Accurate Synthetic Generation of Realistic Personal Information

Agus Pudjijono and Peter Christen*

Department of Computer Science, The Australian National University,
Canberra ACT 0200, Australia
peter.christen@anu.edu.au

Abstract. A large proportion of the massive amounts of data that are being collected by many organisations today is about people, and often contains identifying information like names, addresses, dates of birth, or social security numbers. Privacy and confidentiality are of great concern when such data is being processed and analysed, and when there is a need to share such data between organisations or make it publicly available. The research area of data linkage is especially suffering from a lack of publicly available real-world data sets, as experimental evaluations and comparisons are difficult to conduct without real data. In order to overcome this problem, we have developed a data generator that allows flexible creation of synthetic data with realistic characteristics, such as frequency distributions and error probabilities. Our data generator significantly improves similar earlier approaches, and allows the creation of data containing records for individuals, households and families.

Keywords: Artificially created data, data matching, data linkage, privacy, data mining pre-processing.

1 Introduction

Today, massive amounts of data are being collected by many organisations in both the private and public sectors. A large proportion of this data is about people, such as employees, customers, patients, tax payers, or travellers. Normally, personal identifying information (like gender, names, addresses, dates of birth, telephone, social security or driver's license numbers) is stored in such databases together with application specific information (such as employment details, customer orders, medical details, tax payments, or travel details).

When such databases are being analysed and mined within an organisation, then normally, depending upon the desired mining outcomes, only parts of the personal information is used in an analysis (for example age, postcode of residence, or gender). In these cases, privacy and confidentiality are generally not of great concern, as the data is being mined within an organisation and the results of an analysis are also used within the organisation.

* Corresponding author

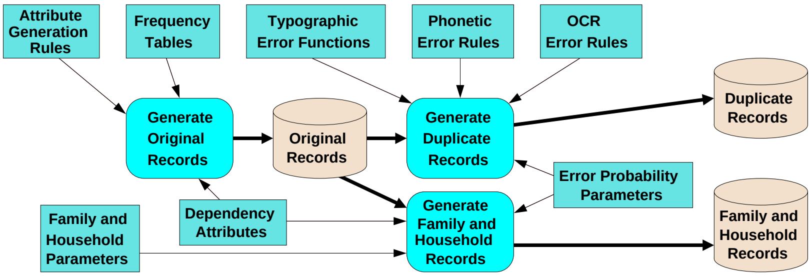
However, when data is being shared between organisations, privacy and confidentiality become of paramount importance, because personal identifiers are commonly required to link records between different organisations [4]. The objective of such linkages is to match and aggregate all records that refer to the same entity. Because real-world data is often dirty [13] (i.e. contains errors and variations, is out-of-date or even missing) and commonly no unique entity identifiers are available, sophisticated approximate matching algorithms are required that use the available personal identifiers for linking [6, 9, 21].

The field of *data linkage* (also called *record linkage*, *entity resolution* or *data matching*) has in the past decade been recognised as an important and challenging problem, and has attracted interest from the data mining, machine learning, information retrieval and database communities. A variety of novel linkage algorithms have recently been developed [6, 9, 21], addressing the major technical challenges of matching accuracy and scalability to very large databases. Another major challenge for the data linkage research community is the lack of publicly available real-world test data sets that allow comparisons of newly developed algorithms and techniques. This lack is mainly due to privacy concerns, because organisations simply cannot publish data sets that, for example, contain personal details of their customers or patients. As a result, data linkage researchers have to use publicly available data sets [2], or use their own, confidential, data which prevents other researchers from repeating experimental studies.

An alternative approach, which is the topic of this paper, is to use synthetically generated data. While this approach has it own challenges, as discussed below, it has several advantages. First, a user can control the size (number of records) and quality (error characteristics) of the generated data sets. Second, such data can be published, and thus allows other researchers to repeat experiments and compare algorithms. Third, the generator program itself can be made available to other researchers, allowing them to generate data sets that are specifically tailored to their use, for example to their country or application domain. Finally, because it is known which of the generated records match with each other and which do not, it is possible to calculate matching rates [6].

