

Personal Named Entity Linking Based on Simple Partial Tree Matching and Context Free Grammar



Sirisuda Buatongkue

Department of Computer Science

Heriot-Watt University

This thesis is submitted for

Doctor of Philosophy

April 2017

I would like to dedicate this thesis to my loving parents ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Sirisuda Buatongkue

April 2017

Acknowledgements

I would like to express my grateful to the following:

First and foremost I would like to thank Dr. Lilia Georgieva, my supervisor, for her encouragement, patience, expert advice and personal support. Secondly, my two examiners have also helped shape this thesis, so would like to express my deep gratitude to Valentina Dagiene and Idris Skloul Ibrahim for their time, guidance, effort and care. Thirdly, Ministry of Science and Technology and Thai Government Student's Office for financial support and encouraging throughout my PhD studies. Finally, special thanks go to my friends, my teachers and family who have supported me throughout my research.

Abstract

Personal name disambiguation is the task of linking a personal name to a unique comparable entry in the real world, also known as named entity linking (NEL). Algorithms for NEL consist of three main components: extractor, searcher, and disambiguator.

Existing approaches for NEL use exact-matched look-up over the surface form to generate a set of candidate entities in each of the mentioned names. The exact-matched look-up is wholly inadequate to generate a candidate entity due to the fact that the personal names within a web page lack uniform representation. In addition, the performance of a disambiguator in ranking candidate entities is limited by context similarity. Context similarity is an inflexible feature for personal disambiguation because natural language is highly variable.

We propose a new approach that can be used to both identify and disambiguate personal names mentioned on a web page. Our NEL algorithm uses: as an extractor: a control flow graph; AlchemyAPI, as a searcher: Personal Name Transformation Modules (PNTM) based on Context Free Grammar and the Jaro-Winkler text similarity metric and as a disambiguator: the entity coherence method: the Occupation Architecture for Personal Name Disambiguation (OAPnDis), personal name concepts and Simple Partial Tree Matching (SPTM). Experimental results, evaluated on real-world data sets, show that the accuracy of our NEL is 92%, which is higher than the accuracy of previously used methods.

Table of contents

Table of contents	xi
List of figures	xv
List of tables	xix
Nomenclature	xx
1 Introduction	1
1.1 Thesis contributions	5
1.1.1 Occupation Architecture for Personal Name Disambiguation (OAP- nDis) and Personal Name Concepts	6
1.1.2 Personal Name Transformation Modules (PNTM) based on Context Free Grammar (CFG)	7
1.1.3 A New Algorithm for Personal Name Disambiguation with Simple Partial Tree Matching (SPTM)	8
1.2 Thesis Outline	9
1.3 Draft Papers	10
2 Theoretical Background	11
2.1 Ontology and Entity on Knowledge Base	12
2.1.1 Wikipedia	12
2.1.2 YAGO	14
2.1.3 Freebase	18
2.1.4 DBpedia	19
2.2 Named Entity Disambiguation	21
2.2.1 Named Entity Disambiguation Framework	22
2.3 Tree Matching	26
2.3.1 Simple Tree Matching (STM)	27

2.3.2	Partial Tree Alignment(PTA)	29
2.4	Conclusion	32
3	Personal Name Surface Form and OAPnDis	35
3.1	Motivation	35
3.2	Personal Name Surface Form Modules(PNSFM)	37
3.2.1	Data Pre-Processing	39
3.2.2	Data Extracting	41
3.2.3	Data Matching	43
3.3	Occupation Architecture for Personal Name Disambiguation(OAPnDis)	46
3.3.1	OAPnDis Architecture	47
3.3.2	Personal Name Concepts	49
3.4	Personal Name Disambiguation Data Catalogue (PNDDC)	52
3.5	Experimental Results and Discussion	54
3.5.1	Data Sets	55
3.5.2	Result and Discussion	55
3.6	Related Work	58
3.7	Conclusions	59
4	Personal Name Transformation With Context Free Grammar	61
4.1	Motivation	61
4.2	Background	64
4.2.1	English Personal Names	64
4.2.2	Context Free Grammar (CFG)	67
4.2.3	String Matching	70
4.3	Personal Name Transformation Modules (PNTM)	73
4.3.1	Context Free Grammar Rules (CFG Rules)	75
4.3.2	Predicate	79
4.3.3	Action	83
4.4	Experimental Results and Discussion	89
4.4.1	PNTM Assessment	90
4.4.2	Text Similarity Metrics Comparison	91
4.5	Related Work	93
4.6	Conclusion	95
5	Personal Name Entity Linking	97
5.1	Motivation	97

5.2	Background	99
5.2.1	AlchemyAPI	100
5.3	Personal Name Entity Linking Framework (PNELF)	101
5.3.1	Personal Name Extractor	102
5.3.2	Searcher	103
5.3.3	Personal Name Disambiguator	106
5.4	Predicting the NIL Value	109
5.5	Simple Partial Tree Matching Algorithm	110
5.5.1	Building the Comparison Tree	111
5.5.2	Simple Partial Tree Matching	112
5.6	Experimental Results and Discussion	113
5.6.1	Data Sets	114
5.6.2	Evaluation Results	115
5.7	Related Work	119
5.8	Conclusion	123
6	Software Specification	125
6.1	Introduction	125
6.1.1	Goals and Objectives	125
6.1.2	System Overview	126
6.1.3	System Functions	128
6.2	Data Design	129
6.2.1	Internal Data Structures	130
6.2.2	Database Description	130
6.3	Work Flow of the Process	131
6.4	User Interface Design	136
6.5	Testing Issues	139
6.5.1	Software summary	139
6.5.2	System Testing	141
7	Conclusion and Future Work	149
7.1	Conclusion	149
7.2	Future Work	151
	References	153
	Appendix A Abbreviation Words Glossary	159

Appendix B Thesis Diagrams	161
Appendix C Reviews of Tools in Fact Extracting and Fact Answering	165
C.1 YAGO	165
C.2 Facts answering from the Bing and Google search engines	169
C.3 Open Information Extraction: REVERB	174
C.3.1 Conclusion	176

List of figures

2.1	YAGO Structure [42]	15
2.2	Snapshot of Freebase Structure [30]	19
2.3	Number of instances in English per class [44]	20
2.4	Snapshot of a part of the DBpedia ontology [44]	21
2.5	Examples of referential ambiguity and differing orthography [50].	21
2.6	Simple Tree Matching (a) Tree A; (b) Tree B; (c) W matrix for the first-level sub-trees; (d) M matrix for the first-level sub-trees [75].	28
2.7	The Simple Tree Matching Algorithm [79].	28
2.8	Expanding the seed tree: (a) and (b) unique expansion and (c) insertion ambiguity [81].	30
2.9	Partial Tree Alignment with multiple trees [81].	31
3.1	PNSFM modules	37
3.2	An example of the dirty data in YAGO knowledge base.	40
3.3	An excerpt of YAGO entertainer class taxonomy	45
3.4	OAPnDis architecture	47
3.5	An excerpt of the occupation taxonomy	49
3.6	A Modified Preorder Tree Traversal(MPTT) algorithm	50
3.7	Examples of personal name concepts of <i>Arnold Schwarzenegger</i> : (a) Politician and (b) Entertainer and Artist	52
3.8	PNDDC structure	54
3.9	Percentage of lexical ambiguity compare with Han and Zhao [37].	57
4.1	Regular grammar for English personal names.	65
4.2	Example of CFG (a) Production rules (P) and (b) Left-most derivations represented as a tree.	69
4.3	Personal Names Transformation Module.	74
4.4	Example of alternative names in a personal name dictionary.	80

4.5	Example of nicknames in a personal name dictionary.	82
4.6	Accuracy comparison of personal name matching among different methods	92
5.1	Example results from GATE information extractor	100
5.2	Personal Name Entity Linking Framework	101
5.3	An example of a personal name extractor module (a) input: a web document and (b) output: alchemy API output values in XML format.	103
5.4	Example of comparison tree	107
5.5	An example conceptual matching score	109
5.6	Example of browsing the NIL value with BingAPI	110
5.7	An example of a flattened tree (a) Original tree (b) Flatten tree	111
5.8	Building the comparison tree (a) initial trees and (b) comparison trees . . .	112
5.9	An example of a simple partial tree matching algorithm	113
5.10	Notation for evaluation searcher performance [34]	115
5.11	Searcher performance.	117
5.12	Disambiguator performance.	118
6.1	Personal name matching screenshot	126
6.2	Personal name matching use case diagram	128
6.3	Personal name matching entity relationship diagram	130
6.4	Personal name matching flowchart	133
6.5	Personal name matching flowchart	134
6.6	Home page	136
6.7	Personal name transformation page	137
6.8	Generating a set of candidate entity pages	137
6.9	The results of Simple Partial Tree Matching algorithm	138
6.10	Final results in personal name matching page	138
6.11	Top ten links of possible people	139
6.12	The URL input form	141
6.13	A set of mentioned names within a web page	142
6.14	A set of mentioned names with prefix and/or suffix transformation under PNTM	143
6.15	A set of mentioned names without prefix and/or suffix transformation under PNTM	143
6.16	A set of candidate entities that are generated from the searcher component .	145
6.17	A comparison tree	146
6.18	Personal name entity linking	148

B.1	Chapter 3: PNSFM modules	161
B.2	Chapter 4: Personal Names Transformation Modules	162
B.3	Chapter 5: Personal Name Entity Linking Framework	163
C.1	An example of entity search about <i>Albert Einstein</i> in YAGO.	166
C.2	An example question and triple patterns with SPARQL syntax.	167
C.3	An (a) SPOTL(X) interface.	167
C.4	SPOTL(X) query result within table form.	168
C.5	The query result from YAGO2 using SPOTLX.	169
C.6	The query results about: JK Rowling birthday from Google.	169
C.7	The query results about JK Rowling birthday from Bing.	170
C.8	The query results about Tom Cruise spouse from Google.	172
C.9	The query results about Brad Pitt spouse from Google.	172
C.10	The query results about Tom Cruise's spouse from Bing.	173
C.11	The query results about Brad Pitt's spouse from Bing.	173
C.12	The query results for " <i>Where was Justin Bieber born?</i> " from Google.	174
C.13	The query results for " <i>Where was Justin Bieber born?</i> " from Bing.	174
C.14	The user interface of the REVERB extraction system	175
C.15	REVERB user interface	176

List of tables

2.1	YAGO's classes distribution [22]	16
2.2	YAGO's instances distributio [22]	17
3.1	The Wikipedia disambiguation page focuses on the personal name: <i>George Bush</i> [71].	36
3.2	Notations used in PNSFM modules	39
3.3	Data set-up	40
3.4	The unique personal name entities in our data catalogue	42
3.5	The data sets in the personal name disambiguation data catalogue	55
3.6	The personal name data catalogue and personal name concepts	56
4.1	Example of personal name variations.	62
4.2	Regular grammar for English personal names.	68
4.3	Example of cosine similarity measurement.	71
4.4	A generative grammar to capture personal name variations.	76
4.5	Personal Name Transformation Rules based on CFG.	77
4.6	An excerpt from given name and family name dictionary.	79
4.7	Example of traditional English nicknames	81
4.8	Example of prefix and suffix	82
4.9	The personal name transformation description.	85
4.10	Expanded framework for processing the personal name <i>George W. Bush, Jr.</i>	87
4.11	Expanded framework for processing nickname <i>Gates, Bill.</i>	88
4.12	Expanded framework for processing alternative name: <i>Bush.</i>	89
4.13	Example of datasets.	90
5.1	Summary data sets.	115
6.1	Software Summary	140

C.1 The contrast between YAGO2, Fact look up in Google search engine, FACTO and Reverb. 177

Chapter 1

Introduction

As a large amount of data on the internet is about people whether in news stories, books, songs, films and sports. The identification of personal names is a key task for areas such as search engines, information retrieval and machine translation. It is the case that of the top five searches in the world in 2013, which are all from the Google search engine, three of them are people's names [31].

Named Entity Disambiguation (NED) is name given to the task of matching the entity that is mentioned in a document to its comparable entry in a knowledge base [34, 45] (e.g. Wikipedia). It is also known as Name Entity Linking (NEL) or record linkage [34].

Dredze et al. [24] describes three challenges in personal name matching: referential ambiguity, lexical ambiguity and predicting the NIL value.

Firstly, referential ambiguity or name variation concerns different names referring to the same person [7]. Typically, an English personal name consists of three parts: the given name, the middle name and the family name, for example *George Walker Bush*. However, there are varying styles used to represent a personal name, such as nicknames, pen names, aliases, short names and abbreviations. For example, *President Bush*, *Dubya Bush* and *Bush* refer to *George W. Bush, 43rd President of the United States*. Furthermore, some personal alias will be completely different from the real name, such as *The Governator* in reference to *Arnold Schwarzenegger*. Additionally, a single name can be represented in multiple patterns. For example, the personal name *Prof. Philippe De Wilde* can be represented at least five different forms.

1. *Prof Philippe D. Wilde.*
2. *Prof P. D. Wilde.*
3. *Prof P. De Wilde.*

4. *Wilde, Philippe De.*

5. *Philippe De Wilde.*

Secondly, lexical ambiguity means that a single name may refer to multiple persons [7]. For example, *Chris Martin* is the name of at least four different people in the public eye:

1. *Chris Martin (born 1977)*, the English front-man of Coldplay.
2. *Chris Martin (artist) (born 1954)*, American painter.
3. *Chris William Martin (born 1975)*, Canadian actor.
4. *Chris Martin (footballer, born 1988)*, Scottish striker for Derby County.

Finally, predicting the NIL value is the technique that is used when the system cannot map the mentioned name to a person in a knowledge base [24].

Hachey et al. [34] introduces a framework for NEL, which involves the use of the tasks of extractor, searcher and disambiguator. Extracting is a task for detecting personal name entities that are mentioned in a document. Searching is the task of generating a set of candidate knowledge base entities to each name mentioned in a document. Disambiguating is the task of best matching an entity with a name when there is ambiguity in the personal name. Certain studies [8, 19, 63] have only focused on precision in the disambiguator task, but Hachey et al. [34] found that the searcher component is more important and has a much larger effect than the disambiguator task on system achievement. Therefore, personal name entity linking tasks not only disambiguate lexical ambiguity but also referential ambiguity. Furthermore, most real-world data is noisy, incomplete or has imprecisely formatted information [16]. Hence, matching the exact term over the surface form that is used in [8, 19] may not be able to process a set of candidate entities for a mentioned personal name. Surface form is a collection of terms that are used to refer to people. For example, matching the exact term returns different results for these two terms: *Barack Obama* vs. *Barak Obama* though both are the same personal name. This is because the letter *c* is missing in the second term.

To solve this problem, we propose Personal Name Transformation Modules (PNTM) based on Context Free Grammar (CFG) and the Jaro-Winkler text similarity metric. PNTM is based on the technique introduced in [3], which transforms personal name variations to a uniform representation. A CFG has advantages because of three reasons. Firstly, CFG is flexible for personal name variations because it can capture multiple name formats. Secondly, CFG can solve the misleading problem in text similarity measurement (e.g. edit distance, cosine similarity or Jaro-Winkler) when the same person is represented using highly

different text (e.g. *Bill Gates* and *William Gates*) by transforming the nickname *Bill* to the given name *William*. Thirdly, CFG is different from other transformation tools because it allows us to understand the internal structure of personal names. For example, the text *Bill* only transfers to *William* when dealing with given names.

The main disadvantage of CFG is that of an ambiguity parse tree, where a single personal name may be produced into more than one parse tree. Arasu and Kaushik [3] solve this problem by using a uniform weight scheme score to rank the candidate names, and then selecting the candidate with the highest score. However, this method suffers when a context involves both given name and last name, because the scores are equal. In our work, a regular grammar in personal name structure is used for segmentation of the components in personal names (specify the location of first name, middle name or last name) and these components are directly matched to the personal name dictionary.

The Jaro-Winkler is designed to deal with typographical errors [25]. Various studies [6, 15, 54, 74] compare the Jaro-Winkler metric to the Levenshtein, Q-gram, Smith-Waterman and TF-IDF metrics. The experimental results show that the Jaro-Winkler metric performs well in terms of personal name matching.

In disambiguator components, several methods [8, 19, 24, 63] have been proposed to handle the lexical ambiguity problem. These methods can be divided into two broad types: the context similarity method and the combining method. The context similarity method or *a bag of word* has been used in [8, 24]. Context similarity uses the terms around the entity mentioned and the Wikipedia page that related with entity to measure similarities between two entities [45, 63]. However, the limitation of the context similarity method is that it requires exact word overlap between the two compared texts, which may become an over-strict constraint because of the flexible use of natural language [45].

Secondly, the combination method is designed for merging multiple techniques together. For example, Cucerzan [19] combines two techniques (context similarity and topical coherence) and Shen et al. [63] integrates three techniques (context similarity, entity popularity and topical coherence) for personal name disambiguation. Topical coherence uses the Wikipedia cross-page links for discovering unity between two entities [45]. This method is based on the assumption that a mentioned entity should be interdependent with other entities in the same document [59]. The entity popularity is a statistical feature that used probability of the mentioned name refer to each entity [45]. However, the effectiveness of topical coherence is limited by the incorrect category in the knowledge base. This is because researchers used knowledge based categories to determine if the entities are coherent. Cucerzan [19] is based on Wikipedia categories to find an interdependence score between ambiguity entity and identification entities. However, Shen et al. [63] argues that the categories in Wikipedia

are dirty and therefore not well-formed. Wikipedia, Freebase and web directories use the thematic domain to classify articles or entities. However, the thematic domain is limited by claiming to represent the correct concepts in each entity [40]. For example, *Stephen King* is assigned to the categories:

Stephen King → *Writers of books about writing fiction* → *Fiction*

but *Stephen King* is not a work of fiction. Shen et al. [63] resolves the problem by changing the knowledge base to YAGO. YAGO has a clean taxonomy of concepts that derived from WordNet and Wikipedia [63]. However, Demidova et al. [22] found that YAGO ontology is noisy, some of the WordNet classes in YAGO are not associated with Wikipedia categories. For example, *Presidents of Clemson University* is under the WordNet class:

head of state president → *representative* → *negotiator* → *communicator* → *person*

but *Presidents of Clemson University* is not the *head of state president*.

To handle the dirty data in Wikipedia and YAGO, we introduce Occupation Architecture for Personal Name Disambiguation (OAPnDis). OAPnDis is a new occupation taxonomy architecture based on web directories and YAGO ontology. OAPnDis consists of four layers: *Person*, *web directory classes*, *YAGO-WordNet classes* and *YAGO-Wikipedia classes*. The two steps: the editing of proper names and consideration of whole names are added to ensure that our occupation taxonomy is clean, well-formed and semantically. Firstly, the thematic domain problem is handled by changing the context in web directories before mapping to our occupation taxonomy (e.g. Arts to Artists). Secondly, the whole name in the Wikipedia category is evaluated before mapping it into the WordNet class. YAGO considers only the head word of category in Wikipedia for linking a Wikipedia category to a WordNet class. This is the source of unrelated links in YAGO taxonomy. For example, *Presidents* is a head word of the category *Presidents of Clemson University* in Wikipedia, so it is mapped to *head of state president* in WordNet class.

To deal with the context similarity problem, we propose a new approach: an entity coherence using Simple Partial Tree Matching (SPTM). We assume that personal names appearing within a single web-page have the same concept or that they are related (e.g. spouse, child). SPTM is based on two algorithms: Simple Tree Matching (STM) and Partial Tree Alignment (PTA). STM is a two-tree matching algorithm that is introduced in [75] to compare the similarity between two computer programmes. PTA was introduced in [81] to align data items from the identified records for the task of web data extraction. STM is advantageous for two reasons. Firstly, it is a top-down algorithm; the two root nodes are

matched first, and if the root nodes are dissimilar this means that two trees are different. Secondly, the algorithm does not allow node replacement or level crossing for computing the maximum number of similarity nodes between two trees. However, giving an equal weight to every nodes may reduce the performance in similarity matching because the levels in occupation taxonomy have different priorities. Additionally, it is insufficient and time consuming to compare the personal name concepts of candidate entities to every personal name concepts in a list. As the result, progressively extending a seed tree technique that is used in PTA algorithm gives us the idea of creating a comparison tree to reduce the matching frequency. SPTM performs a lexical ambiguity and selects the best personal name entity for each ambiguity mentioned using three steps:

1. An individual concept in each personal name entity is generated based on our occupation taxonomy architecture.
2. The identification personal name which has the same root node is merged to create the comparison tree. The comparison tree which has the maximum number of nodes will be considered first.
3. A set of candidate entities is ranked by comparing the similarity between each candidate entity concept and comparison tree. The similarity score is calculated from the number of matching nodes and their weights. The nodes of the different hierarchical levels have different weights. The candidate entity which has the highest score is selected.

Our proposed method is different from the existing approaches because only mentioned personal names are used to disambiguate personal name ambiguity.

