Reconstructing families - Bachelor Thesis

Wouter Joosse
4158407

Contents

Chapter 1

Introduction

Since the cost of data storage has significantly decreased over the last twenty years, modern computer software are more focused on data driven models, where most of this data is related to real persons. Companies, for example, try to create customer profiles in order to predict the needs of their customers by generating recommender systems, governments create profiles of people they think are of interest and health organization keep track of the history of their patients. This data is often stored in data warehouses, where each piece of data is stored in a structured way. Although data architecture practices describe some form of data normalization, most data warehouses are not necessarily compatible with each other and therefor most data concerning the same entity is stored in slightly different ways in multiple data warehouses.

Because of technological progress, the way of data analysis has changed rapidly over the last few years. Data sets are larger, computers contain more calculating power, and the time that it takes to request data and send it to others has has been decreased to nanoseconds. More than ever, analysis of data often requires that different data sources should be joined in order to create more enriched data sets and to discover new links between data.
In order to link data sources with each other these data sources should be able to identify which entities in one date source can be mapped to the same entity in the other data source. This is often where problems arise. Data is not saved according to a universal standard and because of this, each source could save the same information in a different format, making it harder to link this data. For example, dates could be written according to different formats or persons could be mentioned with their full name or with only their initials. Since these idiosyncracies exists, it is still not a trivial task to determine which entities relate to each other.

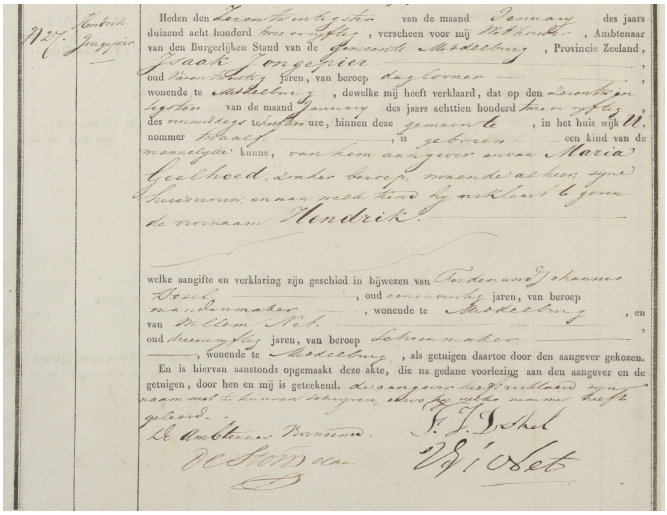
The problem of combining records from different data sources and finding data that are about the same entity is called *Record Linkage* and has been researched extensively in the last few decades. The term *record linkage* was introduced by Dunn [Dunn, 1946], who wanted to assemble a 'book of life' for each person which would describe the person's interaction with health and social security systems, and which would also contain the birth-, death- and marriage-certificates of that person.

1.1 The genlias project

Throughout history, governments held census to identify civilians and keep records about them, in order to register taxations and the people that are allowed to vote, for example. So did the government of The Netherlands in the 18^{th} and 19^{th} century. During that time, The Netherlands was occupied by the Frence empire of Napolean Bonaparte. The French introduced the 'Burgerlijke stand', which was responsible for recording births, marriages and deaths of the people of The Netherlands. The *genlias* project is a project that has started in the last decade of the 20^{th} century with the aim to collect all the certificates that where produced into a single database. This database is now available via `http://www.wiewaswie.nl`.

The dataset gives us great insight into the population of the Netherlands in the 19^{th} century. However, unlike more modern registration practices, the records where not provided with unique identification numbers, which makes it harder to do research. Another complicating factor is that it was not uncommon

Figure 1.1: An example of a birth-certificate. Source: Zeeuws Archief, http://www.archieven.nl



for officials to write different names for the same person on different registrations. For instance, when people married, the persons involved would be registered with the official names. However, when a person passed away, it could happen that the neighbors had to report this event to the civil registration. These neighbors didn't always know the full official name of the person deceased. This is where record linkage could be applied, in order to provide the dataset with unique identifiers per person.

1.2 How to match certificates

The subject of this bachelor thesis are the problems that can arise when one tries to create families out of these certificates when looking at the names that are present on these certificates. In [Schraagen, 2014] a method was proposed in order to link registration certificates of these life events from the *genlias* project. Since the persons in this data set are not provided with an unique id, matching of records is done by matching the names that appear on the certificates. This approach groups the certificates where the names of the persons that are similar to each other in an efficient way. Matches are found by calculating the edit distance between the names that occur on both certificates.

My aim is to assess the quality of the linkages made by this approach and to refine the method that compares two certificates with each other. I will try to find rules that could improve the linkage procedure when trying to link historical names, by taking the matches that are made by the method and apply the rules in order to check the matched registrations for consistency.

The linkage of registrations is done in two phases. At first, for each marriage certificate a set of potential matching certificates is build by matching the names of the bride and groom on each target certificate to the names of the parents on the candidate certificates, using the method that was described by [Schraagen, 2014]. For instance: in the case of marriage registrations, both parent couples are checked, but in order to be considered a match, only one of the couples should match with the target certificate. The (levenshtein) edit distance can be used to regulate the extend in which the algorithm will overmatch or undermatch.
The resulting matches are certificates containing two pairs of four names: the first and last name of a wedded couple on their marriage certificate and the certificate to which is was matched. These four names are compared to there respective counterpart on the other certificate. The second phase is to filter out the matches where these names differ too much.

