->(b:Item)-[t:L_EDGE_BOTH_Y_ABOVE]->(c:Item),
    (a:Item)-[q:L_EDGE_BOTH_Y_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_EDGE_T_EDGE_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.rightmost_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point <= b.backmost_point
    AND a.backmost_point > b.frontmost_point
MERGE
    (a)-[r:L_EDGE_T_EDGE_ABOVE]->(b);


// Relation: L_EDGE_T_EDGE_ABOVE remove extra
MATCH
    (a:Item)-[r:L_EDGE_T_EDGE_ABOVE]->(b:Item)-[t:L_EDGE_T_EDGE_ABOVE]->(c:Item),
    (a:Item)-[q:L_EDGE_T_EDGE_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_EDGE_NONE_Y_ABOVE
```

```
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.rightmost_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:L_EDGE_NONE_Y_ABOVE]->(b);


// Relation: L_EDGE_NONE_Y_ABOVE remove extra
MATCH
    (a:Item)-[r:L_EDGE_NONE_Y_ABOVE]->(b:Item)-[t:L_EDGE_NONE_Y_ABOVE]->(c:Item),
    (a:Item)-[q:L_EDGE_NONE_Y_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: NONE_X_B_EDGE_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.backmost_point
MERGE
    (a)-[r:NONE_X_B_EDGE_ABOVE]->(b);


// Relation: NONE_X_B_EDGE_ABOVE remove extra
MATCH
    (a:Item)-[r:NONE_X_B_EDGE_ABOVE]->(b:Item)-[t:NONE_X_B_EDGE_ABOVE]->(c:Item),
    (a:Item)-[q:NONE_X_B_EDGE_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: NONE_X_BOTH_Y_ABOVE
MATCH
    (a:Item),
```

```
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point < b.backmost_point
MERGE
    (a)-[r:NONE_X_BOTH_Y_ABOVE]->(b);


// Relation: NONE_X_BOTH_Y_ABOVE remove extra
MATCH
    (a:Item)-[r:NONE_X_BOTH_Y_ABOVE]->(b:Item)-[t:NONE_X_BOTH_Y_ABOVE]->(c:Item),
    (a:Item)-[q:NONE_X_BOTH_Y_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: NONE_X_T_EDGE_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point <= b.backmost_point
    AND a.backmost_point > b.frontmost_point
MERGE
    (a)-[r:NONE_X_T_EDGE_ABOVE]->(b);


// Relation: NONE_X_T_EDGE_ABOVE remove extra
MATCH
    (a:Item)-[r:NONE_X_T_EDGE_ABOVE]->(b:Item)-[t:NONE_X_T_EDGE_ABOVE]->(c:Item),
    (a:Item)-[q:NONE_X_T_EDGE_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: NONE_X_NONE_Y_ABOVE
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
```

```
    AND a.pallet_id = b.pallet_id
    AND a.lowest_point + 59 > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:NONE_X_NONE_Y_ABOVE]->(b);


// Relation: NONE_X_NONE_Y_ABOVE remove extra
MATCH
    (a:Item)-[r:NONE_X_NONE_Y_ABOVE]->(b:Item)-[t:NONE_X_NONE_Y_ABOVE]->(c:Item),
    (a:Item)-[q:NONE_X_NONE_Y_ABOVE]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: ON_TOP { reason }
MATCH
    (a:Item) -[:R_EDGE_B_EDGE_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "R_EDGE_B_EDGE_ABOVE"}]->(b);

MATCH
    (a:Item) -[:R_EDGE_BOTH_Y_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "R_EDGE_BOTH_Y_ABOVE"}]->(b);

MATCH
    (a:Item) -[:R_EDGE_T_EDGE_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "R_EDGE_T_EDGE_ABOVE"}]->(b);

MATCH
    (a:Item) -[:R_EDGE_NONE_Y_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "R_EDGE_NONE_Y_ABOVE"}]->(b);

MATCH
    (a:Item) -[:BOTH_X_B_EDGE_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "BOTH_X_B_EDGE_ABOVE"}]->(b);
```

```
MATCH
    (a:Item) -[:BOTH_X_BOTH_Y_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "BOTH_X_BOTH_Y_ABOVE"}]->(b);

MATCH
    (a:Item) -[:BOTH_X_T_EDGE_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "BOTH_X_T_EDGE_ABOVE"}]->(b);

MATCH
    (a:Item) -[:BOTH_X_NONE_Y_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "BOTH_X_NONE_Y_ABOVE"}]->(b);

MATCH
    (a:Item) -[:L_EDGE_B_EDGE_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "L_EDGE_B_EDGE_ABOVE"}]->(b);

MATCH
    (a:Item) -[:L_EDGE_BOTH_Y_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "L_EDGE_BOTH_Y_ABOVE"}]->(b);

MATCH
    (a:Item) -[:L_EDGE_T_EDGE_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "L_EDGE_T_EDGE_ABOVE"}]->(b);

MATCH
    (a:Item) -[:L_EDGE_NONE_Y_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "L_EDGE_NONE_Y_ABOVE"}]->(b);

MATCH
    (a:Item) -[:NONE_X_B_EDGE_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "NONE_X_B_EDGE_ABOVE"}]->(b);
```

```
MATCH
    (a:Item) -[:NONE_X_BOTH_Y_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "NONE_X_BOTH_Y_ABOVE"}]->(b);

MATCH
    (a:Item) -[:NONE_X_T_EDGE_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "NONE_X_T_EDGE_ABOVE"}]->(b);

MATCH
    (a:Item) -[:NONE_X_NONE_Y_ABOVE]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:ON_TOP {reason: "NONE_X_NONE_Y_ABOVE"}]->(b);

// Relation: ON_TOP { gap }
MATCH
    (a:Item) -[r:ON_TOP]-> (b:Item)
WITH
    r,    a.lowest_point - b.highest_point AS gap
SET
    r.gap = gap;
// Relation: ON_TOP remove extra
MATCH
    (a:Item)-[r:ON_TOP]->(b:Item)-[t:ON_TOP]->(c:Item),
    (a:Item)-[q:ON_TOP]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
```

Cypher Query 15: Resulting Cypher query for the ON_TOP_OF relation.