Finally, the approach returns the NIL value when a set of candidate entities in each mentioned name is empty or the conceptualisation of identifications of person differs from other identification person in a web page. Furthermore, the possible persons in each NIL value are predicted by attaching the NIL value entity and the popular occupation from a list of personal name concepts in each web page through the BingAPI.

1.1 Thesis contributions

The objective of this thesis is to establish and assess the new techniques to identify and disambiguate personal names that are mentioned in a web document by linking them to the comparable entry in a knowledge base. We propose a Personal Name Entity Linking

Framework (PNELF) that focuses on two problems in personal name entity linking: the searcher component and the disambiguator component.

Firstly, PNTM and Jaro-Winkler text similarity metric are integrating to improve the searcher performance. PNTM is used to transform name variations to a uniform format. Furthermore, Jaro-Winkler, the string similarity matching is used to instead the exact-match look-up over personal name surface form for generating a set of candidate entities. PNTM and Jaro-Winkler can solve three main problems in referential ambiguity: multiple spellings or typographical errors (e.g. *Beckham* vs. *Beckam*), rearrangement of words (e.g. *Beckham, David* vs. *David Beckham*) and using different words to refer to the same person (e.g. *DB7* vs. *David Beckham*)

Secondly, the disambiguator performance is boosted by solving two problems in NED, including context similarity that requires exact words to overlap between the two compared documents, and the dirty data in Wikipedia categories and YAGO ontology. OAPnDis and occupation taxonomy are used to generate personal name concepts in each person. Furthermore, a new algorithm SPTM requires only personal name concepts to handle lexical ambiguity problems and link a mentioned personal name to a real-world entity.

The effectiveness of PNELF is empirically validated through experimental assessments with real-world data sets. The precision in personal name entity linking is 91.82%. The contributions of this thesis are described below.

1.1.1 Occupation Architecture for Personal Name Disambiguation (OAPnDis) and Personal Name Concepts

Occupation is an advantageous feature that is used to distinguish ambiguity of person [48]. However, the evidence from the literature reviews shows that none of the existing knowledge base ontology is suitable to establish personal name concepts [22, 63]. Wikipedia and Freebase categories are noisy, not well-formed and use the thematic domain structure. The taxonomy in YAGO knowledge is derived from WordNet as a backbone and is connected with Wikipedia categories. However, Demidova et al. [22] argues that some of the WordNet classes are not related to Wikipedia categories. For example, Wikipedia categories *President of FIFA* is under the WordNet class *head of state president*. Finally, DBpedia ontology is well organised, but the hierarchy level is not deep enough to create personal name concepts. To fill the gap in knowledge base taxonomy problems, this thesis introduces OAPnDis; a new architecture for occupation taxonomy. The architecture includes four layers and bases on YAGO ontology and web directories. Layer 0 is a root node to define that an entity assigned under this layer is a person. Layer 1 is derived from web directories that are used

to distinguish personal names in a big picture. Layer 2 is derived from WordNet in YAGO, and layer 3 is derived from Wikipedia categories in YAGO. The conceptualisation in each personal name entity is created from this architecture.

Experimental results on the task of personal name catalogue and personal name concepts show that the maximum number of concepts for each person is two. Only 0.06% had two concepts, which means that people usually work in one career. Additionally, the experimental results showed that 603 (4.84 %) people share both name and occupation (less than 5%), which is similar to the study in [37]. The experimental results suggest that occupation taxonomy is a useful feature to distinguish persons when there is lexical ambiguity in the name.

1.1.2 Personal Name Transformation Modules (PNTM) based on Context Free Grammar (CFG)

Data on the Internet lacks uniform representation because of the heterogeneous data design; as the data comes from multiple sources, it cannot be handled as if it were a single database [11]. Dirty data arises for various reasons, such as lack of design, data entry mistakes and misunderstandings [56]. We can find the context for a name such as *Barak Obama* for *Barack Obama* or *Beckham* for *David Beckham* may be appeared within a web document. However, the previous studies [8, 19, 24, 63] do not focus on cleaning data that are extracted from a web page before it is matched with a personal name surface form.

The study [34] suggests that the searcher component in an NED framework has a larger impact on the system effectiveness than the disambiguator component. This is because personal names that appear on the Internet are variations (referential ambiguity). Therefore, the exact-match lookup over personal surface forms that are used in some studies [8, 19] are insufficient to detect candidate entity. For example, the exact-match lookup cannot detect that *Barak Obama* and *Barack Obama* are the same personal name.

To deal with referential ambiguity problem, we introduce PNTM to transform multiple formats of personal name to a uniform representation. PNTM is based on the study in [3] and consists of three components: grammar rules, predicate and action. The sixteen CFG rules have been created to handle referential ambiguity problems (e.g. different order, nick name, alternative name) to a uniform format. Predicate is a personal name dictionary, which includes the first name, last name, nickname and alternative name. An action is a component to process a result. A single personal name input may return multiple results. For example, the input name *Bill Gates* produces two output names: *William Gates* and *Willis Gates*. A personal name transformation with CFG rules is advantageous because it boosts

the precision and recall in generating a set of candidate entities in each mentioned name.

The experimental results show that PNTM can solve the problem of referential ambiguity by transforming names variations to a uniform format. Furthermore, PNTM and the Jaro-Winkler metric can boost the performance in text similarity measurements.

1.1.3 A New Algorithm for Personal Name Disambiguation with Simple Partial Tree Matching (SPTM)

PNELF consists of three main components: extractor, searcher and disambiguator. "*alchemy API*" is an extractor that be used to extract names mentioned within a web page. In searcher component, the CFG framework is used to transform the personal name to a uniform representation and Jaro-Winkler is used for detecting a set of candidate entities in the personal surface form. To deal with the context similarity problem, the new algorithm SPTM and personal name concepts have been used to disambiguate ambiguous names. A context similarity compares the similarity between a pair of entities (an entity that is mentioned in a web document and an entity in a Knowledge base) by detecting similar terms that occurred around entities in a web document and knowledge base. A limitation in context similarity is flexibility in natural language; we cannot used different words to refer to the same meaning. For example, the given name *William* or *Willis* can be replaced by nickname *Bill*. Our method is different from some studies including [8, 19, 24, 63] in that this thesis uses only mentioned names within a web document are used to disambiguate personal names.

We assume that personal names that have appeared within a single web page have the same conceptualisation or they are related (spouse or child). Layer 1 (e.g. Entertainers and Artists, Sportsmen and Politicians) is a root node for grouping people together. People who have the same concepts mean they have the same root node, but the child nodes may be different.

SPTM means that the two trees matching when computing the similarity score between a candidate conceptual tree and a comparison tree. The depth of node in a tree is used to assign a weighting score to a node (deeper depth equalling higher weight). The comparison tree is created by joining the nodes of identifying people who have the same root node. The comparison tree which has the maximum number of nodes will be selected first. The similarity score is calculated from the total matching node multiplied by the total weighting score. The candidate who has the highest score is selected.

The experimental results demonstrate that our proposed approach performs well when using real-world data sets.

1.2 Thesis Outline

The thesis is laid out as follows.

Chapter 2. Theoretical Background: this chapter reviews the theoretical background and techniques that are related to our work are reviewed. This chapter consists of three parts. The first part discusses knowledge based, ontology and entities from four main sources: Wikipedia, YAGO, DBpedia and Freebase. The second part describes the NED framework and the techniques that are used in NED are described. The final part reviews two algorithms: Simple Tree Matching (STM) and Partial Tree Alignment (PTA), which gave us the idea of introducing the new algorithm Simple Partial Tree Matching (SPTM).

Chapter 3. Personal Surface Form and OAPnDis: this chapter consists of two parts. The first part describes Personal Name Surface Form Modules (PNSFM) that are used to generate personal surface form, occupation categories, the occupation of each personal name and personal name entity relations. The second part introduces OAPnDis, our new occupation taxonomy and discuss how to create personal name concepts based on OAPnDis.

Chapter 4. Personal Name Transformation with Context Free Grammar: this chapter presents Personal Name Transformation Modules (PNTM) based on the CFG and Jaro-Winkler text similarity metric. PNTM and Jaro-Winkler are used to solve three main problems of referential ambiguity or name variations: multiple spelling or typographical errors, rearrangement of words, and the use of different words such as an alias, a nickname or an alternative name to refer to a person.

Chapter 5. Personal Name Entity Linking: this chapter introduces Personal Name Entity Linking Framework (PNELF), including the personal name extractor, the searcher and the personal name disambiguator. In searcher component section describes how to improve the searcher performance using PNTM, Jaro-Winkler and personal name surface form. The final part presents the techniques which are used to link a mentioned name to a real-world entity and discuss how to predict the NIL value. Furthermore, this chapter introduces SPTM and personal name concepts that are generated based on OAPnDis to handle the lexical ambiguity problem.

Chapter 6. Software Specification: this chapter discusses our software specification including system function, data design, database description, the work flow of the process, the user interface design and software testing.

Chapter 7. Conclusion and Future Work: this chapter discusses our conclusion and future work in personal name entity linking.

1.3 Draft Papers

1. An Occupation Ontology Architecture for Crating Personal Name Conceptualisation. To be submitted to AAAI 17.
2. New Framework for Personal Name Disambiguation. A poster presentation (short paper) COLING-2016.
3. Personal Name Matching (Presented at Heriot-Watt Conference December 2014).

Chapter 2

Theoretical Background

In this chapter, we discuss the theoretical background of the name entity disambiguation (NED) framework, and the techniques used in the NED area (i.e. context similarity [8, 19, 63] and topical coherence [19, 62, 63]). Extracting personal name from a web page and mapping them on to a real-world entity can add value to data and give great support to another services like search engines, e-commerces, or travel agencies. However, Dredze et.al. [24] claimed that there are three fundamental problems with NED: lexical ambiguity, referential ambiguity and named entity absence from a knowledge base. Accordingly, matching an entity which is extracted from a web document to a real-world entity requires more effort and greater complexity than with an entity which has come from a database. This is because the web page usually provides partial or incomplete information about each entity [76].

There are two main problems in NED. Firstly, referential ambiguity means that a single person can be referred to in lots of different ways [7]. Secondly, lexical ambiguity means a single name may be referred to multiple persons [7]. Therefore, text similarity measurement is insufficient in NED tasks. The information in knowledge bases such as Wikipedia or YAGO provide useful information in NED tasks and are widely use in NED researches [8, 19, 63]. Therefore, this thesis discusses two features in knowledge bases that are widely used in the area of NED: entity names and knowledge base taxonomy.

This chapter organises the contents into three parts: Section 2.1 discusses knowledge based ontology and entities from four main sources: Wikipedia, YAGO [42, 64], DBpedia [44] and Freebase [30]. Section 2.2 introduces the NED framework and the techniques are used to disambiguate named entities. Finally, Simple Tree Matching [75] and Partial Tree Alignment [79–81] the techniques on which personal name disambiguation are based are reviewed.

2.1 Ontology and Entity on Knowledge Base

Gruber [32] uses the term "ontology" to refer to the "explicit specification of a conceptualisation". The sense in which he uses the word "conceptualisation" [33] is an "an abstract, simplified view of the world that we wish to represent for some purpose". In a knowledge base, the classes and entities are the key components in presenting entity conceptualisation. Class is the hierarchy of the elements that are used for grouping similar entities together. Therefore, the main structure of the knowledge base includes both classes and entities.

2.1.1 Wikipedia

Wikipedia is a free multilingual web-based encyclopedia operated by volunteers via wiki software. An article in Wikipedia is normally represented in an individual topic. Each article has been manually allocated to at least one category in Wikipedia. Categories in Wikipedia are a group of articles that have similar topics. Within a particular page on Wikipedia there is a box containing a list of categories to which an article belongs, which is found at the end of the of page. For example, the *Elton John's* article is assigned to categories: *Elton John, 1947 births, 20th-century composers, 20th-century English male actors, 20th-century English singers* as well as a further 59 categories.

The Portal:Contents/Portals is the Wikipedia root category, and is divided into twelve portals as follows:

- General reference
- Culture and the arts
- Geography and places
- Health and fitness
- History and events
- Mathematics and logic
- Natural and physical sciences
- People and self
- Philosophy and thinking
- Religion and belief systems

- Society and social sciences
- Technology and applied sciences

Wikipedia also divides people into the following broad categories:

1. Association e.g. *People by educational institution, People by company.*
2. Ethnicity, gender, religion, sexuality, disability, medical or psychological conditions.
3. Personal name.
4. Nationality and occupation
5. Place, the place of birth or notable residence. e.g. *Category:People from New York*
6. Year, people are categorised by year of birth and year of death.

Wikipedia categories are organised in the form of hierarchical structures (parent and child categories). However, the hierarchical structure is not well-formed; an article and category do not strictly order (there are many-to-many relationship between articles and categories).

The Wikipedia categories are less organised, longer information, have irrelevant parts and may be inaccurate [66]. Moreover, the category's hierarchy reflects merely the thematic structure so it is of limited use in terms of the ontological purpose [64] because it can not represent a particular instance correctly. For example, *Zinedine Zidane* is assigned under the categories:

French_footballers → *Association_football_players_by_nationality* →
Association_football_by_country → *Association_football* → *Team_sports*.

However, *Zinedine Zidane* is a person and a football player, he is not a football association or a football team.

Wikipedia deals with personal name ambiguity by using various features to make a personal name unique. Wikipedia uses a single feature or combining features to distinguish people who have the same name such as occupation, date of birth, nationality or place of residence. However, occupation is the first priority feature that is used in Wikipedia to handle the problem of lexical ambiguity. The features that are used in Wikipedia for making a unique personal name are as follows:

- Occupation e.g. *Chris Brown (composer), Chris Brown (ice hockey)* and *Chris Brown (dancer)*.

- Date of birth e.g. *Chris A. Brown (born 1964)* and *Christopher J. Brown (born 1971)*.
- Nationality and occupation e.g. *Chris Brown (Canadian musician)* and *Chris Brown (Australian musician)*.
- Place of residence and occupation e.g. *Chris Brown (California politician)* and *Chris Brown (Mississippi politician)*.
- Occupation and date of birth e.g. *Chris Brown (footballer, born 1992)*, *Chris Brown (footballer, born 1971)* and *Chris Brown (footballer, born 1984)*.

Wikipedia is the fundamental source in a knowledge base and a large number of entities and their types stored in the knowledge base are derived from Wikipedia. Therefore, personal name entities and their categories in Wikipedia are the major elements used to create our data catalogue and personal name concepts.

2.1.2 YAGO

YAGO [42, 64] is an ontology and part of the YAGO-NAGA project developed at the Max Planck Institute for Informatics. YAGO stores information in the form of RDF triples: subject(S), property(P) and object(O). This SPO is called a fact. For example,

Nicolas Sarkozy (S) PresidentOf(P) France(O)

is a fact has been stored in YAGO. YAGO collects individual entities and their categories from Wikipedia 'infoboxes' and links them to the clean taxonomy of WordNet. YAGO contains 365,372 classes, 2,648,387 entities and 104 relations [40]. The taxonomy of YAGO is well-formed and meaningful. For example, in the instance shown in Figure 2.1 *Zinedine Zidane* is a soccer player and he is a person.

Zinedine Zidane → *instanceOf* → *Soccer Player*

Soccer Player → *subclassOf* → *Person*

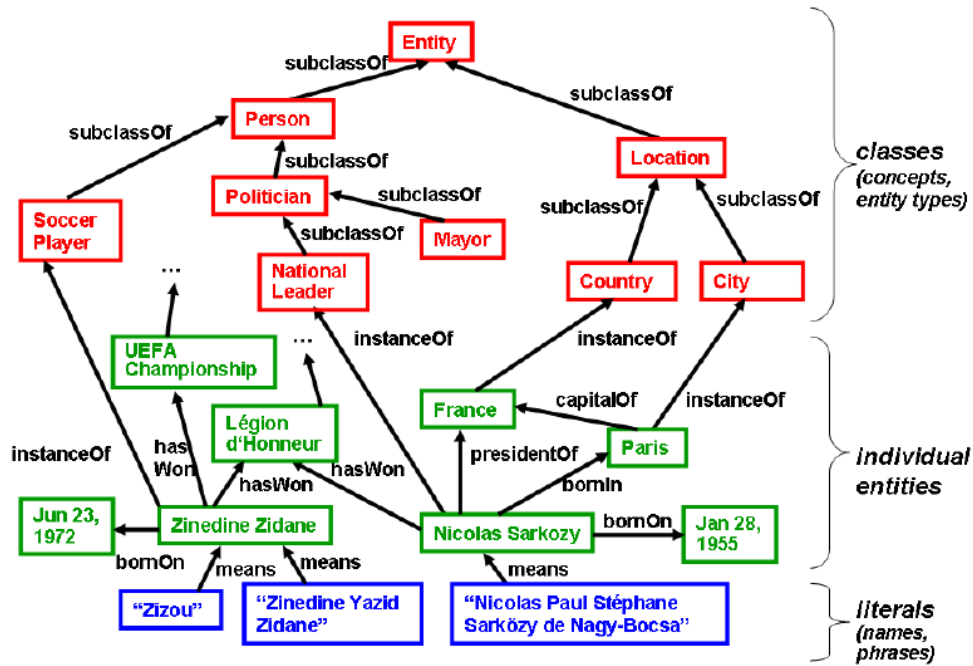


Fig. 2.1 YAGO Structure [42]

The YAGO structure includes of three major components: classes (concepts, entity types), a set of individual entities and literals (names, phrases). Figure 2.1 shows an excerpt of the YAGO knowledge base. These components are described as follows:

1. **Classes:** a class is a group of entities that shared particular characteristics (e.g. *Person*, *Location*, *Country* and *City*). YAGO's class is derived from two the main sources of: WordNet and Wikipedia. YAGO allows each class to be a subclass of one or multiple classes (YAGO taxonomy). The *Entity* is a root class in YAGO taxonomy. The superclass and subclass are connected via property *subclassOf*. An example of the 'subclassOf' connection is as follows:

$$\textit{NationalLeader} \rightarrow \textit{subclassOf} \rightarrow \textit{Politician}.$$

WordNet [69] is a lexical database of English developed by Princeton University. WordNet uses the actual sense of the words for grouping the words. Synset is a set of words that share one sense. Words that have multiple meanings (ambiguous words) can be assigned to several synsets. YAGO considers only nouns and the relationship among synsets (super-subordinate or hyperonymy, hyponymy) to organise taxonomic classes. YAGO establishes class from these synsets and links them to Wikipedia categories.

Table 2.1 YAGO's classes distribution [22]

Depth	Categories in YAGO						Leaf Categories in YAGO				
	Total	Total, n.	WordNet	Wikipedia	YAGO	Geo	Total	WordNet	Wikipedia	YAGO	Geo
0	25	0.00007	24	0	1	0	2	1	0	1	0
1	221	0.00061	16	180	22	3	123	0	113	10	0
2	79	0.00022	35	10	6	28	1	0	1	0	0
3	3,726	0.01032	419	3,286	17	4	3,293	19	3,271	1	2
4	42,979	0.11899	2,465	40,390	5	119	40,262	60	40,134	0	68
5	27,635	0.07651	5,926	21,561	1	147	21,586	165	21,333	0	88
6	67,928	0.18806	9,819	57,950	0	159	57,732	298	57,352	0	82
7	91,455	0.25319	14,954	76,388	0	113	75,234	317	74,857	0	60
8	63,015	0.17445	11,171	51,802	0	42	51,460	154	51,283	0	23
9	28,901	0.08001	8,006	20,880	0	15	19,912	72	19,835	0	5
10	16,115	0.04461	6,257	9,849	0	9	9,498	33	9,461	0	4
11	8,520	0.02359	3,953	4,565	0	2	4,520	9	4,510	0	1
12	4,603	0.01274	2,349	2,253	0	1	2,192	4	2,188	0	0
13	3,294	0.00912	1,270	2,023	0	1	1,910	0	1,909	0	1
14	1,443	0.00399	761	682	0	0	620	0	620	0	0
15	571	0.00158	459	112	0	0	102	0	102	0	0
16	461	0.00128	375	86	0	0	72	0	72	0	0
17	191	0.00053	163	28	0	0	25	0	25	0	0
18	47	0.00013	23	24	0	0	24	0	24	0	0
19	2	0.00001	1	1	0	0	1	0	1	0	0
Total	361,211	1	68,446	292,070	52	643	288,569	1,132	287,091	12	334

The lower classes in the Wikipedia categories are mapped to the higher classes in Wordnet by determining the most frequent sense of the head word in WordNet. YAGO allows only the conceptual categories in Wikipedia to be a class in YAGO [64]. The conceptual category is a category that has the head of word in a plural form. YAGO analyses the head of category name through shallow noun phrase parsing. For example, the '*wikicategory_Afghan_politicians*' has to be assigned to a subclass of the WordNet class 'politician' because 'Afghan politicians' is a head compound word and the word 'politicians' is plural form of 'politician'. The upper and lower class for these connections are as follows:

wikicategory_Afghan_politicians → *subclassOf* → *wordnet_politician_110451263*
wordnet_politician_110451263 → *subclassOf* → *wordnet_leader_109623038*

Table 2.1 shows the total number of classes from WordNet and Wikipedia in each level of YAGO taxonomy. YAGO taxonomy contains 19 depths and most of the classes in YAGO are derived from Wikipedia. 90% of YAGO classes are in depth 4-10.