Chapter 2

Data

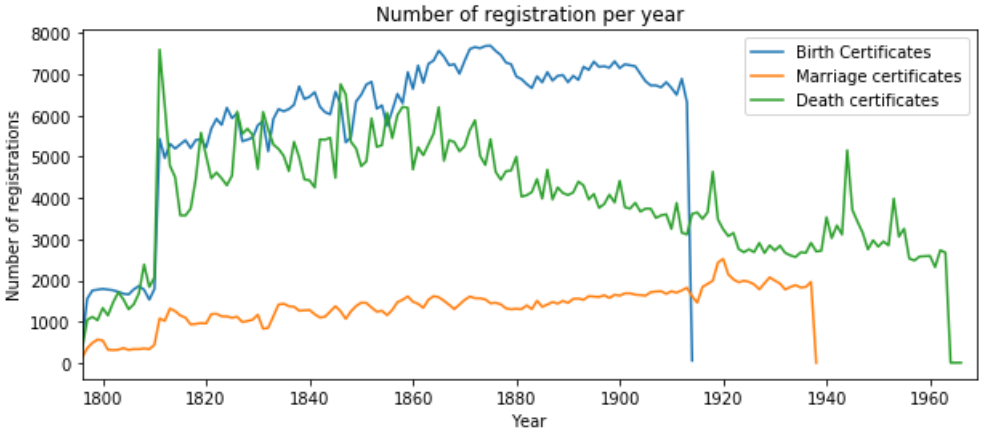
2.1 Description of the records

The dataset we use is provided by the Genlias project and consists of all historical population certificates from the province of Zeeland. The dataset contains 1.558.205 distinct registrations, which are split out into 698.285 birth-certificates, 193.921 marriage-certificates (this type also includes some divorce-certificates) and 665.999 death-certificates. Due to privacy regulations, birth certificates are available until 1913, marriage registrations are available until 1938 and death registrations are available until 1963.

Figure 2.1 shows the number of certificates per year. A sharp increase in the number of registrations can be seen by the year 1811. This is due to the fact that before 1811, only in the southern parts of the province of Zeeland population records were kept. In 1811, when The Netherlands was part of the French empire, the northern parts of The Netherlands started to make civil records.

The original documents consists of handwritten cards on which an official filled in the specific details of the event. In earlier versions, the entire document were completely handwritten. Later on, these cards became pre-printed cards where only the personal details and details of the event needed to be filled. An example of a preprinted birth-certificate can be seen in figure 1.1.

Figure 2.1: The number of registrations per year, split by certificate-type. Birth-certificates are available until 1913, marriage-certificates until 1938 and death-certificates until 1963



The digitalized certificates are stored across three different tables in the database:

1. Persons, which contains data concerning the people mentioned on a registration,
2. Registrations, containing meta-data (such as dates of the registration and the location id),
3. Locations, containing a mapping from location id to names of locations

The Persons table contains records for each person mentioned in a certificate. The number of persons mentioned on a certificate depends on the type of certificate. On birth-certificates, there are three persons mentioned: ego, the mother and the father (this is not always the case), on marriage-certificates, there are six persons mentioned: the bride and groom, and the mothers and fathers of the bride and groom and on death-certificates at least the deceased and his/her parents are mentioned. In the case that the deceased had a partner, this partner is also mentioned, but this is not necessarily the case. Each role that a person has is also specified (e.g. bride, groom, father of the bride, mother of the groom, etc). The name of each person is split into the first name, surname and, if applicable, a prefix.

In table 2.1, an example of how a single birth-certificate is stored is shown. For brevity, not all possible values are displayed. For each person that was born, at least the day, month and year of the birth has been recorded. The same pattern is also valid for both marriage- and death-certificates, but then the date is stored in mar_day, mar_month, mar_year and their respective death-counterparts.

Other information that is also recorded in the database include the place of the event and the date. We also added the latitude and longitude of the places to the dataset.By combining the records on the id of the registration, you can create a single record of the entire certificate.

Table 2.1: An overview of the fields in a birth certificate

id_person	id_registration	registration_maintype	firstnames	prefix	familyname	sex	civil_status	birth_day	birth_month	birth_year
15405	5136	1	maria cornelia	van	oorsel	f	1	17	9	1864
15406	5136	1	willem hendrik	van	oorsel	m	3			
15407	5136	1	neeltje johanna		christiaanse	f	2			

Table 2.2: The number of registrations per set

	Number of registrations
Target set	193,040
Birth certificates	698,199
Marriage certificates	386,080
Death certificates	665,903

2.2 Preprocessing

2.2.1 Initial Matching

The first step is to extract the data that is needed to match the certificates using the method used in [Schraagen, 2014]. Since the names of parents are present on each of the certificates, we can extract these names as strings to match.

The set of certificates are separated, based on the nature of the certificates, into three sets: a set of marriage certificates, a set of birth certificates and a set of death certificates. For each certificate, the first name and family name of the parents are extracted and stored with the certificate id. When multiple names are present for a specific type of name, only the first name is used. For instance: in the case of the first name *bernardus franciscus*, only the name *bernardus* will be used. In the case of marriage certificates, two pairs of parents are present on the certificate (for the bride and groom). These two pairs are both extracted using the role descriptor in the Person table. The pairs are stored on seperate lines in the file used for matching.
The extracted parents are stored in a single file, where each name is separated using a separation token. This way, we can identify the four names (the first name and last name of each parent). The resulting sets of names will be used as the candidate sets.

The target set consists of the names of the bride and groom on the marriage certificates, and these names are also stored in a separate file, in the same format. This way, we can couple the birth, marriage and death certificates to the marriage event of the parents and create sets of families: the life events of the children are coupled to the marriage events of the parents. Table 2.2 shows the number of resulting registrations per set.

Figure 2.2: The spread of the length of the names, split by type and sex.

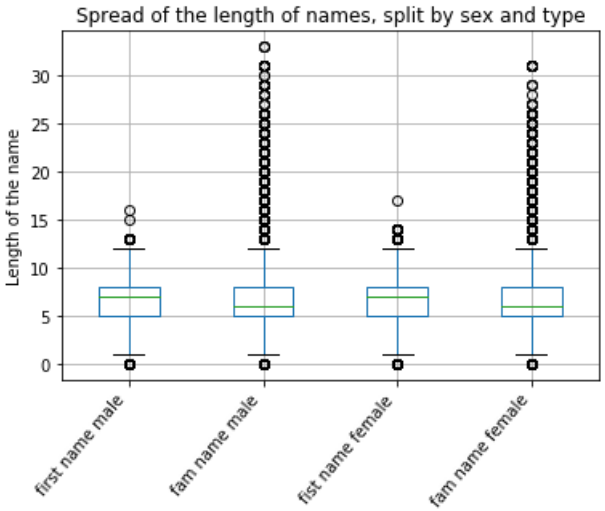


Figure 2.2 shows the spread of the name length per sex and split per name type (first name and surname) over the all candidate registrations. This shows that for the familyname the number of outliers

is far greater than for the first names, and that the spread of the length of the familyname is greater than the spread of the first names. These outliers could pose a problem when increasing the maximum Levenshtein distance, since these could be match more easily to shorter names with a higher threshold. Table 2.3 shows the average length of the name for each group.

