Material Tracking System for Tata Steel

T. Edixhoven
H. van Geffen
B. Kruit
M. Smit

Material Tracking System for Tata Steel

by

T. Edixhoven, H. van Geffen, B. Kruit & M. Smit

to obtain the degree of Bachelor of Science
at the Delft University of Technology.

Project duration:	April 23, 2019 – July 5, 2019	
Thesis committee:	Dr. Ir. H. Wang,	TU Delft
	Ir. O. Visser,	TU Delft
	Dr. Ir. M. Aniche,	TU Delft, Supervisor
	Drs. W. Lamme,	Tata Steel

TU Delft

Abstract

For a steel company it is advantageous to be able to easily track steel through the production process. At Tata Steel this is currently done with the Material Tracking Table. However, generating this table takes months. Therefore a new system had to be developed. This paper describes the building of such a new system, which generates this Material Tracking Table in less than 1 hour, as well as the related systems concerning the acquisition of the input data and the visualisation of the resulting output data.

Preface

Over the past 11 weeks we have been working on our Bachelor End Project. The client of our project was Tata Steel and during the project we have created a new version of the software used at Tata Steel to track materials across different production steps. During these 11 weeks we have worked hard to create a good and functioning product, this report will show the reader what we have achieved and how we got there.

We would like to thank Tata Steel for the amazing opportunity they gave us to work on this project. During the project we have learned a lot, not only about software engineering but also about working within a big company and working with multiple interested people with different backgrounds. Tata Steel has supported us all the way through the project and we were able to work closely together with them, which helped us in developing a product that complies with their wishes and helps them as much as possible.

We would also like to thank our supervisor at the TU Delft, Maurício Aniche. Whenever we had problems or questions we could always direct them at him and he would help us. Lastly we would like to thank the TU Delft for giving us the opportunity to do this project outside of the university, as it has really given us a new perspective on the world of software engineering.

T. Edixhoven, H. van Geffen, B. Kruit & M. Smit
Delft, June 2019

Contents

1 Introduction

Tata Steel annually produces 300.000 coils of steel. Ships containing iron ore and coal arrive at the harbor. The iron ore is then be processed into steel slabs. These slabs are 6 to 12 meters long, 1 to 2 meters wide, 225 millimeters thick and weigh in at about 20 tons each. The steel slabs are then sent to the rolling mill, where they are to be flattened and rolled into coils. The length of these coils can get up to multiple kilometers in length while the thickness is reduced to several millimeters. Following this multiple processes can be applied to the coil, such as rolling, pickling, coating, cutting and welding. These processes are applied by different installations, where each installation is part of a work unit. Of course Tata Steel wants to keep track of what happens to all of their materials and to be able to trace back where each piece of a coil originates from. For this they have monitoring systems at all their installations that measure and store data relevant to the installation in several databases. The relevant data from the databases of all these installations are then combined into a single standardized table called the Entry/Exit-table (EE). In the EE you can find data for the materials on a coil level. However these coils can be cut into multiple parts and these parts can then be welded together with parts from other coils to create a new coil. To get a better overview, Tata Steel also has a system in place that can track parts with a unique production history. These parts with a unique history are referred to as source pieces. The database where the information about source pieces is stored is called the Material Tracking Table (MTT), which is used in many tools throughout the company. This table is often used by Tata Steel employees to trace faults back to a certain point in production. As it is quite hard to find the required information by looking at the data alone, Tata Steel has a visualisation tool for this. This tool visualises the data from the MTT as a tree and shows information, such as direction switches and cuts in an intuitive manner.

2 Problem statement

The goal of our project is to build the third iteration of the Material Tracking Table (MTT3), a table containing the data necessary to track final products throughout production. Building and verifying MTT3 consists of three main steps, namely:

1. collecting and preprocessing the data from all work units
2. using the data from step 1 to construct MTT3
3. verifying the results using a visualisation tool

To properly tackle the problem each step has to be implemented into our project. First step 1 is discussed in section 2.1, step 2 and 3 are discussed in section 2.2 and section 2.3 respectively. In section 2.4 the problems stated in the first three subsections are used to create a list of requirements.

2.1. Entry/Exit-table

Tata Steel wants to be able to track their steel throughout the production process. In this process the product will pass through different installations, all processing the product in varying ways. Each individual work unit keeps track of different sets of data concerning the corresponding process. At the moment, this data does not follow a coherent style. Column names differ, weights use different units, some fields are unavailable, etc. As such a translation step is needed to unify this data. The Entry/Exit-table (EE) is this unification and serves as an input for the MTT3, described in section 2.2.

The EE is thus a selection on an overlapping set of Columns split between different work units, each with their own databases and table structures. Currently this translation and unification is done by a system using a set of scripts written in the programming language COBOL. The problem with this is that the current situation is difficult to maintain. Fixing bugs and adding support for data from new work units or new data sources, is at the moment not that easy to achieve. As the amount of people familiar with the COBOL programming language at Tata Steel and in general has dropped, it is in the interest of the company to replace this system.

As such, a new system for EE construction has to be made. The main goal is to create an easy to understand and manageable architecture for this new system, written in a more well-versed programming language. This allows work units to be added to the EE in a modular and systematic fashion. Due to the scope of the project and the time it takes to understand how the many various work units store their data, it is the goal of this sub-project to build a framework with expandability and adaptability in mind, to allow Tata Steel to expand this implementation to incorporate all work units. To showcase the effectiveness of this framework and to validate its correctness, the framework will be implemented for two installations.

2.2. MTT3

The main goal of MTT3 is to allow employees to track final products throughout production and identify where possible flaws have been introduced. Furthermore the table should provide data that is useful for other applications. The table is constructed using the data from the previously mentioned EE. Each entry in the EE corresponds to a process applied to a piece of material. In the EE, when a process has been applied to a material its ID can change. Therefore each entry contains the necessary keys to identify the input piece as well as the output piece and other necessary information. If a piece of material is cut into multiple pieces a new entry is created for each output piece.

The main difference between MTT3 and the EE is that MTT3 contains a route identification field for each path from a source piece to a final product, therefore allowing all final products to be easily tracked back throughout the entire production. A visualisation of this process is given in Figure 2.1. In this specific process two pieces of material are created. These pieces are then welded together and in the final step the welded material is split into three parts. The number left of the material represents its unique identifier and each letter represents the route identification.

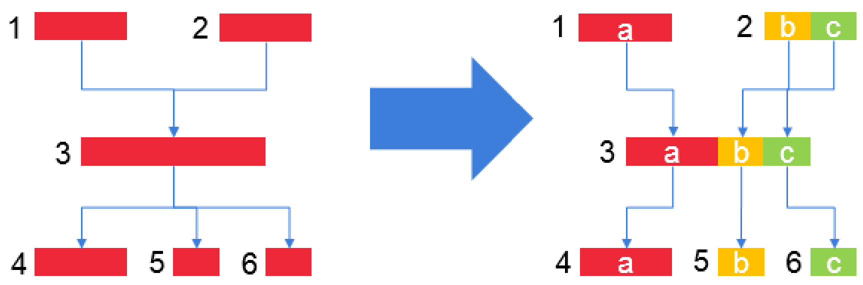


Figure 2.1: Tracking material in the EE (left) and in the MTT (right)

Each downward arrow in Figure 2.1 represents an entry in the respective table. A basic representation of how these entries would look in the EE and the MTT is given in Table 2.1.

Table 2.1: EE and MTT simplified data entries

Table 2.2: EE entries

ID_INPUT	ID_OUTPUT
1	3
2	3
3	4
3	5
3	6

Table 2.3: MTT entries

ROUTE_ID	ID_INPUT	ID_OUTPUT
a	1	3
b	2	3
c	2	3
a	3	4
b	3	5
c	3	6

However building a table such as Table 2.3 introduces a few problems. For example since the entries are dependent on their successors, already existing entries could have to be updated when new entries are introduced. Another problem is fitting the width and length of each final product such that their position in the previous coils is as accurate as possible. This second problem can either be introduced by measurement errors in the EE or pieces of material that have disappeared due to incorrect documentation. Due to these errors multiple outputs from a single piece of material might not fit exactly in the input material. This process is made harder by the fact that coils can be cut both in the width direction as in the length direction, introducing cuts as visualised in Figure 2.2.

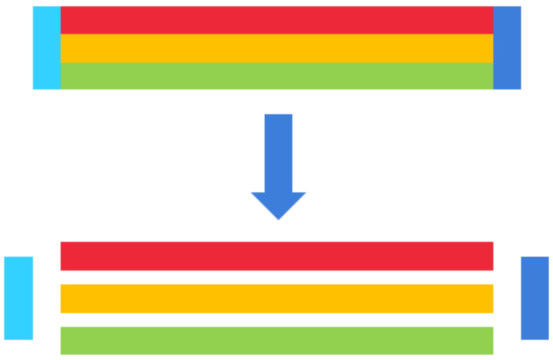


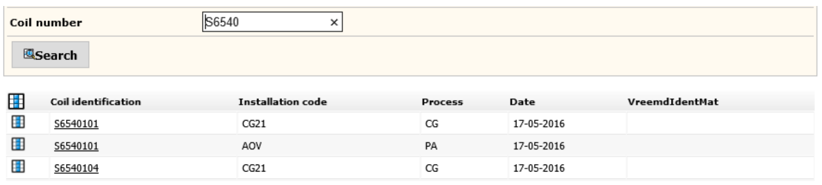
Figure 2.2: Material being cut into multiple parts

As of now two Material Tracking Tables exist, namely MTT1 and MTT2. However due to the bad scaling for larger data sets the construction of these tables from the existing EE takes anywhere from 1 to 6 months, and is therefore generally considered as unfeasible to do. Furthermore updating values in the table also introduces a lot of recursive updates within the table, making this also a challenging and time-expensive task. Even though daily updates are possible, fixing errors in previously used data is still a problem. As of now these these fixes would have to be done by recalculating the entire MTT, making it undesirable to do.

The goal of MTT3 is the same as for MTT1 and MTT2, but the fundamental difference is that MTT3 should be constructable in at most a couple of hours and should also be updateable without a complete recalculation.

2.3. Visualisation tool

The last of the three steps needed to complete our project is the creation of a new visualisation tool. The main goal of this tool is to be able to visualise the data of the older MTT2 and the newly developed MTT3 to give a better overview of how products developed by Tata Steel are changed during their production. A visualisation tool is necessary to make the data easier to understand for users as well as to be used as a debugging tool for MTT2 and MTT3. The reason why MTT1 is not taken into account here is because there already is a tool that can visualise this data which was developed by an external team. This tool works as follows: The user can provide a coil number (the number used to identify said coil) and is presented a list of options on the selection screen as shown in Figure 2.3. Should the user then click on "S6540104" for instance then a "family tree" of this coil will be constructed. A family tree is essentially the visualisation of the routes all the source pieces in a coil have traversed to come to this point (this is the coil that was selected) and where they have gone afterwards. In the family tree of "S6540104" shown in Figure 2.4, one can for instance see that this particular coil has gone through 5 more facilities before it got cut up into 6 pieces, of which 2 finally went on to the customer.