2. Entities: a set of individual entities consist of instances such as people, building, class or country. YAGO [40] divides entities into six categories: people, groups(e.g. music bands, football clubs, universities or companies), artifacts (e.g. buildings, paintings, books, music songs or albums), events (e.g. wars, sports competitions like the Olympics or world championship tournaments), locations and other. Each individual entity could be an instance at least one class and is connected to its class via the

property *type*. The connection between instance and its class is as follows:

Zinedine_Zidane type Soccer_Player

Table 2.2 YAGO's instances distributio [22]

Depth	All Instances in YAGO					
	Total	Total, n.	WordNet	Wikipedia	YAGO	Geo
0	32	0.00001	3	0	29	0
1	31,140	0.00429	0	895	30,245	0
2	5	0.00001	0	5	0	0
3	124,679	0.01718	26,039	98,594	37	9
4	1,251,280	0.17246	4,450	1,234,315	0	12,515
5	402,104	0.05542	11,249	380,542	0	10,313
6	1,561,758	0.21525	26,181	1,529,220	0	6,357
7	2,152,886	0.29672	12,819	2,128,606	0	11,461
8	1,051,682	0.14495	4,985	940,515	0	106,182
9	305,063	0.04205	2,712	302,314	0	37
10	200,300	0.02761	85	200,173	0	42
11	85,298	0.01176	58	85,232	0	8
12	50,678	0.00698	92	50,586	0	0
13	27,193	0.00375	0	27,192	0	1
14	8,470	0.00117	0	8,470	0	0
15	1,451	0.00020	0	1,451	0	0
16	862	0.00012	0	862	0	0
17	536	0.00007	0	536	0	0
18	155	0.00002	0	155	0	0
19	12	0.00001	0	12	0	0
Total	7,255,584	1	88,673	6,989,675	30,311	146,925

A previous study [22] showed that 96.34% of individual entities in YAGO come from Wikipedia. Furthermore, most of the individual entities are located in the leaf classes and 90% of individual entities are located in depth 4-9 of the YAGO taxonomy. The details about instances of distribution are illustrated in Table 2.2.

3. Literals: YAGO deals with ambiguity and synonymy by mapping alternative names via relation *means*. The quotes are used to distinguish literals from the entities. The alternative names are derived from Wikipedia redirect pages. An example of literals is as follows:

"Zizou" means Zinedine_Zidane

YAGO has been used by many researchers. Melo et al. [21] integrates entities from YAGO into the Suggested Upper Model Ontology (SUMO). SUMO is a large scale formal ontology with a specific domain, such as countries, cities, companies or actors; the result is formal ontology on a rich scale. A further study [57] applied YAGO to automatically generate

group of queries and to match the search results into an appropriate category. Limaye et al. [46] used entities, types and their relationships in YAGO to identify entities that are extracted from web tables. Furthermore, Hu et al. [41] used YAGO and the Internet Movie Database (IMDB) to recommend movies and actors for users.

YAGO is an interesting knowledge base to create our data catalogue, and it is a primary source for us to create a conceptualisation of personal name entity. This is because YAGO contains a large number of personal name entities that are automatically extracted facts from Wikipedia and WordNet. Furthermore, YAGO taxonomy merges Wikipedia categories with the concepts of WordNet. Therefore, YAGO taxonomy is well-formed, semantically accurate and provides multiple levels of a hierarchical taxonomy.

2.1.3 Freebase

Freebase [30] is a knowledge base which uses graph technology to store data. It is operated by community members. Freebase contains more than 800,000 personal name entities and more than 2,000 occupations [37]. The Freebase structure includes domain, type, property and topic. Freebase contains more than 39 million topics about real-world entities such as people, places and organisations. For example, 'Bob Dylan', 'Hotel California (song)' and 'love' are topics found within Freebase. 'Type' is a group of similar topics, and topics can be mapped into one or more types. A structured set of properties are used to define type. For example, the 'Football player' is a type that consists of a set of various properties, such as 'Number of Career Goals', 'Matches played' or 'Position(s)'. 'Domain' is a group of related types that is the highest layer in freebase structure (e.g. the domain *Soccer*). The topic, type, property and domain can be mapped as follows:

David Graham : */soccer/football_player/position_s* : *Forward*

Where *David Graham* is a topic, */soccer/football_player/position_s* is a predicate (*soccer* is a domain, *football_player* is a type and *position_s* is a property) and *Forward* is an object or value. Figure 2.2 shows an excerpt of the Freebase knowledge base, where *David Beckham* is a topic or an instance and he is a *football player* (type or class). A *football player* type is a member of the domain *soccer*.

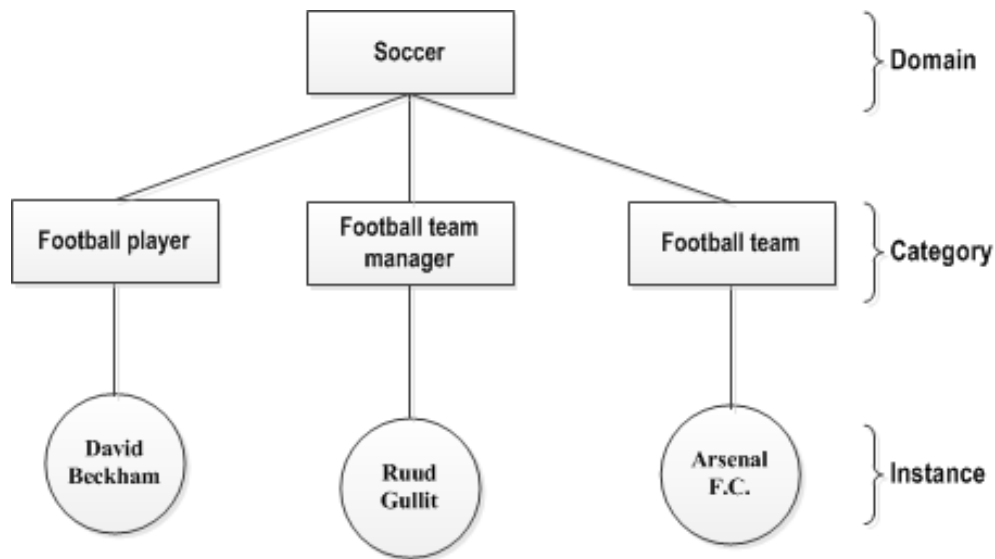


Fig. 2.2 Snapshot of Freebase Structure [30]

Taxonomy in Freebase has multiple depths. However, it is a thematic category. For example, *Milla Jovovich type Film/actor* means that *Milla Jovovich* is an actor, and the actor class is assigned to the domain *Film*, but *Film* is not a person. Therefore, Freebase taxonomy describes incorrect information about *Milla Jovovich*.

2.1.4 DBpedia

DBpedia [20] is a multilingual knowledge base which integrates structured data from Wikipedia. It is maintained by the DBpedia user community. DBpedia has its own ontology. The DBpedia ontology contains 320 classes and 1,650 properties with a maximal depth of five [44]. Figure 2.3 illustrates a total number of instances in some DBpedia classes (e.g. *Person* contains 763,643 instances).

	<i>en</i>
Person	763,643
Athlete	185,126
Artist	61,073
Politician	23,096
Place	572,728
Popul.Place	387,166
Building	60,514
River	24,267
Organisation	192,832
Company	44,516
Educ.Inst.	42,270
Band	27,061
Work	333,269
Music.Work	159,070
Film	71,715
Software	27,947

Fig. 2.3 Number of instances in English per class [44]

Figure 2.4 shows an excerpt of the DBpedia ontology including class and property. *Thing* is an ancestor class and a property *subClassOf* is used for mapping between super-class and lower-class. For example, *Place*, *Agent*, *Organization*, *Species* and *Work* are assigned to the super-class *Thing*. The connections between lower class and upper class are as follows:

$$Athlete \rightarrow Person \rightarrow Agent \rightarrow Thing$$

The class *Person*, consists of two properties: *birthDate*, and *deathDate*.

The taxonomy in DBpedia is clean and it is sensibly adjusted. However, it provides less depth of hierarchical structure for building personal name entity concepts. DBpedia classifies a person by occupation, and this mostly contains only one depth. However, it is too general to define people using only one depth occupation.

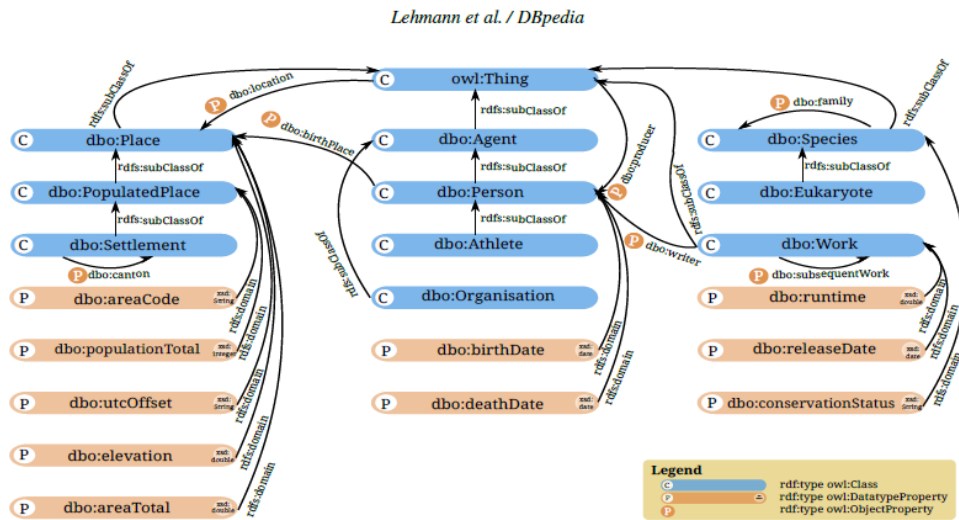


Fig. 2.4 Snapshot of a part of the DBpedia ontology [44]

2.2 Named Entity Disambiguation

Named Entity Disambiguation (NED) is the task of matching an entity that is mentioned in a document to its comparable entry in a knowledge base [34, 45]. NED is also known as entity linking or record linkage [24]. Dredze et al. [24] define three basic challenges in entity disambiguation: name variations, entity ambiguity and absence.

1. Name variations may be defined as a referential ambiguity, where a single person has multiple names. When this is the case we can use difference names to refer to an individual entity, such as full name, alias, nick name or professional position. For example, the personal names are highlighted in Figure 2.5 are used to refer to *George W. Bush*, 43rd President of the United States.

January 20, 2001 After a disputed election and bitter recount battle in Florida whose outcome is effectively decided by the Supreme Court, George W. Bush is sworn in as the 43rd president of the United States. In foreign affairs he promises an approach that will depart from the perceived adventurism of his predecessor, Bill Clinton, in places such as Kosovo and Somalia. (“I think the United States must be humble,” Bush said in a debate with his opponent, Al Gore.) In domestic affairs Bush pledges to cut taxes and improve education. He promises to govern as a “compassionate conservative” and to be “a uniter, not a divider.” He comes into office with a \$237 billion budget surplus.

Fig. 2.5 Examples of referential ambiguity and differing orthography [50].

2. Entity ambiguity is a lexical ambiguity (single name: multiple persons) where a single name refer to multiple entities. For example, *Chris Martin* refer to more than ten persons, such as the English front-man of Coldplay or a Scottish footballer.
3. Absence means that some entities may not exist in a knowledge base (NIL) as a knowledge base does not have information about every person in the world.

2.2.1 Named Entity Disambiguation Framework

Hachey et al.,[34] defines three main components in the named entity disambiguation framework: extractor, searcher and disambiguator.

1. Extractor is the task of detection and preparation of named entities; they are represented in the document and may be used to extract the context
2. Searcher is the process for generating a set of possible entities (candidate entities) that are mentioned in the document. The effective researcher should provide the correct entity in a small set of candidates. Bunescu and Pasca [8] used three sources in Wikipedia: Title pages, Redirect pages, Wikipedia Disambiguation pages for creating a named entity dictionary. The exact-match lookup function and the named entity dictionary are used to generate a set of candidate entities.
 - (a) Wikipedia article titles (a title in each Wikipedia article) are used for extracting the entity name. The title in the Wikipedia article is unique and used a parenthetical expression to separate lexical ambiguity. For example, in the article about *Chris Martin*, Wikipedia uses professional categories to identify a specific person such as *Chris Martin (artist)* or *Chris Martin (cricketer)*. Furthermore, if an occupation is duplicated, the date of birth is used to separate lexical ambiguity, e.g. *Chris Martin (footballer, born 1988)* and *Chris Martin (footballer, born 1990)*.
 - (b) Wikipedia Redirect pages are used to extract alternative names or referential ambiguity in each entity.
 - (c) Wikipedia Disambiguation pages are used to extract referential ambiguity entities.

Cucerzan [19] developed a data catalogue using four sources from Wikipedia article titles, redirect pages, disambiguation pages and references to entity pages in other Wikipedia articles. The catalogue consists of two components, the entity surface form

and entity information. The surface form is a list of referent words that are used to mention each entity in a document. The entity surface form includes of a set of terms and a set of entities that used this referent term. The exact-match lookup function and the entity surface form are used to generate a set of candidate entities. However, the researchers [34] argued that the link anchor texts decrease the system performance.

Shen et al. [63] extend the entity surface form in [19] by adding count information to define popular entities in each surface form for generating a set of candidate entities. The record of count information is used to calculate the link probability score and this score is used to remove the candidate entities that have a very low link probability (the rarest entities that used this referent term). Link probability $LP(e|m)$ or popularity score for entity e can be calculated with equation 2.1. Given $count_m(e)$ is the frequency of links that entity e represented under surface form m (using term m). An E_m is a number of entities that uses the surface form m .

$$LP(e|m) = \frac{count_m(e)}{\sum_{e_i \in E_m} count_m(e_i)} \quad (2.1)$$

The count information is useful to reduce the number of candidate entities because the system can remove the entity which has the smallest number of total count (an entity that rarely used the surface form m for reference; they are not well known). However, the unpopular candidate that is removed from a set of candidate entities may still actually be the correct entity.

Dredze et al. [24] aim to design an open knowledge base data catalogue. The researchers address this by aiming to solve a major problem in discusses in [19] and the study [8] that the data catalogue is dependent on Wikipedia. Name variations are to be considered for candidate entity generating. The researchers select not only the exact match, but also select the entities for which all or part of the contents are mentioned: if the first letters match, if they match an alias or if they have a strong string similarity score.

Cucerzan [19] and Bunescu and Pasca [8] are deep dependence on Wikipedia to generate a set of candidate entities. Dredze et al. [24] introduce a new idea about the independent knowledge base. In addition, the name variations have been considered by combining multiple methods with text similarity metrics to produce a set of candidate entities (to minimise the candidate sets and maximise recall). In generating candidate entities, the exact-match lookup over the named entity dictionary is an insufficient technique to handle name variations and dirty data (such as miss spellings) in each occurrence.

3. Disambiguator is the process for selecting the best entity among ambiguous entities. NED may be classified on the basis of features they used to discriminate ambiguous entities into the two fundamental methods of context similarity and topical coherence.

- (a) **Context similarity** is the bag of words model that is used for comparing the similarity between the mentioned texts around the entity and the document related with the entity in Wikipedia [45, 63]. The study [8, 24] use the context similarity model for ranking a set of candidate entities.

Bunescu and Pasca [8] present two techniques: a context-article similarity and taxonomy kernel for entity disambiguation. The first feature is based on the cosine similarity between query texts and the candidate Wikipedia article. The second feature is the candidate taxonomy kernel which is derived from Wikipedia categories and the 55-word contexts of the query text with a support vector machine(SVM) ranking model. Bunescu and Pasca also returned the NIL values to refer that each mention not covered in Wikipedia by setting e_{out} entity in the named entity dictionary. In the named entity dictionary, the article is given to $e_{out}.T = \theta$ and a set of categories is $e_{out}.C = \theta$. The SVM ranking model returns the entity in Wikipedia that has the highest score (the highest score should be more than a fixed threshold τ), otherwise it returns e_{out} .

Dredze et al. [24] used multiple features consisting of entity categories, popularity (from Google's ranking), document similarity, name similarity and coverage of the co-referential noun in the knowledge base node text. The SVM ranker is used to calculate the similarity score and return the candidate which has the highest score, otherwise it returns NIL.

The main drawback of context similarity is that it is not flexible for natural language because it requires precise word overlap to measure similarity, and the performance depends on a particular term co-occurring between a couple of comparisons [45, 63]. Furthermore, this approach does not consider the interdependence that exist between the entity [36, 63] that is an important feature for solving lexical ambiguity. Finally, these works directly derived the taxonomy of the categories from Wikipedia, which is dirty and not well formed [59, 63]. Therefore, it is insufficient to organise the hierarchy and semantic purpose of a class.

- (b) **Topical coherence** is used in the Wikipedia cross-page links for finding unity between two entities [45]. This method is based on the assumption that an entity mentioned in the same document as another should be interdependent with the

other entities [59].

Cucerzan [19] is the first researcher who combined the topic coherence feature with NED. Cucerzan uses two features that are derived from Wikipedia, including context clues (extracted from the first paragraph in each entity page and other articles that refer exactly to this entity) and category information. The vector space model is employed in the disambiguation process to calculate the similarity between the topic coherence derived from each entity that is mentioned in a single document and the context clues in each candidate entity that appears in the document, as well as the agreement among categories which overlapped with the candidate entities. Shen et al. [63] argue that the assumption about every entities are stored in the knowledge base is wrong because some entities are absent e.g. unpopular entity.

As the categories in Wikipedia are dirty and not well-formed, the next method [62] solves this problem by deriving the taxonomy from YAGO.

Shen et al. [62] proposed LIEGE (Link the entities in web lists with the knowledge base) to map each entity mentioned in the web lists with a real-world entity. The researcher assumes that the entities which appear on the web list in the single document have the same conceptual type. LIEGE uses five features, including a set of entities mentioned in a web list, taxonomy classes from YAGO [42, 64], the entity popularity from Wikipedia and the context in the external document corpus (Wikipedia articles) where the entities appear. The list item within the web lists can refer to multiple candidate entities. Therefore, Shen et al. [62] introduces prior probability to measure the popularity in each candidate entity. Given a collection of web lists L with $|L|$ items. Let l_i be a set of web lists L where $l_i \in L$. The prior probability equation in each candidate entity for the list term l_i is shown below.

$$P_{pr}(r_{i,j}) = \frac{\text{count}(r_{i,j})}{\sum_{u=1}^{|R_i|} \text{count}(r_{i,u})} \quad (2.2)$$

- Prior probability $P_{pr}(r_{i,j})$ is a mention-entity popularity for each candidate entity. The set of candidate entities for the list item l_i is denoted by R_i . Given $|R_i|$ is size of R_i , and use $1 \leq j \leq |R_i|$ to index the candidate entity in R_i . The candidate entity with index j in R_i is denoted by $r_{i,j}$. Where $\text{count}(r_{i,j})$ is the number of links which point to the candidate entity $r_{i,j}$ under the mentioned term l_i and $\sum_{u=1}^{|R_i|} \text{count}(r_{i,u})$ is the total of links which point to the set of candidate entities R_i under the mention term l_i .

- Coherence means that all entities in the web lists should have similar type. Therefore, the candidate entity which has the same conceptual type as the other linking type should be considered. The coherence is calculated from two features: the type hierarchy based on similarity and the distributional context similarity.

The study [62] assumes that all the list items can be linked to the existing knowledge base, but in fact some entities are absence from the knowledge base. Additionally, some of the YAGO hierarchy classes are dirty so the linking quality may not gain an advantage over the type hierarchy based similarity.

2.3 Tree Matching

In this section, we introduce the two tree matching techniques: Simple Tree Matching (STM) [75] and Partial Tree Alignment (PTA) [79–81]. STM and PTA algorithms provide a useful idea for us to represent a new algorithm called Simple Partial Tree matching (SPTM) in personal named entity disambiguation. Tree matching is the algorithm for calculating the maximum similarity between tree A and B [43], and can be defined as the minimum set of operation cost for converting tree A into tree B [47]. The set of the operation cost consists of node removal, node insertion and node replacement where a node is a set of letters in the personal name. Tree matching is useful for mapping two documents which are organised according to a hierarchical structure, or a nested of elements, e.g. XML, HTML or meta-data.