Synthetically generated data that contains personal information is not just useful for data linkage research. Any application where data containing personal details is required for research purposes can benefit from synthetically generated data, because such data removes privacy and confidentiality concerns. Examples include research into privacy-preserving data sharing [4], publishing and mining, statistical micro-data confidentiality, or replacing real identifying information with randomly generated values in order to allow publication of data sets.

The challenges when generating synthetic data are that it is not easy to create data with characteristics that are similar to real-world data. The frequency and error distributions of values have to follow real-world distributions, and dependencies between attributes have to be modelled accurately. Our work in this paper describes a data generator with such characteristics. It is a significant improvement over earlier data generators [1, 2, 12], which created data in much simpler and less realistic ways.



Attribute Generation Rules
Frequency Tables
Typographic Error Functions
Phonetic Error Rules
OCR Error Rules
Generate Original Records
Original Records
Generate Duplicate Records
Duplicate Records
Error Probability Parameters
Dependency Attributes
Family and Household Parameters
Generate Family and Household Records
Family and Household Records

Fig. 1. Overview of the data generation process.

In the following section we describe our approach to synthetic data generation of personal information in detail, and in Section 3 we evaluate our data generator and show that the created data has realistic characteristics. An overview of work that is related to our research is then given in Section 4, and we conclude the paper in Section 5 with an outlook to possible future work.

2 Synthetic Data Generation

As illustrated in Fig. 1, the data generator works in two steps. First, a user specified number of *original* records is created using look-up tables with real values and their frequencies and dependencies, or based on specific attribute generation rules [2]. In the second step, randomly selected original records are modified and stored as *duplicate* records. Alternatively, family and household data can be generated based on various parameter settings. Each record is given a unique identifier that facilitates matching of the original records with their duplicates, allowing to calculate matching rates [6].

2.1 Original Record Generation

The original records are randomly created using frequency look-up tables for name (like given- and surname) and address attributes (such as street number, name and type, suburb name, postcode, and state name). Such frequency tables can, for example, be extracted from telephone directories or other data sources as user has access to. For date, telephone and social security number attributes, a user can specify generation rules that, for example, determine the range of dates (such as start and end birth dates), or the number of digits in a telephone number. In the following, we describe the two major novel features of our data generator [20]: attribute dependencies and family and household data.

Attribute Dependencies A dependency occurs if the values in one attribute depend upon the values in one or more other attributes. Examples include: given names depend on the gender and the cultural background of a person, surnames

```
ACT  :  Acton;5, Ainslie;10, Amaroo;7, Belconnen;12
NSW  :  Albanvale;3, Albert Park;6, Alberton;4, Albion Park;9
QLD  :  Allansford;1, Allendale;11, Allestree;4
```

Fig. 2. Sample from a combined dependency–frequency look-up table with Australian state names on the left, and suburb names and their frequencies on the right.

depend on the cultural background, suburb names depend on the state they are located in, and postcodes depend on both the suburb and state they are located in. These dependencies are based on look-up tables, such as the one shown in Fig. 2, that can be extracted from many publicly available data sources.

When generating the original records, the *key* attributes (i.e. attributes that others depend on) are generated first, and according to a selected key attribute value, one of the corresponding dependent attribute values is randomly selected according to their given frequency. For example, using the values from Fig. 2, if the state *'QLD'* has been selected, the suburb name *'Allansford'* would be selected with likelihood 6.25%, *'Allendale'* with likelihood 68.75%, and *'Allestree'* with likelihood 25%, as these are their given relative frequencies.

