Exploring Debate Dynamics: A Data Visualization Tool for Inference Anchoring Theory

Bachelor Thesis of

Martin Gruber

At the Department of Informatics and Mathematics
Chair of Theoretical Computer Science

Reviewers:	Prof. Dr. Ignaz Rutter
Advisors:	Prof. Dr. Ignaz Rutter
	Prof. Dr. Annette Hautli-Janisz
	Zlata Kikteva

Time Period: 1st March 2024 – 27th May 2024

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Passau, September 18, 2024

Abstract

Television debates have a significant impact on shaping public opinion, but the rapid exchange of viewpoints in these forums often makes it difficult to understand the underlying dynamics. While techniques such as Inference Anchoring Theory (IAT) exist to analyze the relationships between statements, existing tools lack the ability to automatically integrate IAT annotated data for comprehensive visualization. In addition, many people get their information from summaries or short video clips, which are naturally biased.

We present a new visualization tool that is designed to address these challenges. Using IAT-annotated data, our tool generates an interactive visualization of debate dynamics in form of a timeline, enabling non-specialists to understand the connections between statements and improve their comprehension. By providing an effective way to explore debates without depending on biased data, our tool contributes to a deeper understanding of IAT annotated debates and the combating of Fake News.

This thesis provides a detailed description of the data used, discusses necessary data transformations and the implementation for an effective interactive visualization tool.

Deutsche Zusammenfassung

Fernsehdebatten haben einen signifikanten Einfluss auf die Formung der öffentlichen Meinung, aber der schnelle Austausch von Standpunkten in diesen Foren macht es oft schwierig, die zugrunde liegenden Dynamiken zu verstehen. Während Techniken wie die Inference Anchoring Theorie (IAT) existieren, um die Beziehungen zwischen Aussagen zu analysieren, fehlt es bestehenden Werkzeugen oft an der Fähigkeit, IAT-annotierte Daten automatisch für eine umfassende Visualisierung zu integrieren. Zudem beziehen viele Menschen ihre Informationen aus Zusammenfassungen oder kurzen Videoausschnitten, die naturgemäß voreingenommen sind.

Wir präsentieren ein neues Visualisierungstool, das entwickelt wurde, um diese Herausforderungen anzugehen. Mit IAT-annotierten Daten generiert unser Tool eine interaktive Visualisierung der Debattendynamik in Form eines Zeitstrahls, was es Nicht-Spezialisten ermöglicht, die Verbindungen zwischen Aussagen zu verstehen und ihre Komprehension zu verbessern. Indem wir einen effektiven Weg bieten, Debatten zu erkunden, ohne von voreingenommenen Daten abhängig zu sein, trägt unser Tool zu einem tieferen Verständnis von IAT-annotierten Debatten und der Bekämpfung von Fake News bei.

Contents

1. Introduction

Television debates, such as "Question Time" in the UK or "Anne Will" in Germany, reach large audiences and are a cornerstone of political discourse. Question Time, for example, had about 1.4 monthly viewers in 2020 [Tea20]. These television debates provide a compact yet rich source of information that often plays a key role in shaping public opinion. However, the rapid exchange of points of view in these forums presents a challenge: the wealth of information compressed into a short period of time can make it difficult to see the underlying dynamics and connections between what is being said.

While techniques exist to analyze the relationships between statements without adding a bias, such as Inference Anchoring Theory (IAT) [Cen17], the insights gained from these analyses often require specialized background knowledge to use. The only existing tool to explicitely display IAT annotated data visually, OWA/OWA+ [JLR14], cannot be used to provide an overview, and it also requires prior knowledge of the extraction process to understand. On the other hand, there are numerous visualization tools for debates and temporal analysis, such as DebateGraph [Ltda], which allows to create topic maps, or Tiki-Toki [Ltdb], which specializes in timelines. However, they cannot use IAT annotated data to automatically create a presentation of both the temporal information and the relationships.

In addition, many people do not have the time to watch full episodes of debates and instead rely on summaries of articles or short video clips. However, such summaries often introduce bias and interpretation into the content, and may even spread misinformation by presenting statements out of context.

In response, this thesis presents a visualisation tool designed to address these challenges. Our tool automatically uses Inference Anchoring Theory annotated debate data to create an interactive visualisation that allows individuals with no expertise to get a quick overview of the debate, or even more deeply explore parts of the debate they are interested in to understand the connections between statements and develop a clearer understanding of the discourse. By allowing a wider audience to explore debates more effectively, our tool aims to contribute to a better understanding of debate shows and the fight against fake news.

This work begins with a detailed description of the data on which our tool operates. We then discuss the necessary data transformations and operational methods required to enable effective visualization. Then we provide a detailed explanation of the implementation strategies of the data transformation. Finally, we describe the visualization and the methods for interacting with the tool.

2. Inference Anchoring Theory and the QT30 Dataset

Before we can discuss the visualization of the data, we must have an understanding of what the data is and how it was created. The data used for the visualization tool comes from QT30 [HJKS+22]. QT30 is both the "largest corpus of analyzed dialogical argumentation ever created" and the "largest corpus of analysed broadcast political debate to date". It is based on the analysis of 30 episodes of the BBC's political debate programme Question Time, produced between 28.05.2020 and 11.11.2021.

The corpus is annotated using Inference Anchoring Theory, a framework for argument mining. It can be found at https://corpora.aifdb.org/qt30.

2.1 Data Annotation via Inference Anchoring Theory

As mentioned above, Inference Anchoring Theory was used to create the QT30 corpus. This section explains its principles and how the application of the IAT works based on [Cen17].

Inference Anchoring Theory (IAT) is a framework used in argument mining and analysis, primarily in the context of dialogue and discourse. It provides a structured approach to understanding how arguments and conflicts are created and responded to in dialogue settings such as debates or discussions. An elementary component are "Argumentative Discourse Units" (ADUs). An ADU is any piece of text that has the following two attributes:

1. Propositional content
2. Discrete argumentative function

In the IAT there is an equivalent concept to an ADU called a locution. The text of each locution is the same as the text of an ADU, except that the speaker is added as a prefix to the locution. For example, the ADU

> "I don't think the british gouvernment did either."

from Helle Thorning-Schmidt would be changed to:

> "Helle Thorning-Schmidt: I don't think the british gouvernment did either" (Figure 2.1).

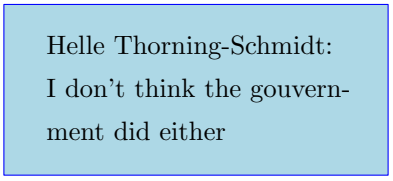

Figure 2.1: Chart of a single locution

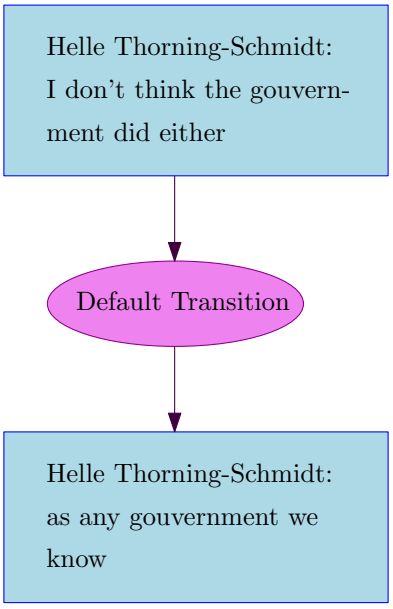

Figure 2.2: Two locutions connected by a transition

There are "transitions" between these locutions that show a functional relationship between them (Figure 2.2).

When a locution has a transition to another locution, the first locution is always chronologically first, so the direction of the transitions is the same as the temporal order. In the majority of cases, there is a transition between subsequent locutions, but some lack one if they are not functionally related. On the other hand, some transitions represent long-distance relations, i.e. they do not connect one locution to the next, but to a more distant locution. Furthermore, a locution can have several transitions, usually between one and two. In IAT maps, transitions are usually referred to as "Default Transitions" without further specification.

Another important part of the IAT are "propositions". Typically, each locution has a proposition, both derived from the same ADU. The difference between a locution and a proposition is that propositions are slightly modified to express the context. For example, the ADU

> "I consider myself part of that public at large"

with the speaker Paul Polman would be changed to

> "Paul Polman considers himself part of that public at large".

The speaker is added to make it clear who is meant by the statement. And

> "society at large has a problem with that"

would be extended to

> "society at large has a problem with the industry spending $3.5 billion to lobby to influence laws"

if " that" refers to lobbying. In this example, the word "that" is replaced by what it refers to. The aim here is to stay as close to the original as possible, while adding as much context as necessary.

Since there is a corresponding proposition to each locution, there is a connection between them. This kind of connection is called an "Illocutionary Connection" (Figure 2.3. These connections always start from a locution and point to a proposition, and they are almost always called "Asserting". Asserting means that a speaker makes a statement about something in order to communicate his opinion about it. In fewer cases, "Questioning", "Challenging", "Popular Conceding", "Disagreeing", or "Agreeing" are used, depending on the goal of a statement. Both "Questioning" and "Challenging" have several more specific versions, such as "Pure Questioning" and "Rhetorical Questioning" or "Assertive Challenging" and "Rhetorical Challenging". If none of the described connection types fit an ADU, "Default Illocuting" is used).

A special case are quotations. When a speaker quotes another speaker or even an absent person, there is not just one locution and one proposition with a connection between them, but two locutions and one proposition. The first locution represents, as usual, what was said. For example,

> "Fiona Bruce: Lawrence says 'if it is safe to go back to school, why isn't it safe for the house of Parliament or are our children test subjects'"

would first have an "Asserting" connection to

> "Lawrence: if it is safe to go back to school, why isn't it safe for the house of Parliament or are our children test subjects",

which also counts as a locution. The second locution is then related to the proposition

> "it is safe to go back to school, why isn't it safe for the house of Parliament or are our children test subjects".

Propositions can also have connections to each other, there are three categories:

1. Rephrase
2. Inference
3. Conflict

These types of connections describe the actual use of an ADU, i.e. how statements interact in the context of the text. For example, if a speaker says something and later refers to it, there would be a "Rephrase" relation between these two propositions. "Rephrase" relations can exist between propositions when a speaker restates, rephrases, or reformulates a statement but does not explicitly repeat a subject. Rephrasing always occurs in the same temporal direction: a proposition rephrases another proposition that was made earlier. This is obvious, since it is not possible to refer to statements that have not yet been made.

Inference Connections, however, are more complex than Rephrase Connections. Whenever a statement supports another statement by providing a reason to accept it, an Inference Connection can be found. There are several types of Inferences:

- Serial Arguments, when each proposition supports the following one
- Convergent Arguments, when different propositions independently support the same proposition

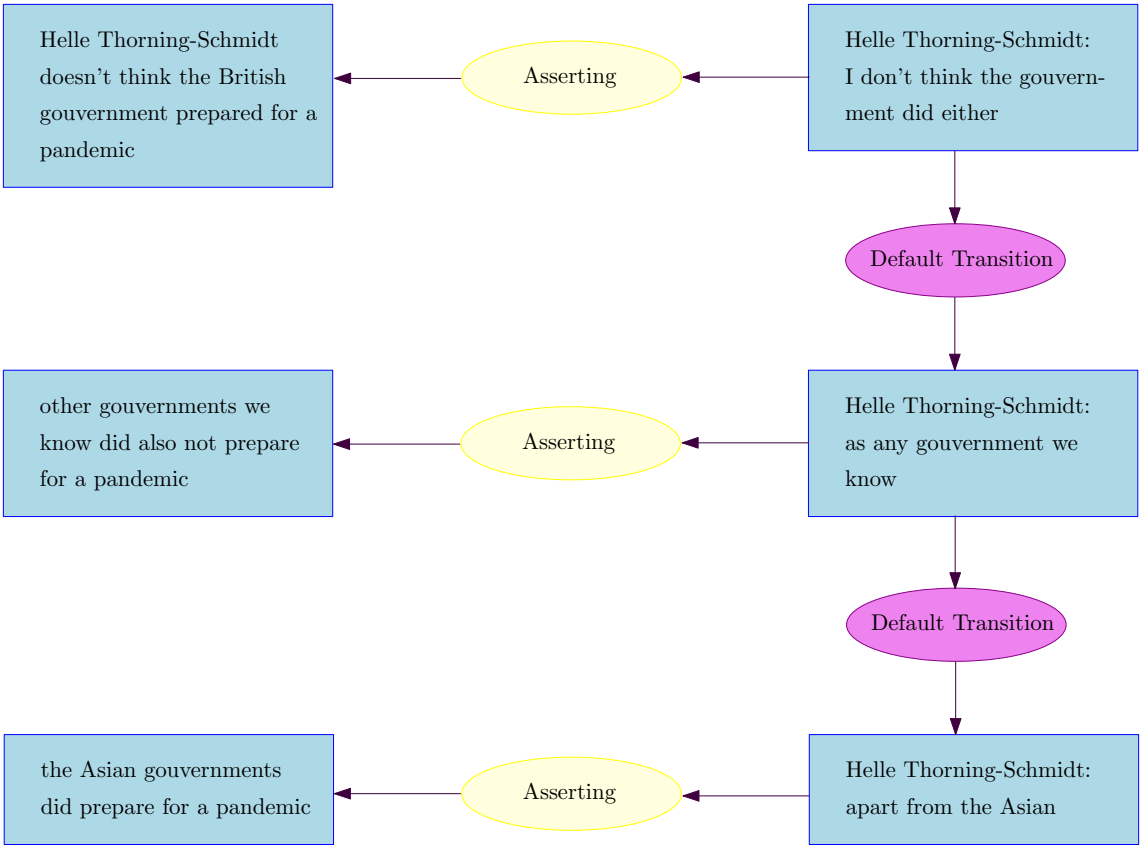

Figure 2.3: Locutions connected to Propositions via Illocuting Connections

- Linked Arguments, when different propositions together support the same proposition
- Divergent Arguments, when a proposition independently supports several other propositions

Unlike Rephrase relations, Inference relations can be pointing to propositions in the past to strengthen arguments already made, or to propositions in the future to create a basis for an argument. Inference relations can have several types, such as "Argument from Expert Opinion". If none of these match, "Default Inference" is used.

The third type of propositional relation is a "Conflict" relation. This type of relation exists between two propositions when one proposition is used to offer a contradictory alternative to another proposition. They come in a variety of forms:

- Rebutting Conflict, when a proposition is targeted
- Undermining Conflict, when a proposition that has an Inference connection to another proposition is targeted
- Undercutting Conflict, when the inference relation between two propositions is targeted

"Conflict" relations can point in both temporal directions, past and future. They also have several types, e.g. "Conflict from Bias" or "Conflict from Propositional Negation". If none of these match the topic, "Default Conflict" is used. (Figure 2.4)

Whenever there is a propositional relation between two propositions, there will always be a corresponding transition between their respective locutions. However, the direction of the transitions is never backwards chronologically, even if the propositional relation refers to something in the past.

