On Identifying Name Equivalences in Digital Libraries

Dror G. Feitelson
School of Computer Science and Engineering
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

Abstract

The services provided by digital libraries can be much improved by correctly identifying variants of the same name. For example, this will allow for better retrieval of all the works by a certain author. We focus on variants caused by abbreviations of first names, and show that significant achievements are possible by simple lexical analysis and comparison of names. This is done in two steps: first a pairwise matching of names is performed, and then these are used to find cliques of equivalent names. However, these steps can each be performed in a variety of ways. We therefore conduct an experimental analysis using two real datasets to find which approaches actually work well in practice. Interestingly, this depends on the size of the repository, as larger repositories may have many more similar names.

1 Introduction

People have names. In fact, most have two or three names, and some have four or more. The names serve to identify people; they can be viewed as labels attached to the named individuals.

Regrettably, the mapping of names to people is not one-to-one. Several individuals may share the same name. Some individuals change their name during their lifetime, e.g. after getting married and establishing a new family. But the most common problem is the inconsistent use of names. People often do not use all their names, or abbreviate them. This leads to a situation where a many-to-many mapping exists from names to people (Fig. 1).

The fact that the mapping is many-to-many rather than one-to-one causes various problems in using digital libraries. For example,

- The ACM portal[1] includes the following distinct entries in its author index:

  DROR G$_\circ$ FEITELSON
  DROR FEITELSON
  D$_\circ$ G$_\circ$ FEITELSON
  D$_\circ$ FEITELSON

[1]The digital library of the Association for Computing Machinery, a large professional society of computer scientists, at URL http://portal.acm.org/.

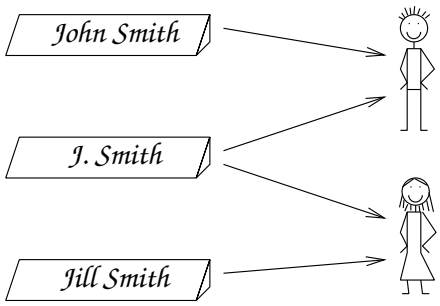

Figure 1: *Naming is a many-to-many mapping of names to people.*

All of these refer to the same person. If this is a common case, the index is four times larger than it need be, requiring extra browsing and scrolling.

- Having different names for the same person makes it harder to use the data in the digital library for automatic classification of papers and identifying equivalent entries. Also, data about co-authorship may be misleading rather than helpful.

- A side effect of the distinction among the different name variants of the same person is that it is impossible to retrieve all the papers by this individual at once. Instead, the papers listed under each name have to be retrieved separately. This is true both when using the author index and when conducting a search. For example, the CiteSeer[2] search instructions say

  > For authors, list all variants that appear in citations, separated by "or", e.g. m jordan or michael jordan or m i jordan or michael i jordan

  but what if you don't know all the variants? A possible alternative is to conduct a more general search, say by specifying only the last name; this risks retrieving many extra documents by other people who have the same last name.

- Likewise, it is impossible to directly count the papers written by an individual, or the citations to the work of an individual. While not the primary goals of a digital library, such actions are often performed in the process of evaluating an author's work. They are also needed for more advanced services, e.g. the generation of an author collaboration graph.

Of course, detecting name equivalences is also important in other applications, e.g. the identification of returning customers [2, 1]. In fact, digital libraries seem to be a relatively easy case, because names are based on how the authors themselves write them in print, rather than on how they are entered into the system by other people.

Nevertheless, current digital libraries do not handle this problem very well. The ACM portal takes the conservative approach of retaining all names as they appear. This includes all variants of each name, some of which are clear spelling errors. For example, the the following five entries in the author index all represent Turing award winner Niklaus Wirth:

[2] A digital library based on the concept of autonomous citation indexing [6], at URL http://citeseer.ist.psu.edu/.

N WIRTH
N$_\circ$ WIRTH
NICKLAUS WIRTH
NIKLAUS E$_\circ$ WIRTH
NIKLAUS WIRTH

The first one is just an abbreviation with the $_\circ$ missing, and the third one is an obvious misspelling.

The CiteSeer web site takes the opposite approach. In its listing of most cited computer scientists, only the first initial is ever used. This causes many distinct names to be bundled together; for example, the top entry is D$_\circ$ JOHNSON, which could be any of at least 18 individuals, and probably more (according to a search in the DBLP database[3]). Somewhat surprisingly, this practice may also split a single author into two: most of the papers by W$_\circ$ DANIEL HILLIS, for example, are attributed to W$_\circ$ HILLIS, but some to D$_\circ$ HILLIS, because the first initial is sometimes dropped.