As attribute dependencies will only create true attribute combinations (for example existing postcode, suburb and state triplets), with a certain (user specified) likelihood the dependency is not followed. Instead, a value is randomly selected from the overall look-up table of the dependent attribute. For example, if the state *'NSW'* has been selected, then, with a certain likelihood, the *'ACT'* suburb *'Ainslie'* might be selected, rather than one of the given *'NSW'* suburbs.

Generating Family and Household Data Family data is generated assuming that all members of a family have the same surname. The records for a family are generated as follows. First, an original record is randomly selected. Depending upon its age and gender value, it is assigned a family role: *husband* if male and age is over 18 (this age can be changed by the user), *wife* if female and age is over 18, *son* if male and age is below 18, and *daughter* if female and age is below 18. The next step is to randomly select how many other records are to be generated for this family, and their age and gender distributions. These records are then generated by keeping the surname value from the first family member record, but randomly selecting given name, gender and date of birth values.

The value of address attributes are generally kept the same for all members of a family, however, depending upon the age value of son and daughter records, a new address will be created with a certain likelihood, assuming the child is old enough and has moved somewhere else. When generating family data, a large number of parameters needs to be set by the user [20], including the probability of assigning a record to a specific family role, the distribution of age gaps, and the probability of parents having a certain number of children.

Household data is generated similarly, with the main difference being that all records generated in a household have the same address, and that all age values are above 18 (again, the user can modify this minimum age). Thus, households

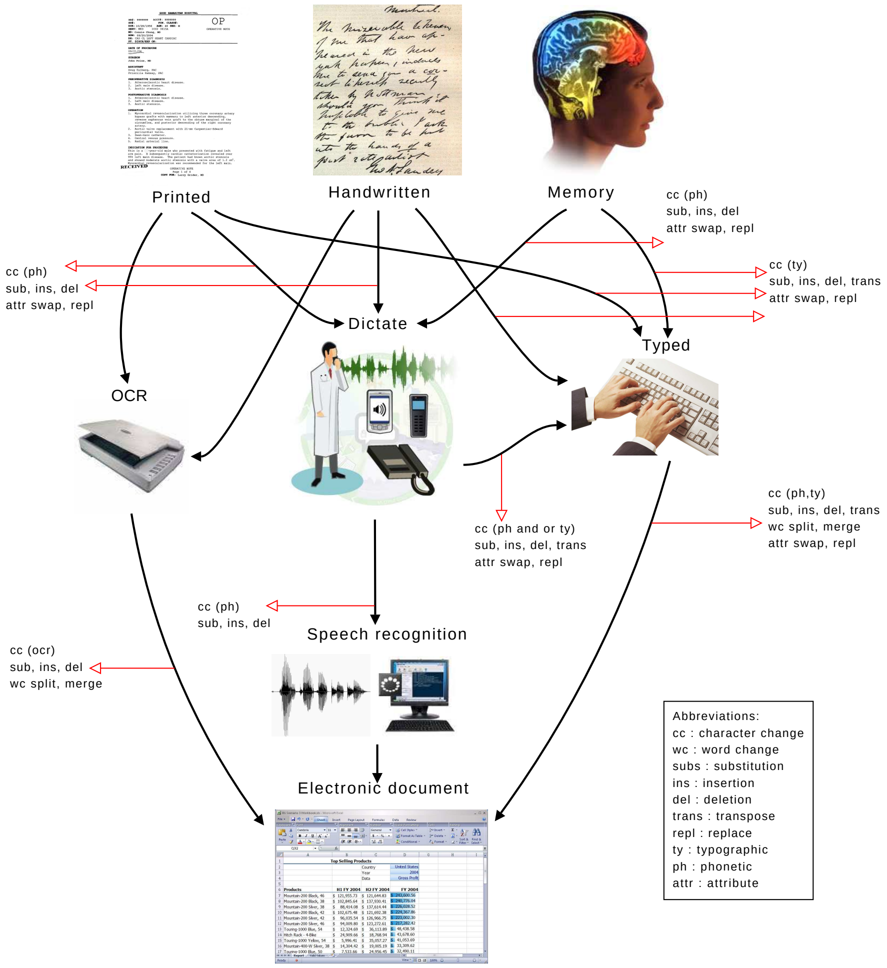


Fig. 3. Model of data sources and possible errors introduced during data entry.

are made of a group of people with different names, and correspond to shared houses as well as unmarried couples. Similar to family data, a large number of parameters needs to be set by the user, for example the distribution of the number of people in a household, and their age and gender distributions.