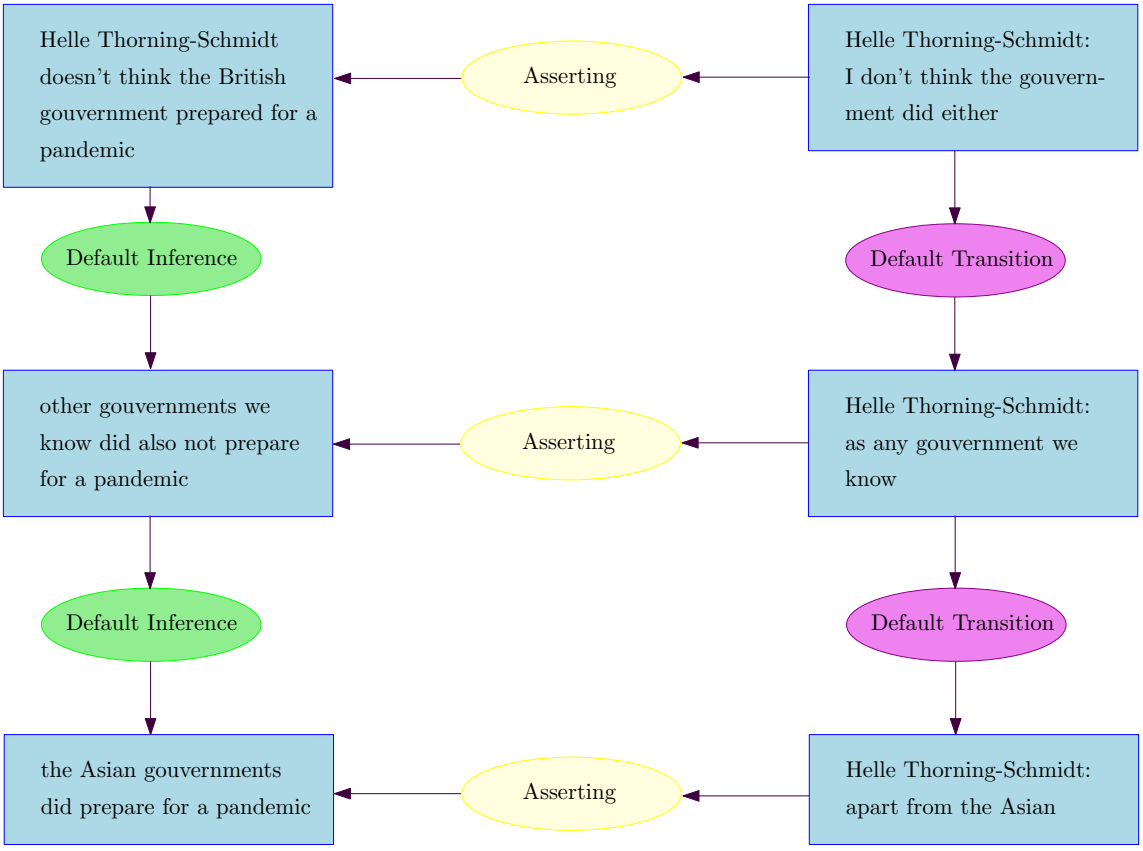

Figure 2.4: Propositions connected via Propositional Connections.

The final type of connection is the one between transitions and propositional relations. These connections are also called Illocutionary connections. They describe the type of propositional relation (Figure 2.5). In the IAT map, a "Disagreeing" connection points toward a conflict relation, an "Arguing" connection indicates an inference relation, and a "Restating" connection points toward a rephrasing relation.

2.2 Data Format Description

In order to discuss the extraction and use of the data in the context of the visualization tool, we must first have a look at the overall format of the data.

2.2.1 Structure of the QT30 Corpus Data

In QT30, IAT was applied to the transcripts of "Question Time" episodes. The text of a transcript contains about 10,000 words and is divided into 40-80 excerpts of 150-250 words each. These are further broken down into ADUs, resulting in 5-20 ADUs/locutions per excerpt. Then, for each locution, the corresponding proposition is generated, and the illocutionary and propositional connections, as well as the transitions between them, are added. Finally, the relations between locutions within different excerpts are identified and collected in separate files.

The full QT30 dataset is available to the public at `https://corpora.aifdb.org/qt30`. The dataset comes in the form of a large number of JSON files (about 1.5k). It contains the analyzed data for 30 episodes, so there are about 40-80 files per episode. Each of these files represents one of the excerpts. There are three different kinds of elements in each file:

1. Nodes

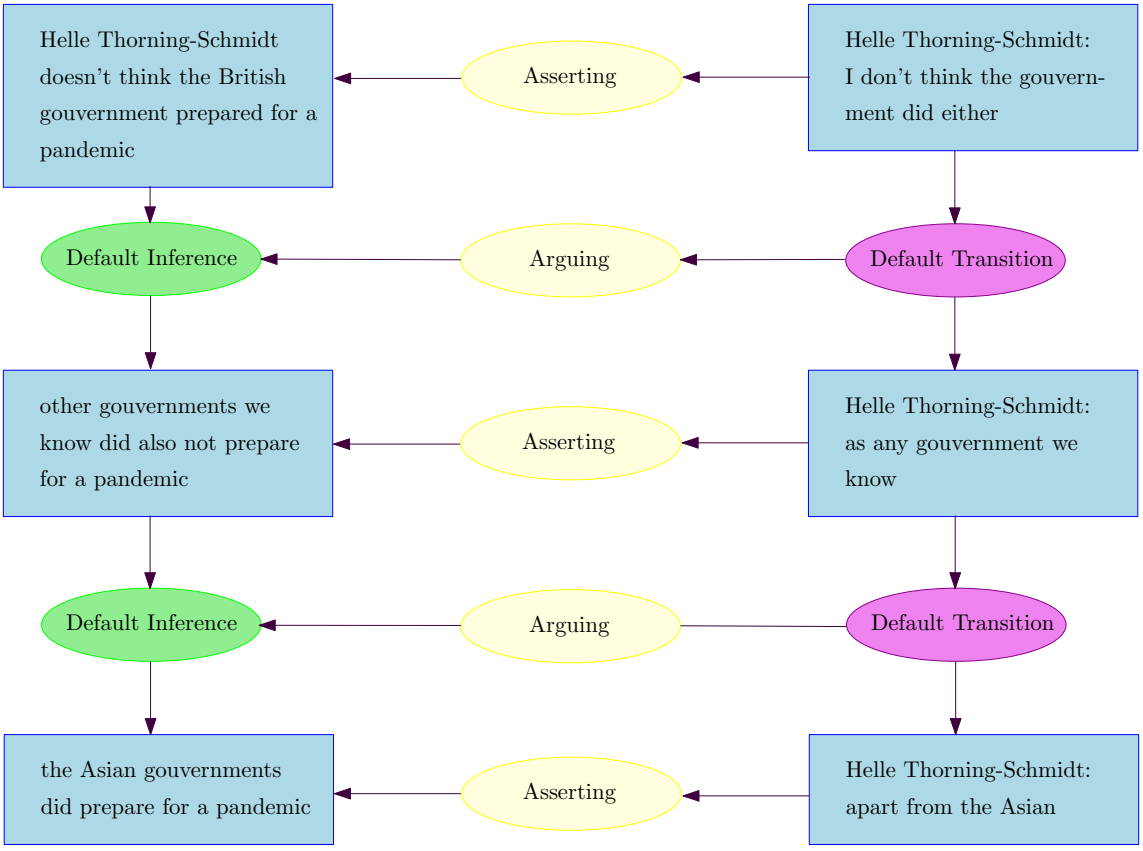

Figure 2.5: Full IAT scheme

2. Locutions
3. Edges

There are different types of nodes, and all of them have the following four attributes: a unique id ("nodeID"), the text contained by the node ("text"), the node type ("type"), and a timestamp ("timestamp"). The timestamp has nothing to do with when a particular statement was made in a debate, but with when a researcher created the node. In the context of this visualization tool, therefore, it is not valuable information.

There are seven types of nodes:

- Nodes of type "L
- Nodes of type "I"
- Nodes of type "TA"
- Nodes of type "RA"
- Nodes of type "MA"
- Nodes of type "CA"
- Nodes of type "YA"

Nodes of type "L" contain the locutions, i.e. they contain the original text from the transcript with the speaker's name prepended, e.g.:

"Tim Stanley: There is a third reason we did not mention"

In each file there are 5-20 "L" type nodes that follow this scheme. However, there are also "L" type nodes that follow a different scheme: In the data of later episodes, for each "L" type node of the

"speaker: text"

scheme, there is another "L" type node of the

"researcher: speaker: text"

scheme, indicating the name of the person who analyzed the particular statement but this is inconsistent in the data set. No additional information about the debate is contained in these nodes. In the following, the term "L" type node" or "L" node" will refer to the nodes of the "speaker: text" scheme, while "not usable "L" type node will refer to the nodes of the "researcher: speaker: text" scheme.

Nodes of type "I" represent propositions, i.e. there is generally an "L" node from which they are derived. In the example above this means that for the node of type "L"

"Tim Stanley : There is a third reason which we have not mentioned"

there is a node of type "I" with the text

"There is a third reason which we have not mentioned".

However, there are a small number of "L" nodes that do not refer to an "I" node, and since "I" nodes cannot exist without "L" nodes, there are no "I" nodes in these cases. This usually occurs if an "L" node contains only a short text such as "No". Consequently, the relation between locutions and propositions is not bijective.

The remaining node types have two additional attributes: First, a "scheme" attribute, which is never different from the "text" attribute. So this attribute does not add any new information. Second, there is a scheme ID ("schemeID") reflecting the "scheme" attribute by assigning a specific number to each different scheme. Again, this is information that cannot be used in the context of the visualization of the data set.

Nodes of type "TA" represent Transitions between locutions. They contain "Default Transition" as text.

Nodes of type "RA" represent Inference relations. They contain "Default Transition" as text.

Nodes of type "MA" represent Rephrase relations. They contain "Default Rephrase" as text.

Nodes of type "CA" represent Conflict relations. They contain "Default Conflict" as text.

Nodes of type "YA" can have different shapes. They describe Illocutionary relations. Thus, the text can be "Asserting", "Default Illocuting", "Questioning", etc., as well as "Arguing", "Restating", "Disagreeing",etc. Yet there is a type of "YA" node in the data set that is not part of the standard IAT maps. These nodes have "Analyzing" as text and connect the unusable "L" nodes with their corresponding "L" nodes via edges. Whether these "analyzing" nodes exist depends on the file, though the data for later episodes are more likely to contain them.

The "edges" in the files represent the edges in the IAT map and are of a generic nature. Each individual edge has a unique ID called "edgeID", the nodeID of the source node as "fromID", and the nodeID of the destination node as "toID". They also have an attribute called "formEdgeID", which is always null.

The term "locution" in the data interferes with the term in the IAT because the locutions in the data are not meant to be equivalent to ADUs, as locutions in the IAT are. As described above, the true locutions are represented by particular nodes of the type "L". For this reason, the "locution" elements from the data are referred to as "locutions" or "locution elements" are enclosed in quotation marks. The actual locutions are referred to without quotation marks. However, the locutions in the data belong to these nodes of type "L", since they even share the same value for the "nodeID" attribute. "Locutions" have the following attributes:

A "nodeID" attribute, just like nodes. Each locution's ID can also be found in an "L" node's nodeID. Although this should be a bijective relation, with each individual L node having exactly one "locution", the "locutions" are sometimes missing or multiple "locutions" exist to one node. "Locutions" also have a personID attribute, which contains the ID assigned to a speaker. The attribute "timestamp", as with nodes, does not represent the actual time at which an utterance was made and is therefore not useful information. The end and source attributes are always null. The "start" attribute, on the other hand, represents the exact time at which a statement was made. Unfortunately, in almost every episode except the first few, the "start" attribute is always null for every "locution". And even in those episodes, the start value is not consistent.

In Addition to the "normal" files there is also one "Inter Map Connection" file per debate in the dataset. Inter Map Connections (IMCs) are the connections between different files. Although file chunking is based on things like topic breaks, there are some connections that cannot be put in one file or another. Also there are some long-distance links. The contents of the IMC files are very similar to the contents of regular files, they also have the same type of nodes, edges, and locutions, except that the "L" nodes in IMC files are "copies" of the "L" nodes that have connections to outside the file they are in.

2.2.2 Additional Data

In addition to the QT30 corpus there are a transcript and a video of the episode that is visualized, and a file with a mapping of speaker names to personIDs.

Transcript

The transcript provided, in the form of a .txt file, is not part of the QT30 data that is publicly available. It was used by the QT30 researchers in applying the IAT and provided as additional data. The transcript text is divided into several parts, with each part corresponding to an excerpt. Thus, a transcript typically consists of 40-80 parts. Each part matches a file in the analysed dataset. A part typically contains 2-4 speaker names, each followed by a timestamp placed before several sentences spoken by the person. A new time stamp is added when there is a change of speaker or topic, or when a person is speaking for an extended period of time. Generally, therefore, there will be no more than 30 seconds between timestamps and often no more than a few seconds. Here is an example from the beginning of a section:

> Part 24
> Words: 206
> Fiona Bruce
> [0 : 41 : 02] Let's hear from some of these people
> AudienceMember 20211111QT14
> [0 : 41 : 06] Personal responsibility is important. Where's the emphasis being placed on corporate responsibility who create the most emissions"

It is notable that audience members are not referred to by name, but by an assigned number.

Video

The visualization tool uses a video of the QT30 episode from 04.11.2021 in addition to the transcript and the QT30 dataset. The video is used to enhance the visualization with an audiovisual dimension. It is publicly available and can be downloaded for free from the BBC iPlayer site: `https://www.bbc.co.uk/iplayer/episodes/b006t1q9/question-time`. (Older episodes may no longer be available)

PersonIDMapping

As described above, the locution elements within files contain a speaker ID, instead of a speakers name. To get the speaker name, a .txt file containing the mapping is used. This file is not publicly available.

3. Data Modeling

This chapter evaluates what information should be displayed and what information is not accessible without operations on the data.

3.1 Objective

The overall idea is to create a timeline that displays the following things:

1. When a speaker says something
2. Who the speaker is
3. The context of a statement in the debate (using information from the IAT)
4. The context of a statement in the text (what was said before, what was said after)
5. Which topics can be found throughout the text
6. A video of a debate for additional audio-visual context

3.2 Required Data Transformation and Refinement

Most of these goals cannot be represented using only the data provided by the dataset, but require additional data and refinements. The first goal even requires two separate tasks, first to find a chronological order, and second to annotate it with timestamps. This section elaborates on the prerequisites and discusses the necessary operations, including special cases that need to be covered.

3.2.1 Chronological Order

To satisfy 1, it is essential to determine the chronological order. But even if there are clues to the temporal order in the data, such as transitions always having the same direction as the temporal order and the order of nodes within a file, it may not be possible to recover the correct order by relying on them. There are not necessarily transitions between two consecutive statements, and some statements have more than one outgoing or incoming transition, resulting in numerous possible orders. And there is no guarantee of a chronological order within the files, even if the order of the nodes in the files follows the chronological order in most cases. A combination of the two clues might produce a

chronological order, or an order that is close to chronological, but that order would still lack timestamps, and guessing a timestamp for each and every node might not be very precise, given that a debate lasts about an hour. The most promising option is to use the transcript, since it obviously contains all statements in chronological order and regular timestamps.

Prerequisites

To locate statements in the transcript, it is necessary to break the transcript text into the smallest possible chunks without risking errors. If the chunks are too small, e.g. if the sentence

> "We have empty museums, empty theaters, empty sports facilities."

were split by commas into

> ["We have empty museums", "empty theaters", "empty sports facilities."]

the locution

> "We have empty museums, empty theaters, empty sports facilities"

could not be found in any of them. Since locutions never extend a sentence, it makes sense to split the transcript into sentences. If a sentence already contains propositional content, it is an ADU by itself, requiring no other sentence. If it does not satisfy the criteria, it is not part of a locution and is omitted. Sentences are therefore the smallest chunk size that can be used without the risk of splitting ADUs. Each individual locution can then be mapped to the transcript by checking whether the locution is a substring of the sentence and assigning a fairly precise location in the transcript. Either way, the transcript must be extracted before a chronological order can be found.