Table 2.3: The average name length per group

	Average name length
First name - Male	6.09
Family name - Male	6.26
First name - Female	6.79
Family name - Female	6.47

Chapter 3

Method

As mentioned in the introduction, life courses are build in three phases: the matching of individual registrations based on the names of the persons that are mentioned on the registrations, creating sets of registrations that are linked to registrations of the marriages of the parents mentioned in the these registrations. These sets are referred to as 'families'. In the second step, the results from the first step are checked for complience on extra constraints, in order to filter out those matches that are accepted in the first step, but where the (allowed) error is too concentrated in a single name. The third step is to check if these families are consistent with a set of constrains, filtering out those that are not.
The accuracy of the matching in the first phase will have a large impact on the size of the families. Since matching in the first phase is done on the Levenshtein distance, the maximum allowed distance (threshold) will determine the extend to which overlinking (or underlinking) will occur. Overlinking might not be considered a bad thing since the results in the first phase will be filtered in the second phase. Although overlinking will be computational more intensive in the second and third phase, underlinking could result in true matches being missed. Matches that have been missed in the first phase can not be retrieved in the second phase.

3.1 Initial matching

The initial matching is done by using the approach as described in [Schraagen, 2014]. This approach is used to create the set of families in an efficient way, in order to avoid matching all the candidate registrations to each of the target registrations. The approach uses a tree sorting algorithm to filter out the registrations that, given some maximum Levenshtein distance, are not to be considered as a match and mapping the target registrations to the candidate registrations that are considered as viable matches. The registrations that are left after filtering are compared to the target registration using the Levenshtein distance. The target registrations are used to build the tree. For each candidate registration, a tree traversal is done. All the target registrations of the vectors in leave nodes that can be reached with a certain edit distance are considered a potential match for the candidate registration. As a result from the tree traversal, each candidate registration is mapped to one or more target registrations, and each target registration has a set of candidate registrations. Two registrations are considered a match when the Levenshtein distance is below the predefined threshold.

Our approach is to use the registrations of the children (that is: birth, marriage and death certificates) as the target certificates, and the certificates of the marriages of the parents as the candidate certificates.

3.1.1 Edit distance

As mentioned above, the method that is used to compare a candidate record to a target record is the Levenshtein distance, which is a function to calculate the distance between two strings. The method was introduced by [Levenshtein, 1966]. Given two strings a and b, the distance between a and b can be defined as the number of operations that need to be executed in order to convert string a into string b (or b into a, since the operations are symmetric).
The original Levenshtein function assumes three operations: deletion, insertion and substitution of characters in a string, where each operations has a cost of 1. This results in an Levenshtein distance that represents the number of operations. Several variations of the Levenshtein distance exists, where either

Table 3.1: The Levenshtein edit distance for the strings. The bold digits represent the path to the final result. *Peter* and *Pieter*

		0	1	2	3	4	5
		␣	P	e	t	e	r
0	␣	**0**	1	2	3	4	5
1	P	1	**0**	1	2	3	4
2	i	2	1	**1**	2	3	4
3	e	3	2	**1**	2	3	4
4	t	4	3	2	**1**	2	3
5	e	5	4	3	2	**1**	2
6	r	6	5	4	3	2	**1**

the cost associated with an operation varies, or the set of operations is altered to allow different operations (such as swapping to characters, also known as transposition) or to disallow certain operations. In our approach, we use the original Levenshtein distance.

The Levenshtein distance is calculated in $O(|a| \times |b|)$ time, where $|\cdot|$ denotes the length of a string. This is done by using a dynamic programming algorithm, that uses a matrix with $|a| + 1$ columns and $|b| + 1$ rows. Each letter in a is associated with a column, and the first column is used for an empty space. An example can be seen in table 3.1, where the strings *Peter* and *Pieter* are compared to each other. In order to calculate the Levenshtein distance, the value 0 is assigned to the cell $d[0, 0]$ (corresponding to the empty spaces). Then, in a recursive manner, the remaining cells are filled according to the following function:

$$d[i,j] = \begin{cases} d[i-1,j-1] & \text{if } a[i] = b[j] \\ minimum \begin{cases} d[i-1,j] & \text{(deletion)} \\ d[i,j-1] & \text{(insertion)} \\ d[i-1,j-1]+1 & \text{(substitution)} \end{cases} & \text{if } a[i] \neq b[j] \end{cases} \tag{3.1}$$

with $0 \leq i \leq |a|$ and $0 \leq j \leq |b|$. The resulting value in cell $d[|a|, |b|]$ is the final value for the Levenshtein distance between string a and string b.

3.1.2 Building the vectortree

Each name on the target registrations will be transformed into a bit vector of eight bits. Each position in the bit vector represents a set of letters, as shown in table 3.2. For each position in the bit vector, the letters corresponding to that position will be searched in the name. If at least one of those letters is found in the name, the position in the bit vector will be true, otherwise that position will be false.

