



Fabio Rutter

Email Search Tool based on Associative Memory

Bachelor's Thesis

to achieve the university degree of

Bachelor of Science

Bachelor's degree programme: Softwareentwicklung - Wirtschaft

submitted to

Graz University of Technology

Supervisor

Kern Roman, Dipl.-Ing. Dr.techn.

Institute of Interactive Systems and Data Science
Head: Lindstaedt Stefanie, Univ.-Prof. Dipl.-Inf. Dr.

Graz, September 2020

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present bachelor thesis.

28.09.2020

Date

A handwritten signature in black ink, appearing to read 'Julio', written over a horizontal line.

Signature

Abstract

A problem that came up during the last twenty-five years is the re-finding of emails. Many different groups of people have thousands of emails in their inboxes, which often causes frustration during the search for older emails. This fact is reason enough to think about new solutions for this issue. Is the continually managing of your emails with folders and labels the best answer? Or is it more efficient to use a memory-based attempt?

In this thesis, we planned and implemented a search tool for Mozilla Thunderbird to test if it is reasonable to use the human's associative memory for re-finding. The first step was to investigate which different things, besides the conventional text and name, people potentially remember to an email. The decision fell on the separation into three additional searching features. They focus on the email partner's primary data, on side facts to the date, and the option to search for a second email, which the user possibly associates with the wanted email.

To check if the tool is applicable, we evaluated it with several test persons by giving them tasks to complete in a test email environment. The results showed a positive attitude toward these new searching ways. Especially the date-related features were rated very high.

These results lead to the motivation of potentially starting further research on the topic. By discovering that dates tend to be remembered quite well, we can improve the tool in this direction before starting a large-scale evaluation with real email data.

Contents

Abstract	iv
1 Introduction	1
2 Background	2
2.1 Associative Memory	2
2.2 State of the Art	2
2.3 Thunderbird	3
2.4 Used technologies	3
3 Methods	5
3.1 Concept	5
3.1.1 Introduction	5
3.1.2 Basic Search Tool	5
3.1.3 Associations	6
3.1.4 Email Association	6
3.1.5 Date Specification	8
3.1.6 Person Specification	8
3.1.7 Multiple Associations	9
3.2 Implementation	10
3.2.1 Thunderbird Plugin	10
3.2.2 Data Import	11
3.2.3 Basic Search Tool	12
3.2.4 Associations	13
4 Evaluation	18
4.1 Methodology	18
4.2 Results	19
4.3 Discussion	19

Contents

4.4	Limitations	20
5	Conclusions	22
6	Future Work	23
7	Acknowledgements	24
8	Appendix	25
8.1	GUI designing with Zurb Foundation	25
8.2	Source code	26
8.2.1	Email Association	26
8.2.2	Euclidean distance calculation	27
8.2.3	Generation of content elements	27
8.3	Evaluation examples	28

List of Figures

3.1	Flowchart for email association	7
3.2	Search Tool	10
3.3	Hierarchical structure of Thunderbird	12
3.4	Email class	12
3.5	Transformation of input values to numeric vectors	13
3.6	Basic Search tool.	14
3.7	Second tab for email association	15
3.8	Email Association	16
3.9	Date Specifications	16
3.10	Person Specifications	17
8.1	Basic HTML structure	25
8.2	Email Association filtering	26
8.3	Euclidean Distance calculation	27
8.4	GUI element generation	28
8.5	Evaluation tasks	28

List of Tables

4.1	Evaluation table	19
-----	----------------------------	----

1 Introduction

The email was created as an asynchronous communication tool to send messages between persons in the first place, but research has confirmed that emails are used for many different applications. For instance, collaborative working, data archiving, management of tasks, and personal contacts. As a result, many persons have a collection of thousands of emails stored, spanning several decades. This fact implies the challenge of re-finding certain emails, especially when they were sent or received some time ago. [4] [5]

There are different kinds of approaches for organizing and re-finding emails. I describe some of them in the background chapter, but in this Bachelor thesis, I focus on the attempt of re-finding emails by providing an extended way of searching, based on the human associative memory. I designed and implemented a search tool for the email client Mozilla Thunderbird to test if it is more efficient to use a memory-oriented approach to help people save time to find emails. More precisely, the tool provides additional searching methods, focused on using remembered context data to individual emails.

Summed up, the first question to be answered in this thesis is if the tool is capable of searching emails by using options to access associated context data. And the second question is, what kind of provided type of data is the most useful one, given the features provided by the tool.

2 Background

In the background chapter, I describe the essential parts of underlying knowledge and technology, used for this project. Furthermore, the state of the art of email re-finding gets treated.

2.1 Associative Memory

The human memory is a mental system that stores and uses information that a person experiences. These kinds of information can not be retrieved as a whole package but can be reconstructed from small pieces of memories in an associative network. The network consists of several links between nodes of data. For instance, these relations are belongingness to a category, or one thing is part of another thing. If one node in this memory network gets activated, the other relations, depending on the link's strength, also get activated. [2]

2.2 State of the Art

The two main types of email management are, on the one hand, the preparatory organization and on the other hand, opportunistic management. [4] The first one is based on folder structures, where the user can move emails to a certain folder, based on the topic or some other organizational parameter. Modern email clients, such as Mozilla Thunderbird¹, also offer automatic filtering, where the user can define specific rules to move emails to the right

¹www.thunderbird.net (Accessed on: 2020-08-07)

2 Background

folder. The second type, opportunistic management, is based on scrolling, sorting, and searching. This method does not need any preparation and focuses on the user's search queries about attributes he or she can remember. [4]

A newer version famously used by Google's Gmail ² web client is the intrinsic organization of emails. [9] Here conversations, which consist of answered emails, are displayed as separate threads. This is a potential improvement to having every single email in an inbox list or folder. Furthermore, Gmail offers an automatic separation between emails written by real persons, advertisements, social media platforms, and notifications. This filtering may also be beneficial for the process of re-finding.