Requirements for the Algorithm

When designing a string matching algorithm for this particular problem, there are several cases to consider: First, there may be sentences in the transcript where no locution matches. This happens when the sentence has no propositional content or argumentative function, and was left out after the IAT was applied. In most cases, however, there is exactly one locution that matches a sentence. For example, the locution

> "We have empty museums, empty theaters, empty sports facilities"

would fit into the sentence

> "We have empty museums, empty theaters, empty sports facilities."

(which is effectively the same sentence, in spite of the beginning and the end of the sentence). In contrast to this example, some sentences contain more than one ADU and are therefore represented by more than one locution. One example is the sentence

> "So why was a decision taken then and how has it worked.".

It contains two locutions, the first is "so why was the decision taken then" and the second is "how has it worked". Both are substrings of the phrase and would be mapped to it. In this case, they are disjoint, meaning that they can be uniquely arranged by another factor within the sentence. Unfortunately, there are cases where a statement matches not only the part of the text from which it is derived, but also another part of the text. For example, the phrase "No" should match the sentence "No". But actually "No" can also be found in the sentence

> "So, the NHS in England and Wales and Scotland and **No**rthern Ireland have put together a delivery plan.".

So there are two possible positions for the locution "No" and it is necessary to choose one. If the wrong one is selected, the correct locution "the NHS in England and Wales and Scotland and Northern Ireland have put together a delivery plan", which is derived from the sentence, would overlap with the locution "No" and it would not be possible to determine a clear order. To avoid this, or at least to minimize the risk of such mismatches, a matching algorithm would have to take measurements against it, otherwise some of the matches will depend only on the order in which the locutions are matched.

The last case is a statement that does not match any sentence in the transcript. There are several possible reasons for this, first of all, of course, errors in the data. Sometimes the transcript contains typos which are the reason why statements do not map. A possible solution might be to allow a certain amount of imprecision, depending on the length of a sentence. Another potential cause is an imperfect regular expression for splitting the transcript. If it splits on every period, question mark, and exclamation mark, statements like 0.5% can be accidentally split in half.

In summary, this string matching problem requires:

- A previously extracted transcript

An algorithm for this problem needs to be able to deal with the following:

- Sentences in the transcript that are not mapped to
- 1:1 mappings
- 1:n mappings including sorting of the n locutions
- Mismatches and the problems they cause
- Unmapped statements

3.2.2 Timestamp Annotation

Timestamp annotation is the second step to fulfill 1 in 3.1.

Prerequisites

The transcript is the only consistent source for timestamps and contains timestamps approximately every 5-30 seconds. As a result, transcript extraction is necessary not only for ordering, but also for timestamp annotation. But the text of a timestamp usually contains several ADUs. Thus, multiple locutions would get the same timestamp. To overcome this problem, the start time of locutions that share their transcript timestamp must be assumed/calculated. However, this requires the data to be in chronological order.

Requirements for the Algorithm

When multiple locutions have the same timestamp but are ordered chronologically, timestamps can be assigned depending on the position of the locutions within this timestamped range and the timestamp of the next timestamped range. For example, if there are the following two timestamped areas

> [00 : 01 : 03] I think, therefore I am
>
> [00 : 01 : 15] Moreover, I think Carthage has to be destroyed.

"I think, therefore I am" contains two ADUs, "I think" and "I am". The start of the next statement is at [00:01:15], so the two locutions "I think" and "I am" were said within twelve seconds. An algorithm would have to identify the locutions of a timestamped area and distribute the given amount of time to them reasonably.

In summary, a timestamp annotation algorithm requires:

- A previously extracted transcript
- Chronologically ordered input data

Also, a simplified version of the graph as described in 3.2.4 might be helpful in the overall context.

An algorithm for this problem must be able to handle the following:

- Assigning each locution a start time and an end time according to its position in the order.

3.2.3 Speaker Identification

To fulfill 2 in 3.1, the person IDs must be mapped to the speaker names. This is best done when the files are first extracted and the "locution" element is associated with its corresponding "L" node. To do this, the speaker mapping file itself must be extracted first. The speaker file has the following format

personID_1 name_1
personID_2 name_2

There are no prerequisites for this process, and there are no special cases that need to be handled.

3.2.4 Graph Conversion

The IAT maps have two dimensions, there are the relations between "I" and "L" nodes, and the relations between "L" nodes and "L" nodes or "I" nodes and "I" nodes. As mentioned earlier, the IAT maps must be displayed as a timeline. The two-dimensional nature of these maps is not ideal, and not all of the data contained is necessary for this purpose. For this reason, a simplified version of the IAT maps has been developed for visualization purposes. The graph conversion is intended to make 3 in 3.1 possible by not only extracting the data for it, but also making the data more manageable.

Prerequisites

Figure 2.5 is an example of an IAT map. The main idea is to merge corresponding "I" and "L" nodes and to remove unnecessary nodes and edges, eventually replacing them with edges that also carry text. "I" and "L" nodes can be merged because there is supposed to be a one-to-one relationship between them. Although this is not always true, since a small number of "L" nodes do not have a corresponding "I" node, it works for the vast majority. The merged node contains the text of both nodes, so the only thing lost is the link between them, i.e. the "YA" connections like "Asserting", "Agreeing" or "Pure Questioning". The majority of these are "Asserting" links, a piece of information that may not be very helpful to people who are not familiar with the IAT. Also, the other links, such as "Agreeing" or "Pure Questioning," may be also clear when looking at the text and do not need to be retained or even displayed. These items were excluded in consultation with researchers who worked on QT30. As a special case, quotations have an additional "L" node in the

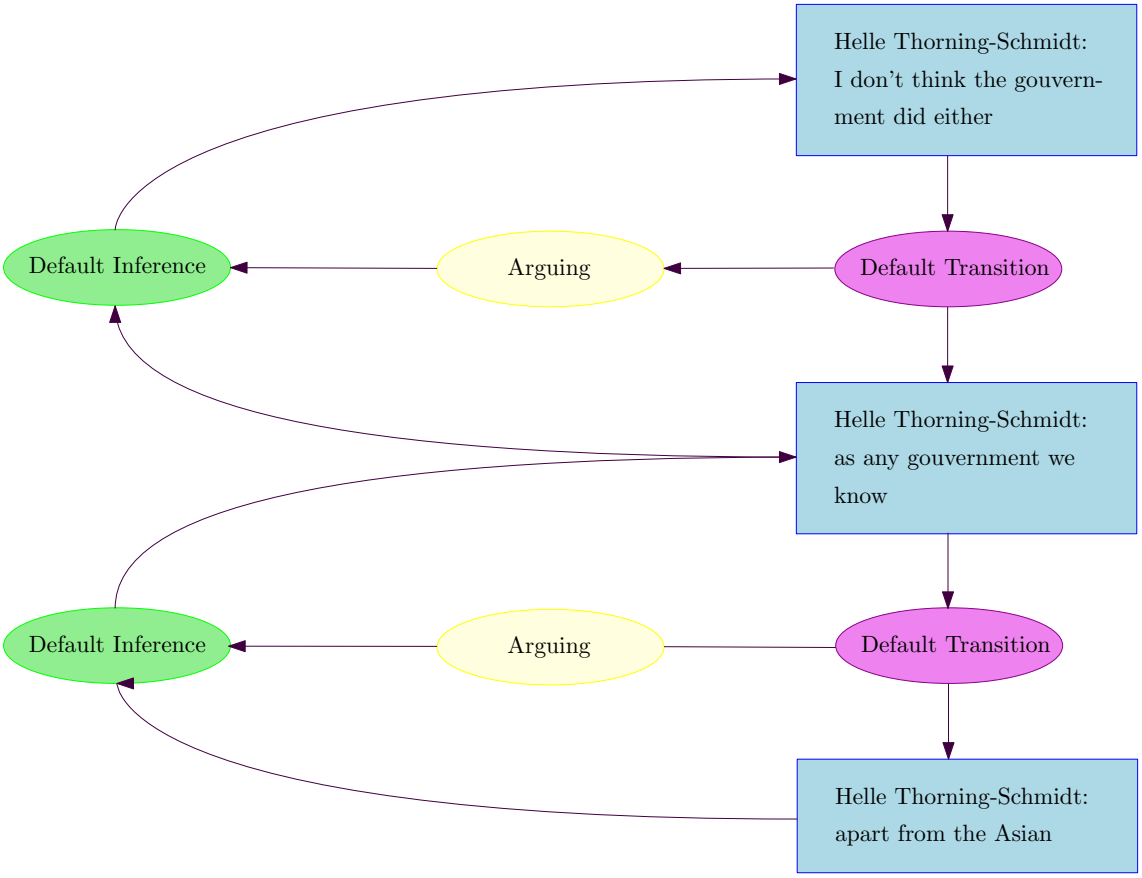

Figure 3.1: IAT map with collapsed "I" and "L" nodes

data. This additional node can be merged with the other nodes, and its text can be used as an additional text to the node. Since these nodes have no edges other than the incoming connection from an "L" node and the outgoing connection to an "I" node, this can be done without risk of losing information. The IAT map with collapsed maps looks like this (Figure 3.1):

Transitions (type "MA"; between "L" nodes) can also be removed. In most cases there is a transition between one locution and the next. These transitions do not carry any essential information, since the simple use of a timeline already reflects the order of the locutions. Another kind of locutions are those that "mirror" the Inference, Rephrase, or Conflict connections. They can be found between any two locutions that have such connections, and could be unambiguously recovered from the simplified map just by looking at the respective propositional connections. The only transitions that might contribute valuable information are those that are missing. Some locutions do not have a Transition because no functional relationship exists between them. It might be valuable information to highlight the positions where a locution does not refer to or has nothing to do with the previous one. However, in consultation with the researchers who worked on QT30, it was decided not to pursue the analysis of missing transitions in the visualization. Hence, the transitions can be removed (Figure 3.2).

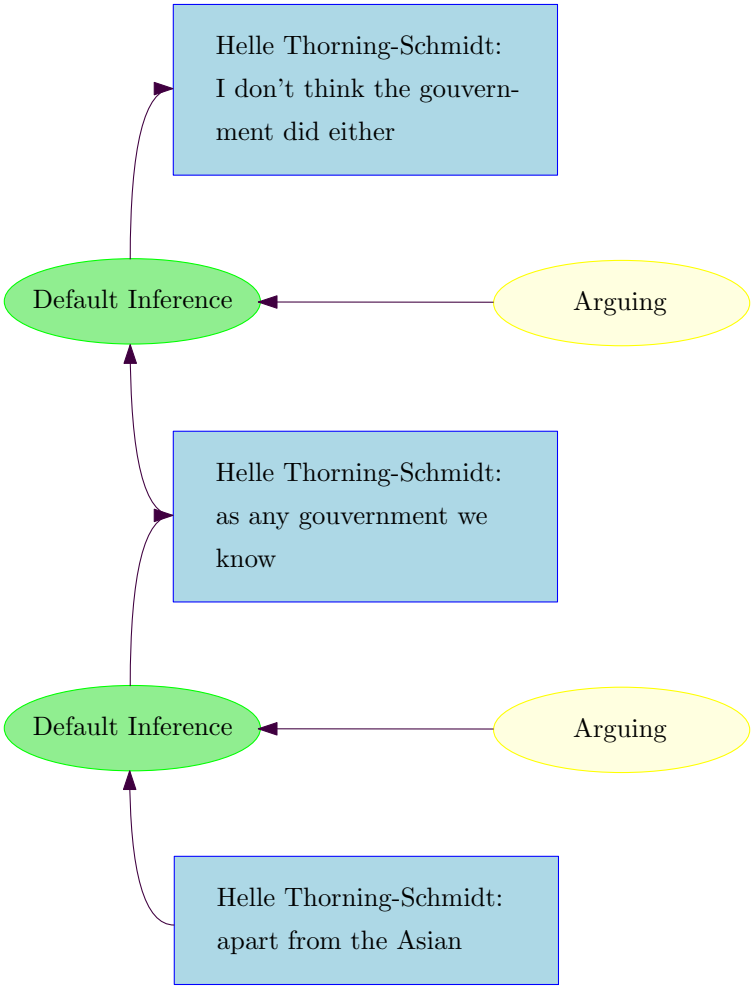

Figure 3.2: IAT map with collapsed nodes and removed transitions.

Now the graph is in a one-dimensional form and reduced to information that is valuable for visualization in the form of a timeline. Another step allows us to simplify further without losing information: The propositional connections (Inference, Rephrase, Conflict) are each represented by two edges and a node. By replacing this with a single edge carrying the text as additional information, the graph can be simplified to have only **one** node type. The text of the "YA" nodes that have an edge to one of the connecting nodes can also be added to the edges as additional information. Again, this does not change or remove any information, as the original could be unambiguously restored by simply recreating the nodes and replacing the single edge with multiple edges. The result looks like this (Figure 3.3):

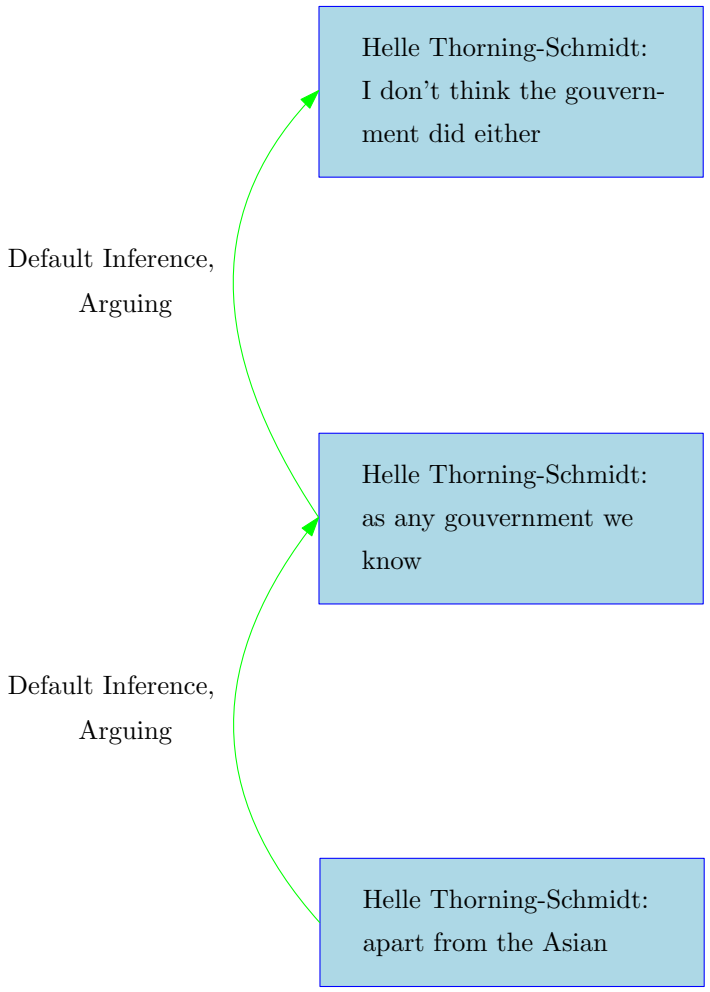

Figure 3.3: IAT map with collapsed nodes, removed transitions, and simplified edges.

This converted version contains nearly all the information of the original graph and, even more importantly, it contains all the information necessary for the visualization in an easier-to-handle form.