Coil number	S6540

	Coil identification	Installation code	Process	Date	VreemdIdentMat
▦	S6540101	CG21	CG	17-05-2016	
▦	S6540101	AOV	PA	17-05-2016	
▦	S6540104	CG21	CG	17-05-2016	

Figure 2.3: Selection screen of the old visualisation tool

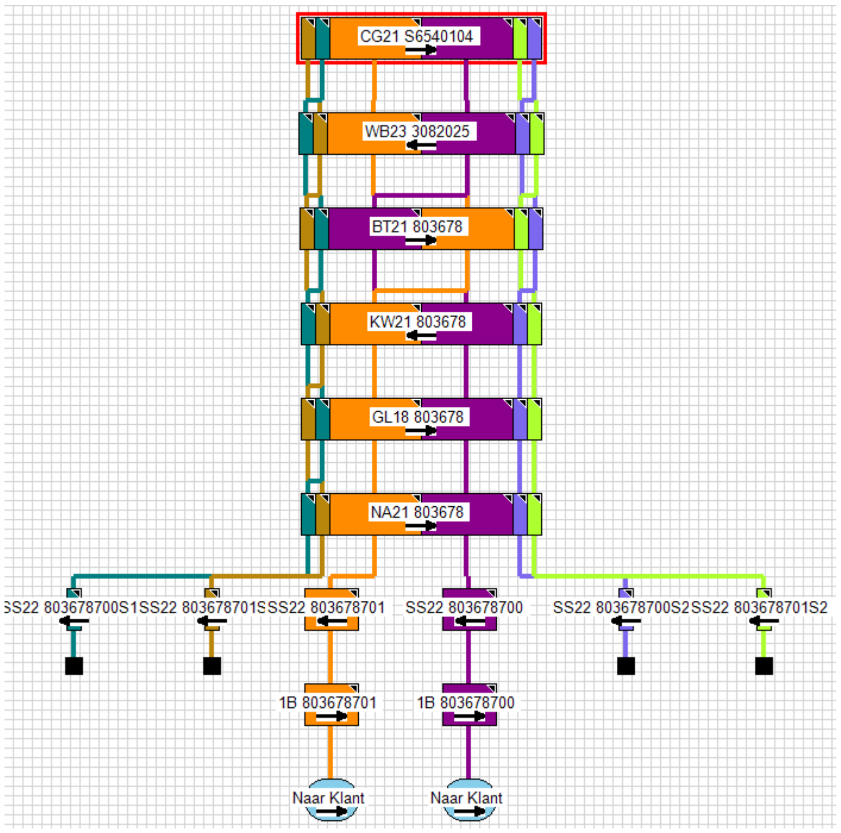


Figure 2.4: Family tree visualised by the old visualisation tool

This tool, however, has its downsides. For instance it was developed in .NET which is a language that only a few, if any, of the developers within the departments using the tool are familiar with. As a consequence, the tool is hard to maintain for the departments that work with it and because it was made by an external team there is also no one available on short notice to work on problems that the tool has. This is why a new tool was desired, to properly show the new and more up-to-date MTT2 and MTT3 and at the same time fix some of the flaws the old tool had, such as not being able to deal with coils that have been slit in the length. The new tool will also add all desired functionality from the original tool as well as add newly desired functionality, such as zooming in on smaller parts.

2.4. Requirements

In appendix B we note the requirements for our solution to each of the previously discussed subproblems. These requirements were made based on the feedback received from multiple Tata Steel employees.

To note the requirements of the problem we use the well-known MoSCoW *(must-haves, should-haves, could-haves, won't haves)* method, as it not only shows the requirements, but also their priority.

3 Process

In this section we outline our work process within our group and the general working structure within the company.

3.1. Working at Tata Steel

During the project we worked at 2 locations. Part of the week we worked at the Tata Steel facility in IJmuiden and the other part at the TU Delft. At Tata Steel IJmuiden we worked in an office with our client. Because we worked at Tata Steel, we also adopted their work style. This meant that we worked using the Agile method. At the start of the day we would have a stand-up meeting where everyone would discuss what they were going to do and we ended the day with another stand-up meeting where everyone was updated on what happened that day, what the plans were for the next day and whether anyone was struggling and needed some help. When we worked in Delft we did these meetings through Skype. Sometimes it happened that our client was not available for the daily meetings, we would then have these meetings without the client and summarize them shortly at the next meeting.

3.2. The wall

Tata Steel also uses the walls of their office to hang slides with information about the current state of the project as seen in Figure 3.1. The idea of this is that you could explain to your colleagues and yourself what is happening with the project at the moment based on these slides. It took some time to get used to making new slides every time you updated a part of the project, but in the end we quite liked this approach as you had all relevant information on one central point. The wall was also often used at the stand-up meetings to aid in your explanations.

Figure 3.1: Wall with slides

3.3. Continuous Integration

To maintain our code base we used pull request based development. This means that new code is first reviewed by at least two other group members, before it can be included into the final code base. This approach helps find errors before they become an actual problem and forces everyone to adhere to strict standards, as your code would not be approved otherwise. Furthermore we used continuous integration to ensure our code base was kept clean. When a commit was pushed a build was made automatically, which checked whether the code compiled, whether it contained no linting errors and whether the tests succeeded.

3.4. Leadership

Early on in the project we identified 3 different sub-problems for our project. To each of these sub-problems a lead was assigned, and as we had 4 member of our group one of the sub-problems had 2 leads. The 2 leads were assigned to creating the MTT, which was later split into 2 smaller tasks: Clustering the input data and processing it in the MTT-engine. This meant that effectively everyone was lead of a task. The lead was responsible for maintaining his section of the wall and updating the planning. At the daily meetings the lead of the sub-problems would update the rest about the progress.

3.5. Presentations

At Tata Steel we were in direct contact with our supervisor Wouter Lamme. However as there were more people at Tata interested in the state of the project we hosted multiple presentations to keep them updated. There was a Kick-Off shortly after the research phase, a Mid-way and a final presentation. In these presentations we would update the interested parties about the current state of the project and they would give their feedback and help steer us in the right direction, if they believed we were off about parts of the project.

4
Design

Our final code base consists of three main components, namely:

1. Build the Entry/Exit-table
2. Build MTT3
3. Visualise the result

When executing the complete process, the first step is to build the Entry/Exit-table by retrieving and preprocessing data from all installations. For each work unit, consisting of one or more installations, a module is written to convert the data to a unified format. The results from these modules are then combined to create the Entry/Exit-table.

Secondly, the new Material Tracking Table is built. This process is divided into two steps, namely clustering the data from the Entry/Exit-table, such that data that is not within the same cluster is not correlated, and processing the clustered data to generate the MTT. The results from this step can then be used for external applications.

The final component is the visualisation tool, which is also an example of an external application using the data resulting from the second component. This tool is able to visualise MTT2 as well as MTT3 and can be used by employees of Tata Steel to more easily understand the production history of a coil.

The complete process discussed above is visualised in Figure 4.1, where each of the components has a differently coloured background.

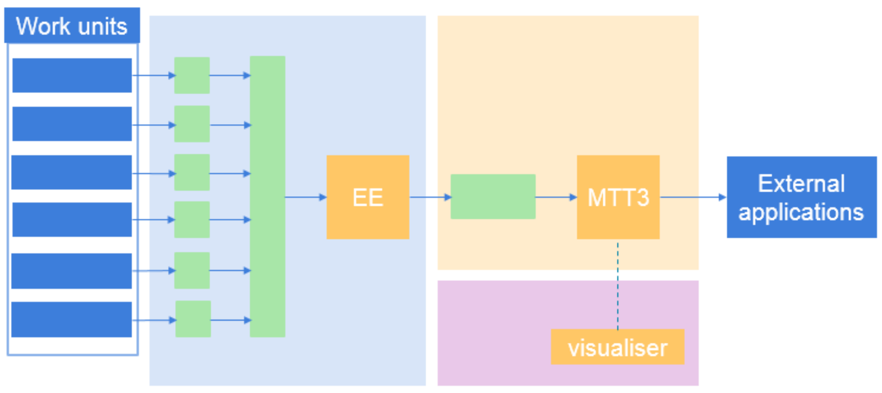


Figure 4.1: The complete process

In this chapter each of the components mentioned above are discussed in greater detail in their respective section.

4.1. Entry/Exit-table

The framework for the new EE engine should be a framework with a focus on expandability and maintainability.

To make this engine expandable one of the key factors is making it easy to add support for new work units. Adding support for a new work units involves two things. A list of column specifications and a join specification for the required data.

The engine works by doing a select on a single prepared data set. This data set can come from any number of sources. As the engine uses Spark it has support for a wide variety of these data sets [1]. Most work units need to access data from a combination of data sources. As such a specification is needed to join these. This join can either be done in a single database environment resulting in the engine getting a single data set. For example, a database containing multiple tables can be joined through an SQL statement and sent to Spark or if the tables are split over multiple different database environment the data could be joined through Spark.

Column	**Specification**	**Implementation Example**
ID_INPUT	The column LOADNR from table A in database X concattenated with the column BATCHNR from table C in database Y.	concat("LOADNR", "BATCHNR")
TS_INPUT	The column INSERT_DATE from table B in database X.	col("INSERT_DATE")
WORK_UNIT	The string "COAL".	lit("COAL")
WEIGHT	The column WEIGHT from table A in database X in kg.	col("WEIGHT") * 1000

Figure 4.2: Example list of column specifications

On such a joined data set the engine then does a translation. For this the column specifications, as seen in figure 4.2, are needed. These mostly involve getting the right columns, setting of constant columns and changing units. Implementing these involves creating a class inheriting from a base class containing a function for every column. In this child class each of these functions needs to be overridden with this selection specification on the joined data set.

The overall EE engine thus involves:

1. Creating a joined data set for every work unit.
2. Passing this data set to an instance of that work units specified class.
3. For each work unit instance calling its EE engine.
4. Unionizing the resulting processed data sets.
5. Finally calculating a set of variables in a post process

Resulting in an Entry/Exit table.

[1] `https://spark.apache.org/docs/latest/sql-data-sources.html`

4.2. MTT3

The main issue with the current MTT-engine is that it is lacking in performance. To tackle this issue we have decided to cluster the data that is fed into the engine. The reasoning for this decision and theory behind the clustering are given in subsection 4.2.1. To properly adjust to the clustered data the MTT-engine also had to be rewritten. The design decisions made while designing the new engine are specified in subsection 4.2.2. These two components are designed so that they can be ran separately. However before the MTT-engine can be ran the clustering has to be ran once. The result from the clustering can then be saved to a file or database, so that the MTT-engine can be directly ran on this data instead of having to cluster the data again.

4.2.1. Clustering

If we have an end product and want to calculate the MTT entries for it not all entries in the data set are relevant. The only entries necessary for calculation are the products predecessors and the other outputs of the direct predecessor. An example of this is shown in figure 4.3 below. To calculate the MTT entries generated by the red material we only need the data of the green entries and not the grey entries as they are not related.

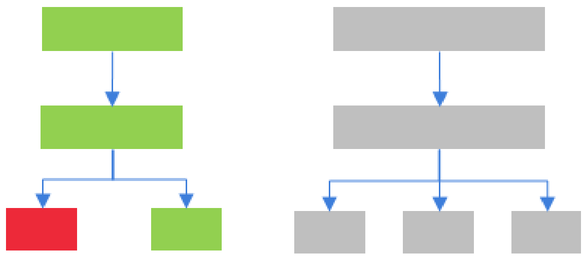


Figure 4.3: EE example

Since the MTT-engine contains a lot of functions with a bad time complexity, limiting the data could limit the size of the data that needs to be queried as well as introduce the possibility of parallelisation.

To easily find these clusters of related entries we have decided to apply Graph theory. We create a Node for all the materials and an edge between nodes if one is produced from the other. We then get a network of materials, some of which are connected to others. We are interested in groups of nodes which are connected to each other. These are the connected components in the graph, which we can easily find using for example a Breadth First Search Algorithm. The result of this would look similar to figure 4.4.