2.3 Thunderbird

The decision, which email client is being used for this project, fell on Mozilla Thunderbird ³. The open-source application is developed by the Mozilla Foundation and is one of the most widely used email clients.

Besides emails it also supports newsgroups, contact-managing, RSS and chat client. With a broad range of different addons, it is possible to extend the basic functionalities, for instance, by a calendar. These additional features could be useful for adding associations to the email search tool. The main benefit, especially for this thesis, is the addon support and the available documentation ⁴ for developing them.

2.4 Used technologies

Mozilla Thunderbird uses the following four technologies to run addons: JavaScript for the algorithms and the intelligent part, Gecko as layout engine,

²mail.google.com (Accessed on: 2020-08-07)

³Version: 60.9.0 for Ubuntu 16.04

⁴developer.mozilla.org/de/docs/Mozilla/Thunderbird (Accessed on: 2020-09-02)

2 Background

XUL as User Interface Language and XPCOM as Cross-Platform Component Object Model. [8]

The most important part of the thesis is the development in JavaScript. JavaScript is an interpreted programming language, which has been developed by Brendan Eich in 1995. Eich was working at Netscape Communications at this time, when they had the most widely spread browser in the world. [1] In the early days, it has mainly been used for scripts that control the site's behavior and add more interaction. [6] Now also non-browser environments, like Thunderbird, use JavaScript.

3 Methods

This chapter contains the concept and the implementation of the search tool and provides detailed information about the used techniques and algorithms.

3.1 Concept

This section gives an insight into why the particular features got selected and how the algorithm filters and ranks emails based on the user input.

3.1.1 Introduction

The approach to use human memory for re-finding implies to neglect the preparatory organization and go with an opportunistic way of searching.

The whole concept is built on two main aspects of filtering the existing emails. On the one hand, it is the usage of a rating, where every email gets a score on how close it is to the given input. On the other hand, it is a boolean of whether it is relevant or not. The result is always a list of emails sorted by the rating and filtered by the “relevance”-boolean.

3.1.2 Basic Search Tool

First of all, the tool had to have the basic functionality of a common email search tool. The following five categories got picked:

3 Methods

- **To/From:** A boolean, which defines whether the email got sent or received by the user.
- **Name** of the person the user is in contact with
- **Text** or subject of the email
- **Date**
- **Daytime**

The tool takes the input from above and calculates the rating from each email to the given input data. This gets achieved by transforming all the emails, as well as the input data, into comparable, five-dimensional vectors. Then it is possible to calculate the Euclidean distance from each email to the input, which finally results in the rating of each email. So the lowest rating is the closest to the searched one and is on top of the result list.

3.1.3 Associations

This section and the following four contain the new part of searching emails, the associations. As mentioned above, the second aspect of the search tool is the “relevance”-boolean. In most cases, this boolean is the only way the association features are influencing the outcome. One particular case is described in subchapter 3.1.7, where different association tools are working in parallel and affect the rating too.

3.1.4 Email Association

The email association tool is based on the idea of people remembering, they wrote another email to a person, and this email has something in common with the email he or she is currently searching for. This feature’s idea came through an example given in the description for this thesis. Since date and time are quite rememberable topics [7], many parts of the search tool are including time-related functionalities.

The picked types of associations are:

- same daytime
- same month

3 Methods

- same year
- same weekday
- same date
- same name

The user has to fill in the same input fields as in the basic search tool and one of the association types, listed above. The algorithm now creates the same kind of ratings for this second email and lists them starting with the lowest. Per default, it takes the first five entries and checks if the rule, defined by the association, is fulfilled or not. These five associations get linked by an “or” disjunction because it is not sure which of the emails is the one he searched for, so it does not matter if one or four of the rules are fulfilled. But those five default picks are probably wrong, or at least some of them are. Therefore the user can pick the correct ones from a separate list and recalculate the association rules to get the new results. The boolean result flows into the “relevance” value of the email item, and the resulting list. This list, which gets displayed to the user, contains the best-rated emails, except those that don’t match the rule.

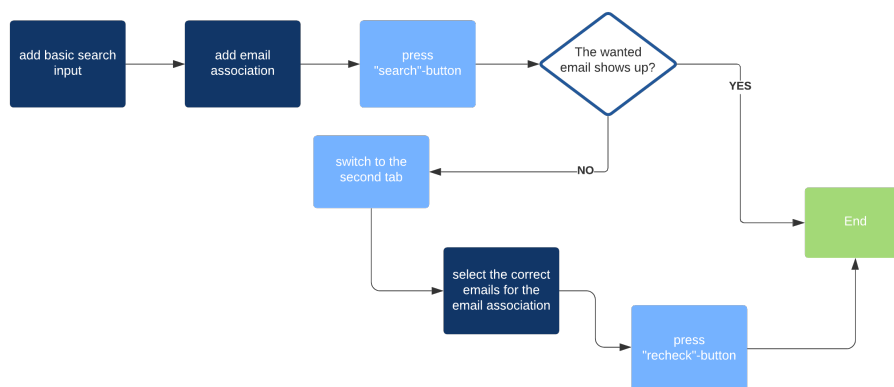


Figure 3.1: This figure shows the flow for the email association feature.