```
// Relation: NEXT_TO.
// Contains 64 sub-queries to create.




// Relation: L_HIGH_Z_TOP_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
```

```
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.frontmost_point < b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:L_HIGH_Z_TOP_Y]->(b);


// Relation: L_HIGH_Z_TOP_Y remove extra
MATCH
    (a:Item)-[r:L_HIGH_Z_TOP_Y]->(b:Item)-[t:L_HIGH_Z_TOP_Y]->(c:Item),
    (a:Item)-[q:L_HIGH_Z_TOP_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_HIGH_Z_MID_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.frontmost_point
    AND a.backmost_point <= b.backmost_point
MERGE
    (a)-[r:L_HIGH_Z_MID_Y]->(b);


// Relation: L_HIGH_Z_MID_Y remove extra
MATCH
    (a:Item)-[r:L_HIGH_Z_MID_Y]->(b:Item)-[t:L_HIGH_Z_MID_Y]->(c:Item),
    (a:Item)-[q:L_HIGH_Z_MID_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_HIGH_Z_BOT_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
```

```
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.frontmost_point
MERGE
    (a)-[r:L_HIGH_Z_BOT_Y]->(b);


// Relation: L_HIGH_Z_BOT_Y remove extra
MATCH
    (a:Item)-[r:L_HIGH_Z_BOT_Y]->(b:Item)-[t:L_HIGH_Z_BOT_Y]->(c:Item),
    (a:Item)-[q:L_HIGH_Z_BOT_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_HIGH_Z_BIG_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.backmost_point <= b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
MERGE
    (a)-[r:L_HIGH_Z_BIG_Y]->(b);


// Relation: L_HIGH_Z_BIG_Y remove extra
MATCH
    (a:Item)-[r:L_HIGH_Z_BIG_Y]->(b:Item)-[t:L_HIGH_Z_BIG_Y]->(c:Item),
    (a:Item)-[q:L_HIGH_Z_BIG_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_LOW_Z_TOP_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
```

```
    AND a.highest_point <= b.highest_point
    AND a.frontmost_point < b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:L_LOW_Z_TOP_Y]->(b);


// Relation: L_LOW_Z_TOP_Y remove extra
MATCH
    (a:Item)-[r:L_LOW_Z_TOP_Y]->(b:Item)-[t:L_LOW_Z_TOP_Y]->(c:Item),
    (a:Item)-[q:L_LOW_Z_TOP_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_LOW_Z_MID_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.frontmost_point
    AND a.backmost_point <= b.backmost_point
MERGE
    (a)-[r:L_LOW_Z_MID_Y]->(b);


// Relation: L_LOW_Z_MID_Y remove extra
MATCH
    (a:Item)-[r:L_LOW_Z_MID_Y]->(b:Item)-[t:L_LOW_Z_MID_Y]->(c:Item),
    (a:Item)-[q:L_LOW_Z_MID_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_LOW_Z_BOT_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
```

```
    AND a.highest_point <= b.highest_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.frontmost_point
MERGE
    (a)-[r:L_LOW_Z_BOT_Y]->(b);


// Relation: L_LOW_Z_BOT_Y remove extra
MATCH
    (a:Item)-[r:L_LOW_Z_BOT_Y]->(b:Item)-[t:L_LOW_Z_BOT_Y]->(c:Item),
    (a:Item)-[q:L_LOW_Z_BOT_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_LOW_Z_BIG_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.backmost_point <= b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
MERGE
    (a)-[r:L_LOW_Z_BIG_Y]->(b);


// Relation: L_LOW_Z_BIG_Y remove extra
MATCH
    (a:Item)-[r:L_LOW_Z_BIG_Y]->(b:Item)-[t:L_LOW_Z_BIG_Y]->(c:Item),
    (a:Item)-[q:L_LOW_Z_BIG_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_BIG_Z_TOP_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.frontmost_point < b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
```

```
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:L_BIG_Z_TOP_Y]->(b);


// Relation: L_BIG_Z_TOP_Y remove extra
MATCH
    (a:Item)-[r:L_BIG_Z_TOP_Y]->(b:Item)-[t:L_BIG_Z_TOP_Y]->(c:Item),
    (a:Item)-[q:L_BIG_Z_TOP_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_BIG_Z_MID_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.frontmost_point
    AND a.backmost_point <= b.backmost_point
MERGE
    (a)-[r:L_BIG_Z_MID_Y]->(b);


// Relation: L_BIG_Z_MID_Y remove extra
MATCH
    (a:Item)-[r:L_BIG_Z_MID_Y]->(b:Item)-[t:L_BIG_Z_MID_Y]->(c:Item),
    (a:Item)-[q:L_BIG_Z_MID_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_BIG_Z_BOT_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.frontmost_point
MERGE
    (a)-[r:L_BIG_Z_BOT_Y]->(b);
```

```
// Relation: L_BIG_Z_BOT_Y remove extra
MATCH
    (a:Item)-[r:L_BIG_Z_BOT_Y]->(b:Item)-[t:L_BIG_Z_BOT_Y]->(c:Item),
    (a:Item)-[q:L_BIG_Z_BOT_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_BIG_Z_BIG_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.backmost_point <= b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
MERGE
    (a)-[r:L_BIG_Z_BIG_Y]->(b);