The goal of this work is to find automatic means to identify the nature of the mapping of names to people. Specifically, we want to identify situations in which multiple names actually refer to the same person. This problem has attracted research for dozens of years, due to its obvious importance in business and services, where names are often entered incorrectly due to mistranscription. One early algorithm is the "soundex" method, in which names are mapped to short codes that represent the salient features of how they sound [4, p. 394]. Another is the "metaphone" method, which preserves more detail and is therefore more accurate [7]. These methods can be used to identify large variations in the way last names (surnames) are written, but may also lead to many false positives.

The present work is limited to identifying variations in *first* names, where the main source of variability is different forms of abbreviation — a topic that has not attracted much research in the past. Moreover, we do so using lexical means, i.e. by only using the names themselves. Extensions based on semantic information are listed as future work at the end of the paper.

It should be noted that the problem we are addressing does not have a single "correct" answer. For example, the names G$_\circ$ HERBERT and HERBERT W$_\circ$ may be two abbreviations of someone with the three first names GEORGE HERBERT WALKER, but they may also be abbreviations of two people with two first names each. It is impossible to know from the names alone. What we need is heuristics that work well *in practice*. We therefore use an experimental approach, and evaluate our proposed heuristics by tabulating their performance on author names from two online bibliographies. This enables the identification of the weaknesses of each heuristic, and the suggestion of a better one.

The next section outlines the framework used for identifying name equivalences. It also presents the environment used for evaluations, and the evaluation results for the different heuristics. This is used to motivate the progression of heuristics detailed in the following sections.

2 Framework and Evaluation

The framework employed by our system is outlined in Fig. 2. The first step is to parse and normalize the input names. This step is required in order to prepare the data for processing: to compare

[3] Another digital library of computer science, at URL http://www.informatik.uni-trier.de/~ley/db/.

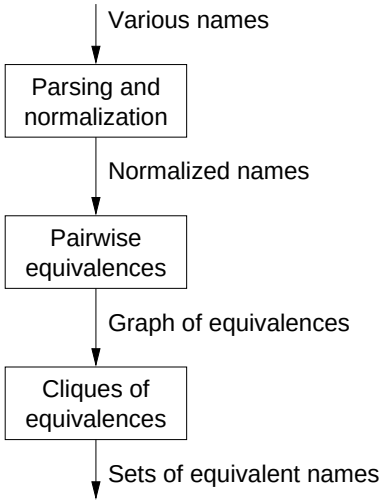

Figure 2: *System framework.*

first names when the last name is the same, we need to know what is a first name and what is a last name. It is also used to avoid simple problems, such as variations that result from unusual capitalization or foreign accents. While not the focus of the present work, we provide some comments on this issue in Section 3.

The equivalence-detection algorithm itself operates in two phases. The first is matching pairs of names to each other. This induces a (very sparse) graph on all the names, where nodes are names, and edges represent equivalences. The second is an analysis of this graph to identify sets of names that are all equivalent to each other. Obviously, these names come from connected components of the graph. These phases are described in detail in Sections 4 and 5, respectively.

The developed heuristics are evaluated by tabulating their performance on two sets of names. The first set comes from the BoW database[4] [3]. This database contains 3872 documents tagged with 5740 author names. These relatively small numbers allow for a manual inspection and an educated guess at which names are indeed equivalent. The output of the heuristics is then compared with the manual evaluation, and deviations are classified as either false positives (the heuristic claims an equivalence that is most probably not true) or false negatives (the heuristic missed an equivalence that most probably is true).

In addition, we also check the performance of our algorithms on a small fragment of the ACM Portal author index. The fragment used contained all the names where the last name starts with the letters FE. There were 3076 such names. In this case a full manual inspection is impractical, because some last names have dozens of instances, making a manual comparison of all pairs very error-prone. However, it is still possible to compare the different heuristics to each other.

The evaluation using a fragment of a large database is important due to non-trivial scaling effects. In a very large database, many more documents are covered. There is therefore a bigger chance to observe variations on an author's name. In addition, there is a bigger change to observe multiple authors that share names or at least initials (Fig. 3). This makes it harder to pick out those names that are indeed equivalent to each other.

[4]This is an experimental bibliographic server, accessible at URL http://www.bow.cs.huji.ac.il/.

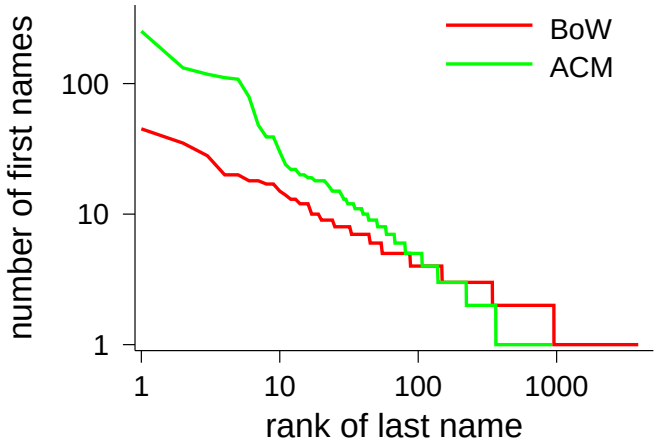

Figure 3: *The distribution of names in both datasets is Zipf-like, but with different exponents. The BoW dataset has an exponent of about 0.55, and the ACM dataset an exponent of about 0.9.*

The results of the evaluations are shown in Table 1. We take the unusual approach of showing the results first, as this will facilitate and motivate the introduction of the various algorithms. Each part of the results is discussed at the appropriate point.