An use case for this tools would be:

John wrote an email to somebody from another department about some flower orders, but he does not know the date or the person’s name. All he knows is, he wrote another email to Jennifer from the human resource

3 Methods

department, about his new hours of labor, around the same period. So he can type in the topic into the basic search and add an association to define the information he knows about the email he sent to Jennifer.

3.1.5 Date Specification

This tool is based on the idea, that a person can more easily find an email by remembering certain details to the sending date and time [7].

So the tool implies three main categories:

- public holidays
- weekdays and workday or weekend
- month and seasons

The user can pick, for instance, Easter Sunday, and the algorithm iterates through all the emails again and checks if it is near Easter Sunday. Since people do not tend to remember exact dates [2], the feature has a tolerance of several days before and after the event. As described above, the results get filtered, and the ones where the rule fits are shown. With this tool, it is also possible to combine two or all three categories. The resulting booleans then get linked by an “and” conjunction.

A use case for this feature would be:

John knows he wrote an email about an order of a new notebook to somebody, some years ago. He only remembers it was a weekend during Christmas time. The tool now provides the opportunity to add this information to ease the search.

3.1.6 Person Specification

This search tool provides the option of entering how much contact someone has had with another person and what number of emails is the wanted email with this person. This feature may be useful if the user does not know the name of the person, he or she is searching for. The first feature is divided in five categories from “once” to “very much”. The second one is more

3 Methods

specific and provides an input for the exact number of the email. Assuming the user can not guess the correct number, results with slight variance are also getting accepted. If both features are used, they also get linked by an “and” conjunction.

An use case for this, would be:

John bought a monitor from a webshop a year ago. He got the recommendation by a comparison-shopping website. The monitor breaks, so he tries to find the email, containing the bill. He can not remember the webshop’s name, but he knows he just once had email contact with this shop.

3.1.7 Multiple Associations

The three types of specifications, described in the last three sections, can also be used together. The email association is the only tool that can be added more than once because it is reasonable to add an association, for instance, for “same time” and another one for “same name”. Where on the other hand, date and person specifications can just be added once.

As mentioned above the state of having more than one association is not just affecting the “relevance” boolean but also the ranking. For every true outcome of an association rule check the rating gets halved. So the email, which matches most of the associations, gets ranked higher than another that only fits one association. So all the emails that have at least one true rule get listed in the final result list.

3 Methods

3.2 Implementation

This chapter is about the technical implementation of individual features, described in the concept chapter. The whole searching program is completely built as a browser application, despite the given email data from Thunderbird and some lines of code to run it as an addon, the application could run in every modern web browser, with JavaScript enabled. This means the business logic is programmed in JavaScript and the graphical user interface in HTML and CSS.

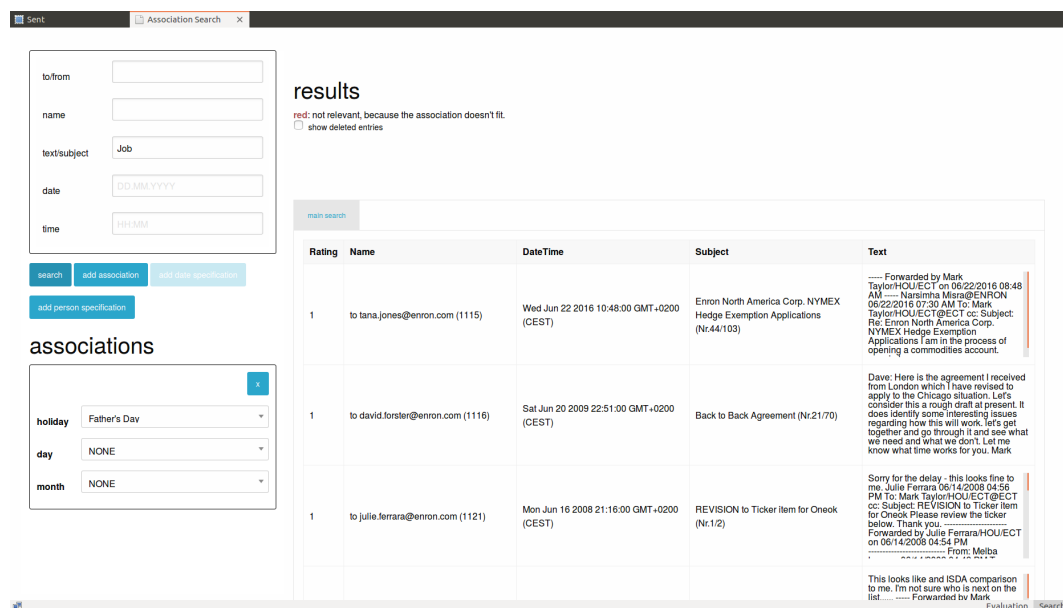


Figure 3.2: The figure shows a screenshot of the GUI of the search tool. On the left side, there is the basic input and all the association fields. On the right side are all the resulting emails, sorted by rating.

3.2.1 Thunderbird Plugin

The thunderbird addon's graphical user interface is made with XUL. The first file has to contain a window tag to create a new page in Thunderbird. The

3 Methods

tag's body includes some JavaScript files and an iframe to integrate the HTML file to the main page of the search tool.