2.2 Error Modelling and Record Modification

As illustrated in Fig. 3, data can be entered in many different formats and through a variety of channels, with each having its own error characteristics. For example, handwritten forms that are scanned and processed using optical character recognition (OCR) software will likely contain substitutions between similar looking characters, such as *'1'* and *'I'*, or groups of characters, like *'rn'* and *'m'*. On the other hand, phonetic errors, like the name variations *'dickson'* and *'dixon'*, are introduced when information is being dictated and entered using speech recognition, or typed manually. Typing itself introduces certain errors more likely than others. Depending upon keyboard layout, mistyping neighbouring keyboard keys, such as *'a'* and *'s'*, can occur. Often, depending upon the data entry channel, a combination of error types can be introduced.

Our data generator allows the specification of typographic, phonetic and OCR errors. For each error type, the user can set parameters of how likely they occur when the duplicate records are being generated. For example, setting the likelihood of typographic and phonetic errors to 0 will result in duplicate records that only contain OCR errors.

Typographic Errors Typographic errors at character level (insertion, deletion, or substitution of a character; transposition of two adjacent characters) are implemented as functions that apply the corresponding modification to a given input string with a certain likelihood (as set by the user), and return the modified string. Following real-world studies of error distributions, the position of a modification is randomly selected such that a modification is more likely introduced in the middle or towards the end of a string, because the initial characters of a name are more likely to be correct than later ones [19].

Optical Character Recognition (OCR) Errors OCR modifications are based on rules that consider shape similarity among characters. We currently have a set of around fifty such rules, for example *'5'* and *'S'*, *'6'* and *'G'*, or *'w'* and *'vv'*. These rules represent the most likely OCR variations that might occur. When duplicate records are being generated, depending upon the input string value and user specified parameter settings, one or more possible OCR modifications will be randomly selected and applied to the input string, and the modified string will be inserted into the duplicate record.

Like phonetic rules (as discussed below), OCR modifications can occur as a single character variation (substitution or deletion), or as a combination of modifications (for example a substitution and deletion, or a deletion and insertion). Different from phonetic rules, however, is that OCR modifications are independent of the position and can occur anywhere in a string.

Phonetic Errors These kinds of errors are usually more complex than simple typographic or OCR errors, as they often include changes of character groups. The basic idea behind our approach in modelling phonetic errors is to employ the encoding rules that are used in phonetic encoding methods, such as *Phonix* [10] and *Double-Metaphone* [17]. In encoding methods, these rules are used to group similar sounding names together, while we use them to modify a name in order to get a similar sounding variation of it. We have developed around 350 such rules, each made of the following seven components.

1. **Position** The position within the input string where the *original* string pattern (see below) can occur. The four possible values are: *ALL* (a pattern can occur anywhere), *START* (a pattern must occur at the beginning), *MIDDLE* (a pattern must occur in the middle, but it cannot occur at the beginning or end), and *END* (a pattern must occur at the end).
2. **Original pattern** This is the string (made of one or more characters) that will be replaced with the *substitute* string pattern if the rule is applied.
3. **Substitute pattern** This is the string (made of zero or more characters) that will replace the *original* string pattern if the rule is applied.
4. **Precondition** A condition that must occur before the *original* string pattern in order for this rule to become applicable. A precondition can for example be that the character immediately before the original pattern is a vowel (*'V'*), a consonant (*'C'*), or a more complex expression [20].
5. **Postcondition** Similarly, a condition that must occur after the *original* string pattern in order for this rule to become applicable.
6. **Pattern existence condition** This condition requires that a certain given string pattern does (*'y'* flag) or does not (*'n'* flag) occur in the input string.
7. **Start existence condition** Similarly, this condition requires that the input string starts with a certain string pattern (*'y'* flag) or not (*'n'* flag).