3 Parsing and Normalization

3.1 Parsing Names

In order to compare names to each other, one must first determine where each name starts, where it ends, what part constitutes the last name, what parts are first names, and whether any salutations or suffixes are present. While definitely important, this aspect of name equivalences lies outside the scope of the present work. This disregard is possible because the BoW data is based on the BibTeX format, which has strict rules that make parsing easy[5] [5]:

1. Sequences of names referring to different people are separated by AND without any commas. Example: JOHN SMITH AND JEFF SMITH AND JACK SMITH.

2. In a sequence of names identifying a person,

   (a) If a comma is present, the part before the comma is the last name, and the part after the comma the first name. If multiple commas are present, use the last one. Example: WATSON, JR$_\circ$, THOMAS J$_\circ$.

   (b) If any of the names starts with a lower-case letter, the last name starts from that point. Everything before it is a first name. Example: JOHN VON NEUMANN.

   (c) If all names start with upper-case letters, the last one is the last name and all previous ones are first names.

[5] Actually there are a couple of additional rules for special circumstances, but these ones give the gist.

Heuristic	BoW			ACM		
	Total cliques found	False positive	False negative	Total cliques found	False positive	False negative
Simple match independ clique	650	31(3)	47	306	8(9)	142
Strict match independ clique	629	4	41	312	1	131
Strict match weighted clique	645	4	26(1)	395	1	49(14)
Strict match full names weighted clique	649	4	22	441	7(2)	7(5)

numbers in parentheses indicate added members or missing members from a real clique.

Table 1: *Effectiveness of the different heuristics. In the BoW data, the correct number of cliques according to a manual inspection is 666, and false positives and negatives are counted relative to these manually identified cliques. For the ACM data such a manual identification is not available, so false negatives and positives are counted relative to a manual inspection of the cliques found by the algorithms.*

The ACM Portal data is from an author index, and thus already provided as individual first and last names. We do however handle simple mundane issues, such as identifying initials without a period with initials that do have a period (e.g. J with $J_\circ$), and partitioning sequences of initials into their components (e.g. $A_\circ B_\circ$ is $A_\circ$ $B_\circ$, with two distinct abbreviated names).

It should be noted that suffixes (such as $JR_\circ$) are actually a qualifier of the first name, but are typically treated as part of the last name. We retain this approach. Our data did not contain any identified instance of a missing or wrong suffix. However, there are variations regarding the separation of the suffix from the last name with a comma. These are handled by the normalization.

We note in passing that several methods can be used when the input data is less forgiving. For starters, one can compare input data with names that have been recognized in the past and are already present in the database. When the data comes from automatic acquisition of documents, it is possible to compare the list of authors at the beginning with the list of references at the end. In particular, self citations may provide important clues regarding the correct parsing of names, and the equivalence of different abbreviations.

3.2 Name Normalization

Once the names have been parsed, it is advisable to normalize them. We performed the following types of normalization:

- Translate all upper-case letters to the corresponding lower-case letters. This is useful in multi-part names, in which the second part is sometimes inconsistently written with different capitalizations.

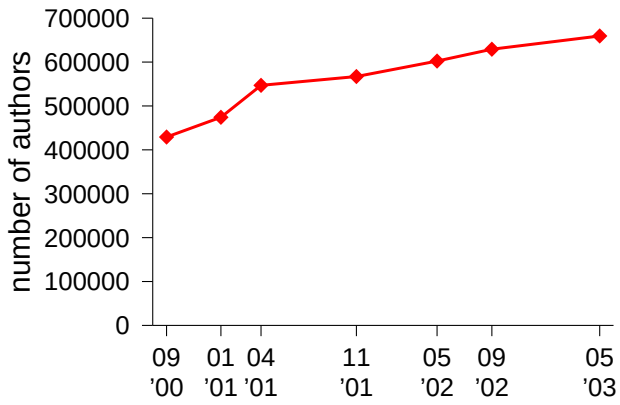

Figure 4: *Number of authors in the CiteSeer database is large and growing.*

- Remove all foreign accents. Again, we empirically find that names are often written inconsistently, either with the wrong accent or without the accent altogether. Note, however, that it is important to conserve the base letter. Example: THOMÁS, THOMÀS, and THOMAS.