// Relation: L_BIG_Z_BIG_Y remove extra
MATCH
    (a:Item)-[r:L_BIG_Z_BIG_Y]->(b:Item)-[t:L_BIG_Z_BIG_Y]->(c:Item),
    (a:Item)-[q:L_BIG_Z_BIG_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_SMALL_Z_TOP_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.frontmost_point < b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:L_SMALL_Z_TOP_Y]->(b);


// Relation: L_SMALL_Z_TOP_Y remove extra
```

```
MATCH
    (a:Item)-[r:L_SMALL_Z_TOP_Y]->(b:Item)-[t:L_SMALL_Z_TOP_Y]->(c:Item),
    (a:Item)-[q:L_SMALL_Z_TOP_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_SMALL_Z_MID_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.frontmost_point
    AND a.backmost_point <= b.backmost_point
MERGE
    (a)-[r:L_SMALL_Z_MID_Y]->(b);


// Relation: L_SMALL_Z_MID_Y remove extra
MATCH
    (a:Item)-[r:L_SMALL_Z_MID_Y]->(b:Item)-[t:L_SMALL_Z_MID_Y]->(c:Item),
    (a:Item)-[q:L_SMALL_Z_MID_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_SMALL_Z_BOT_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.frontmost_point
MERGE
    (a)-[r:L_SMALL_Z_BOT_Y]->(b);


// Relation: L_SMALL_Z_BOT_Y remove extra
MATCH
    (a:Item)-[r:L_SMALL_Z_BOT_Y]->(b:Item)-[t:L_SMALL_Z_BOT_Y]->(c:Item),
    (a:Item)-[q:L_SMALL_Z_BOT_Y]->(c:Item)
```

```
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: L_SMALL_Z_BIG_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.rightmost_point < b.leftmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.backmost_point <= b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
MERGE
    (a)-[r:L_SMALL_Z_BIG_Y]->(b);


// Relation: L_SMALL_Z_BIG_Y remove extra
MATCH
    (a:Item)-[r:L_SMALL_Z_BIG_Y]->(b:Item)-[t:L_SMALL_Z_BIG_Y]->(c:Item),
    (a:Item)-[q:L_SMALL_Z_BIG_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_HIGH_Z_LEFT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.leftmost_point < b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
MERGE
    (a)-[r:F_HIGH_Z_LEFT_X]->(b);


// Relation: F_HIGH_Z_LEFT_X remove extra
MATCH
    (a:Item)-[r:F_HIGH_Z_LEFT_X]->(b:Item)-[t:F_HIGH_Z_LEFT_X]->(c:Item),
    (a:Item)-[q:F_HIGH_Z_LEFT_X]->(c:Item)
WHERE
    id(a) <> id(b)
```

```
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_HIGH_Z_MID_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
MERGE
    (a)-[r:F_HIGH_Z_MID_X]->(b);


// Relation: F_HIGH_Z_MID_X remove extra
MATCH
    (a:Item)-[r:F_HIGH_Z_MID_X]->(b:Item)-[t:F_HIGH_Z_MID_X]->(c:Item),
    (a:Item)-[q:F_HIGH_Z_MID_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_HIGH_Z_RIGHT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.leftmost_point
MERGE
    (a)-[r:F_HIGH_Z_RIGHT_X]->(b);


// Relation: F_HIGH_Z_RIGHT_X remove extra
MATCH
    (a:Item)-[r:F_HIGH_Z_RIGHT_X]->(b:Item)-[t:F_HIGH_Z_RIGHT_X]->(c:Item),
    (a:Item)-[q:F_HIGH_Z_RIGHT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
```

```
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_HIGH_Z_LONG_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
MERGE
    (a)-[r:F_HIGH_Z_LONG_X]->(b);


// Relation: F_HIGH_Z_LONG_X remove extra
MATCH
    (a:Item)-[r:F_HIGH_Z_LONG_X]->(b:Item)-[t:F_HIGH_Z_LONG_X]->(c:Item),
    (a:Item)-[q:F_HIGH_Z_LONG_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_LOW_Z_LEFT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.leftmost_point < b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
MERGE
    (a)-[r:F_LOW_Z_LEFT_X]->(b);


// Relation: F_LOW_Z_LEFT_X remove extra
MATCH
    (a:Item)-[r:F_LOW_Z_LEFT_X]->(b:Item)-[t:F_LOW_Z_LEFT_X]->(c:Item),
    (a:Item)-[q:F_LOW_Z_LEFT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
```

```
DELETE
    q;
// Relation: F_LOW_Z_MID_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
MERGE
    (a)-[r:F_LOW_Z_MID_X]->(b);


// Relation: F_LOW_Z_MID_X remove extra
MATCH
    (a:Item)-[r:F_LOW_Z_MID_X]->(b:Item)-[t:F_LOW_Z_MID_X]->(c:Item),
    (a:Item)-[q:F_LOW_Z_MID_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_LOW_Z_RIGHT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.leftmost_point
MERGE
    (a)-[r:F_LOW_Z_RIGHT_X]->(b);


// Relation: F_LOW_Z_RIGHT_X remove extra
MATCH
    (a:Item)-[r:F_LOW_Z_RIGHT_X]->(b:Item)-[t:F_LOW_Z_RIGHT_X]->(c:Item),
    (a:Item)-[q:F_LOW_Z_RIGHT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
```

```
    q;
// Relation: F_LOW_Z_LONG_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
MERGE
    (a)-[r:F_LOW_Z_LONG_X]->(b);