'T' is a tree and 'T[i]' is the i-th node in tree 'T'. The nodes are ordered from top to bottom and left to right. The letter 'M' is the set of all similarity nodes between two trees, where tree A has n_1 nodes and tree B has n_2 nodes. For any

1. $i_1 = i_2$ iff $j_1 = j_2$;
2. A[i₁] is on the left of A[i₂] iff B[j₁] is on the left B[j₂];
3. A[i₁] is an ancestor of A[i₂] iff B[j₁] is on the left B[j₂];

Intuitively, every nodes in a tree occurs only once and "the order among siblings and the hierarchical relation among nodes are preserved" [47]. One major drawbacks of tree edit distance is high computation cost [28] needed in transforming one tree into another. Yang [75] solves the tree edit distance problem by introducing Simple Tree Matching (STM).

STM was proposed in [75] for similarity measurement between two computer programs. STM is a restricted matching algorithm, as it does not allow node replacement or level crossing for calculating the maximum number of similarity nodes between a pair of trees [47].

STM is a top-down algorithm with the complexity of $O(n_1, n_2)$ where n_1 and n_2 are the size of the trees. The key problem in STM is that every nodes in the tree are equal. Kim et al. [43] enhanced the STM algorithm for web information extraction by assigning different weights to different HTML nodes in the tree and calling it the HTML Tree Matching(HTM) algorithm.

Zhai and Liu [79–81] introduce PTA based on tree matching for aligning data records which are extracted from web pages. This algorithm is called partial tree alignment because only certain matched data fields in a pair of records can be aligned. One limitation in PTA is the equal weight is given to each HTML tag.

Kim et al. [43] develop a Layered and Weighted Tree Matching (LWTM) algorithm for measuring similarity between web data records. They provided different weights for different types of HTML elements, from the assumption that different nodes in similar data records usually have different importance when computing the similarity of these nodes.

In the next sub-section, we give an overview two algorithms. STM and PTA which are used in the personal name disambiguation in this study.

2.3.1 Simple Tree Matching(STM)

STM is a dynamic programming scheme used to identify the syntactic difference between two programs. Figures. 2.6 and 2.7 illustrate the STM algorithm for comparing tree A and tree B. The two root nodes: N1 and N15 are mapping first (in line 1) and if two root nodes have distinct letters, then two trees are completely different. If the two root node are similar, the algorithm then recursively finds the maximum amount of matching nodes between the first level of tree A and tree B and saves it in a W matrix (in line 8). The result values of the W matrix are shown in 2.6 (c). The value of matching N2 and N16 is 3 for the three pairs: (N2, N16), (N6, N18), (N7, N19). The value resulting from matching node N2 and node N17 is 0 because these two nodes contain distinct letters. The value resulting from matching node N3 and node N17 is 2 for the two pairs (N3, N17), (N8, N21). The value resulting from matching node N4 and node N16 is 2. Note that (N9, N10) and (N18, N19) contain identical letters but there do not match because the two pairs of nodes are represented in a different order. The value of matching N5 and N17 is 3 for the three pairs: (N5, N17), (N11, N20), (N12, N22). After that, the M matrix is computed. The final value that results is returned in line 12, M[4,2] is 6 and the maximum number of matching pairs is 7. The seven matching pairs are (N1, N15), (N2, N16), (N6, N18), (N7, N19), (N5, N17), (N11, N20), (N12, N22).

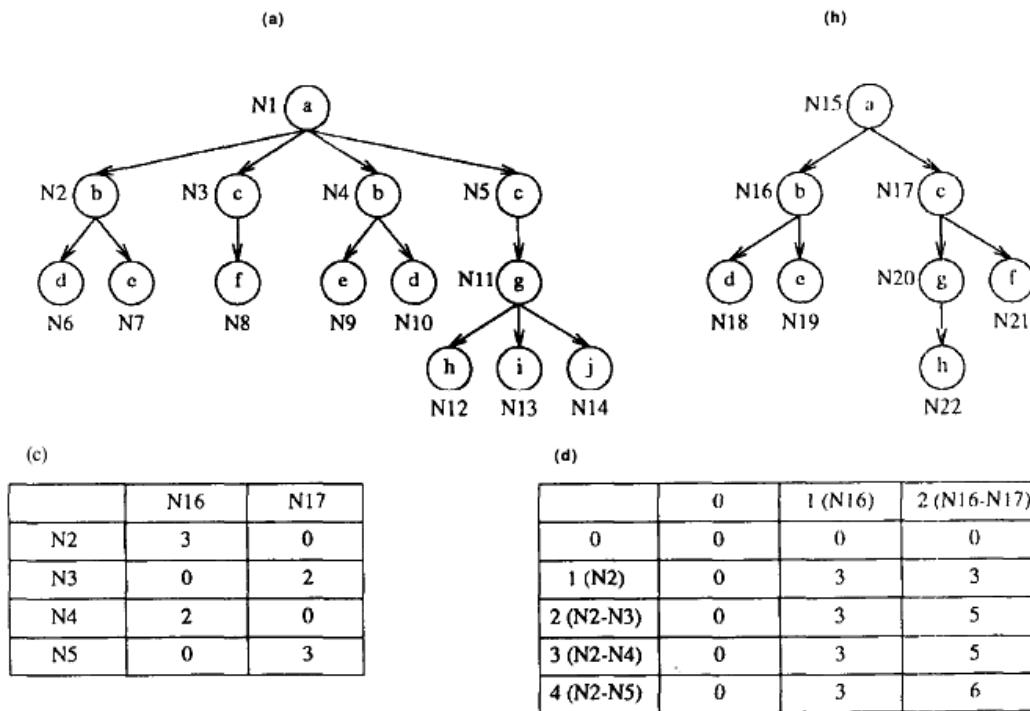


Fig. 2.6 Simple Tree Matching (a) Tree A; (b) Tree B; (c) W matrix for the first-level sub-trees; (d) M matrix for the first-level sub-trees [75].

Algorithm: Simple_Tree_Matching(A, B)

1. **if** the roots of the two trees A and B contain distinct symbols
2. **then return** (0);
3. **else** $m :=$ the number of first-level sub-trees of A ;
4. $n :=$ the number of first-level sub-trees of B ;
5. Initialization: $M[i, 0] := 0$ for $i = 0, \dots, m$;
 $M[0, j] := 0$ for $j = 0, \dots, n$;
6. **for** $i = 1$ to m **do**
7. **for** $j = 1$ to n **do**
8. $M[i, j] := \max(M[i, j-1], M[i-1, j], M[i-1, j-1] + W[i, j]);$
where $W[i, j] = \text{Simple_Tree_Matching}(A_i, B_j)$
9. **endfor**;
10. **endfor**;
11. **return** ($M[m, n] + 1$)
12. **endif**

Fig. 2.7 The Simple Tree Matching Algorithm [79].

2.3.2 Partial Tree Alignment(PTA)

PTA [79–81] algorithm is part of DEPTA (Data Extraction based on Partial Tree Alignment). The alignment is partial because the node in T_i can be inserted into T_s only if the location for the insertion can be uniquely determined in T_s . Otherwise, they will not be inserted and will be left unaligned. This technique is commonly used for alignment of a data item in each data region due to the fact that, in some web pages, the object details or data records are not represented in a connecting segment of the HTML code. Data region is the boundary location in the web page that contains similar data records. In this case, data items from all data records will be rearranged before they are integrating into the database.

PTA aligns multiple trees by progressively growing a seed tree. The seed tree, denoted by T_s , is considered to be from the tree which has the maximum number of data fields. The reason for choosing this seed tree is clear, as it is more likely that the tree has a good alignment with it than with data fields in other data records. After that, for each $T_i (i \neq s)$, the algorithm tries to find a matching node in T_s for each node in T_i . When a match is found for node $T_i[j]$, a link is created from $T_i[j]$ to $T_s[k]$ to indicate its match in the seed tree. If the match cannot be found for node $T_i[j]$, then the algorithm attempts to expand the seed tree by inserting $T_i[j]$ into T_s . The expanded seed tree T_s is then used in subsequent matching.

To clarify PTA algorithm, we will first describe two tree alignment, and after this we will introduce multiple tree alignment.

Partial Alignment of Two Trees

To clarify, PTA algorithm works on, how nodes in T_i can be aligned with nodes in T_s . Firstly, we will demonstrate partial alignment of two trees. After T_s and T_i are matched, some nodes in T_i can be aligned with their corresponding nodes of T_s because they match one another. The nodes in T_i that are not matched should then be, inserted into T_s as they may contain optional data items. There are two possible situations when inserting a new node $T_i[j]$ into the seed tree T_s , depending on the whether a location in t_s can be uniquely determined to insert $T_i[j]$. Otherwise, they will not be inserted and will be left unaligned. The location for insertion $T_i[j]$ can be uniquely decided in the following cases:

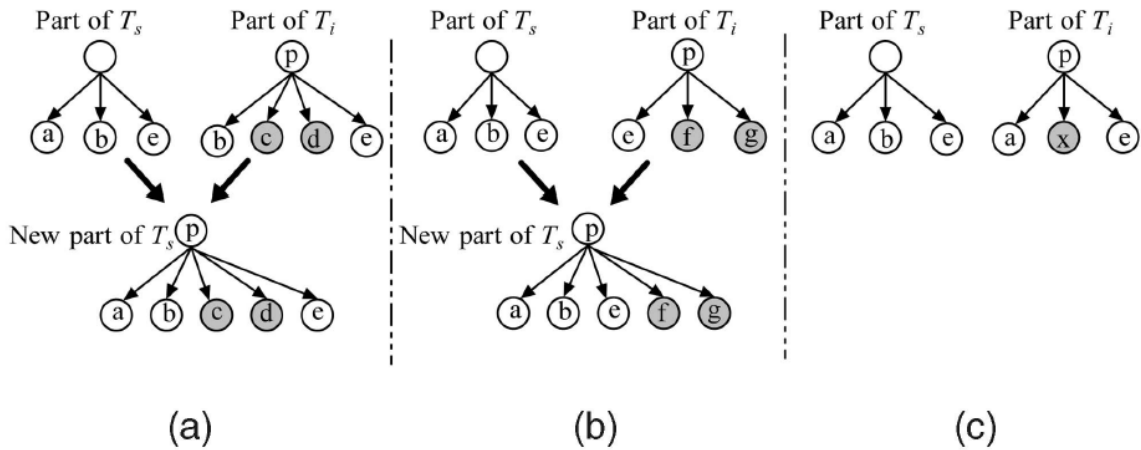


Fig. 2.8 Expanding the seed tree: (a) and (b) unique expansion and (c) insertion ambiguity [81].

1. If $T_i[j] \dots T_i[m]$ have two neighbouring siblings in T_i , one on the right and the other on the left, that are matched with two consecutive siblings in T_s . Figure 2.8a shows such a situation, which gives one part of T_s and one part of T_i .
2. If $T_i[j] \dots T_i[m]$ has only one left neighbouring sibling x in T_i and x matches the right most node x in T_s . The $T_i[j] \dots T_i[m]$ can be inserted after node x in T_s . Figure 2.8b illustrates this case.
3. If, on the other hand, there can not decide a unique location for unmatched nodes in T_i to be inserted into T_s , this is shown in Figure 2.8c. The unmatched node x in T_i may be inserted into T_s , either between nodes a and b , or between nodes b and e . In this situation, we do not insert this node into T_s .

Partial Alignment of Multiple Trees

The alignment algorithm is based on tree matching and uses only HTML tags for comparison between pair of nodes. Figure 2.9 illustrates the multiple tree alignment algorithm.

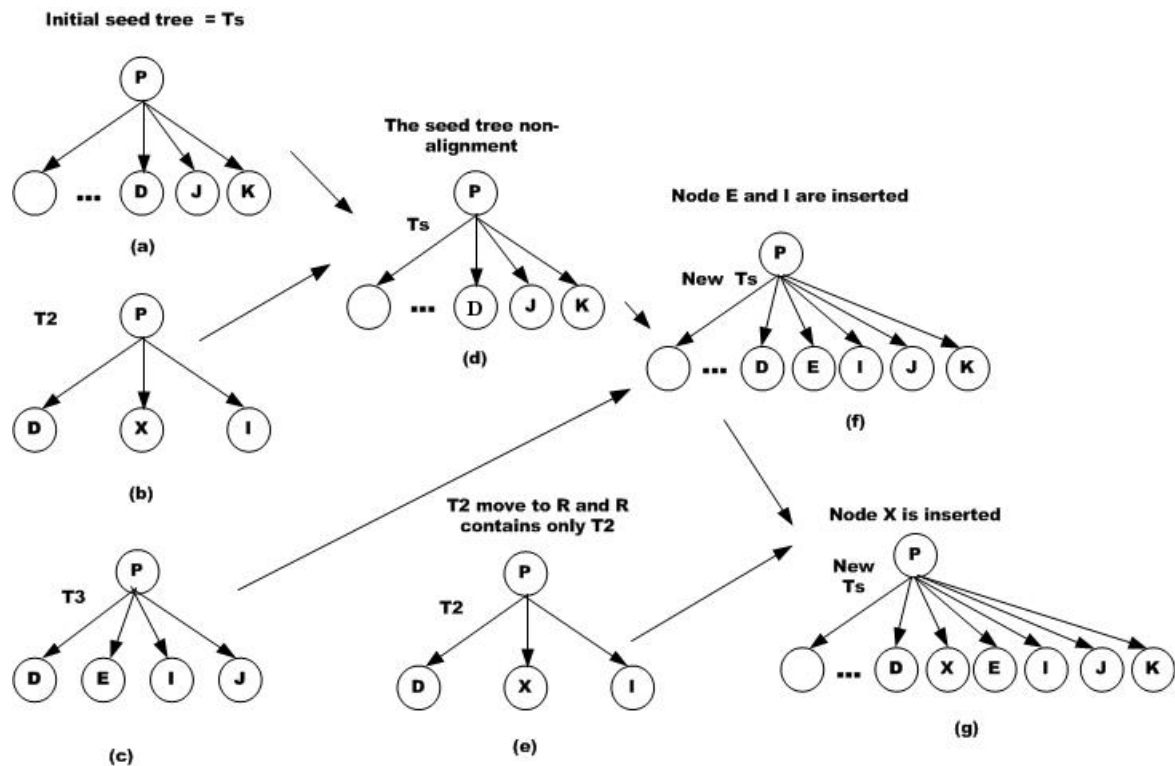


Fig. 2.9 Partial Tree Alignment with multiple trees [81].

The algorithm starts with three trees. T_s , T_2 and T_3 which are to be aligned. The tree which has the maximum number of data items will be a seed tree or T_s . The algorithm is based on two trees matching.

Step 1 A DOM tree is established for each data record. The tree consists of two levels: *parent* and *child*.

Step 2 A DOM tree that holds the maximum number of data items (nodes) is referred to as a *seed tree*. Figure 2.9(a) is a seed tree (T_s).

Step 3 Let T_s be a seed tree and T_i contains a set of other trees in each data region ($i \neq s$). T_i is matched with T_s until end of T_i .

There are two possible situation:

(i) If the location for adding new node can be resolved, the node in T_i will be inserted into T_s . Figure 2.9 (c), (d) and (f) show this situation. The seed tree T_s is expanded.

(ii) If the location for inserting new node cannot be uniquely determined (as shown in Figure 2.9 (a), (b) and (d)), there is more than one possible place to insert the new

node, and node X and I can be added after node D, J or K. Therefore, T_s cannot be aligned and T_2 will be moved to R.

Step 4 When all T_i are processed completely, the trees in each data record will be processed again. If the trees are unaligned in T_s this is the end of the process and, the unmatchable data items will be moved to a single column.

The complexity of this algorithm based on *Big O* notation is $O(k^2)$, where k is the number of trees. Big O notation, also called Landau's Symbol, is a technique which is used in Computer Science to define the performance or complexity of an algorithm. The letter O is defined as the rate of growth of a function, so $T(k) = O(n^2)$ means that T(k) grows at the order of n^2 .

Incorrect alignment of data items happens in the two following situations:

1. Data items of the same attribute are incorrectly aligned into different columns because they are enclosed by tags with different tag names.
2. Data items of different attributes are incorrectly aligned into the same columns because they are enclosed by tags with the same tag name.

The two results of incorrect alignment reveal that tag name is a significant feature for precision in partial tree alignment.

2.4 Conclusion

The occupation taxonomy in existing knowledge bases including Wikipedia, Freebase, YAGO and DBpedia are not suitable to establish personal name concepts. Firstly, Wikipedia and Freebase categories are not well-formed arrangements and are based on thematic domains. While the, YAGO knowledge base fixed the above problems by constructing a backbone class from WordNet synsets before connecting them to Wikipedia categories to form a semantic taxonomy. In this thesis found that some connections between WordNet and Wikipedia are wrong. Some of the WordNet classes are not related to Wikipedia categories. For example, the Wikipedia categories: *President of FIFA* is assigned to the WordNet class: *head of state president*, but the president of FIFA is not a head of state president. Finally, DBpedia ontology is well organised, but the hierarchy level is not deep enough to create personal name concepts.

An occupation is an important feature when distinguishing one person from another [48]. This thesis aims to propose a new occupation taxonomy for generating a personal name con-

cept in each instance. To define a characteristic in each person, the basic element of occupation taxonomy should be semantic and organised in multiple orders. Therefore, we propose a new occupation ontology architecture based on YAGO taxonomy and web directories to create an occupation taxonomy for personal name concepts.

The NED framework consists of three components: extractor, searcher and disambiguator. The extractor fulfils the task of extracting a list of proper names within a document. The searcher generates a set of candidate entities from each proper name by matching a proper name over the entity surface form. However, the exact-match lookup over the entity surface form in the previous works is insufficient to generate a set of candidate entities because the two names (the mentioned name and the entity surface form) are equivalent if all letters are the same. However, one challenge in NED is name variations, as a proper name often has multiple representations and typological errors occur. The disambiguator has the task of solving lexical ambiguity by linking the best candidate to a mentioned name. Two fundamental techniques of contextual similarity and topical coherence are usually used in a NED task. However, contextual similarity is not flexible for natural language. As a result, we introduce a new approach based on tree matching and topical coherence in the disambiguator task.

There are three main advantages to using STM to compare a similarity between two personal name concepts. Firstly, the structure of a personal name concept is a tree. Secondly, the computation needed to make a decision that a pairs of personal name concepts are similar or not is a short one. This is because STM is a top-down algorithm and the two root nodes are compared first, so if the two root nodes are dissimilar this means that there are significant differences in conceptualisation between the two personal names. Thirdly, when considering the similarity between two trees, it is not necessary for every nodes in the two tree to be similar, but enough for some nodes in the two tree to be similar. The STM returns the maximum number of similarity nodes and the matching pairs. However, the performance of STM is limited by assigning equal weight to every nodes in the tree. Additionally, it is time consuming and it does not make sense to compare ambiguous personal name concepts to every identified personal in a web document.

The seed tree growth technique in PTA provides a useful idea to our work for creating a comparison tree. This is because it integrates identification of personal name conceptualisations which have the same root node and because the tree which has the maximum number of nodes becomes the comparison tree. A comparison tree technique can reduce the number of times similarity matching occur between two trees.

As a result, we can introduce the new algorithm Simple Partial Tree Matching(SPTM) for personal name disambiguation by combining the two algorithms of STM and PTA.

Chapter 3

Personal Name Surface Form and OAPnDis

This chapter introduces three main ideas used in personal name entity linking: Personal Name Surface Form Modules(PNSFM), Occupation Architecture for Personal Name Disambiguation (OAPnDis) and the Personal Name Disambiguation Data Catalogue (PNDDC). Firstly, Section 3.2 describes PNSFMs that are used to generate personal name surface form and extracted occupation categories, personal name entity's occupation and their relations. Secondly, Section 3.3 presents an overview of OAPnDis, a new occupation architecture that is used to generate personal name concepts. Finally, Section 3.4 presents a detailed description of the Personal Name Disambiguation Data Catalogue (PNDDC), which is used in personal name entity linking.

3.1 Motivation

Data catalogues are one of the important components of Personal Name Entity Linking (PNEL) and plays a critical role in the precision of PNEL. The major problems of PNEL, however, are lexical and referential ambiguity. Referential ambiguity means that the name can be represented in multiple forms [7, 24] such as abbreviations (e.g. *John F. Kennedy* vs. *JFK*), shortened form (e.g. *Willard Carroll Smith* vs. *Will Smith*), alternate spellings e.g. *Barack Obama* vs. *Barak Obama* and aliases e.g. *DB7* vs. *David Beckham*. Lexical ambiguity means that a single name can refer to different persons [7, 24]. For example, the name *George Bush* may refer to more than six well known people on Wikipedia as shown in Table 3.1.

Table 3.1 The Wikipedia disambiguation page focuses on the personal name: *George Bush* [71].

ID	Personal Names	Professional Category
1	George H. W. Bush	41 st President of the United States
2	George W. Bush	43 rd President of the United States
3	George Bush(biblical scholar)	19th-century biblical scholar and preacher
4	George Washington Bush	First black settler in what is now the state of Washington
5	George Bush (NASCAR)	Former NASCAR driver
6	George P. Bush (born 1976)	Attorney and real estate developer

Traditionally, knowledge bases such as YAGO [42], Wikipedia and Freebase [30] provide information about referential ambiguity for each person. For example, Wikipedia uses 'the redirect pages', YAGO uses the property 'means' and Freebase uses the property 'Also known as' to solve the referential ambiguity of each person. The knowledge bases above therefore provide available data of referential ambiguity of names that is useful for generating a personal name dictionary.