- Replace special characters by commonly-used alternatives. For example, ø is replaced by O, ł by L, and ß by SS. In some cases such a replacement can also be considered for accents; for example, Ä is often written as AE.

- Remove all special marks, such as apostrophes. The only ones retained are hyphens.

Actual examples illustrating the effect of these normalizations on our datasets are given in the appendix. By far the most important is to remove foreign accents, and doing so facilitates many matches that seem to be correct. The second most important is to translate upper-case letters into lower case. We did not observe situations in which these normalizations led to erroneous results.

4 Pairwise Matching

The first phase of finding name equivalences is to identify all pairs of names that potentially match each other. As the dataset may be very large (Fig. 4), it is unreasonable to check all pairs. Rather, we first divide the dataset into disjoint subsets according to the normalized last names. Within each such set, we check for equivalences among the normalized first names. This is in harmony with our focus on the variability among names that is concentrated in the first names, that may be abbreviated in different ways.

The following sub-sections pertain to the second step, i.e. to the matching of first names. We first consider the comparison of two names in isolation, and then the comparison of sequences of first names. Note that we take a conservative approach, and do not attempt to correct spelling errors and find names with small edit distances [1]. Rather, we attempt to apply domain-knowledge regarding the way names tend to be abbreviated.

Name	*Regexp*	*Matches*
J$_\circ$	j$_\bullet$$*$	J$_\circ$, JON, JOHN, JOHNNY, JOHNNIE, JO JOE, JOSEPH, J$_\circ$-H$_\circ$, J-H$_\circ$, JIE-HIE
JOHN	john$_\bullet$$*$	JOHN, JOHNNY, JOHNNIE
J$_\circ$-H$_\circ$	j$_\bullet$$*$-h$_\bullet$$*$	J$_\circ$-H$_\circ$, J-H$_\circ$, JIE-HIE

Table 2: *Turning names into regular expressions for matching.* $_\bullet$ *denotes a match to any character, and* $*$ *means that the previous element (in our case, the* $_\bullet$*) is repeated zero or more times.*

4.1 Matching Single Names

When are two names actually one and the same?

The simplest case is, of course, when they are indeed identical. Another simple case is when one is an abbreviation of the other; for example, we can safely identify J$_\circ$ with JOHN. But there are other abbreviations that are also used, notably nicknames.

Nicknames fall into two categories: Those that are a prefix of the full name and those that are not. For example, ALEX can appear as a short version of ALEXANDER. This is a prefix, and can be handled easily as shown below. But BILL may appear as a short version of WILLIAM. This is not a prefix, and can only be identified using table lookup.

Our proposed approach is to only try and identify nicknames that are a prefix of the full name. This is done as follows. Whenever two names are compared, we use the shorter one to create a regular expression, and match this regular expression with the longer one. the regular expression is obtained by removing trailing $_\circ$, if any, and allowing an arbitrary suffix. In the case of hyphenated names, each part is treated separately. This is illustrated in Table 2.

The problem with this approach is that it may also be the case that one name just happens to be a prefix of another name, without being a nickname. For example, RON may be an abbreviation of RONALD, but it may also be a distinct name by itself. Worse, PAUL is a prefix of PAULA, without being an abbreviation. We therefore tabulate all the matches that occurred in practice in our datasets in the appendix. This indicates that the proposed approach identifies many real matches, while only introducing a small number of false matches. In the BoW dataset, for example, 16 correct matches depended on this feature, and only 3 incorrect matches were introduced. In the ACM dataset, there were 24 correct matches and 12 incorrect ones. The most common correct match was Steve as an abbreviation for Steven; the incorrect ones were all different. In both datasets, these incorrect matches were responsible for most of the false positives in the equivalence finding schemes based on strict matching. However, many nicknames are missed because they are not a prefix; in the BoW dataset, there were 13 such cases, and in the ACM dataset 6 were identified (but it is plausible that there were more, as a full manual inspection was not performed). Mike as an abbreviation for Michael is the most common. Thus using table lookup should be considered.

A special case occurs with hyphenated names. Such names may display various alternative forms: the second part can start with and upper-case or lower-case letter (handled by our normalization), the hyphen may be replaced by a space (turning a single hyphenated name into two independent names), or the hyphen may be deleted altogether (turning a hyphenated name into a regular name, possibly with special capitalization). At this point we chose to only give cursory

treatment to these problems. We retain hyphens and try to match both parts, but do not attempt to treat all other variations. The full treatment of hyphenated names is left to future work.

4.2 Simple Matching Algorithm

Given the regular-expression-based mechanism to compare single names, we turn to the comparison of sequences of names. It is obvious that sequences of first names are often abbreviated, and sometimes some of the names are simply not used. However, we offer the observation that names which are retained will appear in the same order (there were only two exceptions found in our datasets). This motivates the following simple matching rule:

> Each name appearing in the shorter list must be matched with a name in the longer list, in the same order.