// Relation: F_LOW_Z_LONG_X remove extra
MATCH
    (a:Item)-[r:F_LOW_Z_LONG_X]->(b:Item)-[t:F_LOW_Z_LONG_X]->(c:Item),
    (a:Item)-[q:F_LOW_Z_LONG_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_BIG_Z_LEFT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.leftmost_point < b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
MERGE
    (a)-[r:F_BIG_Z_LEFT_X]->(b);


// Relation: F_BIG_Z_LEFT_X remove extra
MATCH
    (a:Item)-[r:F_BIG_Z_LEFT_X]->(b:Item)-[t:F_BIG_Z_LEFT_X]->(c:Item),
    (a:Item)-[q:F_BIG_Z_LEFT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_BIG_Z_MID_X
```

```
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
MERGE
    (a)-[r:F_BIG_Z_MID_X]->(b);


// Relation: F_BIG_Z_MID_X remove extra
MATCH
    (a:Item)-[r:F_BIG_Z_MID_X]->(b:Item)-[t:F_BIG_Z_MID_X]->(c:Item),
    (a:Item)-[q:F_BIG_Z_MID_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_BIG_Z_RIGHT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.leftmost_point
MERGE
    (a)-[r:F_BIG_Z_RIGHT_X]->(b);


// Relation: F_BIG_Z_RIGHT_X remove extra
MATCH
    (a:Item)-[r:F_BIG_Z_RIGHT_X]->(b:Item)-[t:F_BIG_Z_RIGHT_X]->(c:Item),
    (a:Item)-[q:F_BIG_Z_RIGHT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_BIG_Z_LONG_X
MATCH
    (a:Item),
    (b:Item)
```

```
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
MERGE
    (a)-[r:F_BIG_Z_LONG_X]->(b);


// Relation: F_BIG_Z_LONG_X remove extra
MATCH
    (a:Item)-[r:F_BIG_Z_LONG_X]->(b:Item)-[t:F_BIG_Z_LONG_X]->(c:Item),
    (a:Item)-[q:F_BIG_Z_LONG_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_SMALL_Z_LEFT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.leftmost_point < b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
MERGE
    (a)-[r:F_SMALL_Z_LEFT_X]->(b);


// Relation: F_SMALL_Z_LEFT_X remove extra
MATCH
    (a:Item)-[r:F_SMALL_Z_LEFT_X]->(b:Item)-[t:F_SMALL_Z_LEFT_X]->(c:Item),
    (a:Item)-[q:F_SMALL_Z_LEFT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_SMALL_Z_MID_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
```

```
    AND a.backmost_point < b.frontmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
MERGE
    (a)-[r:F_SMALL_Z_MID_X]->(b);


// Relation: F_SMALL_Z_MID_X remove extra
MATCH
    (a:Item)-[r:F_SMALL_Z_MID_X]->(b:Item)-[t:F_SMALL_Z_MID_X]->(c:Item),
    (a:Item)-[q:F_SMALL_Z_MID_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_SMALL_Z_RIGHT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.leftmost_point
MERGE
    (a)-[r:F_SMALL_Z_RIGHT_X]->(b);


// Relation: F_SMALL_Z_RIGHT_X remove extra
MATCH
    (a:Item)-[r:F_SMALL_Z_RIGHT_X]->(b:Item)-[t:F_SMALL_Z_RIGHT_X]->(c:Item),
    (a:Item)-[q:F_SMALL_Z_RIGHT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: F_SMALL_Z_LONG_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.backmost_point < b.frontmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
```

```
    AND a.rightmost_point <= b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
MERGE
    (a)-[r:F_SMALL_Z_LONG_X]->(b);


// Relation: F_SMALL_Z_LONG_X remove extra
MATCH
    (a:Item)-[r:F_SMALL_Z_LONG_X]->(b:Item)-[t:F_SMALL_Z_LONG_X]->(c:Item),
    (a:Item)-[q:F_SMALL_Z_LONG_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_HIGH_Z_TOP_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.frontmost_point < b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:R_HIGH_Z_TOP_Y]->(b);


// Relation: R_HIGH_Z_TOP_Y remove extra
MATCH
    (a:Item)-[r:R_HIGH_Z_TOP_Y]->(b:Item)-[t:R_HIGH_Z_TOP_Y]->(c:Item),
    (a:Item)-[q:R_HIGH_Z_TOP_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_HIGH_Z_MID_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.frontmost_point < b.frontmost_point
```

```
    AND a.backmost_point > b.frontmost_point
    AND a.backmost_point <= b.backmost_point
MERGE
    (a)-[r:R_HIGH_Z_MID_Y]->(b);


// Relation: R_HIGH_Z_MID_Y remove extra
MATCH
    (a:Item)-[r:R_HIGH_Z_MID_Y]->(b:Item)-[t:R_HIGH_Z_MID_Y]->(c:Item),
    (a:Item)-[q:R_HIGH_Z_MID_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_HIGH_Z_BOT_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.frontmost_point
MERGE
    (a)-[r:R_HIGH_Z_BOT_Y]->(b);


// Relation: R_HIGH_Z_BOT_Y remove extra
MATCH
    (a:Item)-[r:R_HIGH_Z_BOT_Y]->(b:Item)-[t:R_HIGH_Z_BOT_Y]->(c:Item),
    (a:Item)-[q:R_HIGH_Z_BOT_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_HIGH_Z_BIG_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.backmost_point <= b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
```

```
MERGE
    (a)-[r:R_HIGH_Z_BIG_Y]->(b);


// Relation: R_HIGH_Z_BIG_Y remove extra
MATCH
    (a:Item)-[r:R_HIGH_Z_BIG_Y]->(b:Item)-[t:R_HIGH_Z_BIG_Y]->(c:Item),
    (a:Item)-[q:R_HIGH_Z_BIG_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_LOW_Z_TOP_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.frontmost_point < b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:R_LOW_Z_TOP_Y]->(b);