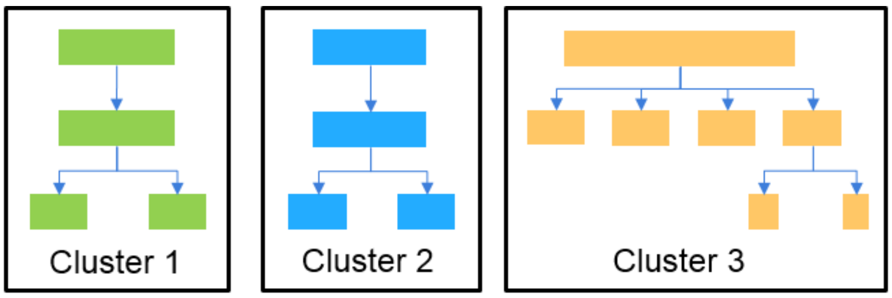


Figure 4.4: Clustered EE example

The input data for this step can be quite large if the entire MTT has to be recalculated, and as Tata Steel produces more material every day, this keeps growing. To make sure our algorithm is able to handle these amounts of data even in the future we decided to use an approach similar to Divide and Conquer. If the dataset is too big to be processed all at once, the data is split into two or more batches of equal size. The algorithm will then find the connected components in these batches one by one and save the results to a temporary file. After all batches have been processed the data is collected from file and combined. In this combination step we search for connections between the different batches. This means that the two materials are connected and should therefore be in the same cluster. The corresponding cluster ID's of the materials are then saved so that the can be merged later on. After all connections between the batches have been found the clusters are merged by changing all occurences of one of the clusters to the other cluster. When all clusters have been finalized the resulting data will be saved so that it can be used by the MTT-engine.

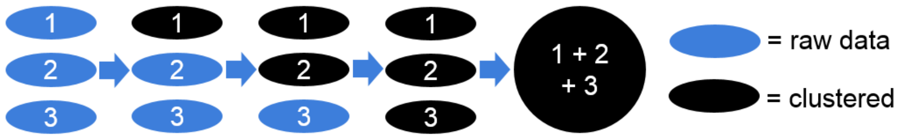


Figure 4.5: Clustering in batches

As we do not want to rebuild the entire MTT every time new data is appended to the EE, there needs to be an update function that can be run on a regular basis (daily). Adding new data to an already clustered dataset is quite easy. We can see the new data as a new batch and the rest of our data as the already clustered data. If the new batch is clustered it can be combined with the clustered data with the same join as described above. After this a list of affected batches is returned, so that the MTT-engine knows which clusters need to be updated.

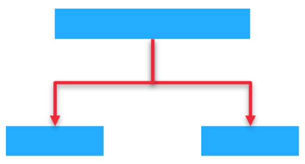
4.2.2. MTT-Engine

Once the data has been clustered it still has to be processed to generate the new MTT. However the current MTT-engine is not built to run on clusters. Therefore we have decided to write a renewed engine, as it allows us to better understand the structure of the MTT, as well as to implement optimisations based on the knowledge that it is ran on clusters.

The main difference between our MTT-engine and the existing one is that the new engine processes clusters in parallel. Because clustering allows us to conclude that items that are not in the same cluster have no influence on each other, we can process multiple clusters in parallel.

When processing a single cluster, entries with the same input are processed simultaneously, forming a group of entries. All groups of entries can be assigned to one of three groups based on whether the entries have a predecessor in the already processed data and the amount of entries in the group:

No predecessor The first case is when the selected entries have no predecessor in the already processed data, meaning that the materials have not been seen before. This case is visualised in Figure 4.6a, where arrows in red represent the entries being processed.Figure 4.6b represents the MTT entries generated by the given EE entries, also marked in red.


(a) Data entries from EE

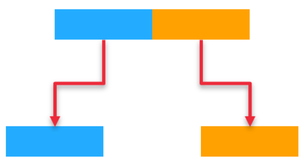
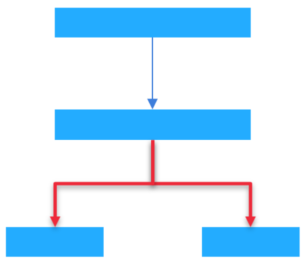

(b) Respective data entries in MTT

Figure 4.6: First case of processing entries

Multiple entries and a predecessor The second case is when there are multiple entries being processed and they have a predecessor in the already processed data. Using the same format as the previous case this case is visualised in Figure 4.7. It is important to note that all arrows in Figure 4.7b are red, as previous entries have to be altered due to the multiple outputs from a single input material.


(a) Data entries from EE

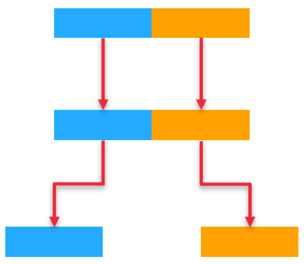
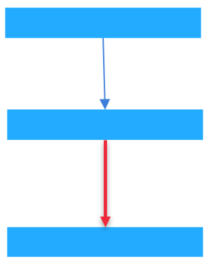

(b) Respective data entries in MTT

Figure 4.7: Second case of processing entries

Single entry and a predecessor The last case is when there is only one entry being processed and it has a predecessor in the processed data. Using the same format as the first case this case is visualised in Figure 4.8. Due to the single output no recursive updates have to be executed.


(a) Data entries from EE

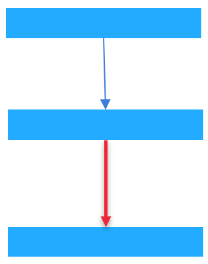

(b) Respective data entries in MTT

Figure 4.8: Third case of processing entries

Each case requires a different approach for processing the data, as some need recursive updates and data from predecessors while other approaches might not.

An important conclusion we can make from these different approaches is that the order in which the entries are processed is of huge importance. This means that the engine needs to sort the data before it is processed. However due to the clusters, the engine can sort each cluster individually. Doing this not only decreases the amount of data that needs to be sorted at once, but also allows the data to be sorted in parallel.

Furthermore, rather than rebuilding the entire MTT for each daily update, we have added an update function. This function limits the rebuilding of the MTT to the clusters affected by the data added that specific day.

4.3. Visualisation tool

As discussed in section 2.3 the main goal of the visualisation tool is to properly visualise the data from MTT2 and MTT3, while keeping desired functionality of the older tool. Properly visualising this data means that the data from MTT2 and MTT3, which is not fully included in MTT1, will be used to display the source pieces as they were during the development process. This new information can be used to fix some of the more major shortcomings of the old tool. Aside from fixing shortcomings of the tool, different functionality has been added and unnecessary functionality has been removed, therefore this section will be split in three parts. In subsection 4.3.1 the shortcomings of the old tool are discussed and how the new tool fixes these. In subsection 4.3.2 the functionality that is left out from the previous tool as it is considered redundant will be explained.

4.3.1. Fixing shortcomings

Drawing according to MTT2 and MTT3 data The first and most major shortcoming of the old visualisation tool was not being able to deal with coils that have been slit in the length. The old tool now just takes the percentage the size of the source piece is in regards to size of the whole coil and takes that as the length of that part in the drawn coil. All these parts are then displayed next to each other regardless of their original placing in the coil. In the new tool the data used to draw the coils is extracted from MTT2 and MTT3 which both have more information about a coil than MTT1 has. Using this information it is possible to deduce the origin coordinate of the source piece in the coil and draw the source piece from there using its own width and length. The result is the change visualised in Figure 4.9. In this new layout it is easy visible where a source piece would be in the coil and how they are ordered inside of the coil.

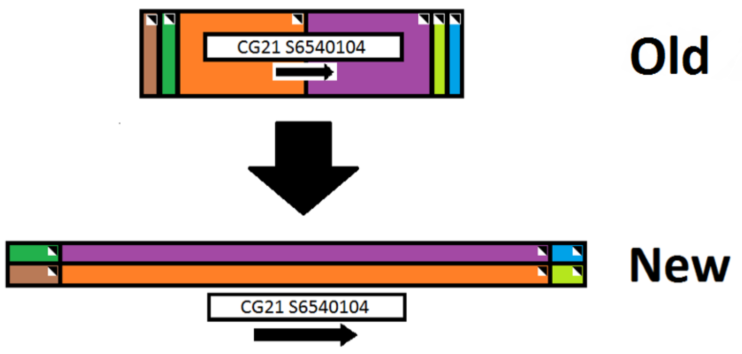


Figure 4.9: Difference between old and new visualisation tool

Removing switching parts Another shortcoming of the old tool is not immediately a lack of functionality but more of a by-product of the first shortcoming. Namely when constructing the family tree in the old tool the positions of two source pieces with the same starting position in a coil is decided by which piece is processed first. As there is no sorting in this process this is seemingly the same as taking a random one from the two and drawing that one first. As a consequence parts can randomly change positions with other parts in the old visualisation tool as is shown in Figure 4.10, causing lots of confusion in users of the application that do not have much experience with the tool and also making it harder to judge if the data is correct as errors in the data with regard to the position of a source piece in a coil are now almost unverifiable. Therefore the new implementation will be using the position data present in MTT2 and MTT3 to let this issue disappear completely.

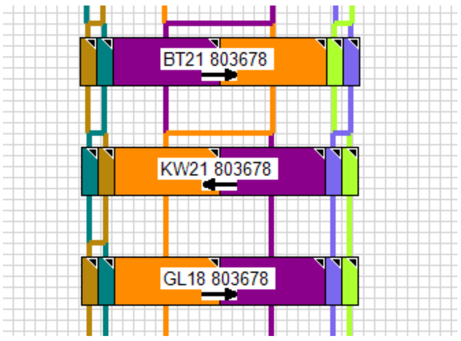


Figure 4.10: Changing part locations in old tool

Switch screen for getting extra information A last shortcoming that the new tool improves upon the old tool is that of displaying extra information about source pieces and coils. In the old tool to get extra information about source pieces such as their route-ident or their thickness, a new page would be loaded showing this data. This means that the family tree, which is a visualisation to better understand the data, disappears if you want to look at said data. In the new visualisation tool this is changed so that hovering over a source piece with your mouse shows a pop-up containing important information of this source piece so that it is available information while looking at the family tree.

Overlapping information In the old visualisation tool, the family tree is generated with a fixed size. This means that some parts are small and information on these is sometimes not displayed correctly or overlapping as shown in Figure 4.11. This is why in the development of the new visualisation tool some new functionality was added, namely zooming and dragging of the family tree. By zooming in on the tree the small parts will be more easily distinguishable and by allowing dragging functionality the family tree is not limited to a fixed width or height as it can be moved until the correct place is in view. By having these extra options the ordering of the family tree can be done in a less limited way, therefore getting rid of the overlapping text the old tree displays.

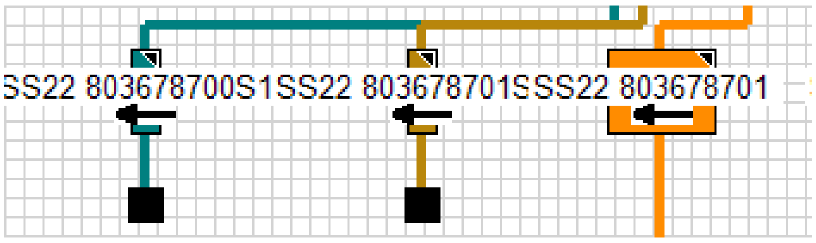


Figure 4.11: Small pieces in family tree with overlapping text

4.3.2. Removed functionality

Replace route-ident lines In the old visualisation tool, lines are used to further clarify the association of source pieces with the same route-ident in the family tree. This association is however also shown by the color of the source pieces. This further visualisation is not necessarily something bad but it can get messy very fast in some facilities as shown in Figure 4.12. Therefore this is replaced in the new visualisation tool with lines connecting coils rather than source pieces.