Note that we do not place a special emphasis on the first name, and do not distinguish it from the middle names. This is because there are quite a few people who prefer their middle names, and therefore tend to discard the first name and retain the middle one when abbreviating.

However, this simple matching rule produces dismal results, and is especially prone to false positives (see top row in Table 1). The reason is that there is no preference to the matching of full names as opposed to abbreviations. For example, it is clear that it is unlikely that DAVID P$_\circ$ is a variant of PAUL, but if we skip the DAVID in the longer name, and then match the initial P$_\circ$ with the name Paul, we abide by the above rule.

4.3 Strict Matching Algorithm

To give full names their proper due, we propose the following rule:

> If we match any abbreviation in a sequence of names, we must also match all full names that appear in that sequence.

Abbreviations, for the purpose of this rule, are single letter names such as J$_\circ$ or J$_\circ$-H$_\circ$. This solves problems like the one posed by the above example, because if we want to match P$_\circ$ with Paul, we must also match DAVID. If we cannot do so we declare that the two name sequences do not match. Note that in implementing this rule we need only count matched names in the longer sequence, because if all names in the shorter sequence are not matched, the matching fails anyway.

However, this does not fully solve the problem. Consider matching the names DAVID P$_\circ$ with the names D$_\circ$ PAUL. This will abide by the rule that all full names need to be matched. What is missing is a sense of directionality: if we match a full name from one sequence to an abbreviation in another, we should not do so in the other direction as well. This is formulated by

> Matching full names to abbreviations should only be done in one direction. For this rule, skipping a full name qualifies as matching it to an abbreviation.

Note that in implementing this rule we should also check names in the longer sequence that are left over after all names in the shorter sequence are matched.

The experimental results in Table 1 show that using the strict matching rules eliminates practically all false positives. We therefore base the quest for cliques of equivalent names on these matching rules.

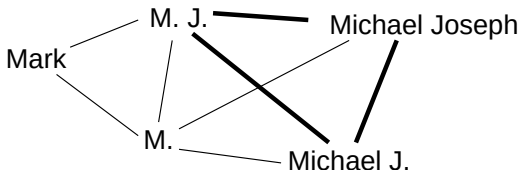

Figure 5: *Example of finding a high-weight clique in a graph of equivalences weighted by names matched.*

5 Choosing Among Alternative Matches

Given the set of pairwise equivalences among names, we want to find sets of names that are all equivalent to each other — in other words, we want to find cliques. Finding maximal cliques is in general NP-complete, but efficient algorithms are known that work on large graphs [8]. In our case in particular this is not a problem, because we work on sets of names that have the same last name, and these are typically of limited size. The variants of clique matching we use are described below.

The description here considers the algorithm as an off-line process. In a real digital library it should be on-line: we have an existing database with equivalences that have been identified in the past, and need to add new authors as they are introduced. But this can easily be done by re-computing all equivalences for the new author's last name, thus reverting to the off-line version.

5.1 Independent Cliques

The simplest approach to identifying cliques of equivalent names is to require independent cliques. This is a simple quadratic algorithm: for each name, traverse its list of equivalences, and verify that this set forms a clique. If they do, they are all considered equivalent. If any of them have additional equivalences outside the set, then the members of this set are not considered equivalent to each other.

The results in the second row of Table 1 indicate that this works pretty well for small repositories like BoW. However, it is too restrictive for large repositories like that of the ACM. In large repositories there may be very many names that share the same last name. It is then highly probable that several names will share an initial, and thus not be independent of each other. We therefore need to allow cliques that have some external neighbors. The difference between the clique members and the external neighbors is that the members are connected to each other more tightly. For this, we need to quantify the strength of the matching between names.

5.2 Weighted Cliques

Weighted cliques are cliques based on high-weight equivalences. The weight is defined simply by the number of names matched: matching three names creates a stronger connection than matching only two, and matching two is better than only one.

A simple example of the effect of such weighting is given in Fig. 5. The initial $M_\circ$ matches all other names, and therefore no independent cliques are possible. But if we use weighting by the number of names matched, we can find that the names $M_\circ$ $J_\circ$, MICHAEL $J_\circ$, and MICHAEL

JOSEPH form a clique of weight 2 (heavy lines in the figure). All the others are connected to this clique by links of weight 1.

The heuristic for finding cliques must now be modified to acknowledge the weights on the links. In particular, the order in which names are considered becomes important. Our solution is as follows, and is illustrated in Fig. 6.

1. First sort the names according to their heaviest equivalence, from heavy to light. In the example the first are $A_\circ$ $B_\circ$ $C_\circ$ and ABE BOB $C_\circ$, then ACE $D_\circ$ $E_\circ$, $A_\circ$ $D_\circ$, and ABE $B_\circ$, and finally $A_\circ$ and ABE $F_\circ$ $G_\circ$. (Note that the maximal equivalence does not necessarily correspond to the number of names.)