Knowledge bases uses multiple features such as occupation, nationality and birth of date for solving problems of lexical ambiguity by inserting these feature at the end of personal name. Occupation is widely used in Wikipedia or Google snippets to distinguish people who have the same name. Furthermore, the experimental results in a study [48] shows that occupation is the most important feature for solving the problem of lexical ambiguity. Therefore, building personal name entities and a corresponding 'occupations catalogue' is an important task in PNEL.

Knowledge bases and web directories such as Wikipedia, Freebase and Dmoz [23] use a thematic domain hierarchy for grouping contents and entities. The categories have a hierarchical arrangement; however, they are not well-formed and barely effective for their ontological purpose [63, 65]. For example, *Stephen King* is assigned to the super-category *Fiction*, even though *Stephen King* is a *writer* and not fiction. Therefore, thematic domain hierarchy provides sometimes meaningless to a person and may affect the performance of personal name disambiguation.

YAGO and DBpedia [20] provide an ontology and semantic classes. However, the taxonomy in YAGO suffers from the error connections between WordNet and Wikipedia class in that, some classes are not related [22]. The major problem in DBpedia is that it is offering a smaller number of level in hierarchy structure so it does not do enough for generating the personal name concept for each person. It is more valuable in personal name disambiguation if the system can create a concept of each person based on their occupation taxonomy, or if

people are classified according which their top view of occupation taxonomy. For example, when describing the entity *Chris Martin*, the system provides the information not only that he is a singer, but also details such as that he is a *20th-century English singers, a British alternative rock musician* and in the upper level, that he is an *Entertainer and Artist*.

The goal of this chapter is to describe OAPnDis, personal name concepts, personal name surface form and investigate that occupation feature can be used to disambiguate lexical ambiguity.

3.2 Personal Name Surface Form Modules(PNSFM)

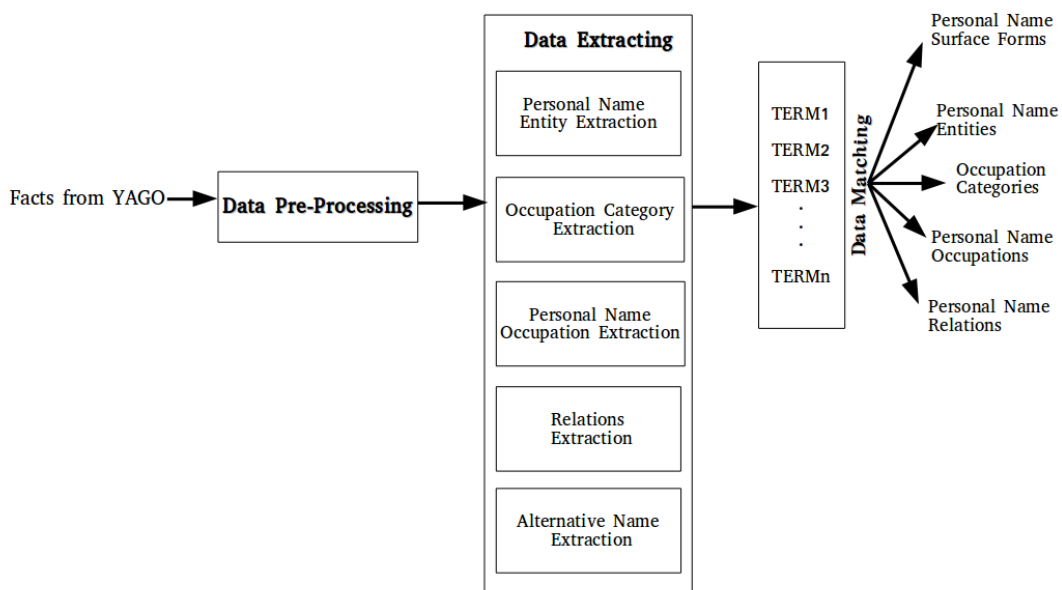


Fig. 3.1 PNSFM modules

PNSFM is a set of component that are used to create a personal surface form and generate information for creating occupation taxonomy and personal name concepts. Figure 3.1 shows this in three stages: data pre-processing, data extracting and data matching. The input is a fact from YAGO, while the outputs are personal name surface forms personal name entities personal name occupations and personal name relations .

PNSFM differs from previous studies [8, 19, 63] because our modules include a data pre-processing stage for removing duplicate and dirty data before integrating it into the personal surface form. This is because dirty data may reduce the performance of personal name matching that relies on character comparisons [25] and consume storage space. The overview of the data catalogue system is outlined in Figure 3.1.

1. The data pre-processing stage focuses on removing unwanted data, detecting duplicate data and transforming representation in Unicode characters into ASCII code characters. While facts in YAGO are usually clean, some dirty data may be present; this is a basic problem with extracted data from the Internet.
2. The data extracting stage works by extracting personal name entities, their alternative names, their occupations, their relations and their occupation categories from YAGO facts. These components are extracted from subjects and/or objects among three properties: *actedIn*, *type*, *isMarriedTo*, *hasChild*, *means* and *subclassOf*.
3. The data matching stage aims to map data and detect duplicate data before storing it in a database. The terms in Figure 3.1 are personal name entities, their alternative names, their occupations, their relations and their occupation categories.

The previous three stages are described in more details in the following sections. Due to the time constraints, this thesis has focused on the following three major types of occupation categories under a *Person* class in YAGO that are usually mentioned on the web pages. The three groups of occupations considered are listed below.

1. Entertainers and artists are people who are working in the area of entertainment and art, including actors, singers, models, writers, dancers and producers.
2. Sportspeople are people working within sport, such as players, team managers and coaches.
3. Politicians are people who are working within politics, such as governors, presidents, Chancellors and senators.

YAGO2 is a source for extracting personal name entities, their alternative names and their occupation categories. Table 3.2 shows the variables and their meanings.

Table 3.2 Notations used in PNSFM modules

Notation	Meaning
$P = \{p\}$	Set of personal name entities
$A = \{a\}$	Alternative names
$O = \{c\}$	Occupation categories
$S = \{s\}$	Personal name surface forms
$R = \{r\}$	Personal name relations
$A(p)$	Set of alternative names in each person
$O(p)$	Set of occupations in each person
$P(s)$	A list of persons who share a surface form
$\{p,r,(p_1,p_2,p_3,\dots,p_n)\}$	Set of person and their relations

3.2.1 Data Pre-Processing

When data from the web is stored, one needs to ensure that it is clean of dirty data in order to optimise efficiency and the precision of data mining [10, 51]. Furthermore, the detection of duplicate data is an important step when integrating data from multiple sources [55]. Data pre-processing is an important stage in PNSFM because it significantly improve personal named disambiguation that is based on similar text matching. Data pre-processing is a part of the data cleaning technique which is used to detect duplicate data, remove unwanted data and transform Unicode characters to ASCII characters in YAGO facts.

On the one hand, YAGO has a large amount of personal name entities, binary relationships and a clean taxonomy of occupation categories which are derived from WordNet and Wikipedia. However, the contents of YAGO could also be dirty or duplicated as shown in Figure 3.2. As an illustration, Figure 3.2(a) shows noisy facts in YAGO that are totally unreadable as they are represented in the form of Unicode characters that always occur in a data set and should be converted to ASCII characters. Unicode is a particularly persistent problem when a web page is generated in different languages such as Egyptian, Russian or Thai. Figure 3.2(b) represents two duplicate data records. This is why data pre-processing is an important stage before data from the YAGO knowledge base is matched.

id	relation	arg1	arg2
#167081640	actedIn	Alicia_Silverstone	Batman_\u0026_Robin_\u0028film\u0029
#167081644	actedIn	Uma_Thuman	Batman_\u0026_Robin_\u0028film\u0029
#167081648	actedIn	Arnold_Schwarzenegger	Batman_\u0026_Robin_\u0028film\u0029
#167081652	actedIn	Chris_O\u0027Donnell	Batman_\u0026_Robin_\u0028film\u0029
#167081656	actedIn	George_Clooney	Batman_\u0026_Robin_\u0028film\u0029
#167081660	actedIn	Nicole_Kidman	Batman_Forever
#167081664	actedIn	Chris_O\u0027Donnell	Batman_Forever

(a)

id	relation	arg1	arg2
#164396180	isLeaderOf	Arnold_Schwarzenegger	Jack_O\u0027Connell
#164396640	isLeaderOf	Arnold_Schwarzenegger	John_Chiang_\u0028California_politician\u0029
#164403484	isLeaderOf	Arnold_Schwarzenegger	Barbara_Alby
#168532648	isMarriedTo	Arnold_Schwarzenegger	Maria_Shriver
#168564988	isMarriedTo	Arnold_Schwarzenegger	Maria_Shriver
#166040136	isPoliticianOf	Arnold_Schwarzenegger	California

(b)

Fig. 3.2 An example of the dirty data in YAGO knowledge base.

In this thesis, YAGO2 [40] is considered as a source to create a personal name surface form, occupation taxonomy and personal name concepts for the following reasons:

1. YAGO2 provides a huge number of facts, entities and relations (without GeoNames it contains 124 million facts, 2.6 million entities and 104 relations).
2. Facts stored in YAGO2 have a higher precision as 95% of the facts are true.
3. YAGO provides alternative names for each person via property *means*.

YAGO has a higher precision rate (up to 95%). On the other hand, errors such as the one shown in Figure 3.2 persist, as duplicate facts and Unicode noise is common.

Table 3.3 Data set-up

Type	Amount(record)	Percentage(%)
Facts	400	100
Entities	25	-
Property	19	-
Dirty data	85	21.25
Duplicate	24	6

We evaluated a random sample of YAGO facts (400 sample sets) with different personal name entities and properties. The results shown in Table 3.3 are those of 400 facts, 25 personal name entities and 19 properties. Of them 85 records (21.25%) are dirty (contain Unicode characters) and 24 records (6%) are duplications. 23 records occur twice and one record occurs four times.

In the pre-processing stage, we need to remove unwanted data, transform data and detect duplicate facts from YAGO. This stage includes the tasks of:

1. Unicode character transformation, transforms a Unicode expression into a readable form. For example:

Hook \u002c_Line_\u0026_Sinker_\u00281969_film \u0029

is transformed to Hook,_Line_&_Sinker.

2. Data replacing replaces unwanted symbols, such as multiple occurrences of white-space characters to single white-space and underscore characters to single white-space.
3. Data removing removes unwanted words, such as the first prefix in for example 'wordnet' and 'wikicategory' and the suffix, which is a set of arithmetical values that is represented at the end of strings. For example, a text such as 'wordnet_performer_110415638', after removing the prefix and suffix, will be transformed to 'performer'.
4. Duplication data detecting. The system detects duplicate data before inserting the new data into the data catalogue.

3.2.2 Data Extracting

After a fact has been pre-processed, the next stage is data extraction. It extracts the subject and/or object from the facts that rely on its property. This stage focuses on extracting the personal name entities, their occupations, their alternative names, their relations and their occupation hierarchy. The five following major terms are extracted from YAGO2:

1. **Personal name entity extraction.** Personal name entities are extracted from subjects and/or objects which have properties: *actedIn*, *type*, *means*, *isMarriedTo* and *hasChild*. For example, *Tom Hanks* is a personal name entity that is extracted from the subject in triple form of fact:

Tom_Hanks actedIn Forrest_Gump.

YAGO distinguishes lexical ambiguity using occupation, or occupation and year of birth if occupation is repetition. As shown in Table 3.4, five occupations: *'merican football, artist, comedian, cricketer* and *motorcycle racer* are used to make a personal name *Chris Martin* unique. Moreover, the birth year is combined if personal name with occupation cannot distinguish person. As shown in Table 3.4, the year of birth is used for personal name ID 5 and 6 to distinguish two people who have both the same name and occupation. Therefore, the parentheses and its details after the personal name are retained to make the personal name unique.

Table 3.4 The unique personal name entities in our data catalogue

ID	Personal Names
1	<i>Chris Martin(American football)</i>
2	<i>Chris Martin(artist)</i>
3	<i>Chris Martin(comedian)</i>
4	<i>Chris Martin(cricketer)</i>
5	<i>Chris Martin(footballer born 1988)</i>
6	<i>Chris Martin(footballer born 1990)</i>
7	<i>Chris Martin(motorcycle racer)</i>

2. **Occupation category extraction.** Occupation categories are extracted from subjects and objects which have property: *subclassOf* in YAGO. The property *subclassOf* is used to generate from two properties in YAGO: Property *subclassOf* in YAGO is used for organizing occupation hierarchy in YAGO. For example,

fact = *musician subclassOf performer*
parent = *performer* is an object
child = *musician* is a subject

3. **Personal name occupation extraction.** The property *type* is used for mapping the correlation between individual personal name entity and its occupation. For example,

Tom Cruise type American actors

means *Tom Cruise* is an instance in the occupation category *American actors* or *Tom Cruise* is an American actor, where *Tom Cruise* is a subject and *American actors* is an object.

4. **Relations extraction.** Personal name relations are extracted from two properties in YAGO facts: *isMarriedTo* and *hasChild* for labelling a connection between the entities. For example,

Bill Clinton isMarriedTo Hillary Rodham Clinton.

Where Bill Clinton is a subject and Hillary Rodham Clinton is an object.

5. **Alternative name extraction.** An alternative name is extracted from fact that has the properties *means*, where the subject is an alternative name and the object is a personal name entity. For example,

Samantha Lewes means Tom Hanks.

This means that *Samantha Lewes* is an alternative name of personal name entity *Tom_Hanks*.

3.2.3 Data Matching

1. **Personal name surface form.** Personal name surface form is a term that is used to refer to a person in a document [19]. Each term that occurs in a document can be shared with multiple persons, so that it is complex for a computer to make a decision about identifying exactly who the person is. This problem is called personal name ambiguity. We classify personal name surface forms into two types: an ambiguous personal surface form and a single personal surface form. An ambiguous personal surface form is a surface form that is assigned to multiple personal name entities. For example, the surface form *Alan Curtis* is shared among different three people:

$$P(\textit{Alan Curtis}) = \{\textit{Alan Curtis (American actor)}, \textit{Alan Curtis (British actor)}, \textit{Alan Curtis (footballer)}\}$$

The second surface form is a single personal surface form, i.e. a surface form which has a single member. For example, the surface form *David Beckham* has only one instance to share this term:

$$P(\textit{David Beckham}) = \{\textit{David Beckham}\}$$

Definition 3.1. Given S is a set of personal name surface forms in a data catalogue D , where $s \in S$ is an instance in S . We use the notation $P(s)$ to denote personal name

entities who share each surface form s . We can draw a notation set of personal name entities in each personal name surface form s as:

$$P(s) = \{p_1, p_2, p_3, \dots, p_n\}$$

In this notation $p_1, p_2, p_3, \dots, p_n$ are instances of $P(s)$. For example, a set of persons who share the surface form *Bush* are:

$$P(\textit{Bush}) = \{\textit{Billy Green Bush}, \textit{Laura Bush}, \textit{George W. Bush}\}.$$

Sources of Personal Name Surface Form

We have two sources to create personal surface forms: the name of personal name entities and alternative names.

- Personal name entity is used for creating personal name surface forms. We move occupation data or birth date that is represented in parenthesis at the end of personal name for separating ambiguous entity. For example, the entity name *Chris Martin(American football)* will be transformed to *Chris Martin*.
 - Alternative names are any other names that used to refer to a person within a document, such as a nickname, pen name or occupation position. For example, the personal name *George W. Bush* can be represented in multiple references such as *W. Bush*, *Bush Junior*, *Bush II*, *G. Dub*, *G.W.B.*, *Dubya Bush*, *Bush 43*, *43rd President of the United States*.
2. **Personal name entity** is a set of personal name in a database and the unique identification of each name. YAGO uses occupation, birth year and/or nationality to distinguish lexical ambiguity e.g. *Thomas Cruise (footballer)* or *Mike Jackson (Australian entertainer)*.

Definition 3.2. The set of personal names is P , where each personal name $p \in P$. A personal name entity p is a unique instance in a data catalogue D which contains unique values. For example,

$$P = \{\textit{Bill Clinton}, \textit{Chris Martin}, \textit{Chris Martin}, \textit{Chris Martin(comedian)}, \dots\}$$

3. **Occupation category.** The property *subclassOf* in YAGO are used for forming occupation category hierarchy structures. Figure 3.3 shows an example result of the

occupation categories hierarchy extracted from YAGO2. The highest class is person and connected to subclass and the lowest are the leaf classes from the Wikipedia categories.

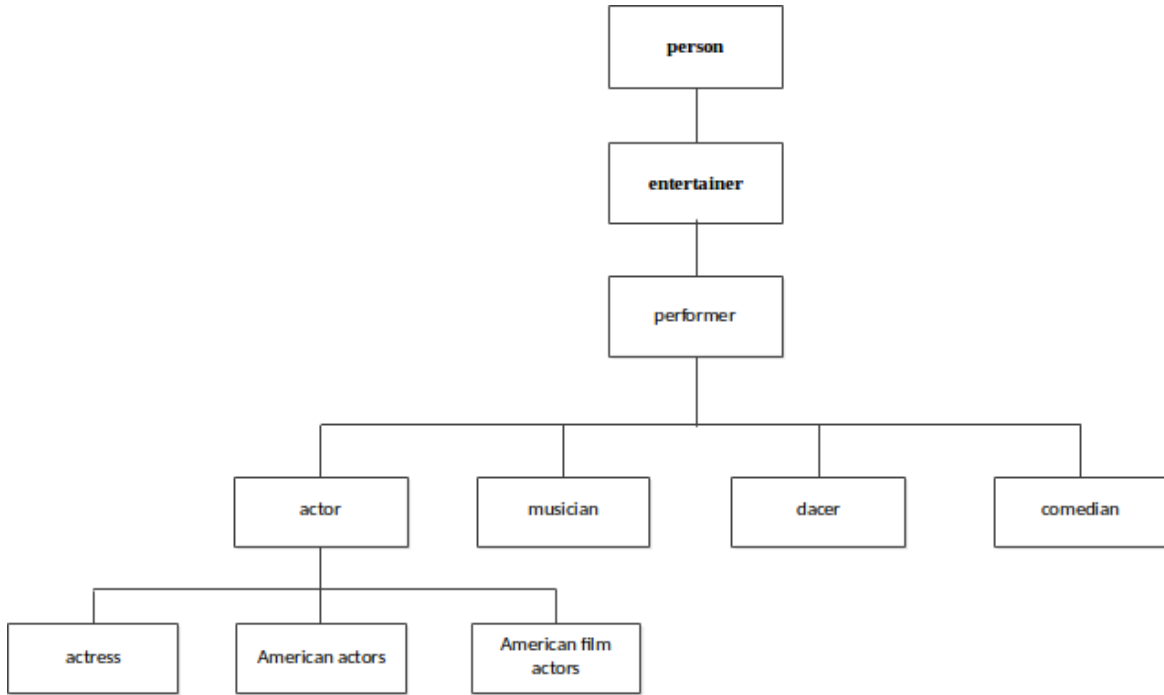


Fig. 3.3 An excerpt of YAGO entertainer class taxonomy

4. **Personal name occupation.** We used the facts with property *type* to form the relations between personal name entity and its occupations.

Definition 3.3. Given O is a set of occupations, where each occupation $o \in O$.

$$O = \{person, entertainer, performer, actor, musician, dancer, American actors, \dots\}$$

Given P is set of personal name entities, where each personal name entity $p \in P$. Instance p may have one or more occupations, written as $O(p)$.

$$O(\text{Tom Cruse}) = \{\text{American actors}, \text{American film producers}\}$$

5. **Personal name entity relations.** Personal name entity relations are created from two properties which extracted from YAGO: *isMarriedTo* and *textithasChild*. For example, entity *David Beckham* *isMarriedTo* *Victoria Beckham* or *David Beckham* *hasChild* *Brooklyn Beckham*.

Definition 3.4. Given the set of binary relations, name is R and $r \in R$ is one relation name. The relationship between the personal name entities is written by $\{p, r, (p_1, p_2, \dots, p_n)\}$. We demonstrate the relations between the entities below:

$R = \{isMarriedTo, hasChild\}$

$P = \{David\ Beckham, Victoria\ Beckham, Brooklyn\ Beckham, Romeo\ James\ Beckham, Harper\ Seven\ Beckham, Cruz\ David\ Beckham\}$

$r_1 = isMarriedTo$

$r_2 = hasChild$

$\{David\ Beckham, isMarriedTo, (Victoria\ Beckham)\}$

$\{David\ Beckham, hasChild, (Brooklyn\ Beckham, Romeo\ James\ Beckham, Harper\ Seven\ Beckham, Cruz\ David\ Beckham)\}$.