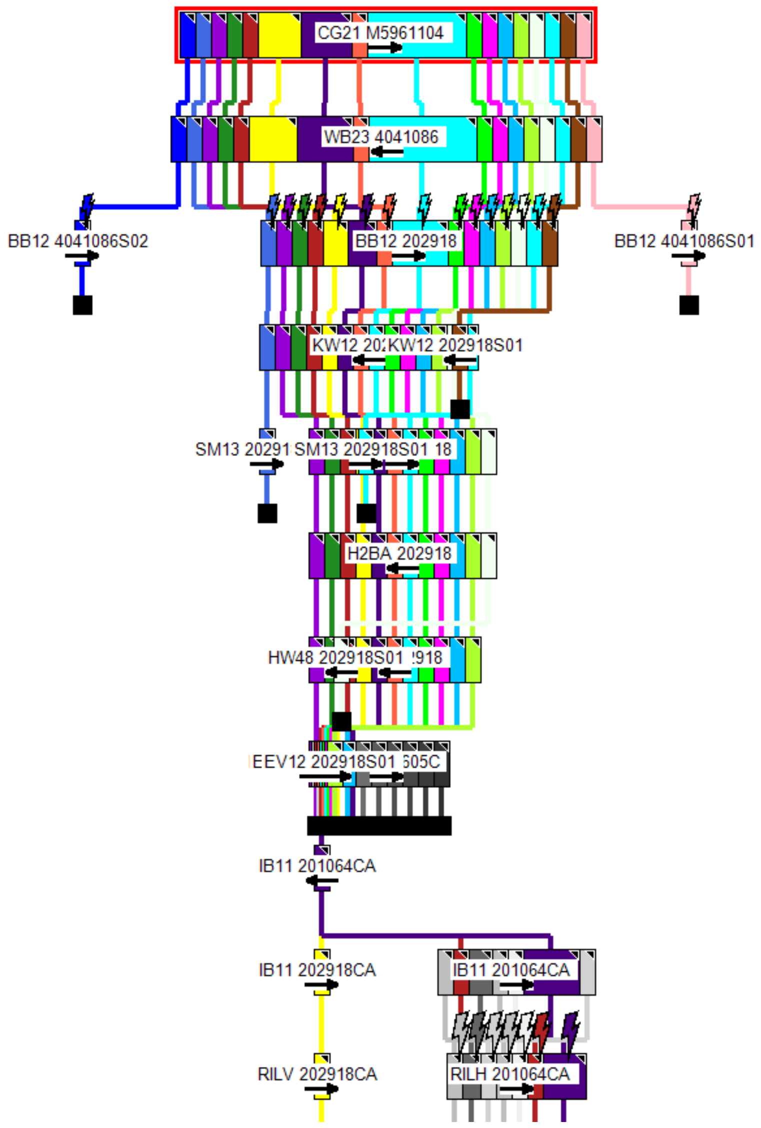


Figure 4.12: Family tree for a coil in packaging

Remove end-of-data-blocks Just as with the route-ident lines, these blocks do not add to the functionality of the tool and can get very messy in certain cases. In Figure 4.12 only three parts come out of the process on EEV12, but where in the material they are from is almost untraceable because of the black bar beneath it, which is a line of end-of-data-blocks. The blocks themselves also do not add any certainty about the state of these source pieces, as it just means that no further data is available from this point on. Therefore this is also removed in the new visualisation tool in consultation with the next users of the tool.

5 Implementation

In this chapter we discuss the choices we made when implementing the system and the challenges that occurred during development.

5.1. Languages & Tools

Spark

As we are working with large data sets for the clustering algorithm we are using Spark. The key point being that Spark can be run on a scalable cluster computing system, allowing Tata Steel to scale up performance based on financial interest. Spark is used at multiple points in the algorithm, from Spark we mostly make use of the DataFrames. These are similar to tables in a relational database, and these represent our input data from the EE. The biggest operations run on Spark are, collection of the data and an inner join used to find clusters that should be merged together. An additional benefit is that Spark is also already being used internally at Tata Steel, so this is a known language for them.

iGraph

Instead of attempting to implement our own algorithm to find clusters within the data, we decided to use a fast and stable library that can do that for us. We ended up choosing iGraph, which is a Python library whose core code is written in C. This means that it is both faster and more memory efficient than a library written entirely in Python. We use this library to create a graph containing all of our input data and to find the connected components within that graph.

JavaScript, HTML, CSS, Node.js

As the old visualisation tool is a web application we decided to follow suit, as this makes it possible to host one server for the whole company. The old visualisation tool was coded in .NET however which was outside of the expertise of developers at Tata Steel, which is why we decided to switch to a more well known approach for web development. This means the website was made in HTML, the scripting was done in JavaScript and the styling was mostly done in CSS, as these are mostly the standards for web development nowadays. For hosting the visualisation tool we decided to use Node.js, as it can be used to obstruct the connection to the database from the user. Furthermore Node is developed with scalability in mind, allowing it to handle a large amount of simultaneous users, which is a very important requirement for the visualisation tool as the tool is to be used by many employees of Tata Steel at the same time.

Python
We decided to use Python in our project for a multitude of reasons. Firstly, as manageability is one of the main requirements, Python is one of the most well known languages and is taught in most studies. Secondly, Python is already being used within Tata Steel. Thirdly, Python is one of the main languages used by Data scientists, which means that new recruits at Tata Steel will most likely have some Python experience, making it easier for them to step into this project. Finally using PySpark we can use Spark within Python.

Azure DevOps
For development we used Azure DevOps, as it offers a lot of useful functionality, such as a built in Scrum board and pipeline management tools. It is also the main development tool used by Tata Steel and allows them to keep the code private. As DevOps uses Git for its version control we also work with this to allow everyone of the team to work on the code base simultaneously.

5.2. Challenges

In this section the different implementation challenges are outlined.

5.2.1. Entry/Exit-table

Legacy code
The old EE engine system involved two separate code bases. One older system written in COBOL and one section written in PL/SQL. The COBOL section generated an older version of the EE which would then be translated by the PL/SQL to a newer version. This process was to be done in a single step. So instead of column $a \to b \to c$, we would go straight from a to c. This could be done for most columns but some did indeed need a step in between, as these relied on the order of and result of those specified columns. Understanding these interactions and was quite a challenge in an undocumented landscape of lesser known programming languages.

Wrong column specifications
Finding out the column specifications for the two implemented work units seemed quite easy with the given documentation. But this documentation turned out to not always match the source code, as such the document became almost useless because every column required a check with the source code. Again the involved programming languages, COBOL and PL/SQL, were written in an almost GOTO style of coding, having global variables being mutated all over the project. This made the understanding of the given translation on these fields hard to grasp. Next to this we know that the old EE engine is not always correct, as such decisions based on what these columns should then be had to be made. With the limited knowledge of the inner workings of all these work units, that becomes quite hard. This is where our supervisor would decide on the final value to be used.

5.2.2. MTT3

RAM
The first problem we ran into when implementing the clustering algorithm was the high use of RAM. Processing 2 years of data requires about 11GB of available RAM. Seeing as the amount of required RAM scales with the amount of years being processed, this introduced a new problem. For this problem we had 2 solutions. The first option was to buy more RAM, while easy this solution was of course not optimal. The second option was to improve the algorithm to use less RAM. This was done via processing in batches. After clustering a batch of data it can be evicted from RAM and a new batch can be loaded. While this affects performance it does allow to process large amounts of data on a PC with limited RAM. We split the data in batches of equal size based on the first and last date in the input. After clustering the individual batches the data is joined with itself to find clusters

that should be merged. In Figure 5.1 below a linear relation between the amount of data and the amount of RAM used can be seen. Using this relation we can determine the optimal batch size for any system if we know the available amount of RAM.

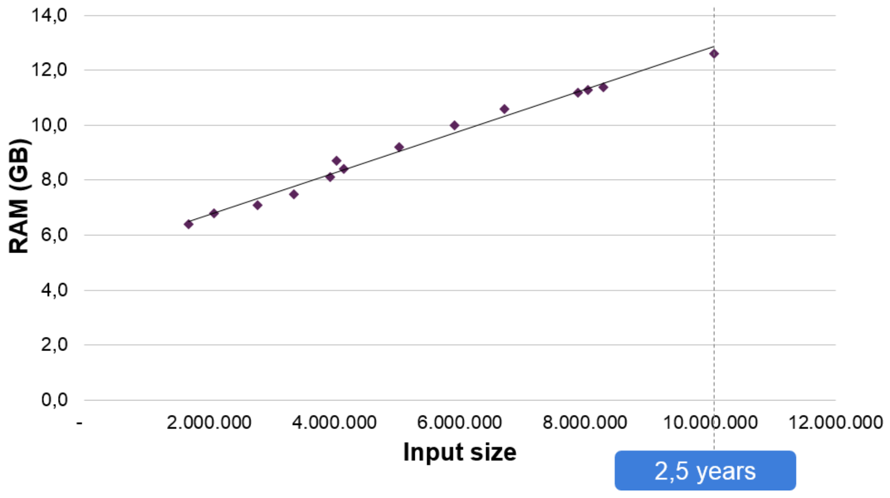


Figure 5.1: RAM usage (clustering)

Because we were mostly limited by the amount of RAM available on the system, we wanted to make sure we were not using that for anything that was not necessary. This became an important focus for the clustering algorithm. To make sure the memory was being used as efficiently as possible we deleted all the data after it was used. A graph displaying the used memory over time is shown below in Figure 5.2. For this graph we clustered a dataset in two batches, this can be seen in the graph by the two spikes to 12GB. From the graph it is also clear that this memory gets freed after the batch is done, which allows us to use bigger batches which makes the program run faster.

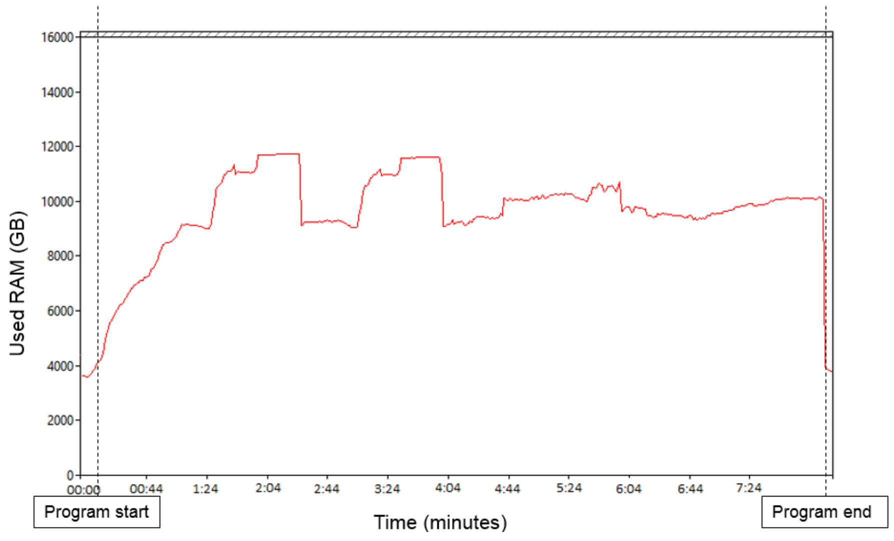


Figure 5.2: RAM usage (clustering)

Large clusters

After finishing the first iteration of the clustering algorithm and looking at the results, we noticed something weird. One of the clusters contained about $\frac{1}{3}$ of the input data. After running some analysis on the sources of the data in the cluster we noticed that all the data originated from Tata

Steel Packaging (TSP), one of the work units at Tata. What happens at TSP is that they take a lot of small pieces of different coils and weld them together to a single big coil. This meant that a lot of data was related and therefore placed in the same cluster. Later on we however learned that materials that are welded together do not affect each other in the MTT. This meant that we could ignore welds in clustering and make smaller clusters. In Figure 5.3 the difference can be seen if the clustering algorithm ignores welds. This change meant that the largest cluster was now a more reasonable number, about $\frac{1}{10000}$ of the input data.