2. For each one, identify the set of its neighbors that all have the highest equivalence score. In the example, starting from $A_\circ$ $B_\circ$ $C_\circ$, it has one such neighbor: ABE BOB $C_\circ$.

3. Verify that this set is a high-weight clique. This means that it is a clique when all edges with lower weights are ignored. Cases of only two nodes, as in the example, are sure to pass this test. If the set is indeed a clique, try to expand it. Otherwise return to step 2.

4. Try to expand the found clique by adding names that have lower weight connections. The criterion for adding to the clique are that the new name should be linked to all current clique members, and furthermore, that all its top-weight neighbors should be in the clique. In the example, ABE $B_\circ$ will be added, but $A_\circ$ will not.

5. When finished with this clique, continue with the main loop (step 2), but skip all names that have already been assigned. In the example, this will find another clique composed of ACE $D_\circ$ $E_\circ$ and $A_\circ$ $D_\circ$.

The results in Table 1 show that using weighted matching identifies many of the missing cliques. But when large groups of names are involved, this is not enough. Consider the example in Fig. 7(a). When all names have the same weight, there are so many equal-weight connections that it is impossible for any clique to stand out. But if we give a higher weight to the matching of full names, two obvious candidates stand out (Fig. 7(b)).

The question is what weight to give to the matching of full names. Specifically, should full names count more or less than additional names? Our results indicate that matching more names is more important than matching full names, so we only give full names a sight advantage: a weight of 1.1 as opposed to a weight of 1 when matching an abbreviation.