// Relation: R_LOW_Z_TOP_Y remove extra
MATCH
    (a:Item)-[r:R_LOW_Z_TOP_Y]->(b:Item)-[t:R_LOW_Z_TOP_Y]->(c:Item),
    (a:Item)-[q:R_LOW_Z_TOP_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_LOW_Z_MID_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.frontmost_point
    AND a.backmost_point <= b.backmost_point
```

```
MERGE
    (a)-[r:R_LOW_Z_MID_Y]->(b);


// Relation: R_LOW_Z_MID_Y remove extra
MATCH
    (a:Item)-[r:R_LOW_Z_MID_Y]->(b:Item)-[t:R_LOW_Z_MID_Y]->(c:Item),
    (a:Item)-[q:R_LOW_Z_MID_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_LOW_Z_BOT_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.frontmost_point
MERGE
    (a)-[r:R_LOW_Z_BOT_Y]->(b);


// Relation: R_LOW_Z_BOT_Y remove extra
MATCH
    (a:Item)-[r:R_LOW_Z_BOT_Y]->(b:Item)-[t:R_LOW_Z_BOT_Y]->(c:Item),
    (a:Item)-[q:R_LOW_Z_BOT_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_LOW_Z_BIG_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.backmost_point <= b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
MERGE
    (a)-[r:R_LOW_Z_BIG_Y]->(b);
```

```
// Relation: R_LOW_Z_BIG_Y remove extra
MATCH
    (a:Item)-[r:R_LOW_Z_BIG_Y]->(b:Item)-[t:R_LOW_Z_BIG_Y]->(c:Item),
    (a:Item)-[q:R_LOW_Z_BIG_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_BIG_Z_TOP_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.frontmost_point < b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:R_BIG_Z_TOP_Y]->(b);


// Relation: R_BIG_Z_TOP_Y remove extra
MATCH
    (a:Item)-[r:R_BIG_Z_TOP_Y]->(b:Item)-[t:R_BIG_Z_TOP_Y]->(c:Item),
    (a:Item)-[q:R_BIG_Z_TOP_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_BIG_Z_MID_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.frontmost_point
    AND a.backmost_point <= b.backmost_point
MERGE
    (a)-[r:R_BIG_Z_MID_Y]->(b);
```

```
// Relation: R_BIG_Z_MID_Y remove extra
MATCH
    (a:Item)-[r:R_BIG_Z_MID_Y]->(b:Item)-[t:R_BIG_Z_MID_Y]->(c:Item),
    (a:Item)-[q:R_BIG_Z_MID_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_BIG_Z_BOT_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.frontmost_point
MERGE
    (a)-[r:R_BIG_Z_BOT_Y]->(b);


// Relation: R_BIG_Z_BOT_Y remove extra
MATCH
    (a:Item)-[r:R_BIG_Z_BOT_Y]->(b:Item)-[t:R_BIG_Z_BOT_Y]->(c:Item),
    (a:Item)-[q:R_BIG_Z_BOT_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_BIG_Z_BIG_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.backmost_point <= b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
MERGE
    (a)-[r:R_BIG_Z_BIG_Y]->(b);


// Relation: R_BIG_Z_BIG_Y remove extra
MATCH
    (a:Item)-[r:R_BIG_Z_BIG_Y]->(b:Item)-[t:R_BIG_Z_BIG_Y]->(c:Item),
    (a:Item)-[q:R_BIG_Z_BIG_Y]->(c:Item)
```

```
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_SMALL_Z_TOP_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.frontmost_point < b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
    AND a.backmost_point > b.backmost_point
MERGE
    (a)-[r:R_SMALL_Z_TOP_Y]->(b);


// Relation: R_SMALL_Z_TOP_Y remove extra
MATCH
    (a:Item)-[r:R_SMALL_Z_TOP_Y]->(b:Item)-[t:R_SMALL_Z_TOP_Y]->(c:Item),
    (a:Item)-[q:R_SMALL_Z_TOP_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_SMALL_Z_MID_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.frontmost_point < b.frontmost_point
    AND a.backmost_point > b.frontmost_point
    AND a.backmost_point <= b.backmost_point
MERGE
    (a)-[r:R_SMALL_Z_MID_Y]->(b);


// Relation: R_SMALL_Z_MID_Y remove extra
MATCH
    (a:Item)-[r:R_SMALL_Z_MID_Y]->(b:Item)-[t:R_SMALL_Z_MID_Y]->(c:Item),
    (a:Item)-[q:R_SMALL_Z_MID_Y]->(c:Item)
WHERE
    id(a) <> id(b)
```

```
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_SMALL_Z_BOT_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.backmost_point > b.backmost_point
    AND a.frontmost_point < b.frontmost_point
MERGE
    (a)-[r:R_SMALL_Z_BOT_Y]->(b);


// Relation: R_SMALL_Z_BOT_Y remove extra
MATCH
    (a:Item)-[r:R_SMALL_Z_BOT_Y]->(b:Item)-[t:R_SMALL_Z_BOT_Y]->(c:Item),
    (a:Item)-[q:R_SMALL_Z_BOT_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: R_SMALL_Z_BIG_Y
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.leftmost_point > b.rightmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.backmost_point <= b.backmost_point
    AND a.frontmost_point >= b.frontmost_point
MERGE
    (a)-[r:R_SMALL_Z_BIG_Y]->(b);


// Relation: R_SMALL_Z_BIG_Y remove extra
MATCH
    (a:Item)-[r:R_SMALL_Z_BIG_Y]->(b:Item)-[t:R_SMALL_Z_BIG_Y]->(c:Item),
    (a:Item)-[q:R_SMALL_Z_BIG_Y]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
```