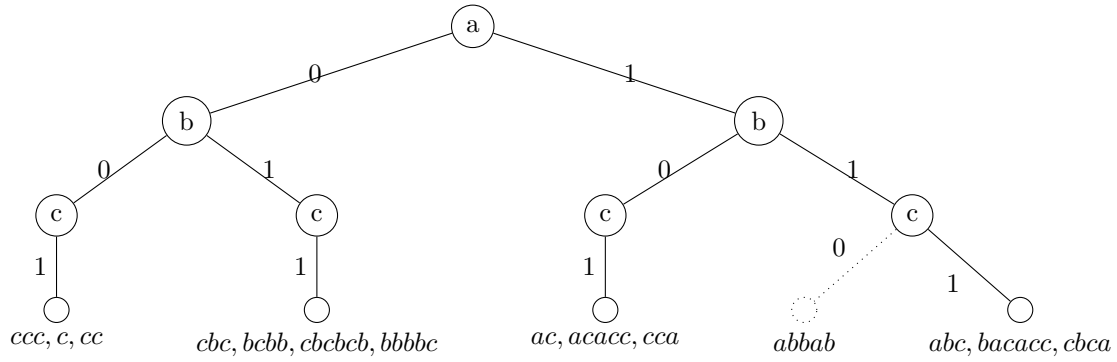
In the case of the target registrations, the resulting bit vector will be used to build the vectortree. The tree starts with a single node, representing the first position in the bit vector. Nodes and edges are added when iterating over the bit vector, by checking for each position in the bitvector if there is an edge from the previous node corresponding with the previous position in the bitvector to a node that correspond to the value of the current position. If this is not the case, a new child node and edge are inserted in the tree. If such edge and node exists, this edge is followed and the next position is evaluated. When the last position of the bit vector is evaluated, the certificate is added to the leave node. This leaf node contains all the certificates with the same bit vector, generating a set of potential target certificates for a candidate certificate.

In order to make this procedure more clear, consider the following example. We are building a vectortree for words that consists of the alphabet a, b, c, and the bit vectors are of length 3, encoding occurrences of these three letters. The word *aaa* will therefore be converted to $\langle 1, 0, 0 \rangle$ and the word *abc* will be converted to $\langle 1, 1, 1 \rangle$.
In figure 3.1 a partial tree is shown. We would like to add the word *abbab* to the tree. The bit vector

Table 3.2: The positions in the bit vector represent certain letters. For each name, eight positions are available and if a letter occurs in the name, the corresponding bit is 'activated'. For example: using the sets of letters in table 3.2 the name *Jobse* will be converted to the vector $\langle 1, 1, 0, 0, 0, 0, 1, 1\rangle$

letters	position
$\{e, g\}$	0
$\{a, l, q, h, j, x, comma\}$	1
$\{r, p, v\}$	2
$\{n, space\}$	3
$\{i, u, w\}$	4
$\{d, f, c, m, z\}$	5
$\{t, s, y\}$	6
$\{o, b, k, other\}$	7

Figure 3.1: A (partial) vectortree when build on words from the alphabet a, b, c. Adding the word *abbab* will add the dotted edge and node to the tree.



will be $\langle 1, 1, 0\rangle$. Since the first position in the bit vector has a value of 1, the edge with label 1 will be taken in the first node. The same holds for the second position. However, for the third position a new edge and node will be added to the three, since none of the previous words have created this edge.

3.1.3 Matching the candidates

The bit vectors of the candidate set are matched by traversing the vectortree, based on the values of the bit vector generated for that candidate. If a path to a leaf node exists, all the target registrations in the leaf node are considered a match, and if no such leaf node can be reached, the candidate is discarded. Tree-traversal is done by exploring all the possible branches in the tree, but a branch is pruned if the difference between the current path and path corresponding to the bit vector (that is, all the paths which labels correspond to the values of the bits in the bit vector) exceeds some predefined threshold.

Since the matches are based on the values of the bit vectors, the strings of the matches do not necessarily need to be exactly the same. Therefor, the Levenshtein distance is used to calculate the exact distance. However, since the bit vector groups certain letters into a single group, potential matches that are within the Levenshtein distance could be missed when only taking the leaf node on the direct path into account. For example, when matches with an edit distance of 1, the legitimate match between the names *Zegers* and *Segers* will not be in the same leaf node and therefor are not considered to be a match when only taking the matches into account that are in the same leaf node. For example: the bitvector of the names *Zegers* and *Segers* are $\langle 1, 0, 1, 0, 0, 1, 0, 0\rangle$ and $\langle 1, 0, 1, 0, 0, 0, 1, 0\rangle$, respectively. Upto bit 5, these vectors are completely the same, and therefor follow the same path.

In order to compensate for this, all the paths that are accessible within the maximum error are also considered, and the registrations in the corresponding leaf nodes are added to the set of potential matches.

Table 3.3: The set of extra rules when matching is done on name pairs. The rules are more strict when the Levenshtein distance increases.

Levenshtein distance	length	length matching prefix
1	shortest > 4	1
2	shortest > 4	2
3	longest > 5	3
4	longest > 7	4
5	longest > 8	4
total length of pair minus Levenshtein distance > 16		1

3.2 Increasing the edit distance

3.2.1 Name pair matching

In the method described above, the edit distance is calculated over the entire string of all names. This could lead to situations where the maximum edit distance is exceeded and the match is rejected, but the match should have been accepted. There are many names in the Dutch language that have alternative writing styles. Examples are first names like *Cornelis / Kornelis* and *Lourens / Laurens* or surnames like *Huizen / Huijzen* or *Belsen / Belzen*. The similarity between these names variants is that the variants share the same semi-phonetic form, although the spelling is different. This adds at least a cost of 1 to the (levenshtein) edit distance, where there is a good reason to ignore these kinds of name variants.

In [Bloothooft and Schraagen, 2014] a set of decision rules was proposed to accept matches based on the edit distance per name. Matching was done not only based on edit distance, but also on the additional constraint that matches must share the same prefix. This was done in order to accept matches for name pairs with higher Levenshtein distances. The constraint of the length of the shared prefix is determined by the edit distance between the two matches, as shown in table 3.3. The constraint on the length demands that, for a given length, either the shortest or the longest name in the name pair is of a certain length, and the constraint on the prefix demands that both names start with the same sequence of characters. There is also an additional rule, demanding that if the the length of the concatenation of both names minus the edit distance is greater than 16 and the names start with the same letter, the match is also accepted.