The second thing to do with XUL is to create an overlay to have an access point, such as a button at the bottom of the Thunderbird GUI. This button triggers the initialization process, which compromises the import of email data and the creation of a new tab for the search tool.

3.2.2 Data Import

The first step to gain access to the emails is to import the MailServices module from the internal resources.¹

After that, it is possible to iterate through the mail accounts belonging to the launched Thunderbird profile. These accounts consist of several nested folders, where the emails are stored. The algorithm recursively goes through these directories and stores their objects combined with their type, which tells whether it is an "Inbox" or "SentMail" flagged folder. The folder objects provide access to all the message headers, containing data about the author, subject, recipient, date, etc. For getting the email body it is necessary to use a stream listener² in combination with the data from the message header to extract the text for each email.

The information gained here is stored in new email objects, shown in 3.4, for later usage at the search tool. Since the program is looping over all emails here, the number of emails with each person also gets calculated and stored for later usage of the person specification.

¹developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Resource_URLs (Accessed on: 2020-09-02)

²developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Reference/Interface/nsIMessenger, developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Reference/Interface/nsIMsgMessageService (Accessed on: 2020-09-02)

3 Methods

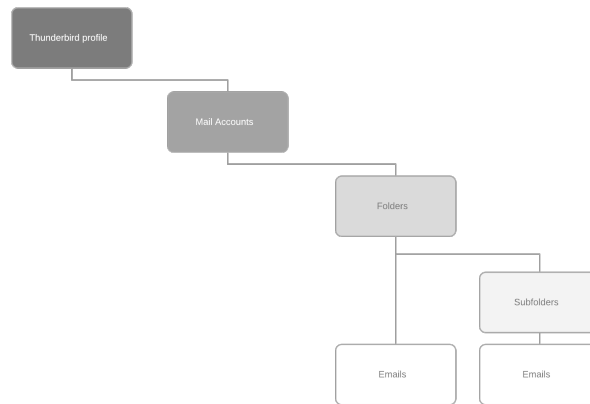


Figure 3.3: The figure displays the hierarchical structure within an Thunderbird user profile.

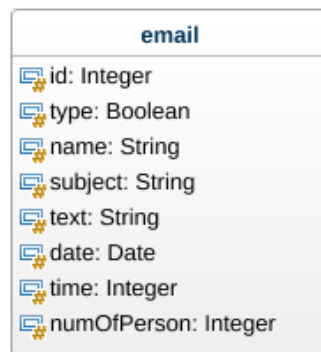


Figure 3.4: The figure shows the data collected and stored to one email during the initialization process.

3.2.3 Basic Search Tool

As shown in 3.6, there are five different input fields to consider. The first step is to determine which ones contain anything to define the vector's number of dimensions. The date, time and "from/to" values get normalized and put into the final vector. The name input gets compared to every other person in

3 Methods

the user's contact list. For this case a JavaScript Library called Fuzzyset.js³ is included. It turns strings into numeric vectors to compare them with cosine similarity. The result for every name is a normalized number, which represents the similarity. The highest one is the best rating, so the input vector gets set to the value one, for later calculation over all inputs. It is also possible to type in more than one name. In this case, the rating gets added up.

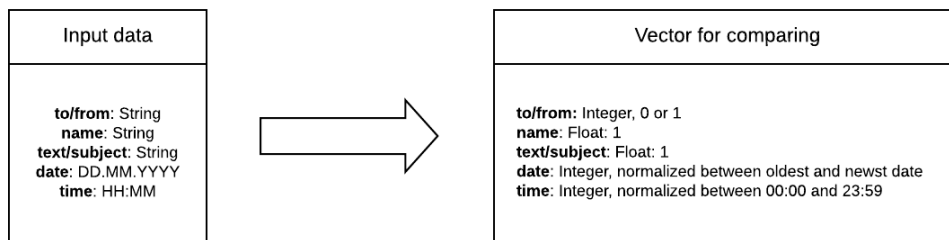


Figure 3.5: The figure displays the transformation of the input data to a comparable vector.

A similar library called Lunr.js⁴ is being in use at the text and subject searching. The difference here is the program searches for multiple keywords in a text and returns a score, which then has to be normalized manually. Again the input vector place for the text field is set to one.

In the end, the Euclidean distance gets calculated for every email vector available. The resulting list gets sorted by the distance and printed in the GUI.

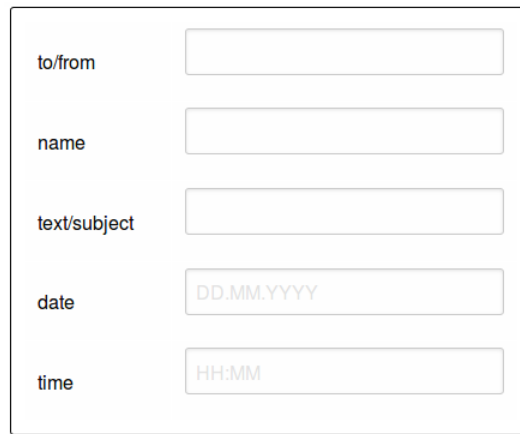
3.2.4 Associations

The resulting list consists of arrays containing the rating, the email object, and a "deleted" flag. The rating is treated in the last section, the new point that is relevant for the association features is this mentioned flag. It defines if an email matches one of the given associations or not and is the indicator

³[glench.github.io/fuzzyset.js/](https://github.com/glench/fuzzyset.js/) (Accessed on: 2020-09-02)

⁴lunrjs.com/ (Accessed on: 2020-09-02)

3 Methods



to/from	<input type="text"/>
name	<input type="text"/>
text/subject	<input type="text"/>
date	<input type="text" value="DD.MM.YYYY"/>
time	<input type="text" value="HH:MM"/>

Figure 3.6: The figure shows the **basic search tool** containing trivial searching fields like name, topic and date.

if the email gets presented to the user. The situation of multiple associations represents a special case and gets described in the last paragraph.