```
// Relation: B_HIGH_Z_LEFT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.leftmost_point < b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
MERGE
    (a)-[r:B_HIGH_Z_LEFT_X]->(b);


// Relation: B_HIGH_Z_LEFT_X remove extra
MATCH
    (a:Item)-[r:B_HIGH_Z_LEFT_X]->(b:Item)-[t:B_HIGH_Z_LEFT_X]->(c:Item),
    (a:Item)-[q:B_HIGH_Z_LEFT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_HIGH_Z_MID_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
MERGE
    (a)-[r:B_HIGH_Z_MID_X]->(b);


// Relation: B_HIGH_Z_MID_X remove extra
MATCH
    (a:Item)-[r:B_HIGH_Z_MID_X]->(b:Item)-[t:B_HIGH_Z_MID_X]->(c:Item),
    (a:Item)-[q:B_HIGH_Z_MID_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
```

```
// Relation: B_HIGH_Z_RIGHT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.leftmost_point
MERGE
    (a)-[r:B_HIGH_Z_RIGHT_X]->(b);


// Relation: B_HIGH_Z_RIGHT_X remove extra
MATCH
    (a:Item)-[r:B_HIGH_Z_RIGHT_X]->(b:Item)-[t:B_HIGH_Z_RIGHT_X]->(c:Item),
    (a:Item)-[q:B_HIGH_Z_RIGHT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_HIGH_Z_LONG_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.lowest_point < b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.highest_point > b.highest_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
MERGE
    (a)-[r:B_HIGH_Z_LONG_X]->(b);


// Relation: B_HIGH_Z_LONG_X remove extra
MATCH
    (a:Item)-[r:B_HIGH_Z_LONG_X]->(b:Item)-[t:B_HIGH_Z_LONG_X]->(c:Item),
    (a:Item)-[q:B_HIGH_Z_LONG_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_LOW_Z_LEFT_X
MATCH
```

```
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.leftmost_point < b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
MERGE
    (a)-[r:B_LOW_Z_LEFT_X]->(b);


// Relation: B_LOW_Z_LEFT_X remove extra
MATCH
    (a:Item)-[r:B_LOW_Z_LEFT_X]->(b:Item)-[t:B_LOW_Z_LEFT_X]->(c:Item),
    (a:Item)-[q:B_LOW_Z_LEFT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_LOW_Z_MID_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
MERGE
    (a)-[r:B_LOW_Z_MID_X]->(b);


// Relation: B_LOW_Z_MID_X remove extra
MATCH
    (a:Item)-[r:B_LOW_Z_MID_X]->(b:Item)-[t:B_LOW_Z_MID_X]->(c:Item),
    (a:Item)-[q:B_LOW_Z_MID_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_LOW_Z_RIGHT_X
MATCH
```

```
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.leftmost_point
MERGE
    (a)-[r:B_LOW_Z_RIGHT_X]->(b);


// Relation: B_LOW_Z_RIGHT_X remove extra
MATCH
    (a:Item)-[r:B_LOW_Z_RIGHT_X]->(b:Item)-[t:B_LOW_Z_RIGHT_X]->(c:Item),
    (a:Item)-[q:B_LOW_Z_RIGHT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_LOW_Z_LONG_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.lowest_point < b.lowest_point
    AND a.highest_point > b.lowest_point
    AND a.highest_point <= b.highest_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
MERGE
    (a)-[r:B_LOW_Z_LONG_X]->(b);


// Relation: B_LOW_Z_LONG_X remove extra
MATCH
    (a:Item)-[r:B_LOW_Z_LONG_X]->(b:Item)-[t:B_LOW_Z_LONG_X]->(c:Item),
    (a:Item)-[q:B_LOW_Z_LONG_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_BIG_Z_LEFT_X
MATCH
    (a:Item),
    (b:Item)
```

```
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.leftmost_point < b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
MERGE
    (a)-[r:B_BIG_Z_LEFT_X]->(b);


// Relation: B_BIG_Z_LEFT_X remove extra
MATCH
    (a:Item)-[r:B_BIG_Z_LEFT_X]->(b:Item)-[t:B_BIG_Z_LEFT_X]->(c:Item),
    (a:Item)-[q:B_BIG_Z_LEFT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_BIG_Z_MID_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
MERGE
    (a)-[r:B_BIG_Z_MID_X]->(b);


// Relation: B_BIG_Z_MID_X remove extra
MATCH
    (a:Item)-[r:B_BIG_Z_MID_X]->(b:Item)-[t:B_BIG_Z_MID_X]->(c:Item),
    (a:Item)-[q:B_BIG_Z_MID_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_BIG_Z_RIGHT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
```

```
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.leftmost_point
MERGE
    (a)-[r:B_BIG_Z_RIGHT_X]->(b);


// Relation: B_BIG_Z_RIGHT_X remove extra
MATCH
    (a:Item)-[r:B_BIG_Z_RIGHT_X]->(b:Item)-[t:B_BIG_Z_RIGHT_X]->(c:Item),
    (a:Item)-[q:B_BIG_Z_RIGHT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_BIG_Z_LONG_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.highest_point > b.highest_point
    AND a.lowest_point < b.lowest_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
MERGE
    (a)-[r:B_BIG_Z_LONG_X]->(b);


// Relation: B_BIG_Z_LONG_X remove extra
MATCH
    (a:Item)-[r:B_BIG_Z_LONG_X]->(b:Item)-[t:B_BIG_Z_LONG_X]->(c:Item),
    (a:Item)-[q:B_BIG_Z_LONG_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_SMALL_Z_LEFT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
```