3.2.2 Allowing more distance

Although these rules are originally applied to name pairs for which the names have been converted into semi-phonetical form, we can use these rules to see if any matches that were rejected when matching on the entire string can be accepted when looking at the distribution of the Levenshtein distance over the four names in the strings. We will focus on the matches with Levenshtein distance 4 and 5. where we accept matches with an Levenshtein distance of 4 or 5 only if all of the names of these matches are accepted by the rules in table 3.3. These rules will give some certainty about the extent to which the names match and therefor can be used to make a more informed decision about the gravity of the mismatch of the strings. When the strings pass the tests in table 3.3 there is at least some overlap in the name pair, and the names are of sufficient length to make sure that the edit distance is in some proportion to the length of the names.

Figure 3.2 shows the distribution of the length of the names in the matches generated by the set of marriage candidates, categorized by the sort of name (first name, surname) and the sex of the person, which shows that for most categories, 75% of the names have a length of 8 or less. This means that, when looking at matches with Levenshtein distance 4 and 5, the constraint on the length of the matching prefix is a sensible constraint in order to make sure that the error is in proportion to the length of the name. In order to check this, we have created two sets of matches for the matches of the birth-certificates: a set where this constraint was dropped and a set where this constraint was applied. By comparing the number of matches that are dropped when the filtering is applied as described in the next section, the effectiveness of the constraint on the prefix can be tested.

When looking at the matches with a Levenshtein distance of 4 or 5, the distribution of the error over the names is also a good factor to take into consideration. An error of 4 or 5 in a single name could mean

Figure 3.2: The distribution of the length of names on marriage registrations

in some cases that more than half of the entire name of the candidate is different from the name of the target. Although the other three names do match completely (since the error is concentrated in a single name), the difference in that single name will render the match dubious. We will not take the type of name the error is in into account, i.e. we will treat an error of 3 in the first name of a male the same as an error of 3 in the last name of a female. We limit the matches we take from matches with a distance of 4 and 5 to those where the error is distributed over more than 1 name. For matches with a distance of 5, we will also ignore the matches which we have labeled '2,3' since these matches differ too much in two names.

3.3 Assessment of birth certificate matches

In the first phase, sets of potential families have been build, where all matches with a Levenshtein distance of 3 or lower have been accepted, and all matches with a Levenshtein distance of 4 or 5 have been accepted if the error was sufficiently distributed over the four names, as described above. All matches to a certain target certificate can be seen as 'children' from the pair on the target certificate, since the names of the ego and ega of the target certificates are matched to the names of the parents on the candidate registrations. The next step is to assess the quality of the matching in the first step. Although the assessment of the quality of the matches can be done on any arbitrary matching, we will now only assess the quality of the matches between the marriage certificates and the birth certificates. The set of matches of the birth certificates to the marriage certificates of the parents are the most important set of matches to check for consistency, since these matches are directly related to the existence of a person: a person cannot exist if it is not born. Every person that has been registered must have been born, and all other events in the life of that person can be linked to the birth registration.

There are four checks that can be carried out:

- Are the birth events after the marriage of the parents?
  Although children can be born before the parents are married, we do not consider these cases. Assuming that all birth events take place after the parents are married simplifies this check significantly.
- Are there at least 10 months between two consecutive birth events within the same family?
  It is very unlikely that children will be born within 10 months from each other, since this is biologically almost an impossibility.
- How long is the overall time span of birth events within the same family?
  If there is a gap of 20 years between two consecutive certificates, the likelihood that those certificate belong to the same family is smaller than when there is a gap of 5 years.
- How long is the time span between the marriage of the parents and the birth certificate? The time span between the marriage and the birth events can not be greater than 25 years. Lowering this

range might make this check more discriminative between families of parents and children with the same name, but it might also pose a problem when looking at large families.

In order to be accepted as a true match, the match must be compliant with all these four criteria.

Chapter 4

Results and discussion

4.1 Initial matching results

4.1.1 Vectortree matching

Table 4.1 shows the number of generated matches using the vector tree, per type of match and per maximum edit distance, as well as the number of extra matches that are generated when using a higher threshold compared to the number of matches with a threshold of 3. In total there are 130,298 extra matches with an edit distance of 4, and 317,311 extra matches with an edit distance of 5, of which 317,287 were made with an edit distance of 5, and 24 with an edit distance of 4. The extra matches with edit distance 4 is due to the extra leaf nodes a registration can end up in when traversing the vector tree, due to the extra registrations with edit distance 5.

Table 4.1: The number of matches per maximum edit distance

	Number of matches			Extra number of matches	
Certificate type	th = 3	th = 4	th = 5	th = 4	th = 5
marriage v.s. birth	556,549	610,647	741,566	54,098	185,017
marriage v.s. marriage	247,476	276,097	348,060	28,621	100,584
marriage v.s. death	394,252	441,831	556,260	47,579	162,008
			Total extra matches:	130,298	447,609

Table 4.2: The degree of overlinking per maximum edit distance. The overlinking is expressed as the number of parents linked per (unique) target certificate. The target certificates are the birth, marriage and death certificates of the children.

	Number of targets			Average match per target		
Certificate type	th = 3	th = 4	th = 5	th = 3	th = 4	th = 5
marriage v.s. birth	111,880	116,023	122,674	4.97	5.26	6.05
marriage v.s. marriage	81,742	86,722	95,142	3.03	3.18	3.66
marriage v.s. death	104,717	109,901	117,952	3.76	4.02	4.72

Figure 4.1 shows the distribution of the error over the names. The edit distance per word is used as the labels for a match: if there are 4 names where the edit distance between the names on the target registration and the candidate registration is 1, the label '1,1,1,1' was assigned to that match, and if there is a match a name with an edit distance of 3 and a name with an edit distance of 1, the label '1,3' is assigned to that match. The order in which the errors occur is not taken into account.
As can be seen from this figure, most of the extra matches are matches that we ignored in our second phase, since the distribution of the error is concentrated in a single name: 62% of all matches with label

Figure 4.1: The distribution of the maximum allowed error over the names, grouped by error per name over all the matches generated in the initial matching using the vectortree algorithm.