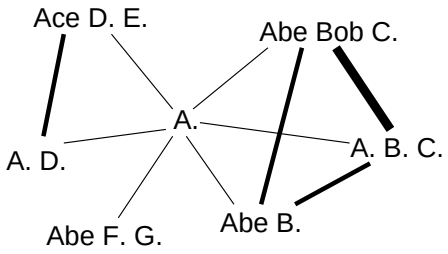

Figure 6: *Illustrative example of the heuristic for weighted cliques.*

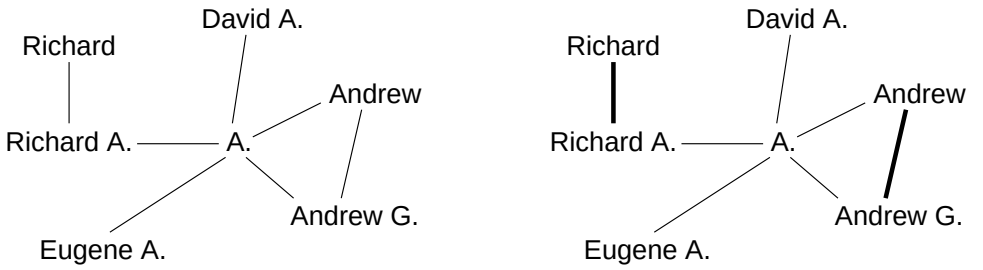

Figure 7: *Adding weight to matching of full names.*

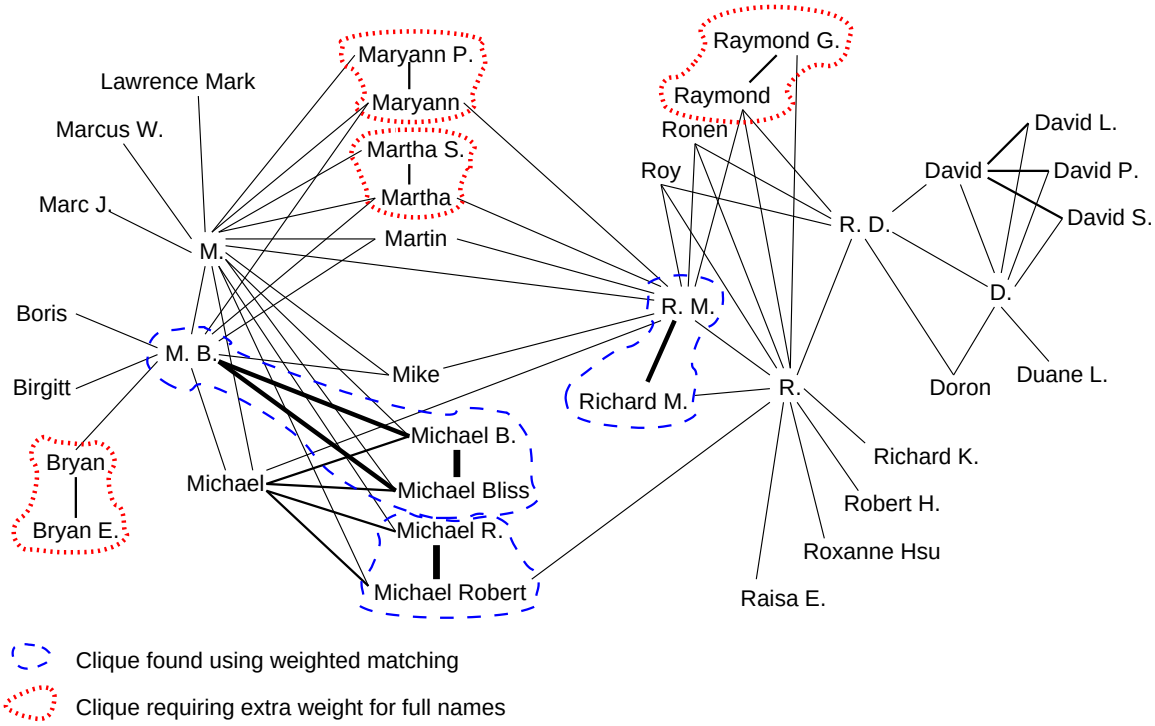

Figure 8: *Largest connected component from the graph of equivalences of people with last name* FELDMAN *from the ACM dataset.*

The final results are shown in the bottom row of Table 1. The improvement is especially striking for the ACM data. The reason is that this is based on a very large database, including some popular names that are repeated dozens of times. As a result, situations such as those portrayed above do occur in practice. An example based on the last name FELDMAN is given in Fig. 8. In this large connected component of the graph, no cliques are found if they are required to be independent. By using weighted cliques, three are found. By giving full names extra weight, another four are found. All of these are considered correct, and no additional real equivalences seem to exist in this example.

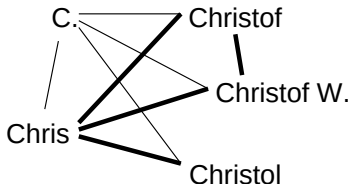

Figure 9: *Correcting a misspelled name (*CHRISTOL*) may allow a new clique to be identified.*

6 Future Work

Our heuristics achieve significant simplifications in the author indices of digital libraries. In Bow, the number of distinct authors is reduced from 5740 to 3580. In the ACM data, it is reduced from 3076 to 1007. In both cases, the vast majority of equivalences found are thought to be true. However, these heuristics only use lexical data, and do not even exhaust this type of data.

The main issue that is yet unresolved is the best handling of hyphenated names. Hyphenation is especially common in Asian and French names. Based on our data is seems that Asian authors at least tend not to abbreviate their names. The rules appropriate to Asian names may therefore be different from those that are best for western names. For example, it seems that partial matching that was good for identifying western nicknames only introduces errors in the context of Asian names. Also, the handling of missing hyphens deserves more work.

Another issue that should be handled is the automatic elimination of errors. Any algorithm can only be as good as its input. It is plausible that at least some misspellings can be caught by comparing with other names already in the database, and selecting the more common version. This can be done based on matching a core code that represents the essence of each name, as was suggested in early applications requiring the retrieval of names [2].

In this context, it should be noted that correcting spelling errors will not only add the misspelled names to existing cliques, but will also allow many additional cliques to be found. The reason is that misspellings cause the current heuristics to think that different names are equally likely; by eliminating such competition, new cliques will emerge (Fig. 9).

A complete new field of study is the use of semantic information. The matching of names can be integrated with checking co-authors, the venues where works are published, and the keywords that appear in the full text of authored articles. It is also possible to glean information from common linking to papers from authors' home pages. Use of such semantic information is expected to be useful for the hardest cases, e.g. when different people actually share the same name.

Finally, the effect of the size of the dataset on the results of the heuristics is very interesting. We showed that as the dataset increases in size, the heuristics need a higher degree of fidelity to extract the real equivalences. It would be valuable to conduct repeated measurements on a variety of database sizes, to verify and characterize this effect.

References

[1] G. B. Bell and A. Sethi, "*Matching records in a national medical patient index*". *Comm. ACM* **44(9)**, pp. 83–88, Sep 2001.

[2] L. Davidson, "*Retrieval of misspelled names in an airlines passenger record system*". *Comm. ACM* **5(3)**, pp. 169–171, Mar 1962.

[3] D. G. Feitelson, "*Cooperative indexing, classification, and evaluation in BoW*". In *7th IFCIS Intl. Conf. Cooperative Information Syst.*, O. Etzion and P. Scheuermann (eds.), pp. 66–77, Springer-Verlag, Sep 2000. Lect. Notes Comput. Sci. vol. 1901.

[4] D. E. Knuth, *The Art of Computer Programming. Vol 3: Sorting and Searching*. Addison-Wesley, 2nd ed., 1998.

[5] L. Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, 2nd ed., 1994.

[6] S. Lawrence, C. L. Giles, and K. Bollacker, "*Digital libraries and autonomous citation indexing*". *Computer* **32(6)**, pp. 67–71, Jun 1999.