Requirements for the Algorithm

Since real data always contains errors and problems, an algorithm for this conversion would have to deal not only with performing the conversion, but also with the errors in the data, and the data contains several errors: Sometimes there are illegal edges, such as backward edges. In the QT30 implementation of the IAT, there should be no bidirectional edges. However, in rare cases they can be found. A conversion algorithm would have to take care of this, maybe by ignoring them where it finds them. There are also some inconsistencies in the data, one of them being the presence of unusable "L" nodes. They are present in some files/debates and need to be filtered out, but they are not present in other files/debates. The same is true for "Analyzing" connections. If there are non-usable "L" nodes in a file, they are sometimes, but not necessarily, connected to usable "L" nodes via "Analyzing" connections. Another inconsistency, of course, is the presence of a "start" value. However, by calculating the start time based on the order in the transcript and the timestamps in the transcript, the start values can (and should) be ignored. Furthermore, the speaker may be missing, i.e. the personID value is null. And there can also be nodes with a missing "type" attribute. Even worse, in very rare cases IDs can be wrong or edges can be missing. The consequence of both of these is that an "I" node and an "L" node cannot be mapped.

Another problem occurs when the same statement is repeated in the same form by the same speaker, such as "The peak follows the lockdown" in

> "The peak follows the lockdown. Lockdown doesn't coincide with the peak. The peak follows the lockdown."

What happens is that these do not end up as separate nodes with separate IDs, but result in **the same node** connected to both "I" nodes. As a result, it is not possible to determine with any certainty which connections and transitions belong to which locution in the IAT diagram. Another error in the data is the number of "locution" elements in a file that can be associated with an "L" node. The correct number, of course, should be one for the usable "L" nodes. The intermediate quotation nodes and the non-usable "L" nodes should not and do not have locutions associated with them. Therefore, the presence of a locution is theoretically a clear indicator for usable "L" nodes. But in some rare cases the locution is missing completely. But the opposite can also occur: for example, the "L" node "Fiona Bruce: The peak follows the lockdown" has a total of 8 (!) "locution" elements sharing its nodeID.

Although the total number of errors in the data is small, all of them could probably cause problems if the corresponding data is not excluded or handled in special cases. Since the conversion of the graph relies entirely on the underlying data, these potential problems must be kept in mind when designing a conversion algorithm.

So in summary, a conversion algorithm requires

- An extracted speaker file

An algorithm for this problem must be able to handle the following

- False bidirectional edges
- Inconsistencies in the presence of unusable "L" nodes and "Analyzing" connections
- Different types of "L" nodes that need to be handled differently
- Missing speaker IDs or missing "locution" elements
- Single "L" nodes connected to more than one preposition
- Multiple "locution" elements for one "L" node

3.2.5 Transcript Distribution

The goal 4 in 3.1 is not to display the statements in isolation, but to allow a person using the visualization tool to see the textual context, i.e. the words/sentences before and after a statement. Since each node represents a locution, which can be a whole sentence or only a part of it without punctuation, and since some (partial) sentences do not end in a locution, concatenating the locutions would not produce an easily readable text. The only way to display the whole text without leaving gaps is to use the transcript again. Also, a chronological order of the data is necessary to produce unambiguous results.

Prerequisites

There are basically two ways to use the transcript for this purpose: One method just displays the transcript and if one needs to highlight something in the transcript, one has to search for the specific part in the frontend. Of course, the transcript could not be used in its original form, the part numbers, timestamps, speaker names, word counts, etc. would be removed first. This option is quite simple as it does not require much data refinement, but it does not solve the problems, it just moves them to the front end. For example, if

there is a question like "Has Tory sleaze returned?" the question mark is not part of the phrase, and if this question is highlighted, the frontend would be responsible for finding a way to associate the question mark with the text. But even more important is how the text is found in the frontend. If the transcript is used directly in the frontend, the string matching problem of 3.2.1 has to be solved again, because the locutions have to be found in the transcript again. Alternatively, all the positional data used to create the order could be transported to the frontend. The other option is to split the transcript text between the locutions, i.e. each locution gets a part of the transcript as additional data, so that the transcript could be restored by reassembling the additional data of the locutions. As a consequence, in the frontend each locution can be mapped 1:1 to a sentence or a part in the transcript, which reduces the complexity for the frontend and reduces the probability of mis-mappings. Although both options are valid, the second option seems to be the better one which requires the transcript to be extracted before that. Also, the data has to be already chronologically ordered, as not mapped transcript sentences could otherwise not be assigned consistently to the same locution.

Requirements for the Algorithm

An important requirement, of course, is completeness. If the transcript is divided into parts, the sum of the parts should be the transcript. In particular, those parts of the transcript where for some reason no matching locution could be found should not be omitted. Consequently, sentences and parts of sentences that are not mapped must be added to another locution as an additional text part. Since there are not only sentences that are not mapped at all, but also sentences that are mapped more than once, we have to decide how to divide these sentences. There are two possible cases for these sentences: Either one locution ends exactly where the other begins, or, as in "why was a decision made then and how did it work", there is a word or another text span between the locutions "why was a decision taken then" and "how has it worked", in this case the word "and".

It is also important to note who is speaking. Of course, it makes little sense to take an unmapped sentence from one speaker and add it to the locution of another speaker speaking after or before him.

In summary, a transcript distribution algorithm requires:

- A previously extracted transcript
- Chronologically ordered input data

Also, a simplified version of the graph as described in 3.2.4 might be helpful in the overall context.

An algorithm for this problem must be able to handle the following:

- Assigning sentences without mapping locution to another locution
- Splitting sentences with more than one mapping among the locutions
- Taking the current speaker or speaking part into account

3.2.6 Topic Exploration

To fulfill 5 in 3.1, fundamentally different methods have to be used, since neither the data nor the transcript can tell what topics the speakers are talking about. There are several ways to extract topics, e.g. machine learning approaches could be used as well as communication theory approaches.

Prerequisites

The visualization prototype will use Latent Dirichlet Allocation (LDA). LDA is a statistical model used to discover hidden themes or "topics" within a collection of documents. Each topic is characterized by a distribution over words. [BNJ01]

In practice, the input consists mainly of two things: an (ideally) large number of so-called "documents", where each document can be represented as a sequence of words or tokens, and the number of topics to be returned. Often, preprocessing is used to remove stop words (common words such as "and", "the", "is") from the data set, as well as to tokenize and lowercase the data. Since LDA requires the transcript data, it must be extracted first.

Requirements for the Algorithm

To apply LDA, the transcript must be divided into chunks and the number of topics must be defined. These topics should not contain verbs like "to be", "to go" or similar, but only more idiomatic verbs and nouns. This requires filtering the words within the chunks to get better results.

Therefore, the topic extraction requires:

- A previously extracted transcript

An algorithm for this problem must be able to handle the following:

- Filtering meaningful topics

4. Data Transformation and Refinement

In this chapter, the transformation and refinements being performed on the data are explained.

4.1 Backend Language and Used Frameworks

The chosen programming language for transforming the input data is Python because of its status as a standard tool in real-world data processing. In addition, Python is a widely used backend language and offers a variety of libraries such as FastAPI.

FastAPI is a modern and minimalistic open-source Python web framework, which makes it suitable for a small application like the visualization tool, and hence the reason for using it [Ram]. The process works simply by calling transformation and refinement methods, followed by sending the processed data to the frontend. In addition to FastAPI, the built-in Python logger is used to keep track of irregularities in the transformation process.

Another Python library used is Network-X. Network-X is an open-source Python library for creating, manipulating, and analyzing complex networks or graphs. Since the QT30 dataset actually consists of graphs, only in the form of JSON objects, it is natural to choose a graph for the internal representation when extracting the data. For this purpose, a directed graph is used, which can have multiple edges between two nodes, as is the case in IAT maps. Another advantage of Network-X is that it allows a fast search in and against the direction of the edges[HSSC08].

4.2 Transformation Process

The optional and non-optional prerequisites explained in 3.2 can be represented in the form of a directed graph, where the required operations are the nodes and the prerequisites are the edges. Thus, if step B has step A as a prerequisite, A has an edge to B. If the prerequisite is optional, the edge is represented by a dashed line. A topological ordering of the resulting graph describes a possible order in which the steps can be executed. There are several possible topological orders, the one used for the data transformation is shown in figure 4.1. The actual procedure uses an additional filtering step.

The following eight steps are taken to fulfill the requirements described in 3.2:

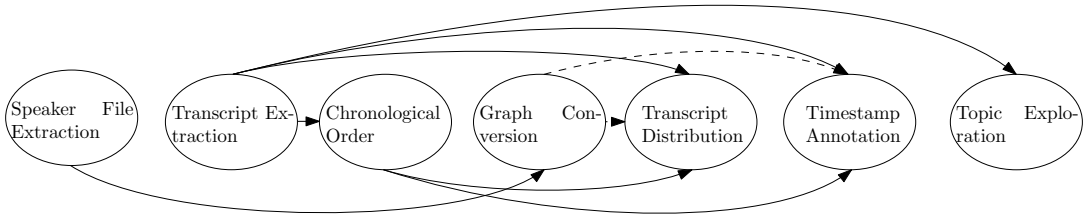

Figure 4.1: One possible step order

1. Extraction of the Speaker File

   This step extracts the speaker file by iterating over the file lines and creating an entry in a mapping for each line. The process is trivial and will not be described in detail.

2. Transcript Extraction

   This step extracts the transcript file and creates a code representation so that positions within the transcript can be mapped to obtain chronological order.

3. Data Extraction and Determination of a Chronological Order

   In this step, all the valuable data is extracted from the files and a graph is created from the data. The graph is a direct representation of the IAT maps. In this step the locutions are also located within the transcript and assigned a position. At the end of this step the nodes can be ordered chronologically. This step also extracts the IMC file and adds long-distance connections.

4. Filtering

   This step removes all isolated nodes, i.e. nodes with no incoming or outgoing edges and all unusable "L" nodes, including the "Analysing" "YA" nodes.

5. Graph Conversion

   In this step, the previously generated graph is simplified and converted into the form described above. The result will contain only one type of node, and its edges will carry some additional information.

6. Transcript Distribution

   This step uses the sorted graph data to distribute the transcript text on the nodes. After this step, the whole transcript should be reconstructable by concatenating the node texts.

7. Timestamp Annotation

   This step uses the timestamps in the transcript to guess the time a statement was made.

8. Topic Extraction

   This step re-extracts the transcript, creates chunks for applying Latent Dirichlet Allocation and returns the topics found.

4.3 Transcript Extraction

The chronological order, the division and distribution of the transcript text among the locutions, and the timestamp annotation are all based on the transcript data, so a proper extraction and code representation of the transcript is essential. The structure of the transcript is described in 2.2.2

The valuable information is

- The item number

  Each file belongs to exactly one part. Knowing where a part begins and where it ends can be used to reduce search times and the potential for mismatches.

- The Speaker

  Information about the speaker can be found in both the transcript and the QT30 dataset. It may be useful to use the speaker to increase the accuracy of string mapping by taking an additional factor into account. This is not currently done, but may be done in a later version of the visualization tool.

- The timestamp

  The timestamps in the transcript are necessary for timestamp annotation, as it is the only reliable source of timestamps.

- The text content

  This is important for the distribution of the text of the transcript and is certainly the most important piece of information in the transcript.

The word count can be omitted as well as any other content such as gestures, actions, or events such as "white_check_mark, eyes, raised_hands".

The code representation of the transcript should be as close as possible to its actual structure, so a dictionary containing the part number as a key is used in the first place. Within a part in the transcript, there are several sequences of a speaker name followed by the timestamp and the text in the next line. In the code representation, each of these sequences is assigned a number and contains a tuple. This tuple contains timestamp, speaker name, and two lists. The first list contains the text that belongs to this timestamp, broken down into individual sentences. The second list has the same number of elements as the first list. But instead of sentences, it contains empty lists. The second list is added in preparation for the mapping problem to be solved for the chronological order.

Thus, the structure of the transcript in the code is as follows:

Part: 1

Subpart: 1

(timestamp1, speaker1, sentences1, mapped_sentences1)

sentences1: [sentence1, sentence2, sentence3]

mapped_sentences1: [[ ], [ ], [ ]]

Subpart: 2

(timestamp2, speaker2, sentences2, mapped_sentences2)

sentences2: [sentence1]

mapped_sentences2: [[ ]]

Part: 2

Subpart: 1

(timestamp1, speaker1, sentences1, mapped_sentences1)

sentences1: [sentence1]

foundstatements1: [[ ]]

To extract the valuable information, the extraction algorithm iterates through the lines of the transcript and uses regular expressions and pattern matching. A new part begins when the regular expression matches, which happens when a line starts with the word "Part" followed by a whitespace and a one- or two-digit number. Matching the speaker is a bit more challenging because the speaker can be made up of two names, or in the case of audience contributions, a name and a number. Therefore, the first word must consist of letters from the alphabet, while the second word can consist of letters and numbers. In addition, '-' can be part of a name, as in "Helle Thorning-Schmidt". A text part can be identified by the timestamp, which always has the format $[hh : mm : ss]$.

If a text fragment is found, it is split into timestamp and text.

The text is further split into sentences using the following regular expression

```
[^.!?]+[.!?]?
```

It splits the text into sentences at the punctuation marks by finding and preserving them.

4.4 Data Extraction and Determination of a Chronological Order

Chronological ordering is done in the same step as extracting data, following transcript extraction. This is because finding the locution position in the transcript does not require much information about the surrounding locutions and, more importantly, the mapping can be done filewise. Since all locutions within a file also share their transcript part, and the transcript has 40-80 parts, this means that the number of potential matches decreases to 1/80 - 1/40 compared to a not file-wise process. Thus, the probability of matching the wrong position shrinks drastically. The combined process of extraction and position determination works as follows.

4.4.1 Creation of the Graph

A loop is used to iterate through all the files in the folder containing the files for the debate to be visualized. During this process, the program makes sure to take valid, not empty, JSON files.

Every valid file is read. First, another loop iterates over all node elements in the data, extracting the text, nodeID, and type, as these three are the usable information. Then it checks if there is a "locution" element with the same nodeID. In the current version, it does not take into account whether the current node is an "L" node or not, which might cause some unnecessary overhead, but this could easily be changed. If a locution is found to a node element, it means that this is a usable "L" node, i.e. a valid locution. These nodes will undergo the process of locating it in the transcript described in 4.4.2.

If a statement could be found in the transcript, a new node is added, containing the three attributes nodeID, text and type, but also its positional data and the timestamp associated with this area in the transcript. Furthermore, the speaker will be part of the information. The personID – speaker mapping created earlier is used for this purpose. However, if a speaker is not found among the speaker names in the mapping, it is because the mapping only contains show guests and not the names of people in the audience. In this case, the speaker name will be set to "Public".

Instead of using the speaker name from the locution elements, the speakers from the transcript could also be used. However, since the existence of "locution" elements with the same id as an "L" node indicates which "L" nodes are usable/represent locutions, this can

also be combined and the speaker can be taken from a "locution node" without additional effort.

If a statement could not be found in the transcript, no graph node is added for this node, since if it cannot be properly ordered, the corresponding data is missing rather than at the wrong position. These unmapped statements are documented by the system logger.

Nodes that do not have a locution will be added directly to the graph with the three attributes mentioned above.

After all nodes have been either added or omitted, all edges are added as edges with the attributes fromID and toID.

Extracting the IMC file is another part of the data extraction. This process happens after the completion of the previous one, but works very similarly: First, the file is read, then a loop goes over the nodes. However, since the "L" nodes and "I" nodes in the IMC file are copies of other nodes, it is not necessary to add them again, even though adding a node with the same ID would not actually add a new node, but rather extend the existing node in network-x. Again, all edges are added to the graph.