3.3 Occupation Architecture for Personal Name Disambiguation(OAPnDis)

The occupation taxonomy in YAGO is an important component and plays a key role in generating OAPnDis. However, we found some evidences to suggest the occupation hierarchy in YAGO being wrong as the child class that is derived from Wikipedia categories is not related to the super-class. For example, in a set of Wikipedia categories:

Presidents of the United States, Presidents of Germany, Presidents of FIFA, Presidents of Marquette University and Presidents of Clemson University are under WordNet classes:

head of state president \rightarrow *representative* \rightarrow *negotiator* \rightarrow *communicator* \rightarrow *person*

and in a set of Wikipedia categories:

Prime Ministers of the United Kingdom, Prime Ministers of France, French Ministers of Justice and Rectors of the University of Glasgow are under WordNet classes:

curate \rightarrow *clergyman* \rightarrow *spiritual leader* \rightarrow *leader* \rightarrow *person*

The occupations above are assigned to the wrong WordNet classes (e.g. a *Presidents of FIFA* is not a *head of state*). This problem may decrease the efficiency and quality in personal

name disambiguation. This is because the occupation taxonomy is an important variable in entity disambiguation [8, 19, 62, 63].

Recently, there have been two major methods used for grouping an entity: an ontology taxonomy and a thematic domain [64]. Firstly, an ontology taxonomy is used in YAGO and DBpedia. DBpedia created its own ontology taxonomy and occupation taxonomy has been used for grouping people. The occupation taxonomy in DBpedia is clean, but provides less depth; in fact, most of occupation hierarchies in it are just one level deep.

YAGO derives taxonomy from WordNet and Wikipedia that provides occupation hierarchy in multiple levels. However, the taxonomy in YAGO is dirty [22]. Freebase, Wikipedia and Dmoz use a thematic domain to classify entity. The main disadvantage of the thematic domain is that it does not represent the correct concept in each entity. For example, *Stephen King* is put in the categories: *Author*→*Books*→*Arts and Entertainment* but *Stephen King* is not a book. As the result, we design our occupation taxonomy to construct the concept of person as described in the next subsection.

3.3.1 OAPnDis Architecture

OAPnDis is an occupation taxonomy which is designed for personal name disambiguation. OAPnDis architecture is based on YAGO classes, but the layer 1 classes are derived from web directories. The architecture of OAPnDis is shown in Figure 3.4. The architecture consists of four layers, as described below.

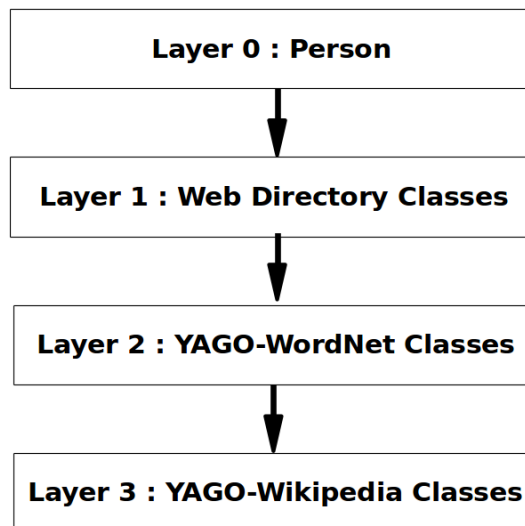


Fig. 3.4 OAPnDis architecture

1. **Layer 0.** The person is the top class in OAPnDis architecture. An instance assigned under this layer is a person.
2. **Layer 1.** Layer 1 classes are derived from the web directory. However, the context of each category is edited before being merging into OAPnDis (e.g. arts to artists, sport to sportsperson) to depart from the thematic directory problem. A key benefit of this step is that the occupations can be brought from the general term (e.g. *entertainer and artist, sportsperson, educator, politician*) to their narrow specific types (e.g. *US president, American actor, English female model*). Furthermore, some unrelated classes are moved. For example, the class *model* is moved from *worker* to *entertainer and artist* and *UK prime minister* is moved from *leader*→*spiritual leader* to *politician*. As result, the mapping of duplicate or unrelated classes is solved.
3. **Layer 2.** Layer 2 are the classes that YAGO derives from WordNet. Some classes in layer 1 may appear in layer 2 (e.g. *entertainer*). A class in layer 2 is discarded when the system detects that it is duplicate class.
4. **Layer 3.** Layer 3 are the classes that YAGO derives from Wikipedia. YAGO maps Wikipedia categories to WordNet classes by considering the head-word of Wikipedia categories. For example, *poets* is the head-word of the category *Canadian poets* in Wikipedia so *Canadian poets* should be a sub-class of *poets* in WordNet taxonomy. However, determining only head-words may cause linking error among WordNet taxonomy and Wikipedia categories. For example, the *Presidents of FIFA* category is a subclass of *head of state president* in WordNet taxonomy like *Presidents of the United States* because they have the same head-word of *presidents*. In our work, we work through the context of Wikipedia categories before mapping them to layer 2.

As the result, our occupation hierarchy is established and shown in Figure 3.5. *Person* is a root node, and has three children: *Entertainer and Artist, Politicians and Sportsperson*. The node *Politician* is a parent of nodes: *President, Governor and Prime Minister*.

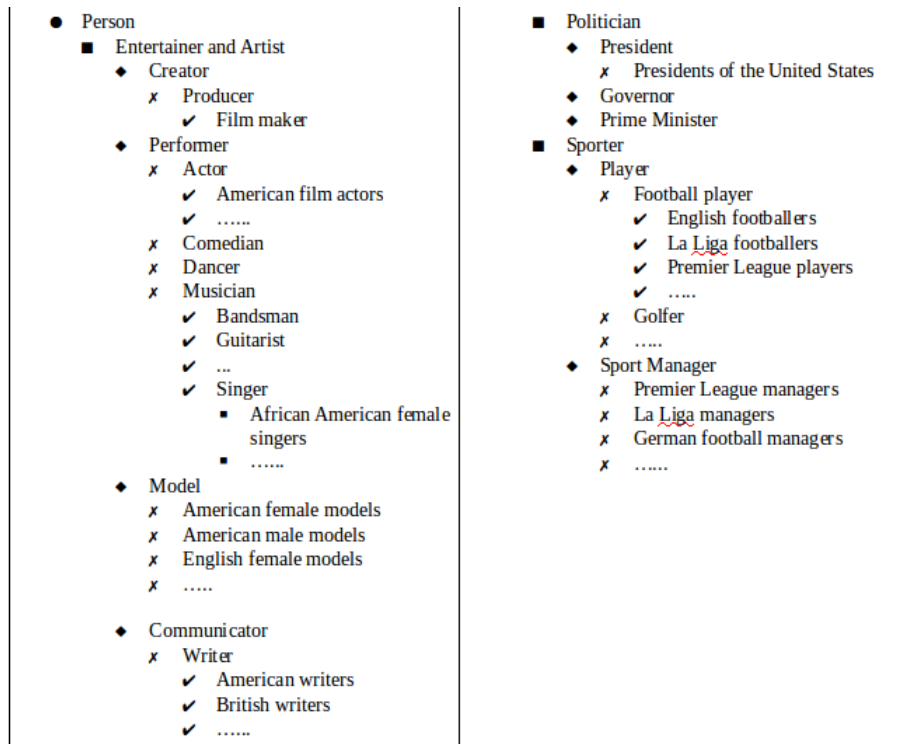


Fig. 3.5 An excerpt of the occupation taxonomy

3.3.2 Personal Name Concepts

The personal name concept is an abstract occupation category that is used to represent the character of person. In this thesis, the occupation taxonomy is used to demonstrate the character of personal name entity. An occupation is the primary feature used to distinguish lexical ambiguity [48]. This subsection is now ready to introduce how to construct the conceptualisation of each personal name, which consists of two steps:

- Building occupation taxonomy based on the architecture of occupation taxonomy.
- Building individual entity concept trees that are derived from occupation taxonomy.

Occupation Taxonomy Tree

The occupation taxonomy is a typical tree which represents the relationship between the super class and the lower class. A class in occupation taxonomy is a single occupation; it becomes a node in the hierarchical structure of the tree.

The main components of occupation trees are nodes and edges. A starter node without a parent refers to a root node; it is an ancestor of all nodes in the tree. The root node in the occupation tree is *Person*. A node without children is a leaf node; most of these are

derived from Wikipedia categories. Siblings are child nodes that have the same parent. The connections between nodes are called edges.

We use the Modified Preorder Tree Traversal algorithm (MPTT) [39, 68] to solve the problem about how to collect hierarchical data in a database. The issue is that a database uses a flat structure to store data. The MPTT approach is to use *"lft"* and *"rght"* attributes (as *"left"* and *"right"* are the reserved keywords in SQL) to store the relationships between parent and child nodes. The MPTT algorithm is shown in Figure 3.6.

This thesis uses an example in Figure 3.6 to describe the MPTT algorithm. The algorithm travels starting from Root node A, from left to right, one level at a time, going down along the edges of tree and assigning a value on the left and right side to every nodes in the tree. The final value is assigned to the right side of the root node.

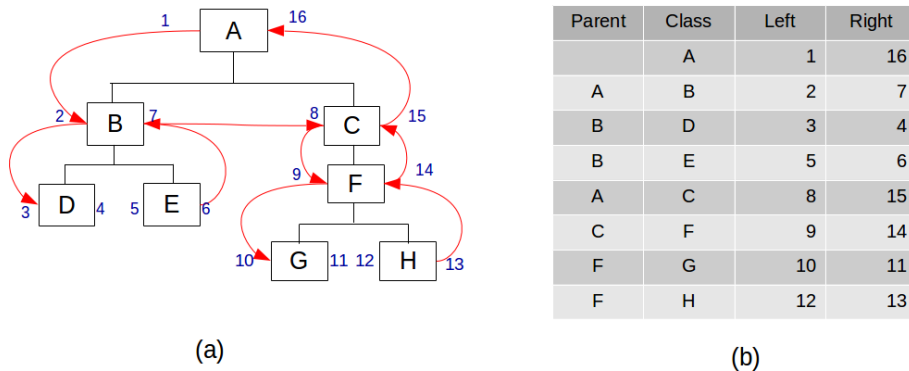


Fig. 3.6 A Modified Preorder Tree Traversal(MPTT) algorithm

A great deal of MPTT algorithm with *"lft"* and *"rght"* values returns the path of node within a single query. For example, if we want to display the path of node *E*, the SQL query could be:

"SELECT class FROM tree WHERE lft < 5 AND rgt > 6 ORDER BY lft ASC;"

The return values of this SQL query are A, B and E. We thus adopt the MPTT algorithm to our work for constructing entity concepts in each individual instance.

We will now describe how to create the personal name concept. The concept is based on OAPnDis, as described in section 3.3.1. The algorithm has two steps to create the personal name concepts in each instance.

1. Building occupation tree in each instance. Given $O(p) = \{0_1, 0_2, \dots, 0_n\}$ is a set of occupation categories in each entity. For example, in an instance *Arnold Schwarzenegger* has five occupations:

$O(\text{Arnold Schwarzenegger}) = \{\text{Politicians, actor, American film producers, American film actors, American film directors}\}.$

The algorithm uses the classes of layer 1 to be a root node of the personal name concept because it can classify people in overview. The occupation categories in $O(p)$ are used to generate an occupation tree in each instance using the SQL query in Section 3.3.1. The occupation node o_i is the leaf node, and it can inherit from their parent including the root node. For example, *Arnold Schwarzenegger* is an actor, so he can be a *performer* and an *Entertainer and Artist*. As the result, when the occupation trees are generated, the number of trees is equal to the number of occupation categories in $O(p)$.

Let $O(p) = \{o_1, o_2, \dots, o_n\}$ is a set of occupation trees in each personal name entity. Where t_i is a set of occupation categories hierarchy in each c_i . For example, $T(\text{Arnold Schwarzenegger})$ will have five trees below:

$o_1 = \{\text{Politicians}\}$

$o_2 = \{\text{Entertainer and Artist, performer, actor}\}$

$o_3 = \{\text{Entertainer and Artist, creator, producer, film maker, American film producers}\}$

$o_4 = \{\text{Entertainer and Artist, performer, actor, American film actors}\}$

$o_5 = \{\text{Entertainer and Artist, creator, producer, film maker, film director, American film producers, American film directors}\}$

2. Building the personal name concepts. After all the occupation, trees in each personal name entity have been created and the personal name concepts for each person have been generated under these trees. The root nodes of each occupation tree are used to identify whether or not these occupation trees have the same concept.

All nodes in the occupation tree that have the same root node merge, and the duplicate nodes are removed to make the node unique in each concept. Given $O(p) = \{o_1, o_2, \dots, o_n\}$ is a set of occupation trees in each personal name entity. Note that any o_i in $O(p)$ are similarity consistent if their root node is equal. Let $C(p) = \{c_1, c_2, \dots, c_n\}$ is a set of personal concepts for each personal name entity.

$$c_i = \bigcup_{i=1}^n o_i$$

where root of all o_i are equal.

We use an example of the personal name entity *Arnold Schwarzenegger* occupation trees to explain how to generate personal name concepts. In general, a conceptual tree

is produced as follows:

- Matching the root node in $O(p)$. After this step, a set of root nodes in $O(p)$ is generated. Hence, $T(\text{Arnold Schwarzenegger})$ has two root nodes: {Politicians, Entertainer and Artist}.
- A tree o_i which has the same root node merges to create personal name concepts. A process loops until the final o_i merges. For example, $C(\text{ArnoldSchwarzenegger}) = \{c_1, c_2\}$, where c_1 and c_2 describe below:

$$c_1 = \{o_1\}$$

$$c_2 = \{o_2 \cup o_3 \cup o_4 \cup o_5\}$$

As a result, the personal name entity *Arnold Schwarzenegger* has two personal name concepts. In the first one he is a *Politician*, and in the second one he is an *Entertainer and Artist*. The details of the concepts which are generated in these steps are shown in Figure 3.7.

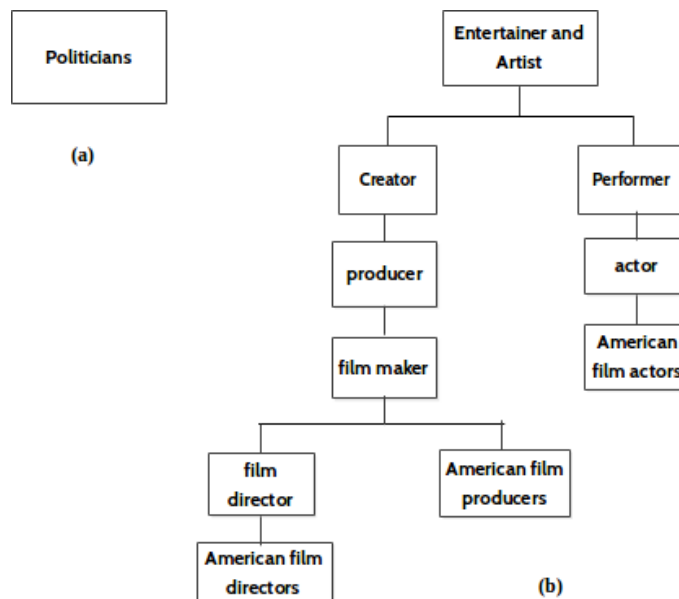


Fig. 3.7 Examples of personal name concepts of *Arnold Schwarzenegger*: (a) Politician and (b) Entertainer and Artist

3.4 Personal Name Disambiguation Data Catalogue (PNDDC)

PNDDC is a back-end database that is used to store personal surface forms, personal name entities, personal name concepts, occupation taxonomy and personal name entity relations.

PNDDC is a knowledge base which is used in Chapter 5 for personal name disambiguation. This thesis has designed the structure of our data catalogue similarly to Cucerzan [19], including in the personal name surface form and personal name entity details. A major advantage of this structure is that it can distinguish between the component of generating candidate entity and the component of entity disambiguation.

Cucerzan created surface form from four sources in Wikipedia:

- The titles of entity pages.
- The titles of redirect pages.
- The disambiguation pages.
- The references to entity pages in other Wikipedia articles.

In the second part, entity details contain three items:

- The entity name that is extracted from the entity title.
- Tags or categories that are assigned to each entity.
- Contexts are the contextual clues for each entity; these are extracted from the first paragraph of an entity page and the corresponding pages refer back to the target entity.

In our work, we use the data sources and technique from Cucerzan, but we do not use catalogue structure. Firstly, the data is 'pre-processing' to remove dirty data before storage. The data pre-processing task can boost the efficiency and the precision of data matching [17]. Secondly, the sources of our data catalogue are derived from YAGO. Finally, the second part of our data catalogue contains different information from Cucerzan because the features and the methods in entity disambiguation are different.

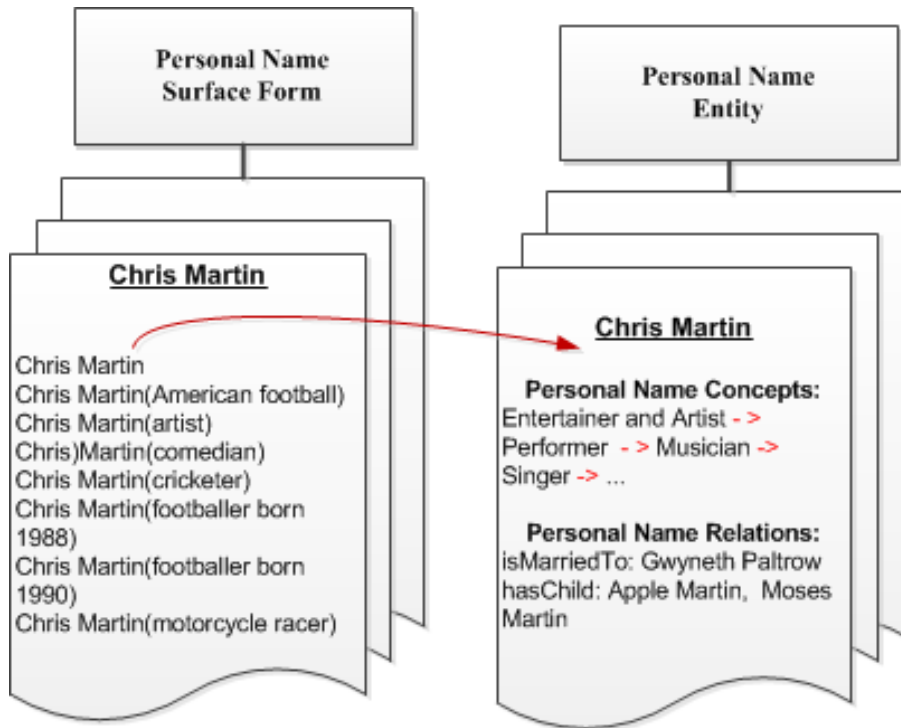


Fig. 3.8 PNDDC structure

Figure 3.8, above, shows a PNDDC structure. The personal name surface form is for the term *Chris Martin*, which is shared by eight different persons. The figure shows above how to create the link for mapping between each term and personal name entity. For example, the personal name *Chris Martin* in the surface form is mapped to eight persons, and the first name *Chris Martin* maps to the personal name *Chris Martin*, a lead vocalist, pianist and co-founder of the British rock band Coldplay. The entity's name, concepts and relations are represented in the personal name entity details section.

3.5 Experimental Results and Discussion

In this section, the data sets and results that are used to build PNDDC are described. PNDDC consists of personal surface form, personal name entity, personal name concepts, occupation taxonomy and personal name entity relations.

3.5.1 Data Sets

Table 3.5 shows the sources from YAGO which are used to construct PNDDC. We consider six properties of YAGO fact:

actedIn, means, type, subclassOf, isMarriedTo, hasChild.

PNDDC stores information about people who have a job in three areas: entertainer, sportsperson and politician. We model PNDDC independently from the data source; this means that PNDDC can create from any knowledge bases which provides personal name, occupation hierarchy, name variations and their relationships. The property *actedIn* is used to extract personal name entities, because the subjects in these facts are actors. For example, in the fact that *Tom_Hanks actedIn Forrest_Gump*, the property *actedIn* seems to imply that the subject *Tom_Hanks* is an actor. The property *means* is the collection of the alternative names of each personal name entity that are useful for us in building our personal name surface form. For example, *DB7* means *David_Beckham*. This means that *DB7* is an alternative name for *David Beckham*. The property *type* and *subclassOf* are used to construct our personal name concepts and the property *isMarriedTo* and *hasChild* are the personal name entity relations. Table 3.5 shows the amount of facts in YAGO that are used to create PNDDC.

Table 3.5 The data sets in the personal name disambiguation data catalogue

Properties	Facts
<i>actedIn</i>	126,636
<i>means</i>	6,741,479
<i>type</i>	8,414,398
<i>subclassOf</i>	367,040
<i>isMarriedTo</i>	27,708
<i>hasChild</i>	15,471

3.5.2 Result and Discussion

The results obtained from PNDDC are summarised in Table 3.6. As a results, PNDDC has 107,058 entities, 145,638 personal surface forms, 4,203 personal name entity relations, 321 occupations and 105,543 personal name concepts. We found that the terms: *Chris, James,*

Christopher, Gina and Philip are the highest sharing. There are shared with 18 people. In addition, the personal name: *John Anderson, John Brown, Robert Brown* refer to 11 different people. Most people have no more than two personal name concepts. Only 0.06% have two concepts.