Email Association

The email association feature includes the same input fields as the basic search and an additional combobox to select the association type. The first step is to calculate the results for these new search inputs and store them separately. After that, the five highest-rated findings get extracted for further processing. Every email from the main result list then gets compared to these five emails regarding the selected type of association. If an email fits the given rule, it stays. Otherwise, it is not visible on the main list anymore.

The users can now look at the results, and if the email they are searching for does not show up, they can access the second tab. In this tab, as shown in the 3.7, the email association feature results are presented in the same way as the main results, except that the users can select the correct ones. After selecting, they have to click “recheck association” to recalculate the “deleted” boolean for the main results.

3 Methods

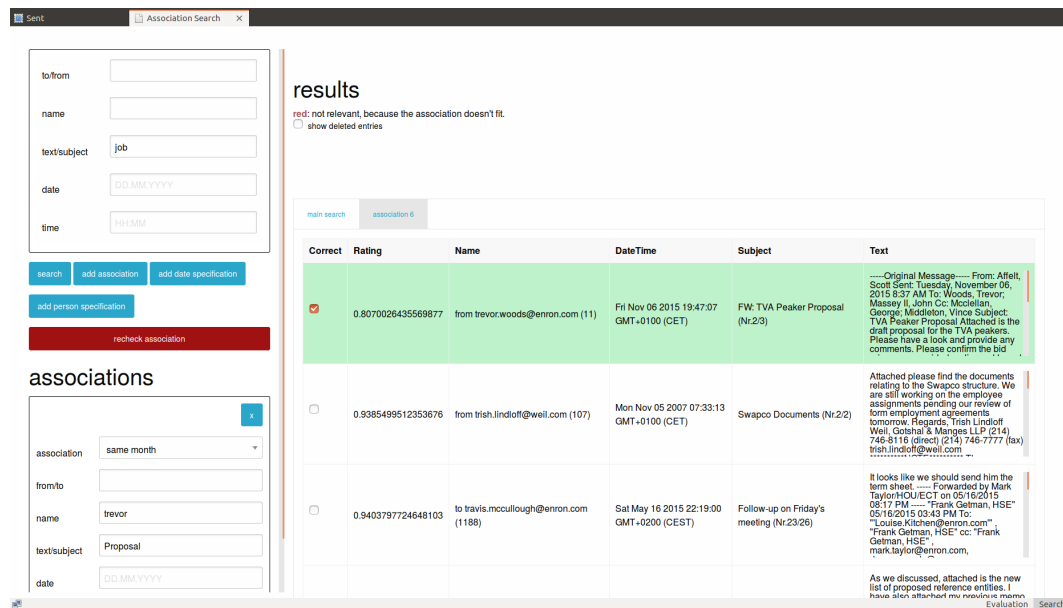


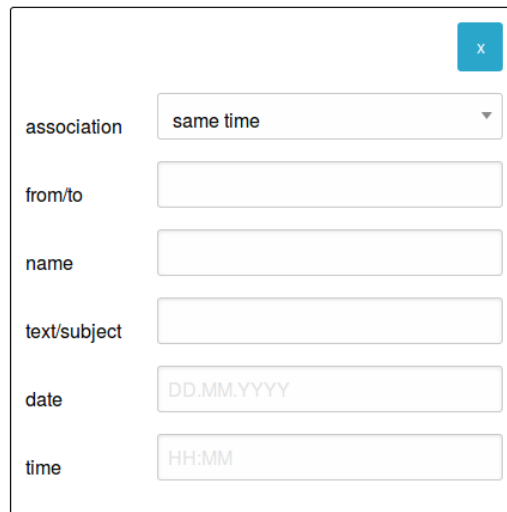
Figure 3.7: The figure shows a screenshot of the second tab, which contains the email association feature results. The green region marks the correct emails, which the user can update. As soon as the email selection changes, a red button "recheck association" appears and leads the user back to the main result list containing the new updates.

Date Specifications

As described in the 3.1.5 concept chapter, the date specification feature consists of three main parts: Public holidays, weekdays, and months. The calculation is similar to the email association above. The algorithm checks if an email is fulfilling the definition and then gets marked as deleted or kept. The weekdays, months and seasons can be stored in constant lists. The public holidays, on the other hand, have to be loaded from an API ⁵ because they can change over the years and some events like the Easter holidays are not on fixed dates every year.