```
    AND a.leftmost_point < b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
    AND a.rightmost_point > b.rightmost_point
MERGE
    (a)-[r:B_SMALL_Z_LEFT_X]->(b);


// Relation: B_SMALL_Z_LEFT_X remove extra
MATCH
    (a:Item)-[r:B_SMALL_Z_LEFT_X]->(b:Item)-[t:B_SMALL_Z_LEFT_X]->(c:Item),
    (a:Item)-[q:B_SMALL_Z_LEFT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_SMALL_Z_MID_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.leftmost_point < b.leftmost_point
    AND a.rightmost_point > b.leftmost_point
    AND a.rightmost_point <= b.rightmost_point
MERGE
    (a)-[r:B_SMALL_Z_MID_X]->(b);


// Relation: B_SMALL_Z_MID_X remove extra
MATCH
    (a:Item)-[r:B_SMALL_Z_MID_X]->(b:Item)-[t:B_SMALL_Z_MID_X]->(c:Item),
    (a:Item)-[q:B_SMALL_Z_MID_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_SMALL_Z_RIGHT_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.rightmost_point > b.rightmost_point
    AND a.leftmost_point < b.leftmost_point
```

```
MERGE
    (a)-[r:B_SMALL_Z_RIGHT_X]->(b);


// Relation: B_SMALL_Z_RIGHT_X remove extra
MATCH
    (a:Item)-[r:B_SMALL_Z_RIGHT_X]->(b:Item)-[t:B_SMALL_Z_RIGHT_X]->(c:Item),
    (a:Item)-[q:B_SMALL_Z_RIGHT_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: B_SMALL_Z_LONG_X
MATCH
    (a:Item),
    (b:Item)
WHERE
    id(a) <> id(b)
    AND a.pallet_id = b.pallet_id
    AND a.frontmost_point > b.backmost_point
    AND a.highest_point <= b.highest_point
    AND a.lowest_point >= b.lowest_point
    AND a.rightmost_point <= b.rightmost_point
    AND a.leftmost_point >= b.leftmost_point
MERGE
    (a)-[r:B_SMALL_Z_LONG_X]->(b);


// Relation: B_SMALL_Z_LONG_X remove extra
MATCH
    (a:Item)-[r:B_SMALL_Z_LONG_X]->(b:Item)-[t:B_SMALL_Z_LONG_X]->(c:Item),
    (a:Item)-[q:B_SMALL_Z_LONG_X]->(c:Item)
WHERE
    id(a) <> id(b)
    AND id(a) <> id(c)
    AND id(b) <> id(c)
DELETE
    q;
// Relation: NEXT_TO { reason }
MATCH
    (a:Item) -[:L_HIGH_Z_TOP_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_HIGH_Z_TOP_Y"}]->(b);
MATCH
    (a:Item) -[:L_HIGH_Z_MID_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_HIGH_Z_MID_Y"}]->(b);
MATCH
    (a:Item) -[:L_HIGH_Z_BOT_Y]->(b:Item)
WHERE
```

```
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_HIGH_Z_BOT_Y"}]->(b);
MATCH
    (a:Item) -[:L_HIGH_Z_BIG_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_HIGH_Z_BIG_Y"}]->(b);
MATCH
    (a:Item) -[:L_LOW_Z_TOP_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_LOW_Z_TOP_Y"}]->(b);
MATCH
    (a:Item) -[:L_LOW_Z_MID_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_LOW_Z_MID_Y"}]->(b);
MATCH
    (a:Item) -[:L_LOW_Z_BOT_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_LOW_Z_BOT_Y"}]->(b);
MATCH
    (a:Item) -[:L_LOW_Z_BIG_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_LOW_Z_BIG_Y"}]->(b);
MATCH
    (a:Item) -[:L_BIG_Z_TOP_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_BIG_Z_TOP_Y"}]->(b);
MATCH
    (a:Item) -[:L_BIG_Z_MID_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_BIG_Z_MID_Y"}]->(b);
MATCH
    (a:Item) -[:L_BIG_Z_BOT_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_BIG_Z_BOT_Y"}]->(b);
MATCH
    (a:Item) -[:L_BIG_Z_BIG_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
```

```
    (a)-[:NEXT_TO {reason: "L_BIG_Z_BIG_Y"}]->(b);
MATCH
    (a:Item) -[:L_SMALL_Z_TOP_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_SMALL_Z_TOP_Y"}]->(b);
MATCH
    (a:Item) -[:L_SMALL_Z_MID_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_SMALL_Z_MID_Y"}]->(b);
MATCH
    (a:Item) -[:L_SMALL_Z_BOT_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_SMALL_Z_BOT_Y"}]->(b);
MATCH
    (a:Item) -[:L_SMALL_Z_BIG_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "L_SMALL_Z_BIG_Y"}]->(b);
MATCH
    (a:Item) -[:F_HIGH_Z_LEFT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_HIGH_Z_LEFT_X"}]->(b);
MATCH
    (a:Item) -[:F_HIGH_Z_MID_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_HIGH_Z_MID_X"}]->(b);
MATCH
    (a:Item) -[:F_HIGH_Z_RIGHT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_HIGH_Z_RIGHT_X"}]->(b);
MATCH
    (a:Item) -[:F_HIGH_Z_LONG_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_HIGH_Z_LONG_X"}]->(b);
MATCH
    (a:Item) -[:F_LOW_Z_LEFT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_LOW_Z_LEFT_X"}]->(b);
MATCH
```