The last four components of a rule (its conditions) can be set to *'None'* if they are not required for a rule. In the following we give some illustrative examples.

- `ALL, h, @, None, None, None, None`
  mustapha → *mustapa*
  In this rule there are no conditions, so any occurrence of the character *'h'* is being removed (replaced with an empty string – denoted with a *'@'*).
- `END, le, ile, C, None, None, None`
  bramble → *brambile*
  The precondition in this rule is *'C'*, which means the character before the original pattern must be a consonant (any character other than *'a'*, *'e'*, *'i'*, *'o'*, *'u'*). The character before the pattern *'le'* in this example is a *'b'*, so the modification *'le'* into *'ile'* can be applied.
- `MIDDLE, ge, ke, None, None, None, y;van;von;sch`
  van geraldus → *van keraldus*
  This rule states that in order to replace the pattern *'ge'* in the middle of the input string, it must begin (start existence condition set to *'y'*) with one of the three patterns *'van'*, *'von'* or *'sch'*.

3 Experimental Evaluation

Our data generator is implemented as part of the *Febrl* (Freely Extensible Biomedical Record Linkage) open source data linkage system [5],[1] and is written in the *Python* programming language. Due to the availability of its source code, it can be modified and extended according to a user's needs.

[1] Available from: `https://sourceforge.net/projects/febrl/`

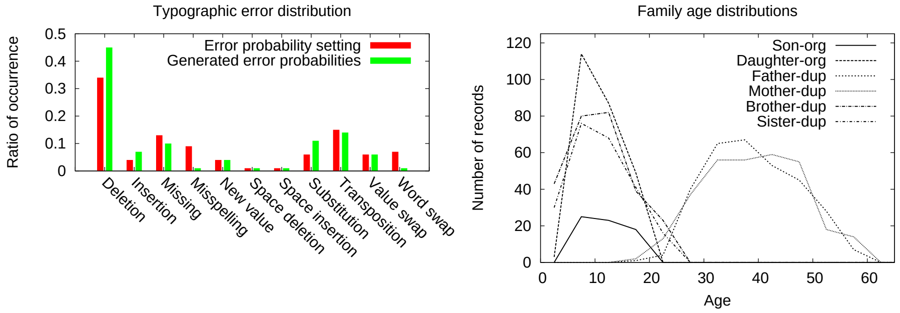


Fig. 4. Example distributions of typographic errors and family age values from synthetically generated data.

A large number of parameters can be set by the user, including the number of original and duplicate records to be generated, the frequency and dependency look-up tables to be used, the distributions of household and family records, and the various error characteristics to be applied when the duplicate records are being created. A detailed description is available elsewhere [20].

We used a variety of data sets to create our look-up tables, including a data set containing 99,571 names and their culture of origin (37 different cultures) [18], a data set with Australian postcode, suburb and state values as available from the *Australia Post* Web site,[2] and the various look-up tables supplied with the *Febrl* data linkage system [5]. Error and modification probabilities were set according to real world studies on typographic and other errors [7, 11, 14, 16, 19].

Due to space limitations, we can only show a selection of experimental results that illustrate the characteristics of the generated data. A more detailed analysis and discussion is provided elsewhere [20]. A user can repeat our experiments by downloading the *Febrl* system and run the generator program supplied with it (and possibly change parameter settings according to her or his needs).