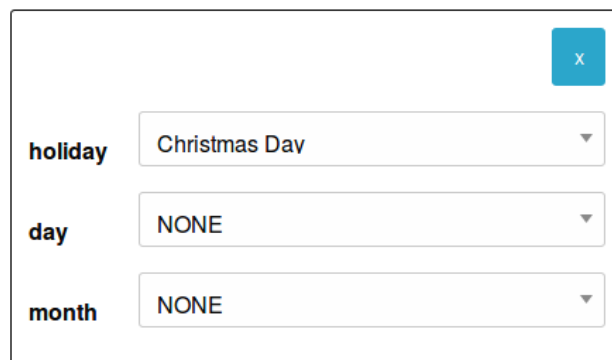
⁵holidayapi.com/ (Accessed on: 2020-09-02)

3 Methods



A screenshot of a web form titled "email association" in the top left corner. The form is enclosed in a light gray border and has a blue "X" button in the top right corner. It contains several input fields: a dropdown menu for "association" with "same time" selected, and text input fields for "from/to", "name", "text/subject", "date" (with a placeholder "DD.MM.YYYY"), and "time" (with a placeholder "HH:MM").

Figure 3.8: The figure shows the **email association** feature, where users can search for another email that is, in some way, related to the email they are searching for.



A screenshot of a web form titled "person specification" in the top left corner. The form is enclosed in a light gray border and has a blue "X" button in the top right corner. It contains three dropdown menus: "holiday" with "Christmas Day" selected, "day" with "NONE" selected, and "month" with "NONE" selected.

Figure 3.9: The image displays the **person specification** feature, which provides input fields for public holidays, days and months.

Person Specifications

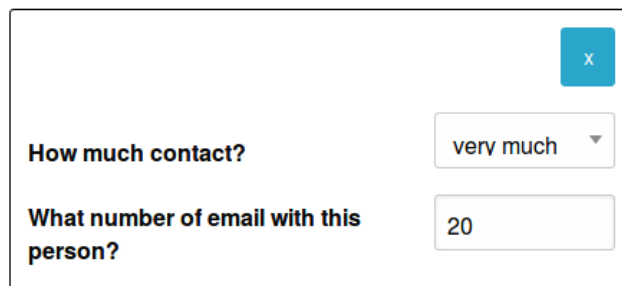
This feature's implementation varies from the others because it needs further information beyond the email itself. The first input field provides the option to enter how much exchange there has been with a person. Since it would be too computationally expensive to calculate these values at every search run,

3 Methods

a dictionary, with email addresses as keys and counter integers as values, gets filled up in the initialization step.

The second field is about which number of emails, with a specific person, it is. For implementing this feature, the email object has to have a second index, which only regards the person's emails. As shown in figure 3.4, this index is named "numOfPerson" and gets calculated during the data import.

These two inputs then get compared to every email's values, and again the relevance boolean gets modified if they match.



The figure shows a rectangular form with a thin black border. In the top right corner, there is a blue square button with a white 'x' icon. Below this, the form contains two rows of labels and input fields. The first row has the label 'How much contact?' on the left and a dropdown menu on the right showing 'very much' with a small downward arrow. The second row has the label 'What number of email with this person?' on the left and a text input field on the right containing the number '20'.

Figure 3.10: This figure displays the **person specifications** feature, which provides fields for defining how much contact the user had with a specific person and what number of email it is.

Multiple Associations

As described in 3.1.7, if more than one association feature is in use, there are three different outcomes for one email. If no association fits the input, it gets filtered out as usual. If just one of the association fits, the email gets displayed in the resulting list. And if two association feature inputs suit, the distance value gets halved. The same happens when a third or fourth association input is added. This is necessary to ensure these emails are ranked higher.

4 Evaluation

4.1 Methodology

With the shift of the main questions for this thesis, described in the limitations chapter, I started using the Enron email data set. Enron is a formerly large energy concern, which became insolvent in 2001. The dataset got later collected and preprocessed by several researchers. It includes about 500.000 emails from 150 users. [3]

I took one person called Mark Taylor from the set and created a Thunderbird profile for him. He was chosen because he has a large data set, which is important for providing a realistic evaluation basis. Then, all his emails got imported by using an addon called ImportExportTools NG¹. Since the Enron data set mainly consists of emails between 1999 and 2001, the years got randomized between 1999 and 2020 to have a wide date range for better testing.

The testing process started with asking the person about age, job, and currently used email search methods. Then I presented every feature step by step and gave them a few minutes to get familiar with the tool. Afterward, they had to complete four email re-finding tasks, consisting of a short text containing hints to different associative context data types. These short texts are adjusted to the email data of Mark Taylor. The evaluation ended with a final question about what feature they think is the most useful.

¹addons.thunderbird.net/de/thunderbird/addon/importexporttools-ng/ (Version: 4.1.0)

4 Evaluation

4.2 Results

Three men and four women participated in the evaluation. The table 4.1 shows all the gained information. The first columns present the basic facts to the person, with their currently used email clients and their ways of searching for older emails. The last two columns contain the actual results, consisting of the most useful feature and the average time needed for an evaluation task.

#	Job	Gender	Age	Current way of searching	Email Client	Most useful feature	average Time needed
1	banking employee	m	53	folder-based	Lotus Notes	email association	1.5 min
2	self employed	f	50	query-based	Thunderbird	date specification	2 min
3	bachelor student teacher education	f	22	query-based	Gmail	date specification	1.5 min
4	master student software engineering	m	24	query-based	Gmx	date specification	1 min
5	software engineer	m	23	query-based and labels	Thunderbird, Gmail	date specification	1 min
6	master student computer science	f	23	query-based	Outlook, Gmail, GMX	date specification	1 min
7	bachelor student midwifery	f	19	query-based and labels	Gmail, Outlook	date specification	1.5 min

Table 4.1: The table contains the results and basic information of the seven persons, who evaluated the search tool. They had to complete four tasks with emails from an test email data set.