4.4.2 Finding the Location Inside the Transcript

The first thing to do to find the correct position of a statement in the transcript is to determine the part of the transcript to which this file belongs. Before the part can be found, the locution text to be mapped must be adjusted. First, the speaker name prepended to the text is removed, which is done using a regular expression. The string matching currently has no tolerance for small errors and typos, but the comparison is case-insensitive, since upper and lower case differences between the transcription and the locution are quite common. To reduce the likelihood of mismatches for short words, especially the word "no," this case-insensitivity only applies to words with more than five letters. Another way to increase the accuracy of the mapping, comparing the speaker in the transcript with the speaker in the locution, is also not included in the current version, but may be added in the future.

Once the text has been adapted, the part number can be identified. This is done in a very straightforward way: The transcript is simply searched for the first sentence where the first locution of a file matches. Then, the part number where that sentence is located can be used for all locutions of that file. The actual search is done by iterating through the code representation of the transcript part by part, subpart by subpart, and sentence by sentence. In this way, each sentence in the transcript is effectively compared to the adapted locution text at most once. If the adapted text is found in a part, the part number is returned.

This method seems to have several shortcomings, first of all performance, since it requires searching the whole transcript in the worst case. But since this only happens once for a file, since all other nodes in that file are obviously in the same part, this is only necessary 40-80 times[HJKS+22] for an episode. In addition, once a file has been associated with a part, the part will be ignored by the files that come after that file. This is achieved by simply keeping track of the part numbers that have already had a matching file. In fact, the performance penalty is negligible, since the expected time to find the first match only requires searching through half of the **remaining** files each.

Another potential problem with this method might be a lack of accuracy, since a statement might appear twice in the text. This may indeed be true, although it is very unlikely, since the parts are deliberately separated and usually start with a non-trivial, distinguishable sentence. In practice, this method has been a reliable way to determine the correct part number for three different debates. Another option would have been the use of an

additional file that contains a file name-part number mapping, similar to the personID-speaker mapping file. However, to minimize the amount of additional information necessary, this problem was solved algorithmically.

Once the part has been determined, the following process takes place for each locution: A loop iterates through the subparts and sentences within that transcript part. After a locution has matched a phrase in one of the subparts of the code representation of the transcript, there are two cases: either it is the first match for that entry, or there have already been other matches.

In the first case, the phrase, together with the position of the first letter and the position of the last letter in the phrase, can be added to the corresponding list of matches. The corresponding list of matches is part of the code representation of the transcript and is located at the same index as the phrase within the list of phrases, but at the fourth position of the tuple. So for instance

> "I agree we want to get the kids back in school if we can as quickly as possible"

would be added to the list of matches of

> "**I agree we want to get the kids back in school if we can as quickly as possible**, but it's got to be done as safely as possible."

as a tuple together with the indices "0" and "80".

This is done to allow efficient ordering within a list of matches. It also enables not only 1:1 mappings to be handled, but also 1:n mappings.

So the position of a node can be defined by:

1. The part number
2. The subpart number
3. The sentence number
4. The start index of the locution, if several locutions match the same sentence

If the list of matches is no longer empty, because one or more other phrases already have a match for the same sentence, we have to check whether the new locution is disjoint to all the others which are already in the list. This means that the index of the character at which the new locution begins in the sentence must not be less than the index of the last character of a previous match, while at the same time the index of the last character of the new locution must not be greater than the index of the first character of the same previous match. This has to be true for every previous match. For example, if "I agree we want to get the kids back to school" and "we want to get the kids back to school as soon as possible" were locutions, they would not be disjoint. The sentence "It doesn't and it's disgusting and the conference should have been as virtual as possible." on the other hand has three disjoint locutions (Figure 4.2).

However, if the locution can be mapped disjointly, it is added to the list of mappings (again together as a tuple with the index of the first and last letter in the sentence). The list is then sorted in ascending order according to the first index of the sentence.

But if the sentence cannot be matched unambiguously, it means that one of the sentences has been mis-matched. To solve this type of problem, we rely on a heuristic and take the first match when there are two non-disjoint matches. So we assume that the second match is the wrong one for this particular sentence and could match somewhere else. The reason we chose to use this heuristic is that it allows us to use the order of the statements in the file. Although it is not guaranteed that the files are perfectly ordered chronologically, they

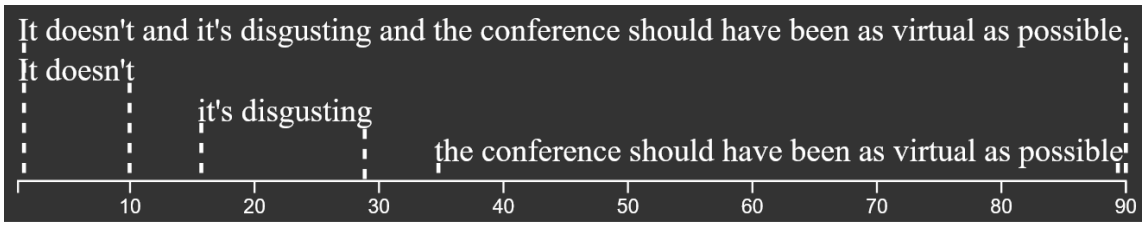

Figure 4.2: Disjoint sentences

usually are, which means that the first occurrence of a matching statement is most likely already the correct one. So the non-disjointly mapped sentence would go into another iteration, as it has not yet found the correct position in the transcript.

As explained in 4.4.1, the unmapped locutions are ignored and do not end up as nodes in the graph. Since their percentage is usually less than 1% of the locutions of an episode, the impact of their loss is insignificant.

4.5 Filtering

After the graph is created, there are some superfluous nodes that can be removed. First, there are nodes that have zero incoming or outgoing edges. These isolated nodes are either the result of errors in the data or unusable "L" nodes without "Analysing" connections. In either case, they can be removed without the risk of losing important information. Filtering must be a separate process, since it is not possible to exclude all isolated nodes during extraction, due to the fact that connections to and from nodes can be added even after a node has already been added, and therefore the number of incoming and outgoing edges can only be determined after creation.

All isolated nodes are collected in a list, along with the nodes that do not have a "type" attribute, whose use would otherwise have to be assumed. Then the nodes are iterated through to find the "Analysing" nodes. These are added to the list, as well as the "L" nodes that have an edge to them, since this is only the case for non-usable "L" nodes.

Then all these nodes are removed from the graph, which means that all their edges are removed as well.

4.6 Graph Conversion

The next step is to convert the graph to the improved form described in 3.2.4. There are many ways to perform this conversion, but the algorithm used by the visualization tool consists of two main steps:

1. Creating a mapping for corresponding "I" and "L" nodes
2. Graph conversion

4.6.1 Node Mapping

There are several possible starting points for the mapping. It can either be started at the "L" nodes and searched along the edges until the corresponding "I" node is found, or it can be started at the "I" nodes and searched along the edges for the corresponding "L" nodes in the opposite direction of the edges. Since Network-x allows both without additional overhead, both methods are valid. The id of the "L" was determined to be also the id of the merged nodes, because these are the ones with corresponding locutions. As the

only essential connections that need to be present in a converted graph are prepositional connections, i.e. "I" to "I" connections, it is better to use the id of the "I" node as a key in the map. Thus, the "I" nodes were chosen as the starting point to create a map with the id of the "I" nodes as the key and the id of the "L" nodes as the value.

Consequently, the node id mapping algorithm iterates over all "I" nodes in the graph. For each "I" node, it takes all predecessors that belong to Illocutionary Connections (i.e., of type "YA"). In the episode from 04.11.2021, about 98% (643 of 659) of the "I" nodes have only one "YA" node as a predecessor, but there are some that have more than one.

This is usually the case when an "L" node does not have a corresponding "I" node. Instead, it will normally have a connection to some other "I" node. In this case, it is not clear which of them is the correct one, so if both connections actually lead to "L" nodes, the second one will overwrite the first one in the mapping. If only one leads to an "L" node and the other is the result of an error in the data, the correct one can be determined unambiguously.

The next step, however, is to determine the predecessor of the "YA" node. If it is not an "L" node, this is a sign of an error in the data and will be logged. But if it is an "L" node, there are two possibilities: Either it is a quotation and there is another "L" node with a connection to it, or it is an "L" node representing a locution. In the QT30 episode created on 04.11.2021, there are actually no quotations at all.

The id of that "L" node is set as the value for the current "I" node if there are no more "YA" nodes to an "L" node at that time. But if there is another incoming edge from a "YA" node, it must be checked again if this node has an "L" node as a predecessor. If so, the id of that "L" node is used for the mapping, otherwise there must be an error in the data. Once every "I" node has gone through this process, the id mapping is finished.

4.6.2 Graph Conversion

Instead of manipulating the existing graph, a new graph is created and populated with all the nodes and edges that are needed. This way there is no need to explicitly remove transitions.

The algorithm again iterates over the "I" nodes of the existing graph. For each "I" node, a node is created which has the id of the mapped "L" node and all of its data, effectively creating a copy of the "L" node, and the text of the "I" node is added to it. Although there are "L" nodes connected to two "I" nodes, which means they would be created twice, this results in only one node, since creating a node with the same id as an existing node only extends the data of the existing node in Network-x. Consequently, this node will receive all connections from both nodes. This does not represent the node correctly in the IAT map, but it does represent the node exactly as it exists in the dataset. After creating a new node, the algorithm iterates over the outgoing edges of the respective "I" node in the existing graph. These are usually between zero and two, and rarely more than three. If the node has a prepositional connection, nodes of type "CA", "RA", or "MA" are found. These nodes always have an Illocutionary connection from a transition node and therefore an incoming edge from a "YA" node. For example, a "MA" node ("Default Inference") usually has an incoming edge from a "YA" node with the text "Arguing". The text of this node is taken and the outgoing edges from the "CA", "RA" or "MA" nodes are used to find the target "I" node of the Prepositional Connection. If the target node is not in the node-id mapping, an error message is logged, otherwise a new edge is added with the following data:

1. The id of the source node
2. The id of the target node
3. The text of the "CA", "RA" or "MA" node

4. The text of the "YA" node that has an edge to the "CA", "RA" or "MA" node.

The ids of the source and target nodes are looked up in the node-id mapping. As a result, edges in the new graph look like this

('775927', '775933', 'Default Rephrase', 'Restate')

This way there is only one node type in the new graph, all other node types like "I", "YA", "TA", "CA", etc. have either been replaced by merged nodes, omitted, or only their text has been taken and added to an edge instead.

4.7 Transcript Distribution

At this point the data exists in the form of a graph. Since the division of the transcript text into locutions requires a chronologically ordered list rather than a graph, the first step is to take the node data of the graph instead of the graph itself. The data can then be ordered by sorting the nodes according to three parameters: 1. the number of the part, 2. the number of sub-part, 3. the number of the sentence. Those sentences with multiple matching locutions are already sorted and don't need to be sorted again.

As explained in 3.2.5, two things need to be determined:

1. What happens to the sentences that are not mapped?
2. How to divide those sentences that have several disjointly mapped locutions.

There is also a need to have a look at the current speaking part of a part in the transcript.

For the first question it is helpful to look at the data. In fact, the sentences that are cut out are in most cases introductory, so that there is usually a locution by the same speaker following the first one. Therefore, it makes sense to splice together the deleted sentences and their subsequent sentences. For example, "Andy, let's start with you." is an introductory sentence by the moderator and has no ADU and therefore no locution. However, the statement is followed by the introduction of a new topic with the words "Obviously, some schools are going to go back on 1 June", to which there is a mapped locution.

By placing unmapped sentences before the first mapped locution that comes after them in the text, almost any part of the transcript can be divided between the locutions, except for those that are at the end of a speaking part. Placing sentences from the end of a speaking part before the next locution that follows could result in one speaker being given the text of another, which is unacceptable. Therefore, this is a special case that needs to be dealt with, and in this case the text should be added to the previous locution. Currently, this special case is not part of the implementation of the transcript division algorithm, so the 1-2 sentences/sentence parts in the visualized debate affected by it are not included. Therefore, the algorithm does not meet the criteria of completeness at the present time. However, as these sentences do not have an argumentative function, their absence is not a significant problem.

The distribution of a sentence in the transcript to several disjoint statements that are mapped to it is done fairly straightforward: The first statement gets the text from the beginning of the sentence up to the index of its last letter. This index has already been determined in the process of locating the statement in 4.4.2 and can be found in the data of the code representation of the transcript. The second statement is mapped to the first character after the end of the previous statement up to its last index, and so on up until the last statement (Figure 4.3).

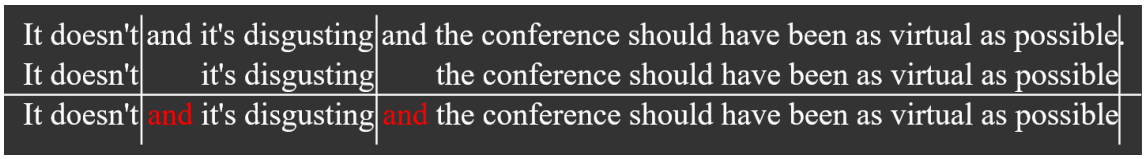

Figure 4.3: Text distribution

The algorithm iterates over the sorted list, tracking the part number, the subpart number and the sentence number. If the current node has a different part number or timestamp number, the sentence number is reset to 0, otherwise there are two cases: Either there are several locutions for a sentence, or there is only one. (As we iterate over the nodes, unmapped sentences will be "skipped").

If there is only one statement, it is mapped directly, i.e. the locution is assigned the whole sentence. Otherwise, the sentence is split and the locution is mapped to the part of the sentence that is assigned to it, using the rule from above. This ensures that each locution is mapped appropriately. If the sentence number of a locution is higher than the sentence number of the previous one by more than one, this means that there are unmapped sentences before it. Then, all previous sentences are concatenated and the number of sentences is increased until the sentence number is equal to the number of sentences of the locution. Finally the current locution text is concatenated with the previous sentences and the loop goes into the next iteration until all elements of the list have been assigned a part of the transcript.

4.8 Timestamp Annotation

Now the individual nodes can be annotated with timestamps. Again, the sorted graph data is used. A loop goes through all the nodes and takes the timestamp string, which is originating from the transcript, of each node. If it is the same timestamp as the previous node, the node is added to a list and the next iteration starts. As soon as there is a node with a different timestamp, the list of nodes with the same timestamp is taken and each of them is given its own timestamp.

This is done by calculating the difference between the timestamp shared by these nodes and the timestamp of the new node. The result of this calculation is the amount of time to be distributed between the nodes. The time is evenly distributed, so the number of seconds is divided by the number of items in the list to calculate an average time. Then a loop iterates over all the items in the list, tracking the current start value and adding it to the timestamp from the transcript as the start time. The end time of a node is therefore its start time plus the average time. The tracked start value is then incremented by the average time so that the start time of the next node is the same as the end time of the previous node and there is no overlap or gap.

After the nodes have assigned timestamps, the list is cleared and the new node with the different timestamp is added to the list. This process is repeated for all nodes until the loop terminates and the remaining list is annotated with timestamps. The last list cannot take the start time of the next section in the transcript as the next timestamp, so the duration of the last section is set to thirty seconds.

4.9 Topic Extraction

The topic extraction requires the transcript to be extracted in a different way than the previous parts. Instead of a statement-wise partitioning, it is more suitable to take the

timestamp-annotated areas for the application of LDA. For this purpose, the transcript is re-extracted into a list of "documents", where a text of a timestamp-annotated area is considered as a document.

Then, custom stop words are defined, i.e. words that are excluded from the topics. An example of an excluded word is "blah". This word is often used when an audience member asks a question:

> "Without the likes of Russia, India and China on board, are this week's COP-26 breakthrough pledges such as the reduction of methane emissions a glimmer of hope or more blah blah blah?".

The moderator, Fiona Bruce, repeats the phrase "blah blah blah" twice in the context of the debate, resulting in "blah" becoming a word in a topic word list. Obviously, this word does not add much content to the topic, so it is filtered out along with other words.