The left-hand side of Fig. 4 shows how the error distribution in the generated data does follow the parameter settings provided by the user. The differences that can be seen are mainly due to the characteristics of the data. For example, misspellings are much less frequent in the generated data than as set by the user, because they are based on a look-up table of name values and their possible misspellings. Thus, no misspelling modification can be applied if a name value in an original record is not listed in the corresponding misspellings look-up table. Similar, word swaps within an attribute (such as *'paul thomas'* ↔ *'thomas paul'*) can only be applied when an attribute value contains more than one word.

The right-hand side of Fig. 4 illustrates the age distributions of family records, where son and daughter records were originally generated, followed by the generation of records that correspond to other family members. As can be seen, the

[2] Available from: `http://www.post.com.au/postcodes/`

(a) Typographic errors

rec_id, age, given_name, surname, street_number, address_1, address_2, state, suburb, postcode

rec-1-org, 33, ***madison***, solomon, ***35***, tazewell ***circuit***, trail view, ***vic***, ***beechboro***, ***2761***
rec-1-dup-0, 33, madison, solomon, 35, tazewell circ, trail view, viv, beechboro, 2761
rec-1-dup-1, 33, madison, solomon, 35, tazewell crct, trail view, vic, bechboro, 2761
rec-1-dup-2, , madison, solomon, 36, tazewell circuit, trail view, vic, beechboro, 2716
rec-1-dup-3, 33, madisoi, solomon, 35, tazewell circuit, trail view, vic, beech boro, 2761

rec-2-org, ***29***, soida, perera, ***416***, marchant place, ***weemilah***, ***nsw***, belmont, 2280
rec-2-dup-0, 29, soida, perera, 414, marchant place, wemilah, nsw, belmont, 2280
rec-2-dup-1, 92, soida, perera, 416, marchant place, weemilah, naw, belmont, 2280

(b) Phonetic errors

rec_id, age, given_name, surname, street_number, address_1, address_2, state, suburb, postcode

rec-3-org, 29, ***jalisa***, ***wane***, 25, ***prisk*** place, ***seabank***, , wa, latham, 6616
rec-3-dup-0, 29, ghialisa, wane, 25, prisk place, zeabank, , wa, latham, 6616
rec-3-dup-1, 29, jalisa, whane, 25, prisc place, seabank, , wa, latham, 6616
rec-3-dup-2, 29, jalissa, wane, 25, prisk place, seapank, , wa, latham, 6616

rec-4-org, 39 , desirae, ***contreras***, 44, maltby street, ***phillip*** lodge, nsw, ***burrawang***, 3172
rec-4-dup-0, 39, desirae, kontreras, 44, maltby street, phillip lodge, nsw, burrawank, 3172
rec-4-dup-1, 39, desirae, contreras, 44, maltby street, fillip lodge, nsw, buahrawang, 3172

(c) OCR errors

rec_id, age, given_name, surname, street_number, address_1, address_2, state, suburb, postcode

rec-5-org, 28, ***phyliss***, winter, 20, ***aspinall*** road, , qld, ***wairewa***, ***3887***
rec-5-dup-0, 28, phyliss, winter, 20, aspinaIl road, , qld, wairewa, 3881
rec-5-dup-1, 28, phyl'lss, winter, 20, aspinall road, , qld, wajrewa, 3887

rec-6-org, ***81***, ***madisyn***, sergeant, 6, ***howitt street***, creekside cottage, vic, ***nangiloc***, 3494
rec-6-dup-0, 87, madisyn, sergeant, 6, howitt street, creekside cottage, vic, nanqiloc, 3494
rec-6-dup-1, 81, madisvn, sergeant, 6, hovitt street, creekside cottage, vic, nangiloc, 3494

Fig. 5. Three example of generated data with different error types. Original values that were modified are highlighted in bold-italics and their corresponding modifications are underlined. Two modifications were introduced into each duplicate record.

age distribution of all family members follows what one would expect in the real world. Please note that all the settings that control these distributions can be modified by a user according to her or his requirements [20].