4.3 Discussion

The goal was to find out if it is possible to use this search tool for further research evaluations. And the feedback was quite positive, the test persons quickly accepted the new search fields, and everyone was able to complete the given tasks. The IT background users completed the assignments a little faster, but the others also finished rather quickly.

4 Evaluation

As shown in the table 4.1 above, most users rated the date specification feature as the most useful. During the feedback conversation at the end, it was quite clear that most of the test users do not think it is rememberable to know how much contact you had with a specific person or what number of emails it is. One argument, which was mentioned several times, was that they might remember if it was just one email, but higher numbers are hard to separate.

The feature that remains in the middle is the email association feature. Especially the idea that they wrote emails to another topic or person during the same period got evaluated as very useful.

One last mentionable topic, which came up a few times, is the usability and the wording. Mainly persons without IT background sometimes had minor issues with the simple design. For instance, the search button does not stand out from the association buttons. Or the field named "from/to", which defines whether you wrote or got the email, was twice wrongly used as the name field. The fact that every person, except for one, is a German native speaker probably also impacted the outcome here in some cases.

4.4 Limitations

There were multiple different approaches for testing this tool. At first, the question I wanted to explore was to find out if it is beneficial to use the associative memory for re-finding emails. So the evaluation would have needed real historical email data of the testing persons. The three different ideas were, we track all searching operations of a person for one week to test the same searching tasks with the new tool. The other way would have been to obtain access to a person's email history and create searching tasks based on their data. And the third one was randomly generating tasks based on their emails, but without a human ever reading them.

These first two methods of testing appeared to come with many complications since this would be a matter of data privacy and despite the fact that test persons are harder to find, the effort for individually testing each person would have been a task for more than one researcher. On the other

4 Evaluation

hand, the third method turned out to be not realizable since it is necessary to generate actual tasks that make sense to the user.

5 Conclusions

The Bachelor thesis treated how the human associative memory can be used in combination with the information retrieval problem of email re-finding. In detail, this includes the development of an addon for the email client Mozilla Thunderbird and evaluation with several test persons to check if the coded tool is capable of being used for other research purposes and which of the implemented features has the most potential.

The new search tool consists of standard search options, like name, topic, date, etc., and three additional features based on the associative memory. The first one, named email association, provides the possibility to search for another email, to which the user, for instance, knows that he had email contact during the same time. The second feature is named person specification. Here the user can add information he knows about the person he is searching for, like how much email traffic he had with this individual. The last one, date specification, consists of input fields for public holidays, weekdays, months, and season. This can be used if users know details to the date or know it was sent or received around a holiday.

The evaluation consisted of several tasks, where the users had to use the new search features. This testing has shown that the tool is quite intuitive, and even test persons without IT background completed the given tasks in a short time. The most useful feature appeared to be the date specification. Additionally, the email association also was positively mentioned multiple times, especially the idea to know if the user has written emails with somebody else during the same period.

6 Future Work

This Bachelor thesis potentially serves as a starting point for further research. The tool provided here can be used for a large-scale evaluation to test if using associative memory positively impacts email re-finding. To ensure a scientifically acceptable answering of this question, it is necessary to work with the person's real email data of several years. And create practical tasks for each user to test if they re-find emails easier with the new tool or the common ways of searching.

Furthermore, some points could be added or changed, like the fact that the tool currently has a relatively long initialization time. This is mainly caused by the JavaScript library LunnrJs. Besides that, one can think of changing the whole loading process since currently all the emails get loaded and stored into email objects, shown in 3.4, at the start of the searching tab. This works fine for a few thousand emails, but it is too much with having more than ten thousand. It is possible to store the information in a file or database for later re-use to solve this problem. Another possible solution would be to load less information and use Thunderbird's searching system named Gloda¹.

An additional improvement for a larger study would be to add more fields of associations and other search features. As described in the background chapter, a large number of people use folders. Thunderbird also provides folder structures, and it would be simple to add another combo box to select the folder. Since the evaluation has shown that people tend to remember parameters related to dates and periods, an additional point would be a connection to the personal calendar. More precisely, to provide the option to filter by meetings and events in the test person's life.

¹developer.mozilla.org/en-US/docs/Mozilla/Thunderbird/gloda (Accessed on: 2020-09-08)

7 Acknowledgements

I wish to express my sincere thanks to Roman Kern, my advisor, for giving me the chance to write my Bachelor thesis about this topic and for sharing expertise as well as providing support during several stages of the project.

Furthermore, I take this opportunity to express gratitude to all persons who spend their time participating in the evaluation.