The next step is to convert the documents into a matrix of token counts that can be used to perform LDA on nine topics in the following step. These two steps are performed using the Python library "Scikit-learn". It returns a consecutively numbered dictionary with several topic words associated with each number. The first seven topic words of each entry are put into a list and added to the list of topics that will be returned later.

5. Visualization

This section describes the visualization of the data. It explains the elements that are part of the visualization and how they relate to each other.

The visualization is mainly done in JavaScript, which is one of the most popular frontend languages at the time. Specifically, it provides the D3 open-source visualization library, designed for creating graphs and diagrams [BOH11]. D3 is used to create most of the parts of the visualization described below.

5.1 Parts of the Visualization

The visualization consists of five parts: The timeline itself, a slider that allows enlarging a part of the timeline for detailed exploration, the transcript, a video of the visualized episode, and topic bubbles, each of which contains five words representing a topic. The different parts of the visualization are interactive and the parts are interconnected, i.e. interacting with one part affects another part. All in all the tool is created in a way that is uses and adapts to the screen size. On the bottom half of the screen there are two elements: the timeline and the slider. The slider is at the very bottom and the timeline above it with the timeline being double as high as the slider. Both have the full screen witdth and in particular their width is the same. In the upper right corner, taking about one quarter of the screen is the transcript. The video and the topic bubbles share the remaining space in the upper left corner. The topic bubbles are at the very left and the video is between topic bubbles and transcript (Figure 5.1).

5.2 Timeline

The timeline is the central part of the visualization. It displays the data from the dataset and provides an overview by showing short and long-distance connections between statements in a temporal context.

5.2.1 Description

The timeline is a diagram in SVG format with an x-axis representing elapsed time and a y-axis representing the speaker. The x-axis starts at zero and goes to about 1h. On the y-axis are the six names of moderator and guests, and "Public", which represents all questions from the audience. This means in particular that "Public" stands for more than one person.

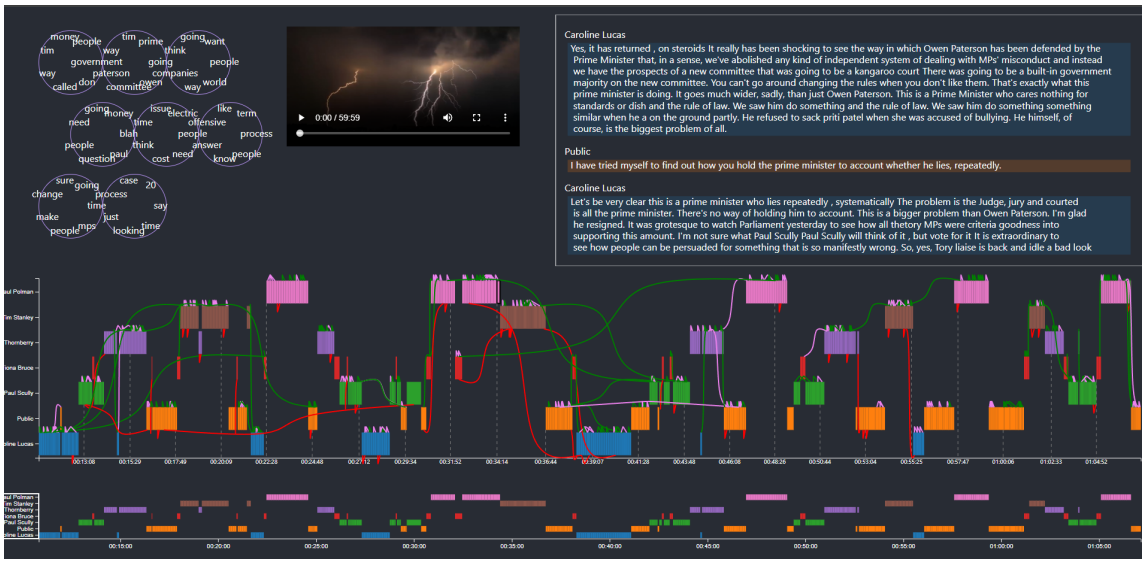

Figure 5.1: Whole Visualization Tool

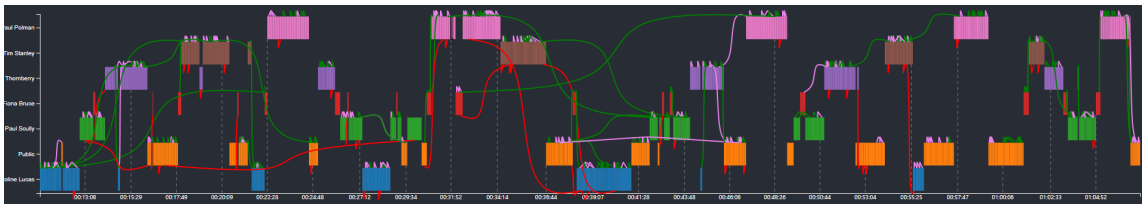

Figure 5.2: The Timeline

Whenever a person speaks, there are bars in the graph, each bar representing a locution/graph node. Thus, each part of a person's speech is usually represented by a number of consecutive bars. The x-value of the bar depends on its start and end time, the y-value on the speaker. There is no overlap between the bars because only one utterance can be made at a time, so the next bar starts when the previous one ends. Each speaker has an associated color.

Attached to the bars are the connections. A connection has a source bar and a target bar, and they start at the center of the top of a bar and go to the center of the top of the target bar if these connections are of the type "Inference" (violet) or "Rephrase" (green). "Conflict" connections (red) start and end at the bottom of a bar. A timestamp is displayed on the x-axis every few minutes (Figure 5.2).

5.2.2 Timeline Creation

To create the timeline, a few steps must be taken. First, we need to retrieve the node and link data from the JSON object sent by the backend. We also need to parse the start and end values of the nodes by defining the time format and creating new dates for each of the time entries as we iterate through the nodes. Then the margins are defined, as well as the height and width of the svg before it is created. The extraction of links and nodes and the conversion of timestamps only needs to be done once for the whole visualization. The defined width and height vary for the different segments of the visualization, but the margins are the same for each SVG as well as the creation process. In the case of the timeline, the height is one third of the screen height minus the margins and the width is the screen width.

After the SVG itself is created, the diagram needs to be generated. For the y-axis, the speakers are obtained by iterating through the nodes and saving the speaker names in a set. The speaker names can then be used for the y-axis using d3 methods. The x-axis is generated using D3 methods with the first start time and last end time data in the nodes. It is generated without ticks, i.e. it is a plain axis. The last thing to prepare before filling the diagram with content is the color scale, which describes the color to be assigned to each y-value, i.e. to each speaker. A standard D3 color scale is used for this purpose.

The next step is to create the bars, sometimes referred to as "nodes", and the edges, referred to as "links", and attach the edges to the nodes. The extracted data is referred to as "node data" and "link data". For that purpose, D3 force simulation is applied on the node data and the link data is used for establishing relationships between the nodes. This ensures that after nodes and links are created in the diagram, the links are attached to the nodes at the appropriate locations. Before the links can be added, a curve is defined based on d3 for a more visually appealing representation of the links. Then the nodes are added. The links are added as path elements in the color of what they represent (blue, violet, red) and have an arrowhead at their end. To determine their path, the y-value of a speaker is used and adapted to be on the upside or downside of a bar, depending on its type. The x value is determined by taking the start time of the source or target node and adding half the difference between the end time and the start time. Then the path is curved using the curve described above. The nodes are added within a group of elements so that they can be moved as one and therefore more efficiently. A node group consists of a node and a timestamp on the x-axis as well as a vertical line between node and timestamp. The nodes are added to the node group as rect elements using the node data. The y-value is the speaker and the x-value is the start time of a node. A node is colored using the color scheme. The other part of the node group are the timestamps for the x-axis and the corresponding vertical lines. They show the start time of a node and are invisible in most cases, except for a few. Whether a node is visible depends on its distance from the last visible node.

After that, the timeline is fully created and listeners are added to handle interactions with the diagram.

5.2.3 Timeline Interaction

There are two ways to interact with the Timeline: By hovering the mouse over a bar or by clicking on a bar.

Hovering over a Bar

This uses the mouseover event. When the mouse hovers over a bar, the following action is triggered: First, all strokes around the bars are removed from the timeline. The bars usually have no strokes, having a stroke is the way the current bar should be highlighted. Then, either all links will be opacified, if the slider is not currently used, or the links inside the enlarged region will be opacified to a lower level (the same level as the visible links outside the region). This step is done in preparation for highlighting the links connected to the respective node, which will be identified in the next step by filtering the links in the link data that have the hovered node as source node.

The next step is to highlight the (sub)sentences in the transcript that represent the data of the target nodes of these links. That is, if node A has text x and a conflict link to node B which has text y, text y would be colored red in the transcript. This is done by iterating over the text elements inside the transcript element and checking whether the text is the text of one of the target nodes of the links that have the hovered node as source node. If the text is found in the target nodes, the text element is colored depending on the type of

Figure 5.3: Hovering over a node

link (Inference, Rephrase, Conflict), otherwise it is colored white, which is the default color. Once all text elements have been colored or left white, the text element for the node over which the mouse pointer is moved has its font weight set to bold. After the connections have been reflected by coloring the connected passages, the connections that have the node as their source must also be highlighted in the Timeline. This is done by simply setting their opacity to 1.0, which reduces the opacity of all links. Then the stroke of the bar the mouse is over is set to highlight the current bar in the diagram. In addition, if there is a topic bubble that contains a word which is part of the text of the highlighted bar, the bubble is also highlighted. This happens via iterating over every text of every bubble and coloring a bubble, if a text inside of it it fulfills the condition.

The last thing that happens when hovering over a node requires a basic understanding of the slider and transcript. The slider is used to magnify the bars in an area of the timeline, highlighting what is within the area. It can be moved to any x-value in the diagram, enlarging the corresponding area. The transcript reflects this by displaying only the text of the nodes that are within that area. As a result, it is common for links to have their source nodes inside the zoomed area, but their target nodes outside the area, especially for nodes close to the boundaries of the magnified part. This means that there is no access to read the statement that another statement is referring to, apart from moving the area, which can be time-consuming if it has to be done each time. That is why these nodes outside the area are not only highlighted by increasing the opacity, which is the level of the nodes inside the area, but they are also appended with their text. This is especially useful for long-distance connections where the source and destination nodes cannot be inside the area at the same time because they are too far apart. To achieve this, the nodes outside the area are identified and their opacity is set to 1.0. Then, text elements are appended to the node elements and placed slightly below the nodes (Figure 5.3).

When the mouse leaves a node, the links are reset to their normal opacity, which depends on their distance from the magnified area. Also, the text elements added for nodes outside the area are removed and the increased opacity of those nodes is decreased again. Also, the bubbles are set back to their default state.

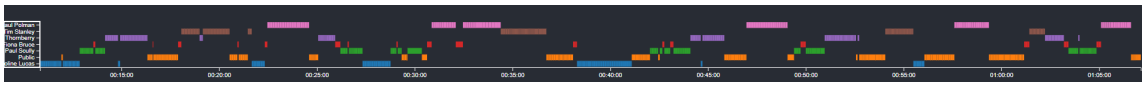

Figure 5.4: The slider

Node Click Action

As seen in the previous sections, the timeline interacts with the transcript and the topic bubbles and also always works with respect to the slider. The "click" action adds yet another interaction, this time with the video. When a node is clicked, the start time of the video is set to the start time of the clicked node. This makes it possible to watch the video starting from the currently observed node, to read along with what is being said, and to analyze the connections of the statements made in real time. Setting the time is as simple as selecting the video HTML element and setting its currentTime attribute.

5.3 Slider

Since the number of bars in the timeline is quite high (601 for the visualized debate), each bar gets very little space on the timeline. As a result, detailed information can hardly be drawn from the diagram. For this reason, the slider can be used to increase the size of a subsection of the diagram by a factor of eight, so that more detailed information can be seen.

5.3.1 Slider Description and Creation

The slider itself looks very similar to the timeline, but smaller. It is essentially the same diagram with the speakers on the y-axis and the time on the x-axis. It has timestamps on the x-axis every five minutes. The bars are displayed as in the timeline with the same color coding. However, there are no connections in the diagram, which makes it appear less complex. There is a white transparent rectangle placed on the speaker names to the left of the first bars which can be moved (Figure 5.4).

The slider not only looks like the timeline, it is also created in a very similar way. It uses the same node data, the same margins, the same width, the same y-scale and x-scale to create the axis, and the same color scale. Only a few things are different: First, it uses a smaller height, only one sixth of the screen height. It also has ticks on the x-axis other than the timeline. And there are no links and no additional timestamps that belong to a node group. The only contents of the diagram are nodes, based on the color of the color scale and positioned in the same way as the nodes for the timeline.

In addition, there is the transparent rectangle with a white border, which is added as a rect element and placed with the right border at x-value zero. It has the height of the slider diagram and a fixed width.

The rectangle is the central element of the slider and the only one in this diagram that has javascript listeners.

5.3.2 Slider Interaction

The slider does exactly what its name implies, i.e. it can be clicked and dragged. However, it has a major effect on the timeline and the transcript. The object to be dragged is the rectangle with the white border, which can be dragged horizontally across the smaller version of the graph. When this is done, the corresponding area in the timeline, i.e. the same bars as them that are inside the rectangle in the slider diagram, are enlarged in the

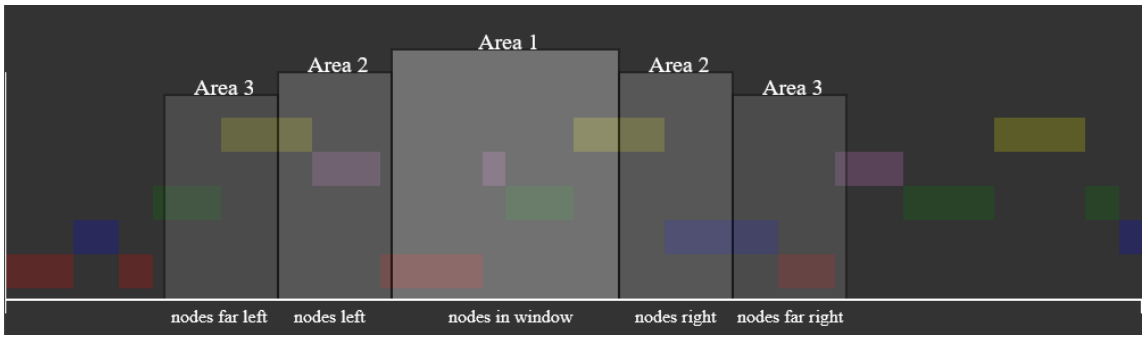

Figure 5.5: The three areas in form of five lists

timeline by increasing the width of the bars. As a result, the total width of the area in the Timeline increases, which means that time is no longer linear. However, the time stamps in the slider remain the same, giving an understanding of the chronological context of the section within the whole debate. On the other hand, the timestamps in the timeline move with the nodes, meaning that within the area, the distance between timestamps increases, indicating that time is "stretched" in that area. Taken together, these two cues provide a good understanding of the temporal context. But this is not the only area where the bars are elongated: To create a "fisheye" effect, two more areas are defined around the area in the rectangle: The areas to the left and right of the central area are magnified by a factor of four instead of eight. And around this area there is a third area where the bars are stretched by a factor of two. The rest of the bars that are not in any of these areas have a reduced length to compensate for the stretching of the others. As a result, the total length of the chart remains the same with or without the slider. Only the part inside the rectangle has full opacity, while the part outside the rectangle has reduced opacity. The opacity of the links also varies depending on which of the three areas they are in. Also, when the transcript is used, only the text of the nodes inside the rectangle is displayed.