Examples of generated data are shown in Fig. 5. As can be seen, the record identifier values designate if a record is an original or a duplicate, and the duplicate records are numbered and refer back to their original record in order to allow the calculation of matching rates [6].

4 Related Work

The first data generator to facilitate research in data linkage and related areas was developed in the mid 1990s [12]. Called *DBGen*,[3] it allowed the generation of records and duplicates using lists of names, cities, states and postcodes, however

[3] Available from: `http://www.cs.utexas.edu/users/ml/riddle/data.html`

without information about the frequency distribution of these values. It allowed various parameters to be set by the user, including the number of records to be generated, the percentage and distribution of duplicates to be generated, as well as the types and amounts of errors to be introduced.

An improved generator was then described in 2003 [1]. It allows attribute values to become missing, and also improved the variability of the created values. It is however unclear if this generator is using real frequency information or not, as not many details were published. To the best of our knowledge this generator is not publicly available.

A first simple version of our generator, called *DSGen* [2], has been freely available as part of the *Febrl* data linkage system. It improved upon earlier generators by including frequency tables of attribute values (for example extracted from telephone directories), more flexible setting of individual error probabilities, as well as inclusion of look-up tables with name variations (to be used for example for nick-names, known phonetic variations, and common misspellings). This generator however does not include attribute dependencies, does not allow creating family or household record groups, and it does not model errors as accurately as the new version described in this paper.

The error model presented in Section 2.2 is based on rules that were originally developed for phonetic encoding methods [3]. These methods have a long history, with the original *Soundex* method, designed for the analysis of US census data, having been patented in 1918 [22]. A common feature of all phonetic encoding methods is that they convert a name string into a code according to how the name is being spoken. This is obviously a language dependent process, and most phonetic encoding methods have been developed for the English language. Names that have the same phonetic code are grouped into the same block or cluster, and this then allows, for example, to find and match records that have differently spelled names but that might refer to the same person [3].

The traditional *Soundex* encoding keeps the first letter of a name string and encodes the remainder with digits based on an encoding table. *Phonex* [15] aims to improve this approach by pre-processing name strings according to their English spelling (for example, *'ph'* will be replaced by *'f'*, or a *'h'* at the beginning of a name will be removed). *Phonix* [10] is an algorithm that goes even further, by applying more than a hundred transformation rules on letter groups (some only applicable to the beginning, middle or end of a name string). These rules were used in the error modelling of our data generator as described in Section 2.2. While the three so far described methods return a phonetic encoding made of an initial letter followed by digits, the *NYSIIS (New York State Identification Intelligence System)* and *Double-Metaphone* [17] algorithms return phonetic codes made only of letters. The *Double-Metaphone* algorithm is also aimed to be more suitable for non-English names, and for certain names returns two alternative encodings. Similar to *Phonix*, it is based on a large number of rules.

Much of our work is also based on the results of various studies that have been conducted in the area of spelling errors and their corrections. An early such study [7] in 1964 found that over 80% of typographic errors were single character errors (i.e. either an inserted, deleted, or substituted character; or two transposed adjacent characters), while another study [11] in 1980 found that OCR and other automatic devices introduce similar error rates for substitutions, deletions and insertions, but not transpositions. Yet another study [19] reported that OCR output contains almost exclusively substitution errors, while this type of error accounts for less than 20% of errors with keyboard based manual data entry. This study also reported that typically 90% to 95% of misspellings in raw keyboard entry only contain one error, and that only around 8% of first letters were incorrect, compared to almost 12% of second and nearly 20% of third letters. Similar results were reported by other later studies [14, 16].

A rule-based approach of error modelling, similar to our method as described in Section 2.2, has recently been applied to spelling corrections [8]. The experimental results reported were better than for other spelling correction techniques based on lexicons of words and their misspellings.