```
    (a:Item) -[:F_LOW_Z_MID_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_LOW_Z_MID_X"}]->(b);
MATCH
    (a:Item) -[:F_LOW_Z_RIGHT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_LOW_Z_RIGHT_X"}]->(b);
MATCH
    (a:Item) -[:F_LOW_Z_LONG_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_LOW_Z_LONG_X"}]->(b);
MATCH
    (a:Item) -[:F_BIG_Z_LEFT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_BIG_Z_LEFT_X"}]->(b);
MATCH
    (a:Item) -[:F_BIG_Z_MID_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_BIG_Z_MID_X"}]->(b);
MATCH
    (a:Item) -[:F_BIG_Z_RIGHT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_BIG_Z_RIGHT_X"}]->(b);
MATCH
    (a:Item) -[:F_BIG_Z_LONG_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_BIG_Z_LONG_X"}]->(b);
MATCH
    (a:Item) -[:F_SMALL_Z_LEFT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_SMALL_Z_LEFT_X"}]->(b);
MATCH
    (a:Item) -[:F_SMALL_Z_MID_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_SMALL_Z_MID_X"}]->(b);
MATCH
    (a:Item) -[:F_SMALL_Z_RIGHT_X]->(b:Item)
WHERE
```

```
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_SMALL_Z_RIGHT_X"}]->(b);
MATCH
    (a:Item) -[:F_SMALL_Z_LONG_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "F_SMALL_Z_LONG_X"}]->(b);
MATCH
    (a:Item) -[:R_HIGH_Z_TOP_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_HIGH_Z_TOP_Y"}]->(b);
MATCH
    (a:Item) -[:R_HIGH_Z_MID_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_HIGH_Z_MID_Y"}]->(b);
MATCH
    (a:Item) -[:R_HIGH_Z_BOT_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_HIGH_Z_BOT_Y"}]->(b);
MATCH
    (a:Item) -[:R_HIGH_Z_BIG_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_HIGH_Z_BIG_Y"}]->(b);
MATCH
    (a:Item) -[:R_LOW_Z_TOP_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_LOW_Z_TOP_Y"}]->(b);
MATCH
    (a:Item) -[:R_LOW_Z_MID_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_LOW_Z_MID_Y"}]->(b);
MATCH
    (a:Item) -[:R_LOW_Z_BOT_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_LOW_Z_BOT_Y"}]->(b);
MATCH
    (a:Item) -[:R_LOW_Z_BIG_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
```

```
    (a)-[:NEXT_TO {reason: "R_LOW_Z_BIG_Y"}]->(b);
MATCH
    (a:Item) -[:R_BIG_Z_TOP_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_BIG_Z_TOP_Y"}]->(b);
MATCH
    (a:Item) -[:R_BIG_Z_MID_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_BIG_Z_MID_Y"}]->(b);
MATCH
    (a:Item) -[:R_BIG_Z_BOT_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_BIG_Z_BOT_Y"}]->(b);
MATCH
    (a:Item) -[:R_BIG_Z_BIG_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_BIG_Z_BIG_Y"}]->(b);
MATCH
    (a:Item) -[:R_SMALL_Z_TOP_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_SMALL_Z_TOP_Y"}]->(b);
MATCH
    (a:Item) -[:R_SMALL_Z_MID_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_SMALL_Z_MID_Y"}]->(b);
MATCH
    (a:Item) -[:R_SMALL_Z_BOT_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_SMALL_Z_BOT_Y"}]->(b);
MATCH
    (a:Item) -[:R_SMALL_Z_BIG_Y]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "R_SMALL_Z_BIG_Y"}]->(b);
MATCH
    (a:Item) -[:B_HIGH_Z_LEFT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_HIGH_Z_LEFT_X"}]->(b);
MATCH
```

```
    (a:Item) -[:B_HIGH_Z_MID_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_HIGH_Z_MID_X"}]->(b);
MATCH
    (a:Item) -[:B_HIGH_Z_RIGHT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_HIGH_Z_RIGHT_X"}]->(b);
MATCH
    (a:Item) -[:B_HIGH_Z_LONG_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_HIGH_Z_LONG_X"}]->(b);
MATCH
    (a:Item) -[:B_LOW_Z_LEFT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_LOW_Z_LEFT_X"}]->(b);
MATCH
    (a:Item) -[:B_LOW_Z_MID_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_LOW_Z_MID_X"}]->(b);
MATCH
    (a:Item) -[:B_LOW_Z_RIGHT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_LOW_Z_RIGHT_X"}]->(b);
MATCH
    (a:Item) -[:B_LOW_Z_LONG_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_LOW_Z_LONG_X"}]->(b);
MATCH
    (a:Item) -[:B_BIG_Z_LEFT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_BIG_Z_LEFT_X"}]->(b);
MATCH
    (a:Item) -[:B_BIG_Z_MID_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_BIG_Z_MID_X"}]->(b);
MATCH
    (a:Item) -[:B_BIG_Z_RIGHT_X]->(b:Item)
WHERE
```

```
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_BIG_Z_RIGHT_X"}]->(b);
MATCH
    (a:Item) -[:B_BIG_Z_LONG_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_BIG_Z_LONG_X"}]->(b);
MATCH
    (a:Item) -[:B_SMALL_Z_LEFT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_SMALL_Z_LEFT_X"}]->(b);
MATCH
    (a:Item) -[:B_SMALL_Z_MID_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_SMALL_Z_MID_X"}]->(b);
MATCH
    (a:Item) -[:B_SMALL_Z_RIGHT_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_SMALL_Z_RIGHT_X"}]->(b);
MATCH
    (a:Item) -[:B_SMALL_Z_LONG_X]->(b:Item)
WHERE
    id(a) <> id(b)
CREATE
    (a)-[:NEXT_TO {reason: "B_SMALL_Z_LONG_X"}]->(b);
```

Cypher Query 16: Resulting Cypher query for the NEXT_TO relation.