Bibliography

- [1] Brent W Benson Jr. 'Javascript'. In: *ACM SIGPLAN Notices* 34.4 (1999), pp. 25–27.
- [2] Yi Chen and Gareth JF Jones. 'Are episodic context features helpful for refinding tasks? lessons learnt from a case study with lifelogs'. In: *Proceedings of the 5th Information Interaction in Context Symposium*. 2014, pp. 76–85.
- [3] William W. Cohen. *Enron Email Dataset*. 2015. URL: www.cs.cmu.edu/~wcohen/enron/.
- [4] David Elweiler, Mark Baillie and Ian Ruthven. 'Exploring memory in email refinding'. In: *ACM Transactions on Information Systems (TOIS)* 26.4 (2008), pp. 1–36.
- [5] Sudheendra Hangal, Monica S Lam and Jeffrey Heer. 'Muse: Reviving memories using email archives'. In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 2011, pp. 75–84.
- [6] Jeremy Keith. 'A Brief History of JavaScript'. In: *DOM Scripting: Web Design with JavaScript and the Document Object Model* (2005), pp. 3–10.
- [7] Liadh Kelly et al. 'A study of remembered context for information access from personal digital archives'. In: *Proceedings of the second international symposium on Information interaction in context*. 2008, pp. 44–50.
- [8] Mozilla. *Building a Thunderbird extension*. 2019. URL: developer.mozilla.org/en-US/docs/Mozilla/Thunderbird/Thunderbird_extensions/Building_a_Thunderbird_extension.
- [9] Steve Whittaker et al. 'Am I wasting my time organizing email? A study of email refinding'. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2011, pp. 3449–3458.

8 Appendix

8.1 GUI designing with Zurb Foundation

The decision to use the Zurb Foundation¹ framework fell quite early during the development because I have been using it for many years, especially during my time working as a web developer. It provides many content elements such as a tab view, used for the association search results displayed in 3.7, or a practical grid system. Furthermore, Foundation has a good looking basic design for form fields, tables, and other fundamental HTML elements.²

```
88
89 <body>
90   <div class="grid-x grid-padding-x">
91     <div id="main" class="columns large-3 cell">
141     <div id="content" class="columns large-9 cell">
171   </div>
172 </body>
173
```

Figure 8.1: This figure shows the basic structure of the search tool, implemented with the grid provided by Foundation. The container with the id named "main" contains the input fields and has a width of three out of twelve columns of the grid. The other box called "content" has a width of nine columns and contains the resulting table.

¹<https://get.foundation/> (Version: 6.4.2)

²<https://get.foundation/sites/docs> (Accessed on: 2020-09-19)

8.2 Source code

8.2.1 Email Association

The email association described in chapter 3.1.4, has the purpose of filtering the emails by a particular term. For instance, the piece of code presented in 8.2 shows the process of the "same month" type. Since there can be multiple mails selected as correct second email, it is needed to collect all the months occurring in those and store them in an array. After that, there are two nested loops to iterate over the months and all the results from the basic search. Inside the inner loop, the month of every email gets compared to the wanted months, and the "deleted"-boolean, stored on the second place of the item, either turns true or false.

```
394 case TYPE.MONTH:
395     var months = [];
396     this.associationArray.forEach(function (item) {
397         let month = item[1].date.getMonth();
398
399         if (!months.includes(month))
400             months.push(month);
401     });
402
403     months.forEach(function (month) {
404
405         main_results.forEach(function (item) {
406             if (item[2]) {
407                 item[2] = !(item[1].date.getMonth() == month);
408             }
409         });
410     });
411 });
412 break;
```

Figure 8.2: This figure shows how the email association feature filters emails by the same month. At first, the valid months get collected, then the main results get filtered by them.

8 Appendix

8.2.2 Euclidean distance calculation

The distances between the input vector and the email object vectors get calculated by the Euclidean distance. Figure 8.3 shows how this calculation is implemented and how the resulting list gets returned.

```
264 calculateDistances(vector, weight) {  
265   var results = []; //distance + email object  
266  
267   //euclidean distance  
268   this.vectors.forEach(function (item, index) {  
269     var res = 0;  
270  
271     for (var i = 0; i < vector.length; i++) {  
272       if (weight[i] > 0) {  
273         res = res + Math.pow(item[i] - vector[i], 2) // Summe -> Differenz2  
274       }  
275     }  
276     res = Math.sqrt(res);  
277     results.push([res, findEmailById(index, list), true, false]);  
278   });  
279  
280   return results.sort();  
281 }  
282 };
```

Figure 8.3: The figure displays the code for the Euclidean distance calculation. At first, I iterate over every email vector. Inside, I square every difference between the vector's items and store the sum's square root into a new object of the resulting list. In the end, the list gets sorted by difference and returned.

8.2.3 Generation of content elements

A big part of the search tool are dynamic content elements like new association search boxes, tables or, new tabs. These elements get generated using JavaScript. At first the container gets determined by the id and then new elements can be added to HTML code. The example below is a rather simple one, it shows the creation of an email association search box, displayed in the screenshot 3.8.

8 Appendix



Figure 8.4: The left code box contains the JavaScript code needed to generate the search box for email associations. The left box contains the resulting HTML code

8.3 Evaluation examples

The hands-on part of the evaluation consists of four tasks the user has to master. Two of them are presented in figure 8.5. The descriptions focus on the new search tool's primary use cases and should provide a check if users get along with the new fields.

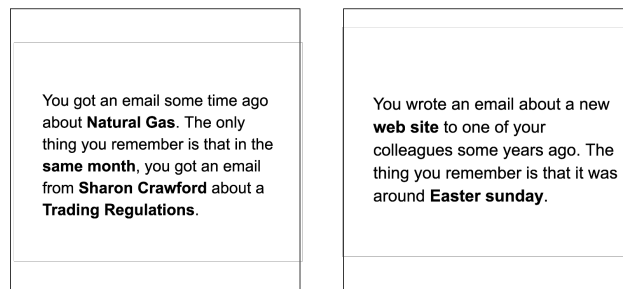


Figure 8.5: The figure shows two tasks used for the evaluation.