5 Conclusions

We have presented a synthetic data generator for personal information that allows the generation of realistic data based on real-world frequency look-up tables, as well as attribute generation rules. Our generator is capable to model dependencies between attributes, it can generate records for individuals, families and households, and it can accurately model typographic, phonetic and OCR errors. Our generator can be used for research where data with private information cannot be shared, linked or published.

There are various possibilities to improve our generator. First, allowing the generation of not just personal information, but also application specific attributes, such as medical, employee, or customer details, would make our generator applicable to the wider data mining community. Second, the generation of family records can be extended to include family roles such as grandparents or cousins, with the ultimate aim to synthetically generate complex family connections as they occur in real life. Third, making the generator more international, by enabling Unicode characters, would also be a useful endeavour, as it would allow the generation of data sets containing, for example, Thai, Chinese, Japanese, Russian, or Arabic characters.

Finally, we also plan to develop a graphical user interface (GUI), which is to be integrated into the *Febrl* data linkage system [5], in order to facilitate the setting of the many possible parameters.

References

1. P. Bertolazzi, L. De Santis, and M. Scannapieco. Automated record matching in cooperative information systems. In *International workshop on Data Quality in Cooperative Information Systems*, Siena, Italy, 2003.
2. P. Christen. Probabilistic data generation for deduplication and data linkage. In *IDEAL'05, Springer LNCS vol. 3578*, pages 109–116, Brisbane, Australia, 2005.

3. P. Christen. A comparison of personal name matching: Techniques and practical issues. In *IEEE ICDM'06 workshop on Mining Complex Data (MCD'06)*, Hong Kong, 2006.
4. P. Christen. Privacy-preserving data linkage and geocoding: Current approaches and research directions. In *IEEE ICDM'06 workshop on Privacy Aspects of Data Mining (PADM'06)*, Hong Kong, 2006.
5. P. Christen. Febrl – A freely available record linkage system with a graphical user interface. In *HDKM'08, CRPIT vol. 80*, Wollongong, Australia, 2008.
6. P. Christen and K. Goiser. Quality and complexity measures for data linkage and deduplication. In *Quality Measures in Data Mining*, volume 43 of *Studies in Computational Intelligence*, pages 127–151. Springer, 2007.
7. F. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.
8. S. Deorowicz and M. Ciura. Correcting spelling errors by modelling their causes. *International Journal of Applied Mathematics and Computer Science*, 15(2):275–285, 2005.
9. A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.
10. T. Gadd. PHONIX: The algorithm. *Program: Automated Library and Information Systems*, 24(4):363–366, 1990.
11. P. Hall and G. Dowling. Approximate string matching. *ACM Computing Surveys*, 12(4):381–402, 1980.
12. M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *ACM SIGMOD'95*, pages 127–138, San Jose, California, 1995.
13. M. Hernandez and S. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
14. K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.
15. A. Lait and B. Randell. An assessment of name matching algorithms. Technical report, DepartmentDeptartment of Computer Science, University of Newcastle upon Tyne, 1993.
16. J. Peterson. A note on undetected typing errors. *Communications of the ACM*, 29(7):633–637, 1986.
17. L. Philips. The Double-Metaphone search algorithm. *C/C++ User's Journal*, 18(6), 2000.
18. C. Phua, V. Lee, and K. Smith-Miles. The personal name problem and a recommended data mining solution. *Encyclopedia of Data Warehousing and Mining, Information Science Reference*, 2008.
19. J. Pollock and A. Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27(4):358–368, 1984.
20. A. Pudjijono. Probabilistic data generation. Master of Computing (Honours) thesis, Department of Computer Science, The Australian National University, 2008.
21. W. Winkler. Overview of record linkage and current research directions. Technical Report RR2006/02, US Bureau of the Census, 2006.
22. J. Zobel and P. Dart. Phonetic string matching: Lessons from information retrieval. In *ACM SIGIR'96*, pages 166–172, Zürich, Switzerland, 1996.