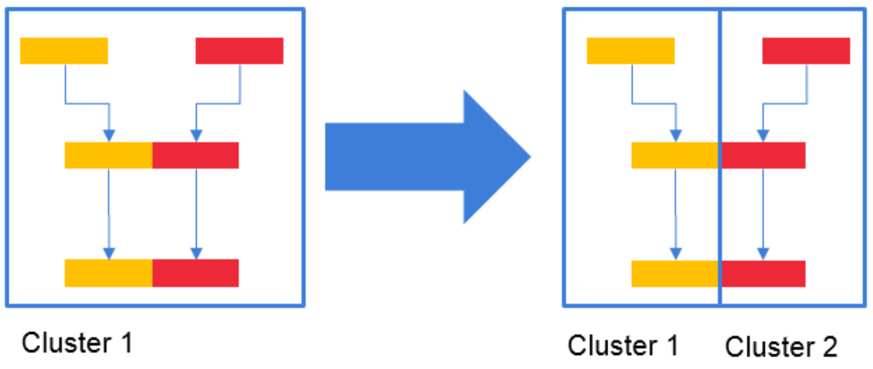


Figure 5.3: Clustering without welds

Legacy code

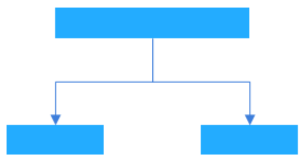
The new MTT-engine is based on the already existing engine. To write the new engine we needed to understand the already existing engine, written in PL/SQL. Luckily Tata Steel also provided multiple documents about the code, however these documents did not always match. Sometimes no or multiple definitions were given for a single attribute. We fixed these issues by looking at the existing implementation and using it to identify the correct definition.

Invalid input data

The data used for the clustering comes from many different factories which work with their own systems and databases. Combine that with the fact that there are 10.000+ new entries every day, you can imagine that from time to time something goes wrong. These errors range from null values to input time stamp which are later than the output time stamp. While most of these errors do not affect the clustering as it only uses a limited amount of columns, some do break the code. During development it would often happen that different settings would yield different results. The cause of these bugs sometimes took a while to find and fix, but currently these invalid inputs do not break the program anymore.

Late entries

Each day updates are ran at a specific time. Due to this when a coil is cut into multiple pieces not all the data about these cuts might be available when the update is ran. Therefore on the the first day the processed data might look as visualised in Figure 5.4.


(a) Data entries from EE

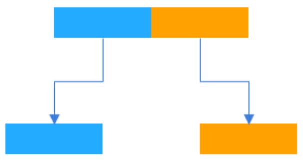
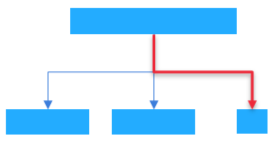

(b) Respective data entries in MTT

Figure 5.4: Data processed before late entry detection

However the next day a new entry can arrive that indicates that the coil should have been split into three parts instead of two, as visualised in Figure 5.5.


(a) Data entries from EE

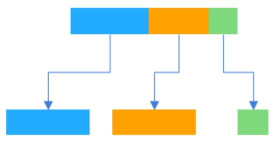

(b) Respective data entries in MTT

Figure 5.5: Data processed after late entry detection

Updating this entry is harder than a normal update, since the data from the previous entries is required. The first step to solving this challenge is to consistently identify late entries. This can be done by retrieving entries with the same input material from the already processed data. For each detected late entry the data from its entire cluster is retrieved from the EE and used to recalculate that cluster in the MTT. The existing data in the MTT for that cluster is then replaced with the recalculated data.

Error correction

It is possible for entries in the EE to contain errors and be corrected after they have been used to calculate the MTT. These entries can be corrected in the EE, but do not necessarily lead to a recalculation of the MTT. To solve this challenge, as well as the late entries, we have decided to recalculate all the clusters affected by the addition of the daily data. This does not guarantee that the correction is implemented in the MTT, however it does make it significantly more likely.

5.2.3. Visualisation tool

Acquiring legacy code

The first problem in the development of the visualisation tool was acquiring the code of the old visualisation tool. In the first two weeks of the project we were talking about how the new tool should be made and what should be kept from the old tool. To know what we needed to keep from the old tool we wanted to take a look at it and therefore asked the old team for the rights to the code. However, we did not get a response. After notifying the higher-ups a search was initiated and another week later the 2 owners of the code were found, one in India and one who was having a vacation in Holland. When asking the owner in India access was denied as our project in Holland was not known at Tata Steel India. Luckily a week later the owner in Holland was back from vacation and gave us the rights to the code, but at this moment 4 weeks had already passed, causing us to have already started our development of a new tool.

Legacy code

When we acquired the code of the old visualisation tool, we got a huge repository with an undocumented file structure. In these .NET files almost every file imported and used data from other files meaning that truly understanding what was written in the file meant first reading 2-10 different files which also needed more files to be read before understanding these. Some of these files were also imported libraries or native functions but because of the lack of documentation it was very hard to see which were and which were not part of their own developed code. Another problem in understanding the code was names that did not make immediate sense, for instance when referring to the percentage the size of a source piece is with regards to the size of a coil the used term is "Vlees" or in English "Meat". If this file structure had been properly documented and had variable names that would better identify what they represent this would have been way less of a challenge to understand.

Ordering of data

When implementing the functionality of the visualisation tool, figuring out how a family tree would be constructed was somewhat of a challenge. As the code of the old tool was not available at the start and as no one in the department we worked in knew how the tree was constructed this had to be thought over once again. By looking at the data piece by piece the order was eventually figured out, but this took some notable time.

Changing demands

During this project we have learned how to work in a company above all else. What was one of the most notable learning experiences we had was that no one in the company wants the exact same from the project. This was mostly seen with the visualisation tool as this is what the end users will mostly use. It therefore came as no surprise that in the timespan between the Kick-Off meeting and the Midterm meeting the demands of other people involved with the project had changed. This brought along that changes had to be made in the implementation, some small others major. Keeping up with these demands was therefore quite the challenge.

Time constrictions
The last and probably most noticeable challenge for the implementation of the new visualisation tool was time. As the bachelor end project only takes 11 weeks making a fully functional system to visualise all data from the materials in a clean and orderly way could be a whole project on its own. Therefore the tool we deliver is used mostly for debugging purposes of the data and mostly implemented with functionality in mind. This means that styling of the web application has been done in a lesser fashion and user testing periods have also been somewhat short, but all the functionality needed for the debugging has been added making the tool fulfill its requirements which are listed in Appendix B.

5.3. Testing

To ensure our system works we have written tests for the crucial parts of the system. To test the system we used unit tests as well as smoke tests. These tests are ran automatically by the continuous integration as part of the build. This means that if a pull request contains failing tests it can not be merged into the master. This ensures that code that breaks functionality, either directly or indirectly, can never end up on the master branch. The final coverage achieved by these tests are given in Figure 5.6.

File	line coverage	branch coverage
cumulative.py	97%	98%
fitting.py	100%	100%
position.py	100%	100%
ratio.py	99%	98%
batch_engine.py	100%	100%
mtt_functions.py	89%	89%
post_process.py	100%	100%

Figure 5.6: Test coverage MTT-engine

The tests were written using dummy data, as this allows us to change the exact values that influence the function that is being tested. To ensure the program was also able to run on the actual data, manual tests on the complete data and subsets of it were also performed.3

6
Results and Discussion

6.1. Entry/Exit-table

There now is a framework specified for the creation of an Entry/Exit table, written in a modern programming language, which has been implemented for two work units.

With the right knowledge of the systems, implementation of a new work units has been made relatively easy. Mostly involving answering the questions, where do I get my data from? and to which columns does the data belong? With this bare minimum knowledge writing the code requires a basic understanding of SQL-like Spark concepts. Most data sources are supported, through Spark's data importing system.

The code is now structured and documented, instead of bundled in a single undocumented file, important functions are tested, and variables are set at a single clear location, not depending on volatile global variables.

6.2. MTT3

The main goal of MTT3 compared to MTT2 was achieving a significant speedup. Currently if there is an error in some of the entries they cannot be recalculated as recalculating the entire MTT takes somewhere between 1 to 6 months. Our goal was to be able to calculate the MTT within 4 hours.

6.2.1. Clustering

The first step in the process is the clustering. This allows us to process multiple clusters in parallel which should help run the program faster. In Figure 6.1 the run time can be seen when running clustering on data sets of different sizes. In this figure we can see two things. First of all there are two jumps in the graph. The first at 3 years and the second at 5 years. These can be explained by our batch system. It is possible to process 2 years of data in a single batch, however 3 years needs to be done in 2 batches. This means that we have to combine batches which takes extra time. The jump at years 5 is caused by the fact that we have to use 3 batches instead of 2. The increase in overhead is less here, as we were already combining batches.

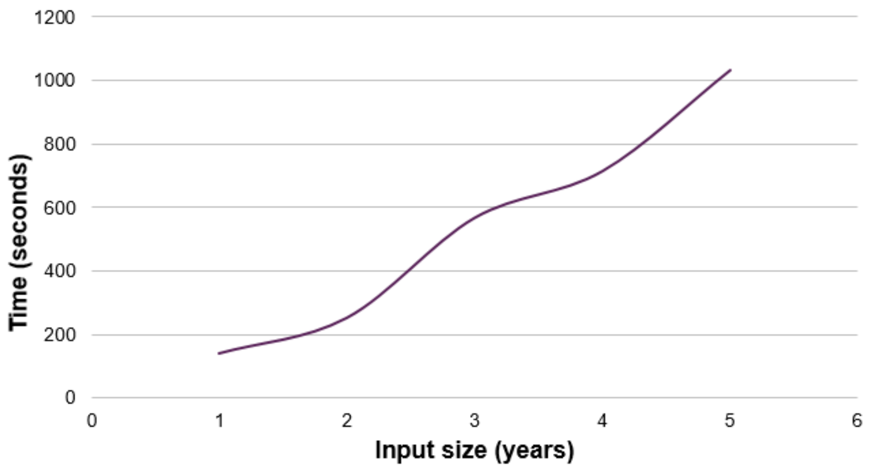


Figure 6.1: Clustering run time

To get a better idea of the difference in performance when processing in multiple batches we have clustered the same data set in different amounts of batches. The results are plotted in Figure 6.2. As you can see there is a big jump from doing it in 1 batch to using 2 batches. This is because the combining step is not done when processing in a single batch.