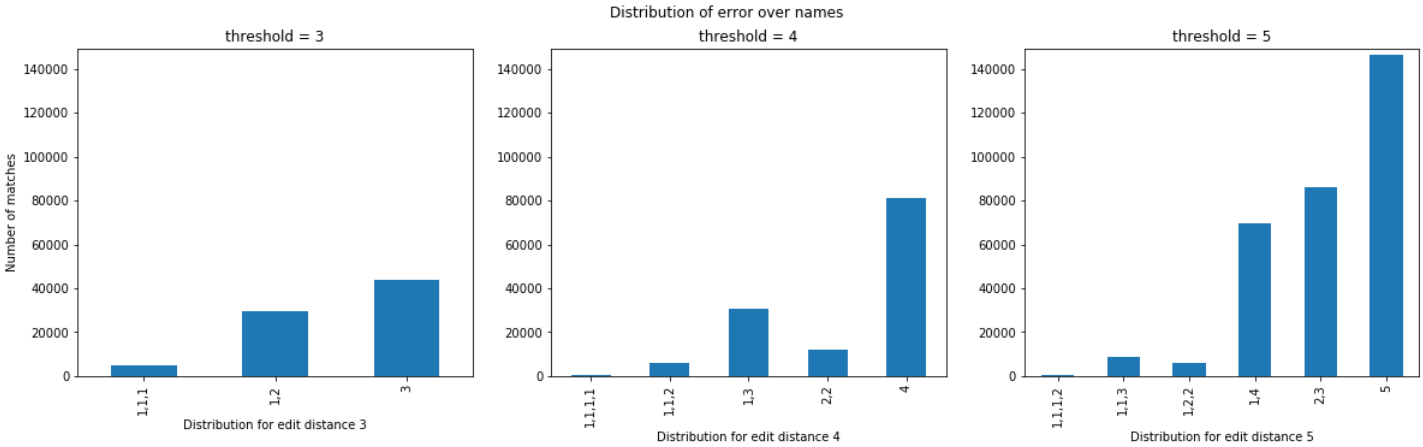
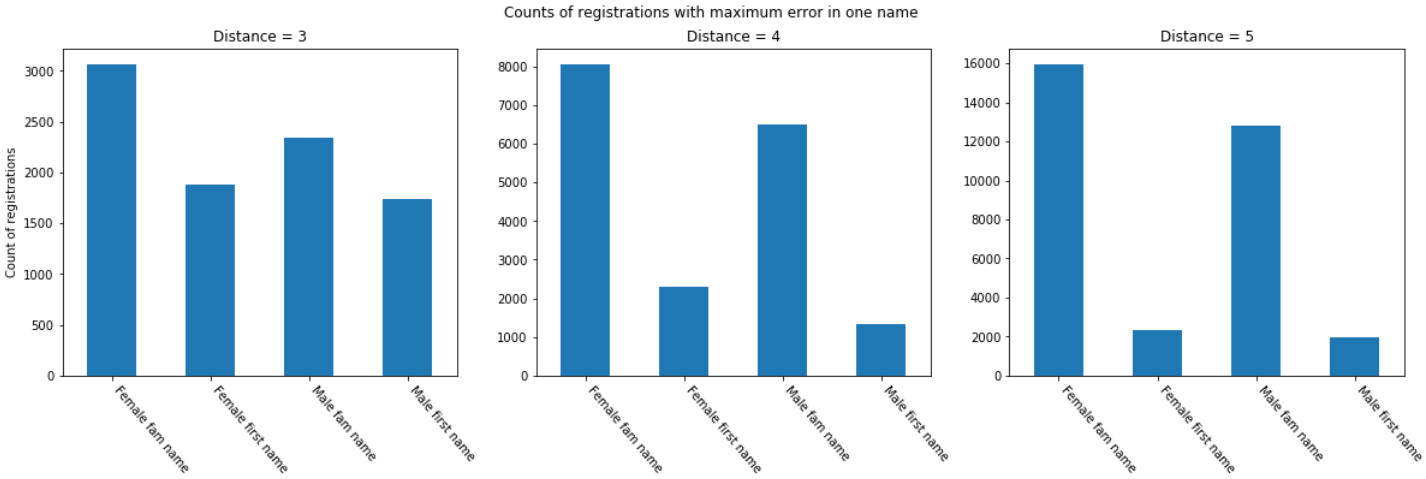


Figure 4.2: Breakdown of the number of matches where the maximum error is concentrated in a single name for the matches generated on the marriage certificates.



'4' for matches with edit distance 4, and 68% with label '5' or '1,4' for matches with edit distance 5. When looking at the distribution of the error for matches with an edit distance of 3, 55.8% has that error in a single name. Although an edit distance of 3 over the entire string seems as a sensible measure, these figures show that a lot of matches were generated where the error is concentrated in a single name. Since the average length of a name between 6 and 7 characters, an error of 3 in a single name means that, on average, the names on the target and candidate registration differ in almost half of the number of characters. This shows that the assumption that matches where the error is concentrated in a single name should be ignored when looking at higher edit distances, and matches where the error is spread over multiple names can be considered as proper matches.

Figure 4.2 shows the number of matches where the maximum distance is concentrated in a single name, broken down by the name type, for the set of matches where the candidate set is the set of marriage registrations of the children. This shows that increasing the threshold will primarily generate more matches in the family name, as discussed above and as indicated in chapter 3, since the number of matches where the error is concentrated in a single first name does not increase as much as the number of matches where the error is concentrated in the last name.

4.1.2 Named Pair based matching results

In total, there are 556,549 matches on birth certificates with a Levenshtein distance of 3 or lower and 54,096 and 130,910 matches on Levenshtein distance 4 and 5, respectively. Ignoring the matches with a distance of 4 and 5, as described in chapter 3, 20,681 matches remain with a distance of 4, and 34,952 matches remain with an edit distance of 5. Table 4.3 shows the breakdown of these matches per label.