[7] L. Philips, "*Hanging on the metaphone*". *Computer Language Magazine* **7(12)**, pp. 38–44, Dec 1990.

[8] E. Tomita and T. Seki, "*An efficient branch-and-bound algorithm for finding a maximum clique*". In *Discrete Mathematics and Theoretical Computer Science*, pp. 278–289, Springer-Verlag, Jul 2003. Lect. Notes Comput. Sci. vol. 2731.

A. Examples of Real data

The following is a listing of data quoted in the text.

In the BoW dataset, the following matches relate to normalization:

- Normalizations that facilitated correct matches
  1. ADAMS, III / ADAMS III
  2. BJØRN / BJORN
  3. BÜLENT / BULENT
  4. CHAO-JU / CHAO-JU
  5. CHUAN-LIN / CHUAN-LIN
  6. BREZÁNY / BREZANY
  7. GARCÍA-MOLINA / GARCIA-MOLINA
  8. HÉCTOR / HECTOR
  9. GÜNTER / GUNTER
  10. JOSÉ / JOSE (2 times)
  11. KI-CHANG / KI-CHANG
  12. LOHR / LÖHR
  13. SHANG-HUA / SHANG-HUA

14. STEELE, JR。 / STEELE JR。
15. THIÉBAUT / THIEBAUT

- Normalizations that caused what are probably incorrect matches

  None found.

- Normalizations that were not identified

1. MÄNNER / MAENNER

In the ACM dataset, the following matches relate to normalizations:

- Normalizations that facilitated correct matches

1. ANTÔNIO / ANTÓNIO / ANTONIO
2. CARRÁ / CARRA’
3. GYÖRGY / GYORGY
4. JÉRÔME / JÉROME
5. JIŘÍ / JIRÍ
6. JOSÉ / JOSE
7. RAPHAËL / RAPHAEL
8. ROGÉRIO / ROGERIO
9. SÁNDOR / SANDOR (2 times)
10. TOMÁS / TOMÀS / TOMAS
11. T.Y. / T.-Y.

- Normalizations that caused what are probably incorrect matches

  None found.

- Normalizations that were not identified

1. BJÖRN / BJOERN

In the BoW dataset, the following matches relate to nicknames:

- Nicknames that facilitated correct matches

1. ALEX / ALEXANDRU
2. ANGELO / ANGELOS
3. DAN / DANIEL
4. GREG / GREGORY (2 times)
5. JEFF / JEFFREY

6. KEN / KENNETH
7. KRITHI / KRITHIVASAN
8. PETE / PETER
9. PHIL / PHILIP
10. PRITH / PRITHVIRAJ
11. RICH / RICHARD
12. RON / RONALD
13. SAM / SAMUEL
14. STEVE / STEVEN (2 times)

- Nicknames that caused what are probably incorrect matches

1. JONG / JONG-UK
2. LI / LIXIA
3. SU / SUKIL

- Nicknames that were not identified

1. AVI / ABRAHAM
2. BILL / WILLIAM
3. CEZARY / CZAREK
4. CHARLIE / CHARLES
5. GARY / GREGORY
6. JIM / JAMES
7. KATHY / KATHERINE
8. MIKE / MICHAEL (3 times)
9. RICK / RICHARD
10. TOM / THOMSON
11. WM / WILLIAM

In the ACM dataset, the following matches relate to nicknames:

- Nicknames that facilitated correct matches (including abbreviations and misspellings)

1. ALEX / ALEXANDER
2. ALEX / ALEXANDRE
3. AW / AWI
4. BRIGIT / BRIGITT
5. CH / CHRISTOPHER

6. CHRIS / CHRISTOPHER (2 times)
7. DEB / DEBORAH
8. DON / DONALD
9. DOUG / DOUGLAS
10. ED / EDWARD (2 times)
11. GREG / GREGORY
12. JEFF / JEFFERY
13. LEO / LEONIDAS
14. PAT / PATRICIA
15. PHIL / PHILLIP
16. RON / RONALD
17. RONALD / RONALDO
18. SID / SIDNEY
19. STEVE / STEVEN (2 times)
20. TH / THOMAS
21. URI / URIEL
22. YA / YAKOV

- Nicknames that caused what are probably incorrect matches

1. ANNE / ANNEMARIE
2. CHIH-CHUN / CHIH-CHUNG
3. DE (as part of SILENE DE FREITAS) / DENIS
4. JOSE / JOSEPH
5. JUN / JUNKANG
6. KE / KEQI
7. MARY / MARY-ELLEN
8. PAUL / PAULA
9. QIAN / QIANGZE
10. TIAN / TIAN-JIN
11. XIANG / XIANG-LI
12. YONG / YONGXIAN
13. ZHANG / ZHANGJUN

- Nicknames that were not identified

1. DAVE / DAVID
2. MIKE / MICHAEL (4 times)
3. RICK / RICHARD