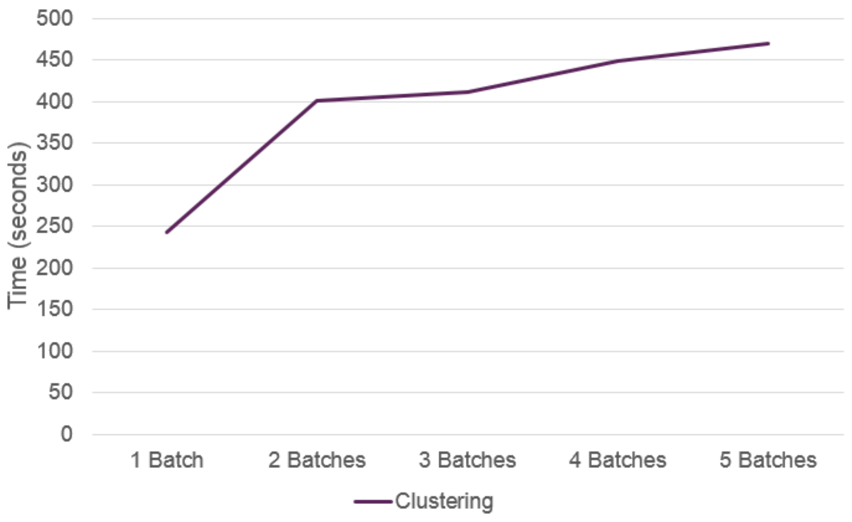


Figure 6.2: Clustering 2 years

As we are working with Spark we have the possibility to assign more workers to the program if we have the cpu power to run those workers. In order to know whether buying more cpu power would significantly speed up the program we can the program with different Spark settings. The results are plotted below in Figure 6.3. From this figure we can see that assigning multiple cores speeds up the program. however the speedup decreases after adding more cores. The speedup appears to have a logarithmic relation with the amount of cpu cores, which means that this is likely not the best way to achieve a big speedup. This figure also contains data about different amounts of batches. We can see that the amount of cpu cores does not affect the performance of multiple batches.

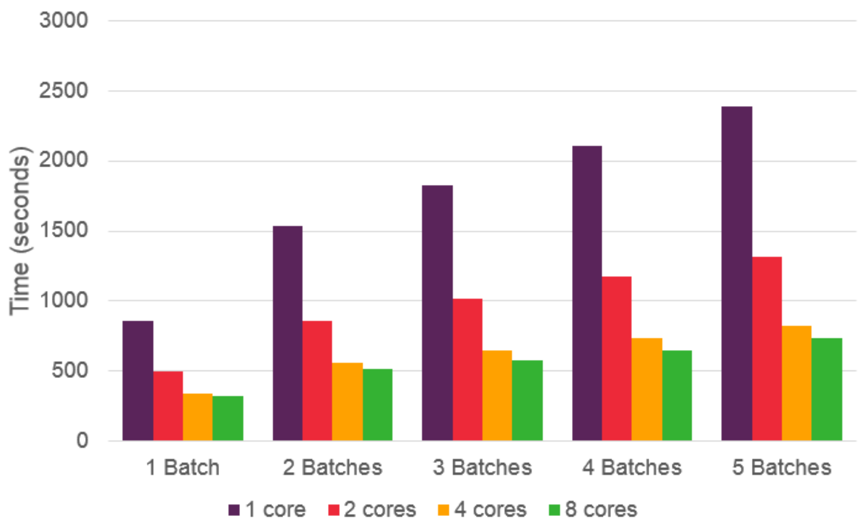


Figure 6.3: Clustering 2 years different cores

6.2.2. MTT-engine

Using the clustering we discussed before we have achieved a significant speed up. In Figure 6.4 the difference between running the engine with and without clusters is shown. The data from Figure 6.4 goes up to 30.000 entries, which is approximately three days of data. From this graph you can see that while the engine without clusters would take around 4 minutes, the engine using clusters takes only 3 seconds.

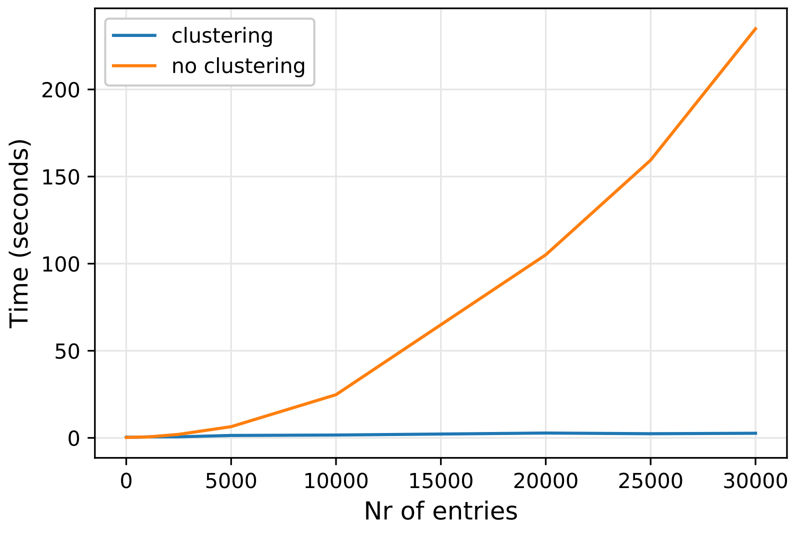


Figure 6.4: Performance of MTT-engine small data size

While there is an obvious difference visible in Figure 6.4, the difference becomes even clearer when looking at larger data sets. An example of which is given in Figure 6.5.

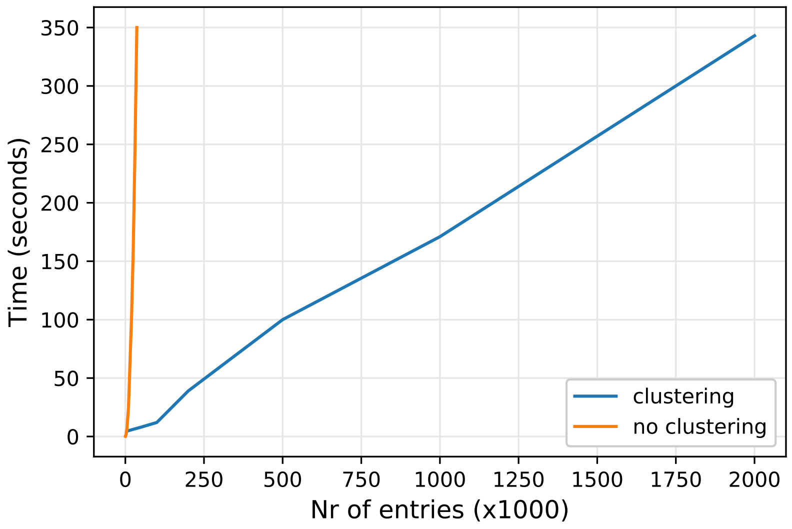


Figure 6.5: Performance of MTT-engine medium data size

While our engine without clusters seems to scale exponentially, our engine with clusters seems to scale in a linear fashion. This is likely because the non-linear functions are executed per cluster, which never exceed the size of 400 and have a mean of approximately 9. Therefore the engine scales linearly with the amount of clusters, which in turn scales linearly with the amount of entries. From this we can also conclude that an increase in production would only slow down the engine linearly, provided that the mean amount of processes on each coil stays roughly the same.

Using our engine on clustered data the MTT containing the data from the past 7 years can be generated in approximately 15 minutes, bringing our total time to construct the MTT from scratch to approximately 40 minutes instead of anywhere from 1 to 6 months. This speedup is achieved due to three important factors, namely:

- **Data size** - Clustering has allowed us to perform the inefficient functions that are necessary for the calculation of the MTT on significantly smaller data sets. Functions such as sorts benefit enormously from the reduced size. However sort functions are by far not the only functions that benefit from the reduced size, as it is also beneficial for simple queries like retrieving outputs or predecessors.

- **Parallelisation** - One of the benefits of clustering is that we can say for certain that two entries that are not in the same cluster will not influence each other. This means that these clusters can be processed in parallel. The amount of speed up this provides is highly dependent on the system that the engine is ran on. The system our benchmarks were ran on had 4 cores, limiting the speed up provided by the parallelisation to a factor of 4.

- **RAM** - The reduced size of each cluster has the added benefit that entire clusters can be kept in memory, which means that all necessary information when processing an entry can be accessed almost immediately. We listed this as an important factor when speeding up the process because transferring data with RAM is usually significantly faster than with a database.

Precisely assigning which factor gives the largest speed up is a difficult task, as we are unable to run the older engine to compare it properly.

To identify a possible optimization we analyzed our engine to find what the optimal cluster size would be. To do this we ran our engine on the same data, but clustered differently. The results of this experiment are shown in Figure 6.6, where the interval of where our cluster sizes lie is marked in light blue.

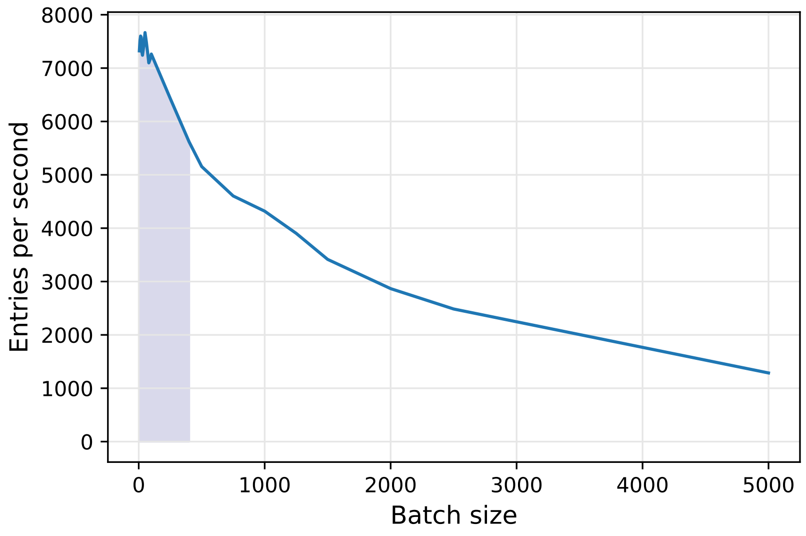


Figure 6.6: Entries per second related to batch size

From Figure 6.6 we can see that our cluster sizes are close to optimal and would have to be reduced further to be faster. However as our clusters are as small as possible, we can not get closer to the optimum.

6.3. Visualisation tool

The goal set for the visualisation tool was essentially to renew the old tool and make it work with the additional data MTT2 and MTT3 provide. This has been done according to the requirements for this task (see Appendix B). Some of the could haves we set for the visualisation tool have been changed due to changing demands and are now leaning more towards won't haves, but nevertheless all the must haves and should haves have been implemented into the tool. In Figure 6.7 the new visualisation tool with a newly created family tree is shown, which is also the new version of the tree shown in Figure 2.4.

In this section, first a general overview of the tool is given in which the requirements we set at the start of the project are discussed. Difficult to envision requirements of the tree are elaborated in greater detail afterwards, as well as parts of the tool that are not visible at first sight.

General overview In Figure 6.7 the menu is shown on the left. This menu is a drop down menu which can be collapsed, when it is not needed anymore, to safe space on screen. Inside of the menu there are options for different search queries, zooming, displaying a grid on the background and printing or exporting the tree.

When a query is sent, a selection screen will open, as shown in Figure 6.8. In this screen the desired coil can be selected and a family tree is made based around this coil. In this newly made tree, the coil used for its creation is now displayed with a thicker border around it.

As you can see in Figure 6.7 coils have lines going in and out of them. These lines indicate the where the parts in the coil have come from and where the parts go to afterwards.