To get a slider behavior, three Javascript events are combined: the "mouseup", "mousedown" and "mousemove" events. Moving the rectangle is only allowed when the mouse is clicked down, when it is clicked up, it cannot be moved anymore. When the slider is moved, the video player is paused, then the position of the mouse is determined and the rectangle is set to the mouse position so that its center in x-direction is at the mouse position. Then it is determined, which nodes are in which area. The three areas are represented by five lists:

- The nodes that are inside the window (i.e. the rectangle)
- The nodes that are in the area to the left of the window (referred to as "left of the window")
- The nodes that are in the area to the right of the window (referred to as "right of the window").
- The nodes that are in the area far left of the window (referred to as "far left of the window").
- The nodes that are in the area far right of the window (referred to as "far right of the window").

The three areas have the same size in x-range. Each list except the first has an x-range half the size of the window, and since two lists represent one area, each area has a window length of x-range. The lists are disjoint, i.e. a node is either in one list or the other, not both at the same time. 5.5

When the slider is moved and the content of the rectangle changes, the timeline is updated. However, this only happens when the content of one of the areas changes. Slight moves of the slider may not change any of the lists, so updating all the nodes would only cost resources without any gain. If the content changes, some necessary calculations have to be done:

First, the length of the areas represented by the lists must be calculated, taking into account the scaling. This is done by taking the difference between the last and the first time value of a node in a list and multiplying it by the scale factor (which is two for nodes in the outer areas, four for those to the right or left of the window, and eight for those in the window). Then the length of the part of the diagram that is not in any area is calculated by subtracting the x-value of the first node in the far left list from the x-value of the last node in the far right list and subtracting this from the total length of the diagram. The "x-value of a node" is by default the x-scale of a time value of a node. Although the time values are never changed, the x-value can be manipulated without changing the data. The result of the computation is the length of the part if nothing was scaled. The length of the remaining part, if all areas are scaled, can be determined by subtracting the sum of the length of all scaled areas from the total length of the diagram. By dividing the portion remaining after scaling by the portion that would remain without scaling, an anti-scaling factor can be defined. This factor describes by what the bar lengths of the nodes outside of any area must be multiplied to keep the total length of the diagram at a constant level.

After calculating the anti-scale-factor, the new positions of the nodes must be defined, because if one node takes up more or less space than before, the start position of the following nodes must be adjusted to avoid overlapping. The first node of the far left area starts at its previous position multiplied by the anti-scale-factor, since every node before it is scaled by this factor. The first node of the left area starts at the position of the first far left node plus the length of the far left area. The first node of the window starts at the first node to the left of the window plus the length of the window, and so on until the first node of the area after the far right area. These values can be used later to quickly calculate the new x-value of a node.

The nodes whose x-value is inside the window have opacity set to 1.0, while the others have opacity reduced to highlight the area inside the window. Then the nodes (i.e. the node groups) are repositioned. Their new x-value is determined by taking the adjusted start value of the area they are in and adding it to the position of the node within the area, multiplied by the appropriate scale factor. The position of a node within its area is determined by subtracting the default start value of the area from the default x-value of the node. The new bar width is set similarly by also deciding which area a node is in and multiplying the default bar width by the appropriate scale factor. After the nodes have been updated, the links need to be updated as well. This involves calculating the new position and width of the source and destination nodes, which is done in exactly the same way as for the nodes. Then the center of the top edge of the nodes can be determined and the new path can be set. The opacity of the links is set based on their source nodes. If a source node is inside the window, the opacity is 1.0, as it is for the nodes. If a link has its source node in the area to the left or right of the window, the opacity is reduced to 0.3, just like the node opacity. But unlike nodes, the opacity of links shrinks even further if they are located in the far right or left areas. And if they are even further away, the opacity is set to zero, making the links invisible. This behavior both enhances the fish-eye effect by gradually fading in and out of the links, and reduces the complexity of the diagram for a person using the tool by showing only the links that are relevant to the current part, rather than exposing them to all the links at once.

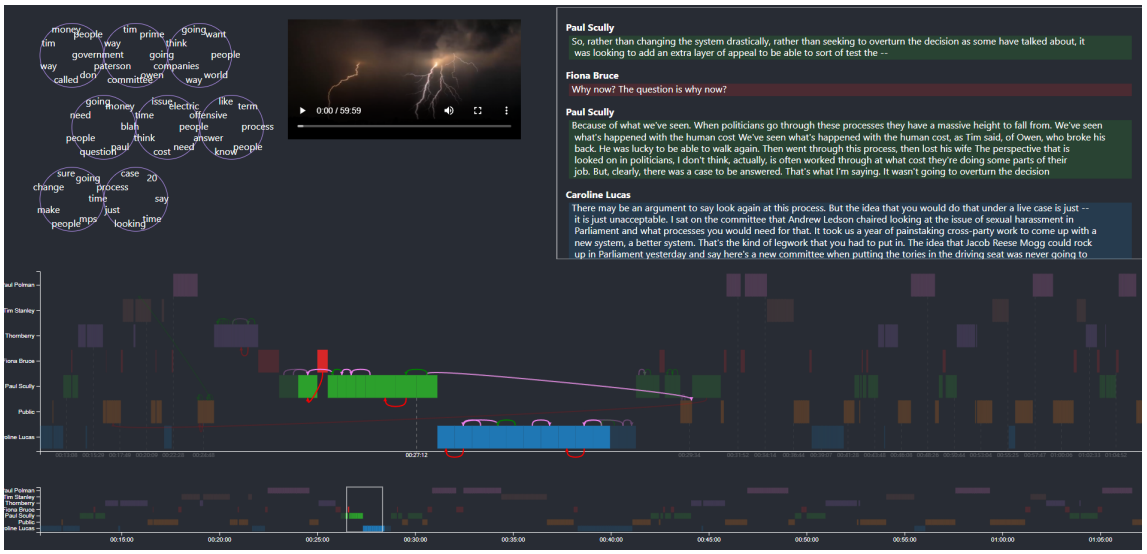

Figure 5.6: Moving the slider

After the positions and opacities have been adjusted, the transcript is re-created and filled with the currently relevant text. Limiting the text to the nodes inside the window is done for the same reason as hiding distant edges, it reduces the complexity of the visualization on the relevant parts. Then the strokes around the nodes are removed, if any remain.

When the slider is dragged back to the speaker names on the left side of the slider diagram and no longer covers any nodes, the original view is restored, i.e. the opacity of all nodes and links is set to 1.0 and the transcript is re-created to contain the full text of all nodes (Figure 5.6).

5.4 Transcript

The transcript gives a different view by displaying the text on the bars. It allows to see what was said and to understand the connections between statements directly in the text.

5.4.1 Transcript Description and Creation

The transcript displays the speakers and their utterances. It shows a speaker's name and below it the text in block form. Below a text block is a short blank space and then the next speaker-text block sequence. The background of a text block is colored in the color assigned to a speaker in the timeline, but with a low opacity. By default, the transcript contains the entire text of the debate, but by using the slider, it contains only the text relevant to the area (Figure 5.7).

The transcript is placed inside a transparent rectangle with a white border called "textbox". This element is added first to the transcript SVG element. The content of what to display depends on whether the slider is used, and if so, its position. If it is not used, the transcript text of all nodes in the data is used, otherwise a subset of it is used by mapping the nodes to the text.

Then the sentences of each node must be assigned a position. The sentences should not end up with one sentence per line. So if a sentence does not fill the given space, another one should start in the same line. On the other hand, if a sentence is too long to fit on one line, it must be split. It is also necessary to group the text by speaker. A block of text should always start with a speaker name, followed by the block of text with the background color, and then an empty line before the next speaker name.

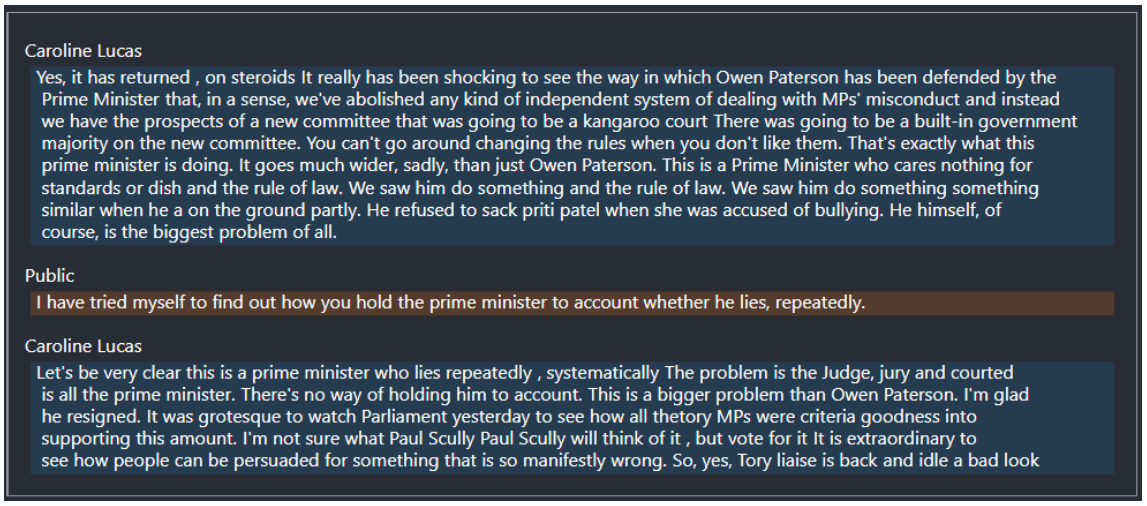

Figure 5.7: The Transcript

The process is as follows: A loop iterates over all (sub-)sentences and tracks several attributes, most importantly the x-value of the previous phrase and the y-value within the SVG element. These x and y values have nothing to do with the timeline or the slider, but refer to the position inside the SVG element. It also keeps track of the index of the char where a sentence was last split, the previous speaker, and the number of chars currently in a line. There is a maximum number of chars allowed in a line that is currently not dynamically determined, which means it may not work for every format. Everything else is dynamically determined to fit different screen sizes.

If the speaker of a text is not the previous speaker (which is true for the first sentence as well), the name of the speaker is appended inside the textbox. If the speaker is not the very first speaker, it is appended on a new line after leaving some space. Also, the height of the rectangle representing the background of the text block before the new one is set to close the previous text block. Of course, this does not happen if the current text belongs to the very first text block. Then a new background rectangle in the speaker's color is added, starting just below the speaker's name. The y-value of the element is stored in a variable to calculate its height when the speaker changes or the loop ends. Then the tracked values are reset (except for the y value, which is incremented) to start a new line, and the previous speaker is updated to the current speaker, whose name has just been added. If the speaker of the current text is also the previous speaker, the text is continued without adding a new speaker name, closing the previous block or resetting the values.

Then, a new text element without text content or position is added to the text box. If the current number of characters in the current line plus the number of characters in the new statement is less than the maximum number of characters, which is 125, the statement will fit in the line. This means that the y-value and x-value can be set to the current y- and x-values, and a unique id can be added. Then, the number of chars in the line must be increased, as well as the current x-value, which requires adding the previous x-value to the computed length of the statement element.

If the new statement does not fit into the current line, it must be split. And since it is quite possible for a statement to span more than two lines, especially if there is already text in the first line, it may have to be split several times. First, a new text element is added with the x- and y-values set to the current position. Then a loop is used to iterate until the entire text has an assigned position. Inside the loop, it is calculated what part of the sentence would still fit into the line. To avoid splitting words, the text part is split at the last occurrence of a space and the split part is taken. If there is no whitespace in the

text part, an empty string is added instead, and the number of characters in the line is set to the maximum so that the next iteration uses a new line. If the rest of the text fits into one line, it is added without searching for the next whitespace. Then a new tspan element is added to the text element containing the text of the split part.

The reason for this is that SVG text elements cannot create paragraphs. So there are two options: Either create a separate text element for each split sentence, or put everything that belongs to a split sentence into tspan elements so that all parts can be associated with one text element. The second option is used to allow faster searching for text elements.

If the split text element is from the beginning of a sentence or empty, it is placed in the current line. Otherwise, a new line is used, the x-value is reset and the y-value is increased, and the y-value of the tspan element is set to the increased y-value. Either way, the x-value of the tspan element is set to the current x-value and the current-x value is then increased by the calculated length of the tspan element. Also, the number of characters in the line is reset or set to the number of characters in the split part. After the last part is added as tspan, the loop ends and after the new text is fully appended, listeners are added to it.

As each text is added, the last background rectangle is given a height based on the difference between the y-value of the element and the current y-value.

5.4.2 Transcript Interaction

There are two ways to interact with the transcript. One is to scroll down the transcript text. The other two are to hover over a sentence in the transcript.

Transcript Scrolling

The number of nodes in the window, the number of different speakers, and the length of the sentences vary depending on whether the slider is used and where it is. As a result, the total height always varies and often takes up more space than intended due to a limited screen size. In order to solve this problem and to be able to display the whole text regardless of its length, a scrolling mechanism is needed. However, scrolling in SVG elements like rectangles is not possible by default, so it has to be implemented using the javascript "wheel" event. To maintain the boundaries when scrolling, the y-values of the first and last text element must be identified. Scrolling should only be allowed when either scrolling up with the first text element not being higher than line one (where it is positioned at the beginning), or when scrolling down with the last text not being scrolled too high. At the moment the "too high" condition is hardcoded, this may be fixed in the future, as it simply requires storing the number of lines of the current transcript in a variable. If the conditions are met, one line (in the unit em) is added to the y-value of each element inside the hover box in the scrolling direction. This happens not only for the text, but also for its background (Figure 5.8).

Hovering Over a Text Element

Hovering over a text element has a very similar effect to hovering over a node. The text elements of connected nodes are highlighted in the same way, the links in the Timeline are highlighted in the same way, the hovered text and the corresponding nodes are highlighted as well. This means that the links in the timeline are also set to a lower opacity and the associated links are identified. Then the text elements are colored if the node they represent has such connections. The links in the timeline that have the node with this text as a source node are set to full opacity. Then the node gets a black stroke and the font weight of the hovered text is set to bold. And if the node with the hovered text has a

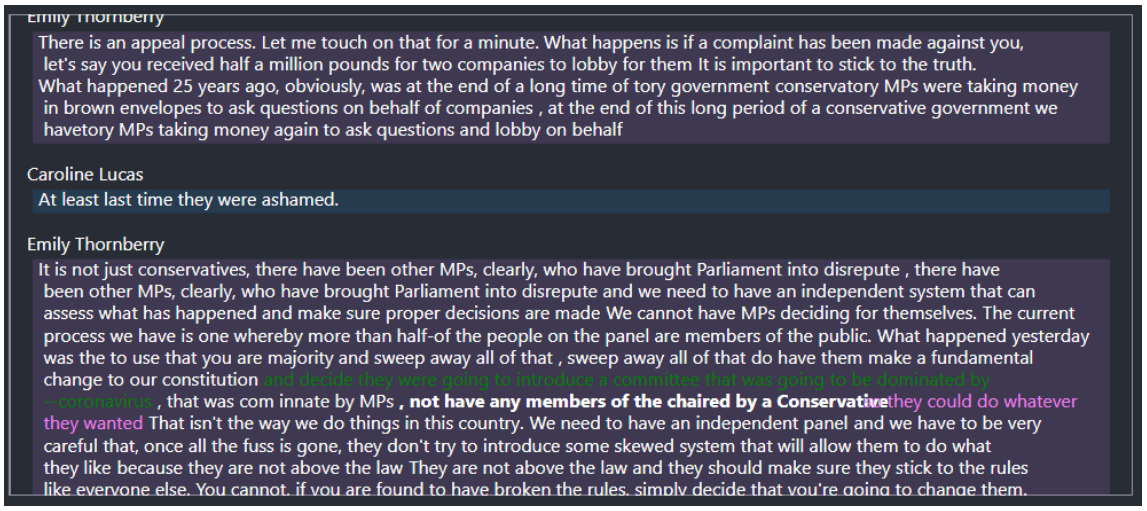

Figure 5.8: Scrolling the transcript

link to a node outside the sliding window, the node's opacity is set to 1.0 and its text is appended and in the end, the topic bubbles are highlighted, just like in 5.2.3.