The matches with edit distance 4 and 5 are checked for compliance with the rules as described in chapter 3.1.4. The acceptance rate for these matches is shown in table 4.4. The acceptance rate, which is the ratio of matches that meet all the constraints as described in section 3.2.1, is shown for matches where the additional prefix constraint is not applied (in the 'Loose' column) and where the additional constraint is applied (in the 'Strict' column). When comparing the acceptance rate for matches with edit distance 4 and 5, the matches with a distance of 5 are clearly less precise. This is expected, as the edit distance itself is a measure of certainty about the correspondence of two strings. Matches where the edit distance is distributed over more than two names seem to be more precise than matches where the error is concentrated on one or two names. This seems to confirm our assumption that matches with label '4', '2,3' and '5' should not be accepted.

The difference in acceptance rate for the 'Strict' approach and the 'Loose' approach seems to be consistent over all labels, indicating that

4.2 Filtering birth certificates

The matches on birth certificates that are accepted as described in the previous section are checked for consistency. Figures 4.3 and 4.4 show the percentages of accepted and rejected matches for the matches with and without the additional prefix constraint, respectively. The name pair matching constraints are only checked for matches with Levenshtein distance 4 and 5.
The left plot in both figures shows the number of rejected and accepted matches when the maximum number of years between the first and last birth match is 15 at most, and the right plot shows the same percentages, but when the maximum number of years is 20. In both figures, the number of accepted matches is higher when the maximum allowed years is 20 years. This is expected, as this requirement is less restrictive on the matches.

Since matches with a distance of 3 where assumed to be correct matches it is more interesting to note that the number of rejected matches increases significantly at edit distance 3. Table 4.5 shows that this is mainly contributed by the matches where the distance is concentrated in a single name. Although the majority is accepted by our consistency checks, the rise of the rejection rate suggests that the assumption - that matches with edit distance 3 are safe to accept - must be adjusted and that matches with a distance of 3 in a single name should be investigated further.
The effect of additional constraints on matches is clearly visible in the matches with edit distance 4 and 5. With no additional constraint on the prefix, the rejection rate of these matches is around 35% to 40% for matches with a maximum of 15 year between the first and last birth certificate. With the additional prefix constraint, the rejection rate drop to 15% to 20%.

Table 4.5 shows the rejection rate per label. The rejection rate of matches without the additional prefix constraint is lowest when the distance is distributed over all names (or over three in case of a distance of 3), like we expected, and increases as the error is more concentrated in one name, as seen in the previous section. The rejection rate for the matches that comply with the extra prefix constraint seems to be evenly distributed. This is further evidence that the constraint on the prefix of the names can be used in order to distinguish matches when there is a larger distance between the names.

Table 4.3: A breakdown of matches per type of registration. The labels represent the distribution of the error over the names of the registrations.

Birth certificates					
Threshold: 3		Threshold: 4		Threshold: 5	
Label	# registrations	Label	# registrations	Label	# registrations
0	374,474	1,1,1,1	119	1,1,1,2	224
1	101,556	1,1,2	2,554	1,1,3	3,546
1,1	19,218	1,3	12,821	1,2,2	2,606
2	27,656	2,2	5,187	1,4	28,576
1,1,1	2,147				
1,2	12,777				
3	18,720				
Total:	556,549		20,681		34,952

Marriage certificates					
Threshold: 3		Threshold: 4		Threshold: 5	
Label	# registrations	Label	# registrations	Label	# registrations
0	166,849	1,1,1,1	61	1,1,1,2	111
1	44,785	1,1,2	1,221	1,1,3	1,907
1,1	16,578	1,3	6,538	1,2,2	1,351
2	9,011	2,2	2,582	1,4	15,857
1,1,1	3,683				
1,2	980				
3	5,608				
Total:	247,494		10,402		19,226

Death certificates					
Threshold: 3		Threshold: 4		Threshold: 5	
Label	# registrations	Label	# registrations	Label	# registrations
0	246,288	1,1,1,1	135	1,1,1,2	202
1	80,103	1,1,2	2,387	1,1,3	3,106
1,1	16,209	1,3	11,297	1,2,2	2,229
2	22,743	2,2	4,482	1,4	25,078
1,1,1	1,910				
1,2	11,007				
3	15,921				
Total:	394,251		18,301		30,615

Table 4.4: The acceptance rates for edit distance 4 and 5 on the ruleset in 3.3. The acceptance rate describes the percentage of generated matches per label that comply with these rules.

Acceptance rate - Birth ↔ Marriage certificates					
Threshold: 4			Threshold: 5		
Label	Loose	Strict	Label	Loose	Strict
1,1,1,1	43.70%	31.09%	1,1,1,2	50.00%	32.14%
1,1,2	59.71%	35.83%	1,1,3	36.91%	15.48%
1,3	47.35%	18.48%	1,2,2	35.03%	12.70%
2,2	47.27%	17.27%	1,4	8.49%	2.61%
Acceptance rate - Marriage ↔ Marriage certificates					
Threshold: 4			Threshold: 5		
Label	Loose	Strict	Label	Loose	Strict
1,1,1,1	32.78%	18.03%	1,1,1,2	44.14%	31.53%
1,1,2	58.23%	34.46%	1,1,3	33.67%	15.21%
1,3	43.27%	16.20%	1,2,2	33.46%	12.06%
2,2	42.72%	12.59%	1,4	7.67%	2.37%
Acceptance rate - Death ↔ Marriage certificates					
Threshold: 4			Threshold: 5		
Label	Loose	Strict	Label	Loose	Strict
1,1,1,1	43.70%	28.89%	1,1,1,2	49.00%	31.68%
1,1,2	60.20%	36.57%	1,1,3	37.54%	17.58%
1,3	47.48%	18.74%	1,2,2	34.99%	14.09%
2,2	46.63%	16.17%	1,4	8.54%	3.77%

Table 4.5: The rejection rate per label shows that the additional prefix constraint filters out a large chunk of the inconsistent matches generated with edit distance 4 and 5 on matches where the maximum difference between the first and last birth certificate in a family is 15 years.

Distance	Label	Loose	Strict
3	1,1,1	0.057	0.057
	1,2	0.132	0.132
	3	0.433	0.433
4	1,1,1,1	0.115	0.162
	1,1,2	0.087	0.079
	1,3	0.281	0.084
	4	0.366	0.137
5	1,1,1,2	0.116	0.135
	1,1,3	0.256	0.082
	1,2,2	0.383	0.121
	1,4	0.385	0.111

Figure 4.3: Percentage of rejected and accepted matches by the extra checks on consistency. These matches are the matches that are accepted when name pair based matching is performed with the extra constraint on matching prefixes.