Table 3.6 The personal name data catalogue and personal name concepts

PNDDC Descriptions	Total
Personal name entities	107,058
Personal name surface forms	145,638
Number of <i>isMarriedTO</i> personal name entity relations	3,872
Number of <i>hasChild</i> personal name entity relations	331
Number of occupations	321
Number of persons who have single personal name concepts	105,482
Number of persons who have multiple personal name concepts	61

The final results show that an occupation is a significant feature in the in disambiguating lexical ambiguity, as can be seen by comparing our results with the previous study of Han and Zhao [37].

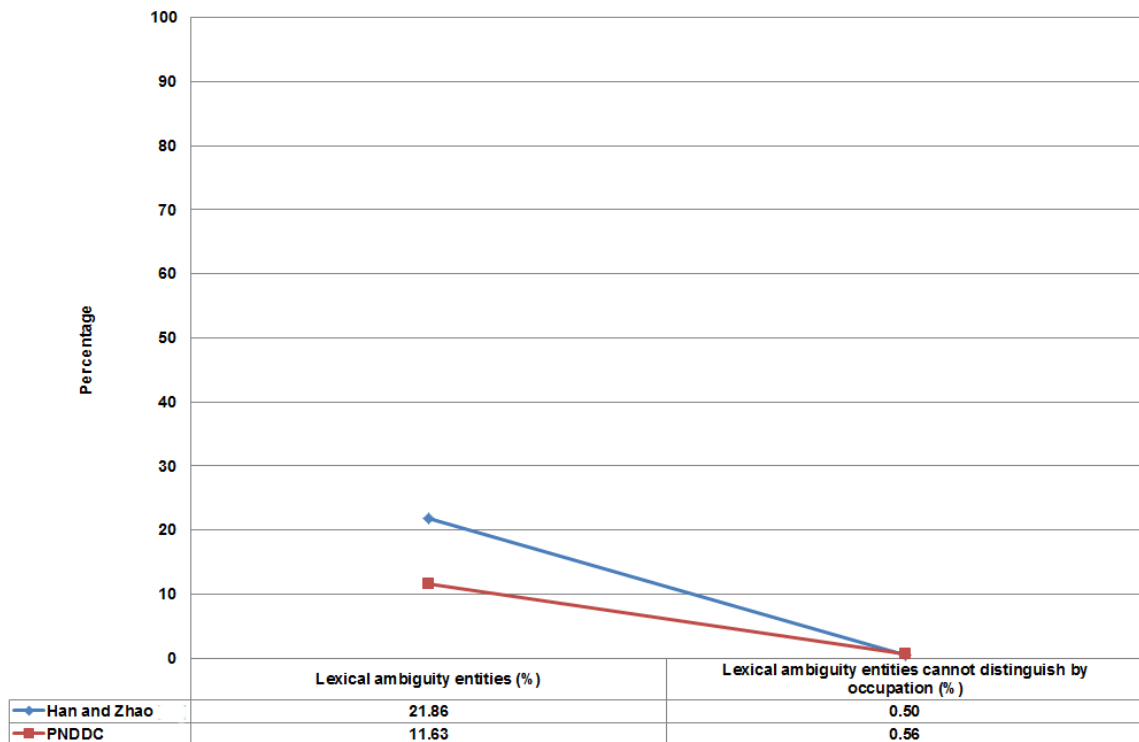


Fig. 3.9 Percentage of lexical ambiguity compare with Han and Zhao [37].

Han and Zhao extracted occupation categories from Freebase, and mined the reference entity tables using a web-querying method. However, Han and Zhao used only a single occupation to define the personal name when a name has lexical ambiguity (e.g. *Andrew Powell(Psychiatrist)* and *Andrew Powell(Musical Artist)*). Our work, instead, uses personal concepts to solve the problem of lexical ambiguity. The details are described in Chapter 5. We conduct 107,058 personal name entities, while Han and Zhao who gather 183,284 personal name entities. Our verification metric for evaluation is the percentage of lexical ambiguity entities that cannot distinguish by a single occupation against Han and Zhao. The results are shown in Figure 3.9.

The results of our study are consistent with Han and Zhao, who suggest that occupation is useful information to distinguish lexical ambiguity. Of all personal name entities, it has 11.63% lexical ambiguity entities compared with 21.86% for Han and Zhao. The most interesting finding is that for just 0.56% of case from our work and 0.50% from previous study [37] occupation cannot be used to distinguish lexical ambiguity because two people have a similar name and career. Occupation nearly always works in sorting out lexical ambiguity and so it is a significant feature in lexical ambiguity disambiguation.

3.6 Related Work

Creating personal name surface form and personal name concepts are significant tasks in personal name entity linking because personal surface form and personal name concepts provide useful information in generating a set of candidate entities and personal name disambiguation. There are several works which have been focused on to create a data catalogue for named entity disambiguation.

We start with Bunescu and Pasca [8], who used Wikipedia as a source for creating an entity names dictionary. There are three sources that Bunescu and Pasca use to construct entity name dictionary: Entity pages, Redirect pages and Disambiguation pages. The entity name dictionary includes two fields: the term, and the set of entities denoted by this term. An entity is extracted from an article title under the heuristic rules, to verify that a title is a proper name. A set of terms in an entity names dictionary is derived from entity names, redirect names and disambiguation names in each entity. For example, the term *John Williams* may refer to three different people: *John Williams (composer)*, *John Williams (wrestler)*, *John Williams (VC)*.

Cucerzan [19] extends the data sources from Bunescu and Pasca by adding the references to entity pages in other Wikipedia articles for constructing the entity name dictionary and applying the term '**surface form**' to refer to the entity name dictionary. However, Hachey et al. [34] find that the link anchor texts not as good data in generating a set of candidate entities. Drredze et al. [24] argue that both entity name dictionaries which are introduced by Bunescu and Pasca [8] and Cucerzan [19] are limited because of Wikipedia specific dependencies.

Limaye et al. [46] introduce a new data source YAGO for creating a data catalogue, which is used for annotating web tables. The researchers annotate table cells by matching them with entity IDs in a data catalogue, annotating pairs of columns with a binary relation in the catalogue and associating one or more types with each column of the table. The catalogue contains types, entities and relations. Types are related by subtype relation $T_1 \subseteq T_2$ and entity E may be instance of one or more types. Each type and entity may be described by one or more string that is called 'lemmas', written by L(T) and L(E) where T and E are each instance in types and entities. The relationship between a pair of column header relations is written by $B(T_1, T_2)$. For example, a relation *BornPlace(Person, Country)* means that the relation *BornPlace* consists of two member types: person and country. Therefore, the value of a couple of cells or the data record is written as $B(E_1, E_2)$. For example, *Steven Gerrard was born in United Kingdom* can be written as *BornPlace(Steven Gerrard, United Kingdom)*.

Bunescu and Pasca [8] and Cucerzan [19] used Wikipedia categories for name entity

disambiguation. However, categories in Wikipedia are dirty, as is the thematic domain representation, so it is not clean enough for purpose of ontology purpose [63]. Han and Zhao [37] use a reference entity table for name disambiguation. The reference entity table is a set of personal names and the corresponding occupation (e.g. *Michael Jordan, Basketball Player*) which is extracted from Freebase. Han and Zhao use only a single occupation to describe a person.

In our work, we propose a new occupation taxonomy architecture: OAPnDis, which is derived from YAGO and the web directory. We employ a clean and well-formed occupation taxonomy which provides a more details to represent a personal concept in each person. Our approach differs from Han and Zhao because we introduce the personal name concept, which is a list of occupations to describe a person.

3.7 Conclusions

This chapter introduces a new technique to create a personal surface form, a new occupation taxonomy OAPnDis and a PNDDC structure.

Firstly, PNSFM is introduced to construct the personal surface form and the personal name entity relations. Unlike the previous study, our technique is independent from the data source and includes a data pre-processing component prior to integration in a database. The independent source means that the personal surface form or the personal name entity relations can be generated from any knowledge base. Data pre-processing is a data cleaning technique that is used to remove dirty data before extracting and matching facts.

Secondly, we propose a new occupation taxonomy: OAPnDis. The architecture is based on two well known knowledge bases: Dmoz and YAGO. The new occupation architecture can create a well-formed and semantic occupation taxonomy. As the result, personal name concepts are created based on OAPnDis.

Finally, we introduce the PNDDC structure. PNDDC can be separated into two parts: personal name surface form and personal name entity details (personal name entity, personal name concepts and personal name entity relations). This structure is advantageous because this thesis wants to distinguish information between the searcher component and the disambiguator component in personal named entity linking, as described in Chapter 5.

We have found that occupation is the best feature to disambiguate lexical ambiguity and that only 0.56% of lexical ambiguity cannot be distinguished using a single occupation.

Chapter 4

Personal Name Transformation With Context Free Grammar

In this chapter we introduce Personal Name Transformation Modules (PNTM) that are based on Context Free Grammar (CFG) rules and personal name dictionary. PNTM is used to transform a variety of name formats (e.g. nickname, alias, different order) into a uniform representation (e.g. *DB7* vs. *David Beckham* or *Beckham*, *David* vs. *David Beckham*).

4.1 Motivation

A proper name is a key component in personal name entity linking due to a lack of primary key representation. Textual similarity metrics (e.g. Jaro-Winkler and cosine similarity) or the learning based approaches (e.g. SVMs, decision tree and naïve Bayes) are widely used in data matching [2]. Jaro-Winkler is a character-based comparison metric that was designed to solve typographical error problems [25]. Cosine similarity is a token-based similarity metric that aims to handle the rearrangement of words. However, several studies [6, 15, 54, 74] have revealed that the Jaro-Winkler metric performs well with personal name data. Furthermore, the study [18] found that the cosine similarity technique is faster than Jaro-Winkler by 16.67 times. Moreover, the cosine similarity metric is used in [8, 19, 24] for name entity disambiguation.

Table 4.1 Example of personal name variations.

ID	Personal names	A unique personal name format
1	George W. Bush	George W. Bush
2	Bush, George W.	George W. Bush
3	Bush, G.W.	G.W. Bush
4	43 rd President of the United States	George W. Bush
5	Bill Clinton	William Clinton
6	William Clinton	William Clinton
7	Timberlake	Justin Timberlake
8	Justin Timberlake	Justin Timberlake

A fundamental problem in personal name matching is referential ambiguity [3, 7]. Actually, a wide variety of contexts can be used to refer to an individual person (a person has many names) such as the official position, nickname, short name or full name. As shown in Table 4.1, we can see that:

1. ID 1-4, four different terms are used to refer to the same person: *George W. Bush*.
2. Personal names ID 1-2, the same personal name but appears in a different order.
3. The nickname *Bill* is used to refer to the given name *William* in ID 5.
4. The alternative name *Timberlake* is used to refer to the personal name *Justin Timberlake*.

Arasu and Kaushik [3] have suggested that text similarity measurement may provide a poor effective similarity value when we use different words to mention the same person. The personal name IDs: 1 and 2 are equal but are represented using a different order and cannot be detected by Jaro-Winkler. On the other hand, cosine similarity can detect the similarity between IDs: 1 and 2 because the order of words is insignificant for this algorithm. Moreover, both Jaro-Winkler and cosine similarity cannot detect the similarity between two strings if two strings are completely used a different letters (e.g. IDs: 1 and 4). Likewise, the similarity scores between IDs: 2 vs. 3 and IDs: 7 vs. 8 based on Jaro-Winkler or cosine similarity are less because only some parts of names are similar.

Christen et al. [16] introduced probabilistic Hidden Markov Models (HMM) for cleaning and standardisation of personal names and addresses. The HMM is used to segment the personal name component. The accuracy results show that a rules based approach is better than HMM. The HMM failed to segment the personal name component when a person has

either two given names or two last names. Furthermore, this technique is insufficient for our problem because this method only solves the different order in a personal name.

Arasu [3] proposed a programmatic framework based on CFG and a personal name dictionary to transform personal names and their affiliations which are represented in multiple forms into a uniform representation. A *programmatic framework means how an input record is transferred to output records and it is specified using a declarative programme* [3]. The advantage of this module is not only dealing with the different sequences of personal components but also transforming a nickname into a given name. The CFG method differs from the black-box similarity function because it enables us to understand the internal structure of a personal name component. For example, the word *Rose* that occurs in a personal name cannot transform into *Roxana* if it is not a given name. However, there are two certain drawbacks with the use of this module:

Firstly, the objective of this framework is to transform personal names that are stored in a database that are less variation than a mentioned name within a web page. In our work, we expand the personal name dictionary by adding alternative names that we extracted from the YAGO knowledge base. Therefore, the new personal name dictionary contains a list of prefixes, suffixes, given names, last names, nicknames and alternative names.

Secondly, Arasu parsed a whole name over personal name dictionary and then the uniform weighting scheme is used to evaluate whether each word in a whole name could be a prefix, suffix, given name or last name. Arasu gave a different non-negative real number weight to a CFG rule and the output which has the lowest weight will be considered. However, the weight value has limitations when a word can be qualified as multiple answers and the weighting scores in CFG rules are equal. For example, the words *Ferguson* or *Black* can be used as both a given name and a last name. To handle this problem, we combine regular grammar of English personal name patterns and a set of CFG rules to segment a sequence of strings in a personal name. After that, the type of each word in a sequence is identified (e.g. it is a prefix, a suffix, a given name, a last name, a nickname or an alternative name) and exactly matched over the type of names in the dictionary. As a result, the uniform weighting scheme is removed from Personal Name Transformation Modules (PNTM).

The purposes of this chapter as follows. Firstly, Section 4.2 describe the background to the regular grammar for English personal name, Context Free Grammar and String matching techniques. Section 4.3 describes the PNTM based on CFG and personal name dictionary to transform multiple personal name patterns into uniform representation. Section 4.4 describes experimental results and discussion. This section aims to compare the performance of two string matching techniques: Jaro-Winkler and cosine similarity within terms of which technique will provide the best results in personal name matching when working

with PNTM.

4.2 Background

The World Wide Web is a huge source of public documents that are published by various organisations and individual users such as government, official website and personal websites [11]. The Internet provides various types of information such as news, financial, products and personal information. On the other hand, unstructured data on the Internet also lacks uniform representation because it has a heterogeneous data design; the data comes from multiple sites of sources so that it is hard to handle like a single database [11].

Dirty data comes for various reasons such as lack of design, data entry mistakes and/or misunderstanding. Hence, data cleaning is needed before integrating data records from multiple sources into a database. Data cleaning is an important part of the ETL (extraction, transformation, loading) in a data warehouse when heterogeneous data sources need to be integrated [35]. The major objective of data cleaning should be supported discovery and elimination of all significant errors and inconsistencies both in a single source and when integrating various sources [55].

In this section, we give an overview of the techniques that have been integrated for PNTM including English personal names, Context Free Grammar (CFG) and two string matching techniques: Jaro-Winkler and cosine similarity.

4.2.1 English Personal Names

Personal names are a group or set of words which are highly constrained grammatically and generally include proper nouns that form given names and family names [67], e.g. *George Bush*. However, recently personal names can be formed using a mixture of proper nouns and common nouns, e.g. *43rd President of the United States* or *Ginger Spice*.

Anglo-Saxon countries are countries which use English language as the official language or English-speaking countries. Particularity, Anglo-Saxon countries include Australia, Canada, New Zealand, the United Kingdom, Ireland and the United States. In this thesis, we deal with the personal names from Anglo-Saxon countries and the term English personal name has come to be used to refer to these names.

English personal names have specific characteristics, forms and structures. The regular grammar consists of five components: prefix, given name (GN), middle name (MN), family name (FN) and suffix. A given name can be shortened or replaced by nickname (NN). For example, we used the nickname *Bill Clinton* to refer to *William Jefferson Clinton, the 42nd*

US President. Prefix, middle name and suffix are optional components. Meanwhile, these components may not be used to represent a personal name in a document. Additionally, person may use a different prefix or suffix for reference e.g. *Mr. John Smith* or *Dr. John Smith*.

The structure of a personal name is close to that of an English sentence because it consists of two main parts: given name(subject) and family name (predicate). A given name may be extended by a prefix and/or a middle name (noun phrase) or replaced by a nickname (pronoun). Figure 4.1 shows how a prefix, suffix, given name or family name are combined to produce a single name. The shaded rectangles including prefix, GN, FN, AN and suffix are the components we need to evaluate or match over the personal name dictionary. The shaded circles including MN and SN are the components that we do not match over any names in the personal name dictionary and we leave the output like the original values. The components in the big white rectangle produce output under CFG rules and the components outside the rectangle (prefix and suffix) produce an empty value.

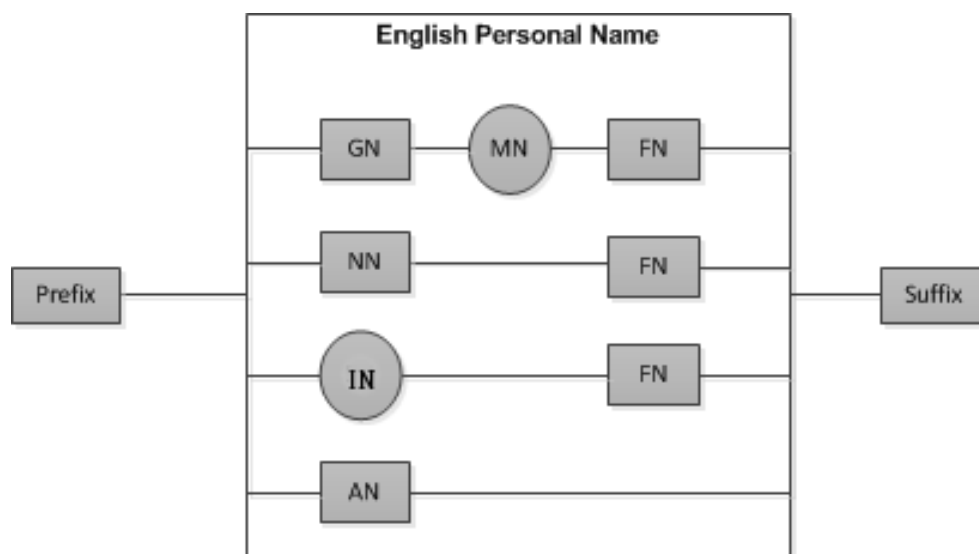


Fig. 4.1 Regular grammar for English personal names.

Note: GN = Given Name, FN = Family Name, MN = Middle Name, NN = Nickname, IN = Initial Name and AN = Alias Name

The list of abbreviated words in Figure 4.1 is explained below:

1. **Prefix** is a word that describes the person status, job or academic degree. Titles will come before a name (prefix) or follow a name (suffix) depending on the type of title. We classify titles into three groups: institutionalised titles, pseudo titles and academic titles.

- Institutionalised titles are used to indicate gender, status or job position. For example, *Mr. Barack Obama* or *Sir Alex Ferguson*.
 - Pseudo titles or job titles can come before or after personal names, however before is preferable. For example, *President Barack Obama* or *Barack Obama, the 44th President of the United States*. A period is used between a personal name and a pseudo title when it comes after personal names.
 - Academic degree titles come after personal names with a period to separate name and degree. For example, *John Smith, Ph.D., Biology* or *Professor Steve Chapman, FRSE, FRSC, C.CHEM*.
2. **Suffix** is a word we use when people in a family share the same name. General suffix titles are Jr., Sr., III and so on. We do not use a period between a name and a general suffix. For example, we write *John Smith Jr.* to refer to John Smith's son or *George Bush Jr.* to refer to *George W. Bush*.
 3. **GN**: Given name is a name that was given when you were born or it is a person's first name and middle name (MN) which is used to refer to a person in a family.
 4. **FN**: Normally, in Anglo-Saxon countries, a baby will have their father's family name.
 5. **NN**: Nickname is an alternate name that is usually used informally. Nickname is normally based on first name, physical characteristics such as *Ginger, Pinky, Shortie*, personality such as *Motor mouth* or from job titles or social standing such as *Doc, Moneybags*.
 6. **IN**: Initial name is an initial letter in a given name or where a middle name is used to abbreviate proper name. In particular, the first letter in a first name or middle name can be used to abbreviate a personal name, e.g. *John Smith* to *J. Smith*.
 7. **AN**: Alternative name is another name that is used to refer to a person e.g. *10th President of the United States* which refers to *John Tyler*. In addition, in this thesis, a personal name which has a single word or a group of words that have at least five words are classified into a group of alternative names e.g. *Bush* or *Da Silva Dal Belo Felipe*.

Moreover, a personal name can be written in multiple arrangements. It can be started with family name comma and followed by given name and/or middle name. For example, *Smith, John* or *Smith, J.* refers to *John Smith*.