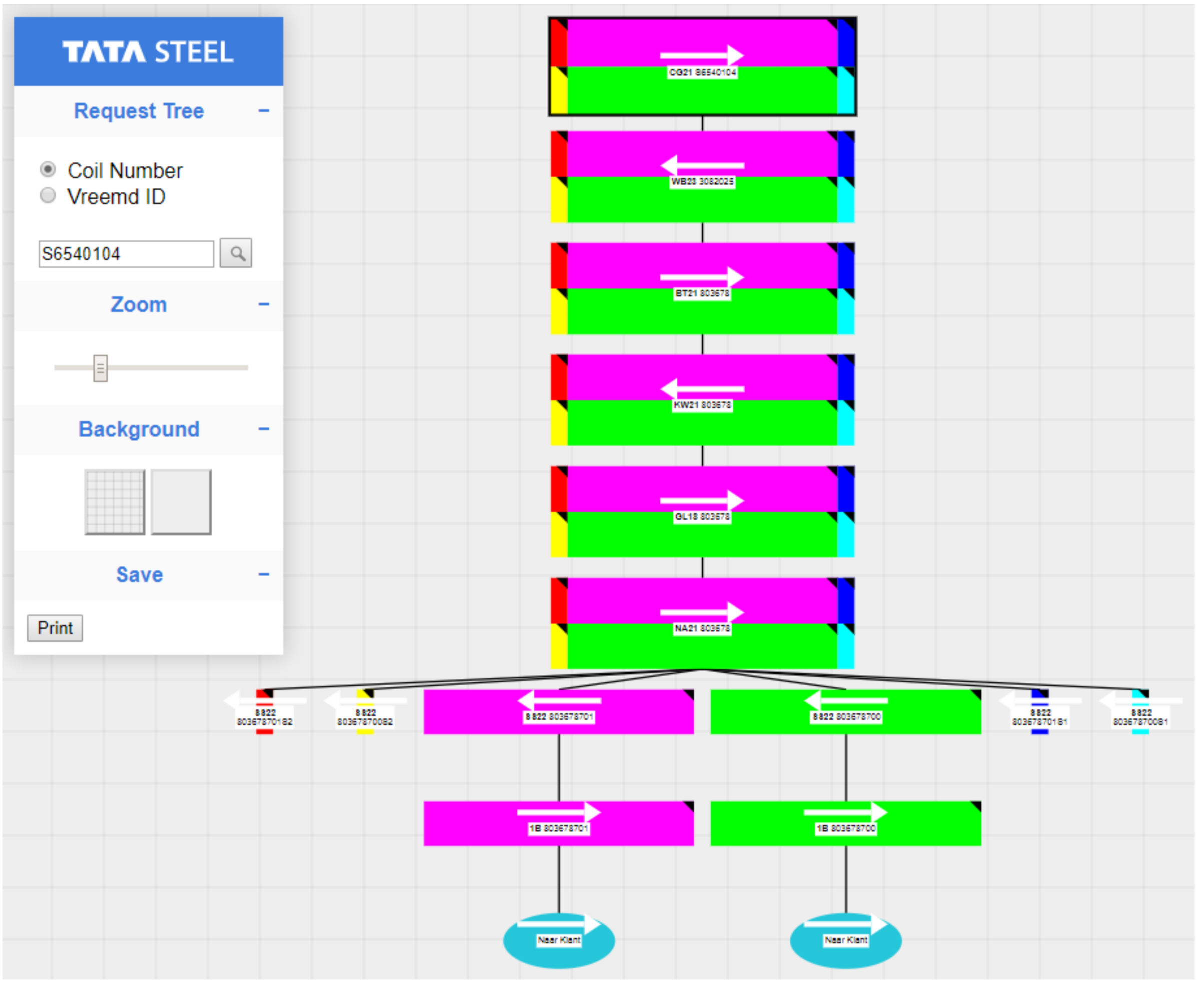
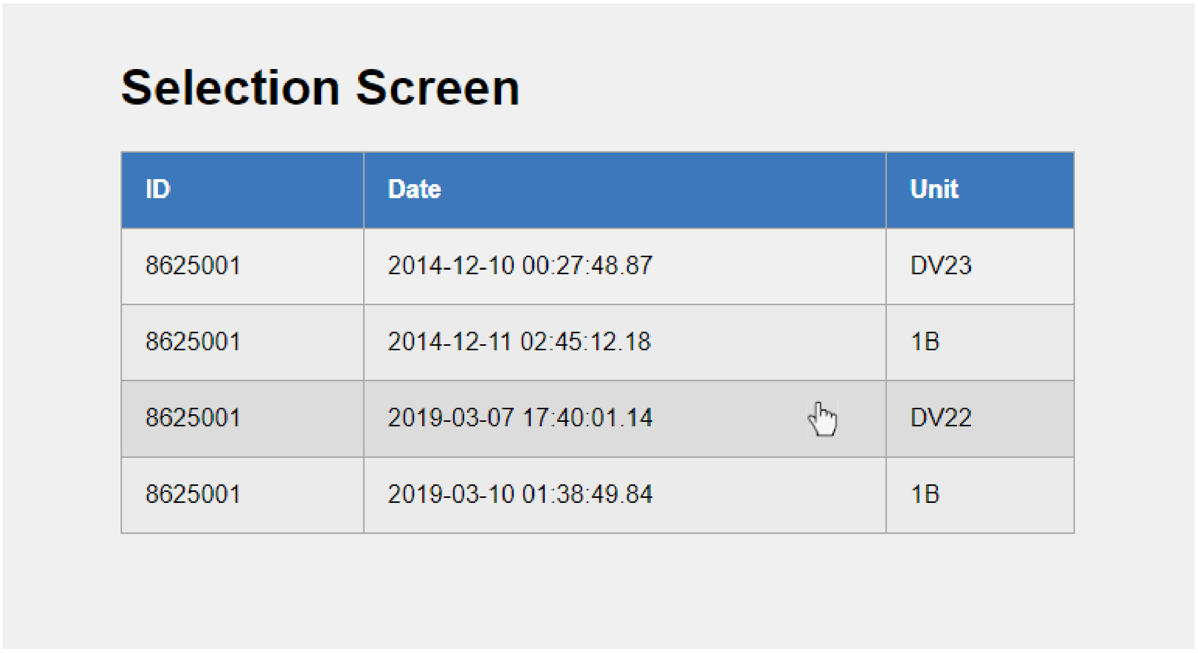


Figure 6.7: Tree in new visualisation tool



Selection Screen

ID	Date	Unit
8625001	2014-12-10 00:27:48.87	DV23
8625001	2014-12-11 02:45:12.18	1B
8625001	2019-03-07 17:40:01.14	DV22
8625001	2019-03-10 01:38:49.84	1B

Figure 6.8: Selection screen in new visualisation tool

In Figure 6.9 a zoomed-in coil is shown. In this coil we also present some specific information about the coil itself. The arrow indicates which side of the coil entered the process first. The code in the middle is made up of two parts. The bold part is the code of the work unit in which the process has been applied to the coil. The other part of the code is the identifying number assigned to a coil. The triangle in the corner of each source piece indicates which side was up with respect to its original position. A black triangle means that the same side as in the first process was up, a white one means it reversed.

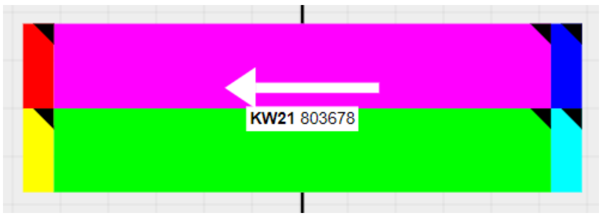


Figure 6.9: Close up of a coil

In the rest of this section the less intuitive requirements are discussed.

Journey between factories The journey of the material is shown in the code in the middle of a coil. The bold part is the one that is important for the journey of the material as this is the number of the work unit in which the process has been applied to that coil. By taking all of these from the top to the bottom of the family tree, you would get the journey of this particular end product.

Inconsistencies in data If a source piece has lost or gained weight during different processes (aside from being cut), this means that something went wrong in this process, as changing the length or width of a source piece should not change its weight. This change in weight is indicated by a small lightning bolt, as shown in Figure 6.10. When the mouse is hovering over this lightning bolt, the percentage in which the weight differs with the previous one is shown.

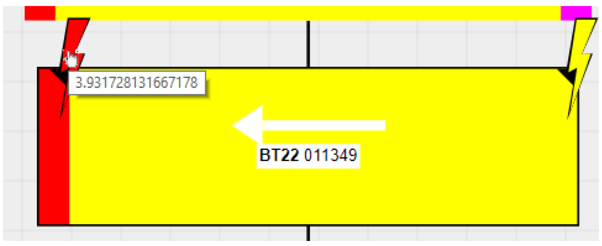
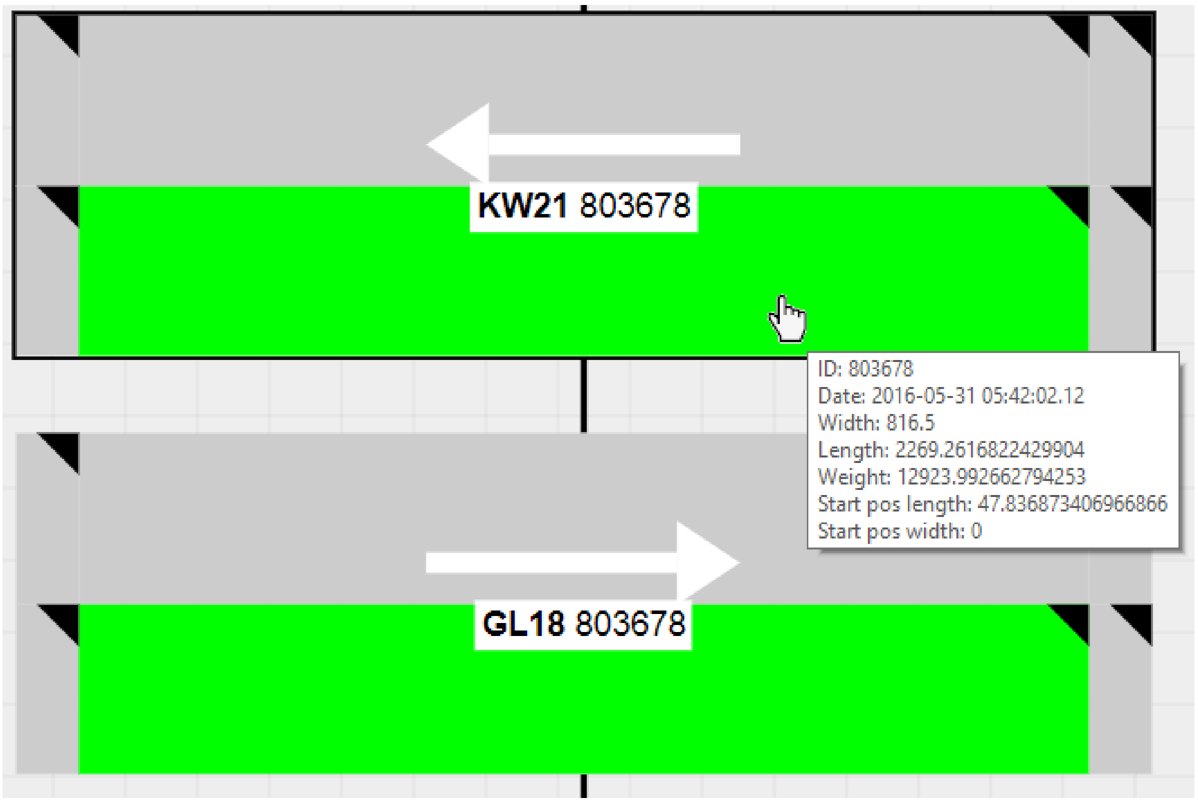


Figure 6.10: Weight change indication symbol

Show meta data of coil The meta data of a coil is not directly shown when the family tree is made. To show meta data of a source piece, a user needs to hover over a source piece and then the information, such as the length and weight of the source piece, is shown on the screen. During this hovering the route ident of the source piece is also selected meaning that in other coils the same route ident is also highlighted indicating the position of the source piece in this coil. An example of this is visible in Figure 6.11



ID: 803678
Date: 2016-05-31 05:42:02.12
Width: 816.5
Length: 2269.261682242990 4
Weight: 12923.992662794253
Start pos length: 47.836873406966866
Start pos width: 0

Figure 6.11: When hovering over a source piece meta data is shown

End destination of Coil When a coil is finished and shipped to a customer, this gets a different icon to indicate it is done instead of the coil which is drawn. In Figure 6.7 this is seen at the bottom of the tree, for the two coils in the middle.