Hovering off a text element is also very similar to hovering off a node. The link colors in the sliding window are reset to what they were before, and the nodes outside the window are set to a lower opacity. In case there was a selected node outside the sliding window and text element added to it, text element is also being removed. Then all text elements are set back to their default color by iterating over them. Finally, the stroke of the node is removed and the font weight of the previously hovered text is reset to normal as well as the topic bubble appearance.

5.5 Videoplayer

The videoplayer adds an audio-visual view of the debate . It is implemented as an HTML "<video>" element and has the "preload" option set to "auto" to cache the video in the browser (Figure 5.9).

The video player can be interacted with depending on the browser and video format. For example, Google Chrome allows changing the playback speed or switching to full screen mode. However, these options are specific to the HTML element, while there is one function of the video player that is closely related to other elements of the visualization. As the video plays, it updates the position of the slider and thus has all the effects on the timeline and the transcript that the slider has.

This is done using the Javascript "timeupdate" event. It takes the start time of the first node of the data and adds it to the current time in case the first sentence did not start in the first second of the episode. Then the x-value of it is determined. The idea behind this is to use the current position of the video in the same way as the position of the mouse in the slider. So the process is exactly the same as in 5.3.2 from this point on.

5.6 Topic Bubbles

The purpose of the topic map is to help the user of the visualization tool gain a deeper understanding of the content by representing the topics covered in the episode. Each five key words are in one bubble, representing a topic. Interaction with the bubbles should highlight the bars on the timeline that represent a statement that contains that topic in its text.

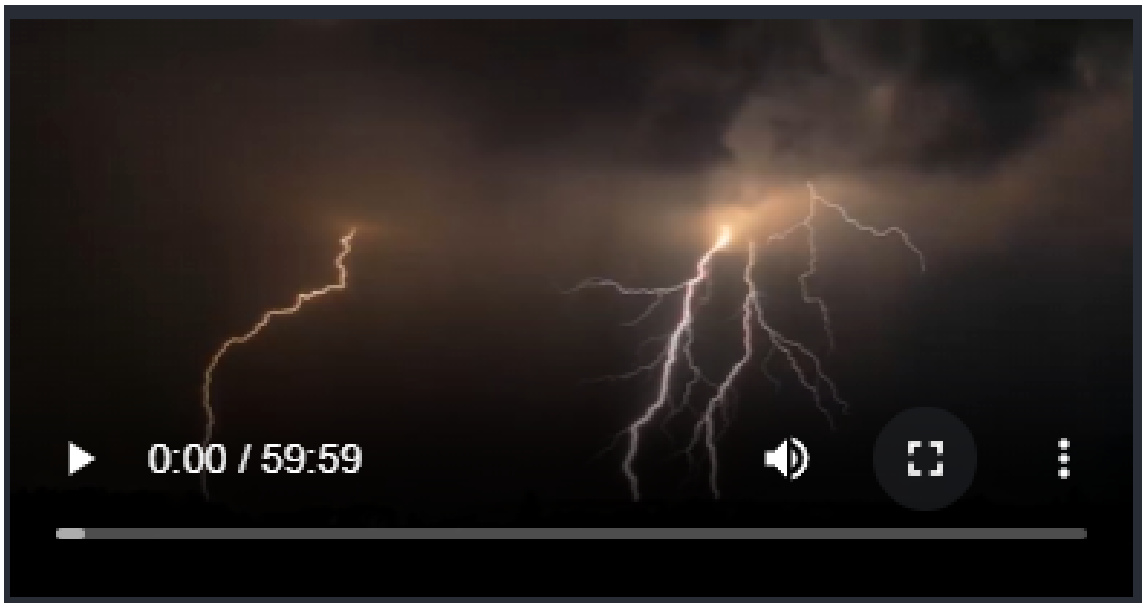

Figure 5.9: The videoplayer

5.6.1 Topic Bubble Description and Creation

The five words of a topic are placed inside a circle, hence the name "topic bubble". There are nine topic bubbles arranged in three rows. The first row contains three bubbles, the second row contains three bubbles, and the third row contains the remaining three bubbles. The second row is shifted so that it is offset from the first and third rows (Figure 5.10).

Each topic bubble is created by first adding a 'g' element to the topic bubble SVG. A transparent circle with a purple outline is added to this element. The radius and position of the element has to be calculated in relation to the given space and the number of bubbles and is described below. Once the circle has been added, the words must be positioned within it. The position of each text element is calculated based on the angle around the circle and the radius of the circle.

The position of each bubble is determined as follows: First, the aspect ratio of the given location is calculated by dividing the width by the height. Multiplying this by the number of bubbles and taking the square root is a heuristic approach to determining an appropriate number of bubbles in a column. The number of columns and the number of bubbles can be used to calculate the number of rows. This allows the maximum diameter of the bubbles to be found by dividing the total height by the number of rows and the total width by the number of columns and taking the minimum of both.

Once the size and number of rows and columns have been determined, the position of each bubble can be decided. The bubbles should not be arranged in a grid or in a circle, but in shifted rows for a more appealing display. A reasonable unit of displacement is the radius of a circle. So the algorithm iterates over the rows and columns, determining the y and x positions and adding the radius of a circle to each uneven row as a shift amount. The x-value, y-value and radius are then placed in a list for later use.

Once all the positions have been determined, it is iterated over the list of topic lists and a bubble for each list created.

5.6.2 Topic Bubble Interaction

There are basically two ways of interacting with the topic bubbles: hovering over the words in the bubbles, or hovering over the bubbles themselves. Hovering over a word will highlight

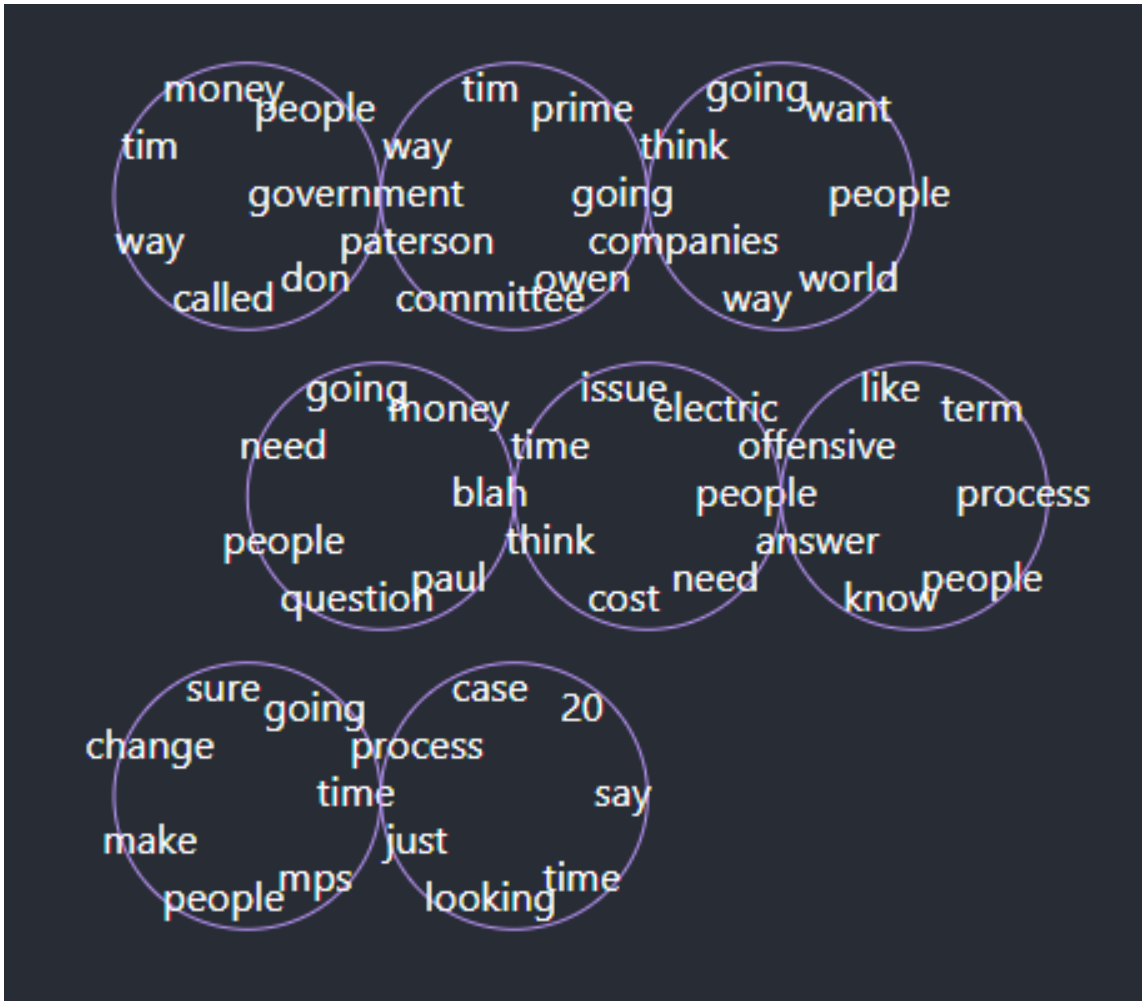

Figure 5.10: The Topic Bubbles

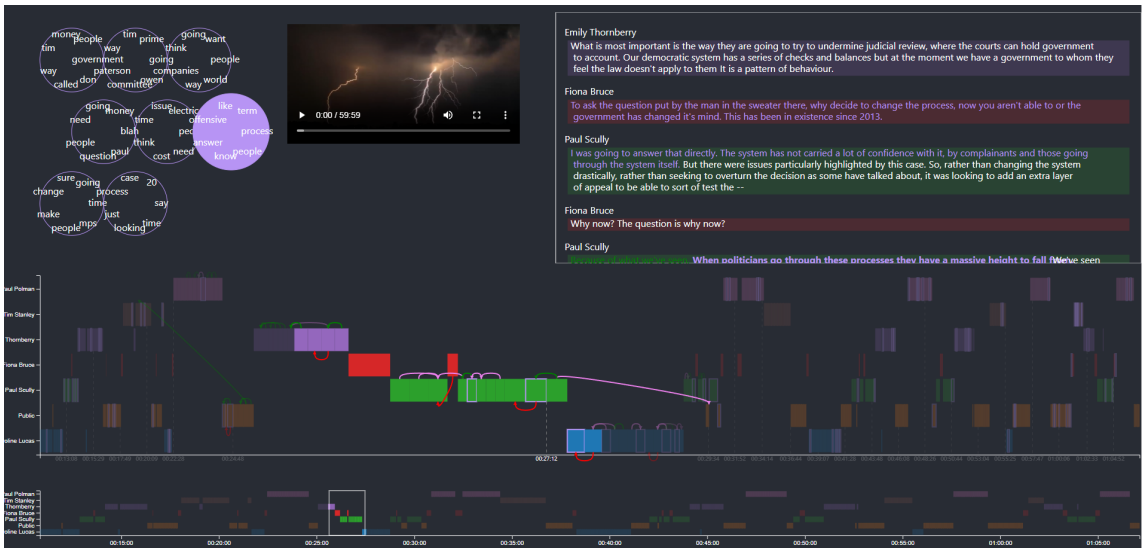

Figure 5.11: Hovering on a Topic Bubble

all bars in the timeline that have a corresponding text containing that word by adding a stroke. Hovering off a bar resets it by removing all the strokes. It also highlights the text elements that contain the topic. Hovering over a bubble has the same effect, but for every text element in it, not just a single text element. To highlight a topic, first the nodes that have a text attribute containing the topic are filtered. Then the stroke attribute for these is set to the color of the bubbles. Next, the text elements that match the condition are filtered and their color is set to the bubble color. Finally, the text itself is highlighted by changing it from white to the bubble border color. There is only one difference between hovering over the text and hovering over the bubble itself: When hovering over text, the text is colored; when hovering over a bubble, the radius of the bubble is increased by 20% in the form of a transition, and the background color is changed to the colour of the bubble's outline to make it stand out even more (Figure 5.11).

6. Conclusion

In conclusion, the Visualization Tool provides a new approach to automatically present and interactively explore the Inference Anchoring Theory annotated dataset. By allowing users to contextualize statements within broader discussions, it allows for deeper understanding of content through an intuitive visual interface accessible on any browser. The tool's multifaceted presentation of data allows users to engage with information in a variety of ways, resulting in a better understanding of the dynamics of the debate.

Looking ahead, there are several avenues for further improvement. First, efforts could focus on refining the accuracy of timestamps to provide more precise temporal context. In addition, adjustments to accommodate different screen sizes would improve the tool's usability across devices. Furthermore, the mapping algorithm could be extended to allow some degree of "error" in the data, allowing a handful of nodes more to be used. And integration of machine-learning processes for topic extraction tailored to debate contexts could enhance the tool's analytical capabilities, allowing for more in-depth insights into discourse dynamics.

Another component could be added to the tool in the future if additional topic data is provided. With provided topics rather than automatically extracted ones, the accuracy would be increased and the nodes containing that topic could be displayed as a graph with further edges to all nodes to which they have propositional connections. As a result, all nodes that are in any way affected by a topic could be analysed in a different way. Another new feature could be speaker-based highlighting, where clicking on a speaker's name in the timeline highlights only the things that speaker said and the statements of other speakers that are connected to them. Also, if the timestamps of speakers were known from the audience who were introducing new questions, it would be possible to have a question selection feature that would extend the time until the next question is introduced in the graph. This feature could help end-users who are not necessarily interested in every topic to focus on the questions that are of interest to them.

By continuing to iterate and enhance the Visualization Tool, we can increase its utility as a valuable resource for analyzing and understanding the complexities of debate interactions.

Bibliography

[BNJ01] David Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[BOH11] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[Cen17] Centre for Argument Technology. A quick start guide to inference anchoring theory (iat), 2017.

[HJKS+22] Annette Hautli-Janisz, Zlata Kikteva, Wassiliki Siskou, Kamila Gorska, Ray Becker, and Chris Reed. QT30: A corpus of argument and conflict in broadcast debate. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 3291–3300, Marseille, France, June 2022. European Language Resources Association.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[JLR14] Mathieu Janier, John Lawrence, and Chris Reed. Ova+: An argument analysis interface. In *Computational Models of Argument: Proceedings of COMMA*, volume 266, pages 463–464, 2014.

[Ltda] Thoughtgraph Ltd. Debategraph. `https://debategraph.org/Stream.aspx?nid=61932&vt=ngraph&dc=focus`. Accessed: April 20, 2024.

[Ltdb] Webalon Ltd. Tiki-toki. `https://www.tiki-toki.com/`. Accessed: April 20, 2024.

[Ram] Sebastián Ramírez. Fastapi. `https://github.com/tiangolo/fastapi`. FastAPI framework, high performance, easy to learn, fast to code, ready for production.

[Tea20] BBC News Press Team. Tweet, November 2020. `https://twitter.com/BBCNewsPR/status/1331526320795365378?ref_src=twsrc%5Etfw`.