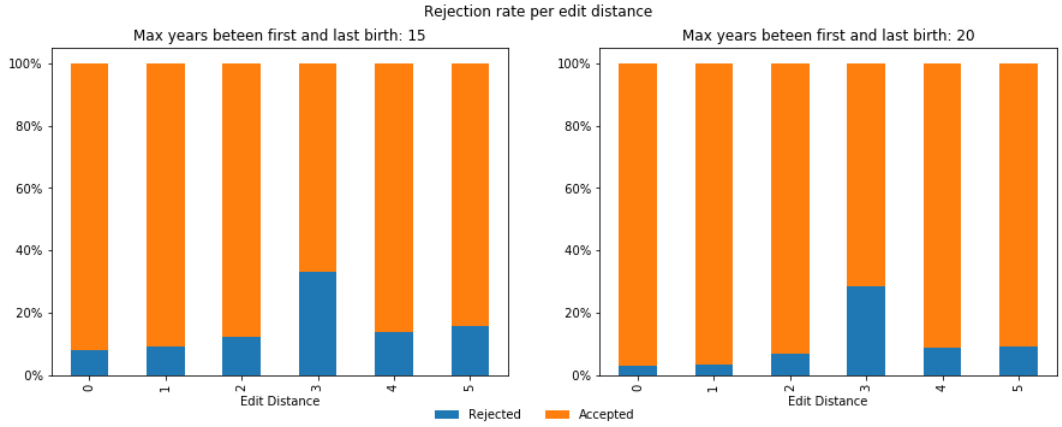
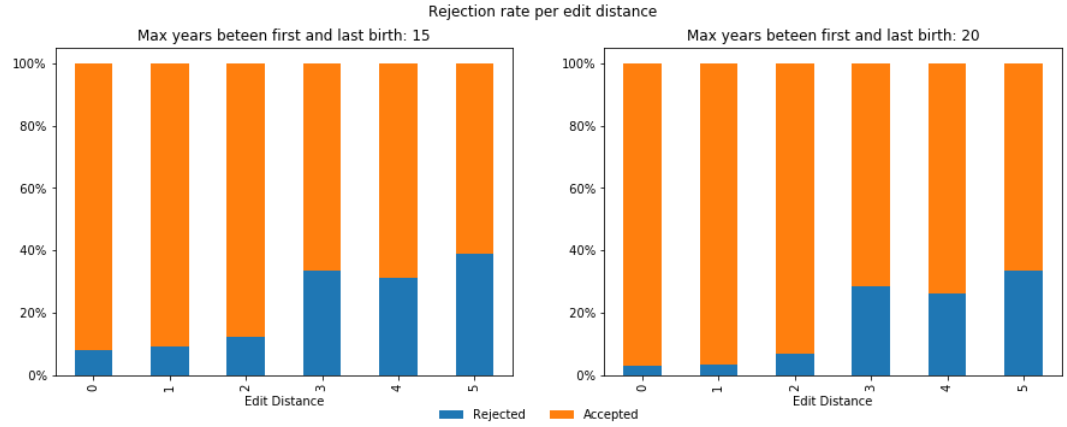


Figure 4.4: Percentage of rejected and accepted matches by the extra checks on consistency. The matches that are checked are the matches that are accepted when name pair based matching is performed without the extra constraint on the matching prefixes.



Chapter 5

Conclusion

In this report, we have investigated the method to link historical life event certificates from official registrations to each other, using an automated approach based on the variance in the names that are mentioned on those certificates.

When increasing the threshold for the initial matching algorithm a lot of extra matches will be generated. Offcourse, care should be taken to accept these extra matches. A great number of extra matches will differ too much in a single name (out of four names), which will lead to incorrect matches.

When choosing to filter these extra matches on name pairs, the rules as desribed in table 3.3 can be used to filter out unwanted matches. As can be seen in table 4.4, these rules filter out a great deal of extra generated matches. This effect seems to be stronger when the allowed edit distance is more focused in single names.

The constraint on the prefix of the name in a name pair seems to be a rule that can be very decisive, since the acceptance rates are much lower when applying this specific rule.
When checking the accepted matches for consistency, the percentages per label of matches that are rejected are much more steady over the set of matches that are accepted by this rule, when compared to the matches that are accepted when this rule is not applied.

We have tested the ruleset on matches with a maximum edit distance of 4 and 5. The ruleset seems to be insensitive for these distances, as the rejection rate of the consistency check seems to be the same for matches made with maximum edit distance 4 and 5.

5.1 Further Research

Since we only checked the ruleset on matches with edit distance 4 and 5, it might be interesting to see if these rules can be applied when increasing the edit distance even further. Another question is whether applying the rule on the prefix on name pairs that are in semi-phonetical form can be more decisive.

Although we did an initial check on the correctness of the birth certificates using the dates of the birth-events, more thorough checks could be done by checking the following criteria:

- Life course consistency: when the matches on marriage and death registrations are also considered, are there registrations that are inconsistent with the previous matched registrations. For instance: a marriage registration is matched with a event date that is later then a death certificate of that person.
- Place / Sex consistency: In the case of two competing registrations, to what extend can the places on those registration be used to rule out a registration? This could also be checked for the sex of the persons on the registrations.
- Name consistency: If there are life events missing, are there registrations in the family where names are name variants of names on other registrations?

List of Figures

List of Tables

Bibliography

[Bloothooft and Schraagen, 2014] Bloothooft, G. and Schraagen, M. (2014). Learning name variants from true person resolution. *Proceedings of the International Workshop of Population Reconstruction.*

[Dunn, 1946] Dunn, H. L. (1946). Record linkage. *American Journal of Public Health and the Nations Health*, 36(12):1412–1426.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

[Schraagen, 2014] Schraagen, M. (2014). *Aspects of Record Linkage.* PhD thesis, Universiteit Leiden.