7 Evaluation

In this section we evaluate the way our project process went. Highlighting some of the important considerations to take concerning the project.

7.1. Process

7.1.1. Legacy systems

One thing that slowed us down significantly is the lack of proper documentation. If for instance all work units were properly documented we could, instead of only for two, have made the EE engine for more of them. Since we were only at the company for a short period, spending a large amount of time truly understanding the inner workings of the different steel processes was not feasible. This did, however, give us a good view at how most companies would probably operate in reality. Working with these Legacy systems did provide a new learning experience for us, as in university these things are rarely taught, while in the real world they are always present.

7.1.2. The scope of the project

During the project we had to implement a new MTT. This came down to the three-sub tasks we have discussed in the rest of this report. This did however make the scope of this project very big. From the start our group of 4 needed to be split in 3 teams where each team would focus on one task. This meant that understanding each others tasks and helping each other out was sometimes challenging and also made us change the scope in which each task would be completed. The EE engine for instance became a framework to which it is easy to add new work units, instead of adding all of these work units ourselves. We think that should we have had only one of these sub-tasks that we could have delivered a product that would be fully integrated in their system.

7.1.3. Requirements

As we develop in agile way and in a large company, requirements are subject to change. As people from many different sections of the company have differing opinions on many aspects of our project. Some of these discussions seemed like they would have been better to have done before we arrived at the company, resulting in a clearer goal at the start of the project.

7.1.4. Language switch

During the development of the project the first aim was to make every task in python so that the code base would be comprised of one main language. To do this we first started to implement the visualisation tool in PyQt5. The first 2 weeks were mostly dedicated to learning the structure of the data and thinking of what needed to be in the tool. After this implementation in PyQt5 started, which at first went quite well. However when trying to combine multiple coils to form a tree this language started to work more against than for us as the basic buttons we used to represent source pieces of a coil were hard to order. Also functionality as zooming and dragging which we wanted to add was only possible in specific ways, for which tutorials online were hard to find. We then took a day for ourselves to see how far we could remake the same implementation in Javascript and found out that development using this language was a lot smoother. This led to the decision to switch languages and re-implement the visualisation tool after 2 weeks of implementing in PyQt5.

7.2. Ethics

One of the things asked from us in this project was to consider ethics. As we recently have taken a class in this subject, specifically concerning the ethics related to computer science, we should be able to reason about the ethics concerning this project. As such we feel that this project is not unethical and in general does not concern ethics much. The project as a whole makes the steel making process easier to understand, increasing efficiency. A lot of data is used, but this data only concerns material things, as such this project generally does not seem to be detrimental to any living being either directly or indirectly. However, as the Material Tracking Table allows its users to track flaws to a place and time in production, it also allows someone to track by whom the flaw was introduced. This could be done by matching the information about the time and place of the error to the known work shift schedule. Though the ethical aspect of this is debatable, it is not something introduced by our system specifically, seeing as this could also be done with previous iterations of the MTT.

7.3. SIG

During the project we had to hand our code in through the Software Improvement Group (SIG) twice. The first moment was to set a baseline and give us feedback to improve our code. The second evaluation was used to determine whether we had improved enough compared to the first evaluation. Below we have summarized the feedback we got from both evaluations.

First evaluation The overall score for the first evaluation was 3.9 stars. This meant that our code was maintainable by market standards. SIG identified two main things about the code that could benefit the most from improvement. These were Unit Size and Unit Interfacing. Unit Size looks at methods that are above average big. These were for us the main methods of the MTT-engine, which have to call a lot of other methods. The second improvement point was Unit Interfacing. This looks at the amount of parameters given to a function. This was also triggered by some function in the MTT-engine, because to calculate these entries quite a lot of data is required. Besides Unit Size and Unit Interfacing, SIG also noticed that our code did not have any (unit)tests yet at the time of evaluation.

Second Evaluation The overall score for the second evaluation remained similar to the first evaluation. While we tried to implement the feedback given by SIG, the large increase in code volume also meant that it was harder for us to implement it everywhere in the project. Because of this the maintainability remained similar to the first evaluation.

8 Conclusion

The new Entry/Exit engine framework satisfies its goal to be expandable. It requires the bare minimum knowledge required to add a new work unit. From there it is easily implemented in a more well-versed programming language. This change of language in cooperation with the addition of structural project organization, documentation and testing also has resulted in a more maintainable program, thus achieving its intended goals of expandabilty and maintainability.

The new MTT-engine is able to generate the Material Tracking Table from scratch in approximately 30 minutes instead of anywhere from 1 to 6 months. These 30 minutes include the clustering as well as the engine itself. This speed up is achieved by reducing the data size, adding parallelism and keeping necessary data in memory.

The new visualisation tool is able to visualise data from MTT2 and MTT3. This means that it fixes shortcomings of the old visualisation tool, such as inability to properly display coils that are slit in the length. It also removes some redundant functionality, such as end-of-data-blocks and adds some new functionality to make the tool more intuitive to work with, such as zooming and dragging.

In conclusion, the requirements we set at the start of the project have been fulfilled and we believe that the concepts shown in our report and codebase will be useful for Tata Steel.

9 Recommendations

In this section recommendations for continuation on our work are given.

9.1. Efficiency update

As of now daily updates to the Material Tracking Table are not implemented in the most efficient way possible. When a daily update is initiated, the new data is clustered and all related clusters are retrieved from the EE. These retrieved clusters are then recalculated using the MTT-engine. This process performs unnecessary calculations when a cluster contains no late- or changed entries. This problem could be solved by writing a separate function for late entries and marking changed entries, so that their respective clusters can be recalculated. Since we estimate that the time gained from this optimisation is only a matter of minutes, we decided to focus our attention on other parts of the project.

9.2. Cloud computing

Since the MTT-engine is highly parallelizable, it could benefit greatly from being executed on a large cluster of computers. However, data might have to be saved locally on these computers for them to speed up the process. The process could be ran on a Hadoop cluster, since they are specifically built for storing and analyzing large amounts of data and work well with Spark.

9.3. User testing

The new visualisation tool has been developed without the opportunity to test it all that much with the eventual end users, so user testing is advised to see if further improvements can be made to the tool. Some of these improvements could be making the tool more intuitive or nicer to look at as for now the tool is mostly developed with functionality in mind.

A
Terminology

Coil - While steel is transported it is coiled, therefore they are referred to as coils.

EE - Entry/Exit-table.

Installation - A factory that is able to process the steel.

MTT - Material Tracking Table.

Source piece - A piece of a coil with a unique process history. Looking at Figure 2.1 on the right each color represents a source piece.

Work unit - A group of related installations.

B
Requirements

B.1. Entry/Exit-table

Must haves:

- The Entry/Exit-table must use the same relational schema as the current Entry/Exit-table.
- The Entry/Exit-table must have all required information to build an MTT3 instance from it.
- The Entry/Exit-table must be modular/expandable with ease, to make sure it is possible to add new installations to it.
- The Entry/Exit-table must have the data from at least one facility.
- The Entry/Exit-table must have all entries from different installations follow the same general format.
- The Entry/Exit-table must be made in a more accessible language than the now existing Entry/Exit-table, which was made in COBOL.
- The Entry/Exit-table must be properly documented, so that other can easily use it afterwards.

Should haves:

- The Entry/Exit-table should be constructable in an efficient manner meaning that it should not be the bottleneck of our project.
- The Entry/Exit-table should be compatible with the newer Smart Steel Work format.
- The Entry/Exit-table should have the information of at least two facilities.
- The Entry/Exit-table should be the only table that is required for MTT3.

Could haves:

- The Entry/Exit-table could have the information of more than two facilities.

Won't haves:

- The Entry/Exit-table won't have integration of facilities that do not have their data in RAS or Smart Steel Work.

B.2. MTT3

Must haves:

- The MTT3 must be constructable in 4 hours.
- The MTT3 must be modular/expandable, to make sure it is possible to add new facilities to it.
- The MTT3 must be able to provide an overview of the material to track it, meaning that it fullfils it job of being an MTT.
- The MTT3 must be able to be fully constructed from the Entry/Exit-table.
- The MTT3 must follow the same structure as MTT2.
- The MTT3 must be properly documented, so that other can easily understand and use it afterwards.
- The MTT3 must be written in a well-known and supported language, such as Python.

Should haves:

- The MTT3 should be constructable while at work, meaning that it should finish in 1 hour.

Could haves:

- The MTT3 could be user tested.

Won't haves: None

B.3. Visualisation tool

Must haves:

- The visualisation tool must be able to show the journey between the factories of all source pieces.
- The visualisation tool must be able to show separate coils as separate parts in the visualisation.
- The visualisation tool must be able to show in which installation the coil has been processed.
- The visualisation tool must be able to show/indicate inconsistencies in data, such as weight loss between installations.
- The visualisation tool must be able to show meta data of the coil during the process, such as length, width and ID's.
- The visualisation tool must be able to show the end destination of the coil.
- The visualisation tool must be compatible with MTT2 and MTT3.
- The visualisation tool must be properly documented, so that others can easily use it afterwards.

Should haves:

- The visualisation tool should be able to visualise how the coil has been cut.

Could haves:

- The visualisation tool could be able to track scrap material as well.
- The visualisation tool could be able to visualise weight representation.
- The visualisation tool could remake some of the extra functionality of the previous tool.
- The visualisation tool could be tested for intuitiveness by other workers.

Won't haves: None

C Project Info Sheet

Title of the project: Material Tracking Table for Tata Steel
Name of the client organization: Tata Steel
Date of the final presentation: 04-07-2019
Final Report: `https://repository.tudelft.nl/`

Description:

For a steel company it is advantageous to be able to track products throughout their entire production. At Tata Steel this is currently done with the Material Tracking Table. However, generating this table can take up to 6 months. Therefore we were asked to develop a new faster system, as well as a new visualisation tool to comprehensively show the generated data.

During the research phase we learned about the current system and the requirements of the new system by interviewing multiple Tata Steel employees. This affected our project significantly, as the requirements differed per employee and we had to find a common ground.

Whilst developing we adopted the standard approach used within Tata Steel, which meant we had two meetings each day to discuss the progress of the project, as well as three presentations in total for the stakeholders within the company.

The final product consists of three main components, namely: (1) A modular framework to create a table containing all relevant information from all installations, (2) An engine that is able to create a table containing the production history of each piece of material within an hour, (3) A tool to visualise the previously mentioned table more accurately than the previously used tool. The innovations of our product will be used by Tata Steel to improve their current system.

Team Members

Name: Tom Edixhoven
Contribution and role: full-stack developer, tester

Name: Hunter van Geffen
Contribution and role: full-stack developer, tester

Name: Bas Kruit
Contribution and role: back-end developer, data analyst

Name: Mels Smit
Contribution and role: full-stack developer, Scrum master

Client
Name: Wouter Lamme
Affiliation: Tata Steel

Coach
Name: Maurício Aniche
Affiliation: TU Delft

Contact
Tom Edixhoven, tomedixhoven@gmail.com
Hunter van Geffen, huntervangeffen@gmail.com
Mels Smit, melssmit13@gmail.com
Bas Kruit, swkruit@hotmail.com