The regular grammar of English personal names is a major key point in Context Free Grammar rules to transform the variation patterns of personal names into a unique pattern and to produce the position of given name and family name. In the next section we describe Context Free Grammar rules which are the highlighted technique we used for personal name transformation.

4.2.2 Context Free Grammar (CFG)

A context-free grammar is a set of rules that defines how the patterns of strings can be generated which can be shortened CFG [52].

Definition 4.1. A context-free grammar (G) is a quadruple $G = (N, \Sigma, P, S)$ Where:

N is a set of characters known as a nonterminal alphabet.

Σ is a set of characters disjointed from a nonterminal alphabet N , i.e. $(N \cap \Sigma = \emptyset)$, known as a terminal symbol.

P is a set of productions, each of which must contain a nonterminal followed by an arrow pointing right by any alphabet in a nonterminal and/or terminal $(N \cup \Sigma)$.

S is a start symbol.

1. A finite set of nonterminal symbols is denoted by N . Generally, the Roman capitals A, B, C, \dots, Z usually stand for nonterminal elements. Nonterminal symbols N can be changed using the production rules.
2. A finite set of terminal symbols is denoted by Σ . The small Roman letters a, b, c, \dots, z are used to represent terminal symbols. Σ disjoint from N or $N \cap \Sigma = \emptyset$ where \emptyset is an empty set. The empty set string can be denoted by \emptyset or $\{\}$.
3. A finite set of productions is denoted by P . A production rule component includes a head and a body. The head or the left hand side may be shortened to LHS, which consists of a single nonterminal symbol. The right hand side may be shorted as RHS or the body which can be a terminal, nonterminal or both. A production rule can be represented in the form of $head \rightarrow body$, which means a head or LHS is replaced or written by a body or RHS.

Given a set of production (A, α) where $A \in N$ or A is a member of the nonterminal elements and α is a rule alternative of A and $\alpha \in (N \cup \Sigma)$. The order set of (A, α) are called productions and can be written in the form $A \rightarrow \alpha$.

If A has n production rules, a rule alternative of A can be written as

$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ where $n \geq 0$; a RHS or α_i could be an empty set or ϵ .

The \rightarrow is called a rewrite arrow. $A \rightarrow \alpha$ is called context free grammar rules or rewrite rules. The algorithm is called context free grammar for the reason that all rules only contain one symbol on the LHS and it is rewritten by RHS.

4. A start symbol is denoted by S. A start symbol is a special symbol chosen from non-terminal symbols. S is an element of N or $S \in N$.

Example 3.1 shows a starter symbol (S), a set of terminals (Σ), a set of nonterminals (N) and a set of production rules (P).

Example 3.1 Given $S = "a\ boy\ sits\ in\ the\ car"$ $N = \{S, NP, VP, PP, DT, Vi, Vt, NN, IN\}$ and $\Sigma = \{the, boy, a, car, sits, in\}$

ID	Productions
1	$S \rightarrow NP\ VP$
2	$VP \rightarrow Vi$
3	$VP \rightarrow Vt\ NP$
4	$VP \rightarrow VP\ PP$
5	$NP \rightarrow DT\ NN$
6	$NP \rightarrow NP\ PP$
7	$PP \rightarrow IN\ NP$
8	$Vi \rightarrow sits$
9	$NN \rightarrow boy$
10	$NN \rightarrow car$
11	$DT \rightarrow the$
12	$DT \rightarrow a$
13	$IN \rightarrow in$

Table 4.2 Regular grammar for English personal names.

Note: S = Sentence, NP = Noun Phrase, VP = Verb Phrase, PP = Preposition Phrase, DT = Determiner, Vi = Intransitive Verb, Vt = Transitive Verb, NN = Noun and IN = Preposition

Left-Most Derivations

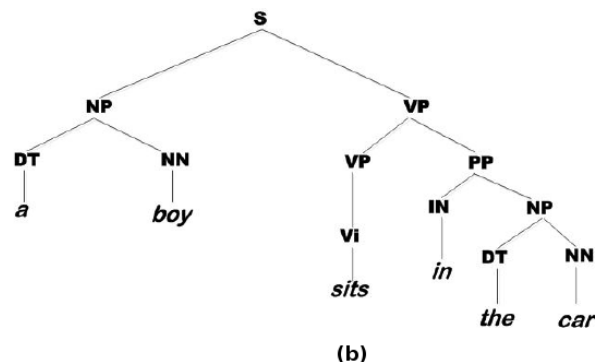
A left-most derivation or top-down parsing is a sequence of parsing production rules or a sequence of generating string $S_1 \dots S_n$ by a grammar or production rules where $S_1 = S$; S is start symbol $S_n \in \Sigma^*$; Σ^* is the free monoid finitely generate by Σ . For example, $\Sigma = \{a, b\}$ then $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$.

Each S_i where $i = \{2,3, \dots, n\}$ is derived from $S_{(i-1)}$ by selecting the left nonterminal in $S_{(i-1)}$ and it is rewritten by α where $S_{(i-1)} \rightarrow \alpha$ is a rule in P. We developed an example 3.1 to describe the details about left-most derivations.

From the sentence $S = "a boy sit in the car"$. We start with the start symbol S in the production rules as S is written by two nonterminals: NP VP using Rule 1. The algorithm goes to the first symbol in the RHS of Rule 1 (it is NP) and replaces NP with Rule 5 ($NP \rightarrow DT NN$) and keeps going like this. Finally, the complete derivation will be finished if every nonterminal symbol is assigned to terminal symbols.

Derivation	Rules Used
S	$S \rightarrow NP VP$
NP VP	$NP \rightarrow DT NN$
DT NN VP	$DT \rightarrow a$
a NN VP	$NN \rightarrow boy$
a boy VP	$VP \rightarrow VP PP$
a boy VP PP	$VP \rightarrow Vi$
a boy Vi PP	$Vi \rightarrow sits$
a boy sits PP	$PP \rightarrow IN NP$
a boy sits IN NP	$IN \rightarrow in$
a boy sits in NP	$NP \rightarrow DT NN$
a boy sits in DT NN	$DT \rightarrow the$
a boy sits in the NN	$NN \rightarrow car$
a boy sits in the car	completely left-most derivations

(a)



(b)

Fig. 4.2 Example of CFG (a) Production rules (P) and (b) Left-most derivations represented as a tree.

CFG has been studied by many researchers in a variety of fields. The studies [38, 73] applied CFG in a natural language processing (NLP) system. Hasan et al. [38] designed context grammar rules for Bangla language and used these grammar rules to develop a Bangla parser. The experimental result showed that a CFG not only support English language but we can also apply a CFG to other NLP, e.g. Bangla language.

Xu et al. [73] introduce an approach for parsing medication sentences based on probabilistic context free grammar. Due to the fact that most patient information is stored in narrative text documents, it cannot be directly accessed through a computer information system. The main drawback in parsing sentences using CFG is an ambiguity; one sentence may provide multiple results in a parsing tree. To solve the ambiguity problem, the Probabilistic Context Free Grammar (PCFG) is used for parsing medical sentences when multiple possible parse tree are produced. The final result is considered by the overall probability of a parse tree. The experimental results showed that the accuracy in PCFG is better than CFG.

Chanda et al. [13] design context free grammar rules to verify three diagrams including the sequence diagram, class diagram and state chart diagram in the design phase of the Object Oriented system. The CFG verification framework consists of syntactic precision

and inter-diagram consistency. The static and dynamic behaviour of a design is formalised and traceability and consistency are included among the diagrams.

Arasu and Kaushik [3] apply a context free grammar for data cleaning. The researchers used the CFG rules and external knowledge base for paring multiple affiliation formats into a uniform representation. The ambiguity parse tree (one affiliation may be generated for more than one parse tree) can be solved using uniform weight scheme score. We extended the study [3] for parsing personal name variations that are extracted from a web page to a standard format. However, we used different techniques for solving the ambiguity parse tree problem. In our work, we used a regular grammar in personal names to find the structure in a personal name representation (the location of the first name, middle name or last name) and is directly matched to the external knowledge base.

4.2.3 String Matching

String matching plays an important role in personal name matching. Various methods have been introduced to deal with duplicate detection. For example, the character-based similarity metrics (e.g. edit distance, Jaro, Smith-Waterman or Q-Grams), token-based similarity metrics (e.g. cosine similarity or Monge and Elkan) and phonetic similarity metrics (e.g. Soundex or New York State Identification and Intelligence System). The character-based similarity metrics have a good performance in catching typographical errors but the effectiveness is lower when the words are rearranged (e.g. *George Bush* versus *Bush, George*) [25]. The token-based similarity metrics aim to fix this problem. The phonetic similarity metrics pay attention to the string-based representation (the words may be represented using different characters but they have similar phonetics). However, the studies [6, 15, 54, 74] found that the best performance for personal name matching was with Jaro-Winkler.

Yancey [74] evaluates Jaro-Winkler metrics and edit distance metrics using a U.S. Census Bureau data set. The results show that Jaro-Winkler metrics perform well in personal name matching. Bilenko et al. [6] compare the performance of the character-based metrics and token-based metrics among various types of data sets. The output shows that the Jaro metrics work well for census data sets. However, the study [6] suggested that we cannot use the same metric for every data set (one metric that is effective in some data sets may perform poorly in another). Peng et al. [54] show that both Jaro and Jaro-Winkler perform well and faster than three string similarity metrics, including Levenshtein, Q-gram and Smith-Waterman in *Electoral Roll* data sets. Finally, the study [15] also found that Jaro and Jaro-Winkler are effective in personal name matching.

Turning now to the experimental result in [18] shows that cosine similarity metrics

spend the smallest amount of time compared with other techniques (e.g. Jaro-Winkler, SoftTFIDF).

In the next section, we describe two string matching techniques, including cosine similarity and Jaro-Winkler. In the two string similarity metrics we want to evaluate which one works well with CFG in personal name similarity matching.

Cosine Similarity

Cosine similarity is normally used in information retrieval for matching between query terms and the documents are stored in a database [49]. This technique is one of the token-based similarity metrics or bags of words. Cosine similarity works well with a high volume of data and the location of words is ignored to detect the similarity between a sequence of words [25] (e.g. *George Bush* vs. *Bush, George*). On the other hand, this metric measure fails to map a pair of words where there are typographical errors. For example, the two words: "*book*" and "*bok*" are absolutely different under the cosine similarity function.

In example 4.1, we want to measure the similarity between a query term P in a document E. Given

$$P(\text{Michael Owen}) = \{\text{Michael, Owen}\}$$

$$E(\text{Michael James Owen}) = \{\text{Michael, James, Owen}\}$$

The similarity between entities E and P is calculated using equation 4.1.

$$\text{sim}(P, E) = \text{Cos}(\theta) = \frac{\vec{V}(P) \cdot \vec{V}(E)}{|\vec{V}(P)| |\vec{V}(E)|} = \frac{\sum_{i=1}^n p_i \cdot e_i}{\sqrt{\sum_{i=1}^n p_i^2} \sqrt{\sum_{i=1}^n e_i^2}} \quad (4.1)$$

Where n is the number of unique tokens in entity E and entity P. The value of each token p_i and e_i is the frequency of each unique token occurring in entities E and P which are shown in Table 4.3.

Table 4.3 Example of cosine similarity measurement.

	michael	james	owen
p_i	1	0	1
e_i	1	1	1

$$\text{sim}(P, E) = \text{Cos}(\theta) = \frac{\vec{V}(P) \cdot \vec{V}(E)}{|\vec{V}(P)| |\vec{V}(E)|} = \frac{1.1 + 0.1 + 1.1}{\sqrt{1^2 + 0^2 + 1^2} \sqrt{1^2 + 1^2 + 1^2}} \simeq 0.82 \quad (4.2)$$

Jaro-Winkler

Jaro-Winkler is character-based string matching to measure the similarity between two strings which is widely used in short string comparison (e.g. first name and last name) [25, 74]. The study [54] shows that the Jaro-Winkler provides a higher rate of accuracy in string similarity matching than Levenshtein, Q-gram and Smith-Waterman and also performs well in typography errors [25].

Given A and B are a set of strings $A = (a_1, a_2, \dots, a_m)$ and $B = (b_1, b_2, \dots, b_n)$. The two characters a_i and b_j are equal if they occurred in the same location or no further than the maximum match distance. The maximum match distance equation is represented below:

$$\text{Maximum match distance} = \frac{\max(|A|, |B|)}{2} - 1 \quad (4.3)$$

The transposition is determined by the number of matching characters that have a different sequence order divided by 2.

The Jaro similarity value for two strings is then given by

$$\text{Jaro}(A, B) = \frac{1}{3} \left(\frac{c}{|A|} + \frac{c}{|B|} + \frac{c-t}{c} \right) \quad (4.4)$$

where c is the number of matching characters and t is the number of transpositions.

The Jaro-Winkler algorithm [60] is represented below:

$$\text{Jaro - Winkler}(A, B) = \text{Jaro}(A, B) + \left(\frac{p(1 - \text{Jaro}(A, B))}{10} \right) \quad (4.5)$$

Where $\text{Jaro}(A, B)$ is the Jaro score and p is the length of the longest common prefix (the maximum number is 4 characters).

For example, given a string $s_1 = \text{Barack}$ and $s_2 = \text{Barak}$ we compute the Jaro-Winkler as follows:

1. $|s_1| = 6$ and $|s_2| = 5$.
2. The maximum match distance = $\max(6, 5) / 2 - 1 = 2$.
3. The number of matching characters = $c = 5$.

4. The sub-sequence of two matching strings is *Barak* and *Barak*. Therefore, there are no transpositions ($t = 0$).
5. $\text{Jaro}(s_1, s_2) = 1/3(5/6 + 5/5 + (5-0)/5) = 0.94$.
6. $\text{Jaro-Winkler}(s_1, s_2) = 0.94 + 0.1 * 4(1-0.94) = 0.964$.

4.3 Personal Name Transformation Modules (PNTM)

In this section we discuss the personal name variation problems and how to handle them using PNTM. One key aspect of personal names that appeared within a web document is name variations or referential ambiguity. We define name variation problems in four categories:

1. Multiple spelling or typographical errors e.g. *Barak and Barack* or *Ian and Iain*.
2. Alternative names such as a nickname, pen name or official name. For example, *43rd President of the United States* and *Dubya Bush* refer to *George W. Bush* or *Lady Gaga* refers to *Stefani Joanne Angelina Germanotta*.
3. Variation forms such as abbreviations (e.g. *JFK* and *John F. Kennedy*), shortened forms (e.g. *Robyn Rihanna Fenty* and *Rihanna*) and initials for given and middle names (e.g. *G.W. Bush*, *George W. Bush* and *George Walker Bush*).
4. Multiple orders such as *George W. Bush* and *Bush, George W.*.

Name variations reduce the efficiency in character-based similarity metrics such as edit distance (e.g. Jaro-Winkler) and token-based similarity metrics (e.g. cosine similarity) for detecting whether two names are similar or not [3]. For example, the similarity value is smaller for *Robyn Rihanna Fenty* and *Rihanna* or completely different between *43rd President of the United States* and *George W. Bush*.

Arasu et al. [1] propose a black-box similarity function for mapping whether *Rob* and *Robert* are equal. However, the black-box similarity function cannot allow us to understand the internal structure of the context [3]. Our method extends Arasu and Kaushik [3] by including alternative names transformation and maps personal name patterns.

PNTM is the module we use to transform multiple forms of English personal names into a standard format. Standard format means a personal name should be uniform representation. Uniform means that the personal name has a single representation for each individual person. For example, multiple name representations of *George W. Bush* (e.g. *Bush*, *Dubya*

Bush, George W. Bush, Jr.) should be transformed into a single representation *George W. Bush*. The most important role for PNTM is to generate a personal name under the transformation rules into a uniform representation. For example, PNTM transforms the name *Bush*, to *George W. Bush, Laura Bush and Billy Green Bush* because these three people use the same alternative name *Bush*. However, the modules do not identify that the name *Bush* refers to *George W. Bush, Laura Bush or Billy Green Bush*.

In addition, our module allows us to understand the grammar in a sequence of personal name tokens. Therefore, each word that appears within a token may be a given name, a nickname or a last name depending on the structure of a personal name pattern. PNTM does not transform every word that appears in the personal name dictionary, but it only transforms the words that are related to the CFG rules. For example, the word *Rose* may be a given name or a middle name but the word *Rose* only transform to *Roxanna* if *Rose* is a given name.

Figure 4.3 describes the PNTM module. The module consists of three main components: CFG rules, predicate and actions. The shaded circles are input and output. The module takes a single personal name and may produce 0-n outputs. The blue shaded areas including personal name patterns, personal name dictionary and prefix & suffix dictionary are a predicate. Finally, the three rectangles inside a big rectangle are the three action steps to transform a personal name into a uniform representation.

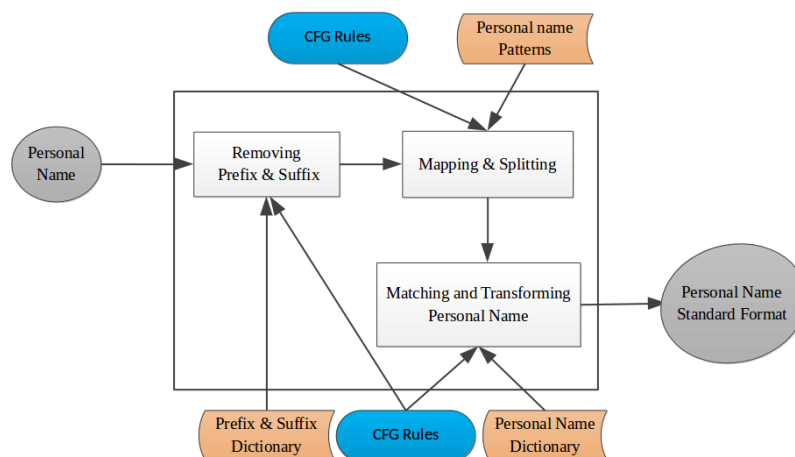


Fig. 4.3 Personal Names Transformation Module.

Given a personal name as input, the PNTM uses three main components to produce a uniform of personal name representation:

1. **CFG rules.** CFG rules or transformation rules are the set of rules that specify how the personal name input can be generated.
2. **Predicate.** Predicate is an external database that consists of a prefix and suffix dictionary, a personal name dictionary and a set of personal name patterns.
3. **Action.** Action is a set of functions we use to produce the output under CFG rules. Action consists of three steps as follows:
 - Removing prefix and suffix use to delete personal name titles (prefixes and suffixes) from personal name input.
 - Mapping and splitting use to map a sequence of personal name strings with a set of standard patterns. Then a sequence of personal name strings splits into given name, middle name, family name or alternative name depending on its location.
 - Matching and transforming personal name. Each word in a sequence of personal names maps directly over the personal name dictionary and then produces a personal name that has a standard format.

4.3.1 Context Free Grammar Rules (CFG Rules)

The transformation rule is a quadruple of $G = (N, \Sigma, P, S)$ Where:

1. N is a set of nonterminal symbols that we use for referencing the collection of given names, middle names or family names.
2. Σ is a set of terminal symbols that refers to a set of names in the personal names dictionary disjointed from nonterminal alphabet N ($N \cap \Sigma = \emptyset$).
3. P is a set of productions or CFG rules that we use to generate a single personal name pattern.
4. S is a starter symbol or an input personal name.

Nonterminal symbol (N) is literal symbols which can be changed using the rules. We use angular brackets to represent nonterminal symbols e.g. $\langle GN \rangle$. An example of nonterminal symbols is shown in Table 4.4. The personal names can be captured in seven nonterminal groups, as shown in the following:

- $\langle Prefix \rangle$ is a symbol referring to a collection of prefixes.
- $\langle GN \rangle$ is a symbol referring to a collection of given names.
- $\langle FN \rangle$ is a symbol referring to a collection of family names.
- $\langle Letters \rangle$ is a symbol referring to a collection of initial names.
- $\langle AN \rangle$ is a symbol referring to a collection of alternative names.
- $\langle NN \rangle$ is a symbol referring to a collection of nicknames.
- $\langle Suffix \rangle$ is a symbol referring to a collection of suffixes.
- $\langle Symbols \rangle$ is a symbol referring to a collection of symbols that may be used in personal name collections (e.g. *O'Connell* or *Hans-Porter*).

Table 4.4 A generative grammar to capture personal name variations.

Nonterminal symbols	Terminal symbols
$\langle Prefix \rangle$	"Miss", "Ms", "Mr", "Mrs", "Prince"
$\langle GN \rangle$	"Aaliyah", "Aamna"
$\langle FN \rangle$	"Aaberg", "Aadland", "Aafjes"
$\langle Letters \rangle$	"A", "B", "C"
$\langle AN \rangle$	"10th President of the United States", "11th Lord Kinnaird"
$\langle NN \rangle$	"Ab", "Abbie", "Abby"
$\langle Suffix \rangle$	"Jr", "Sr", "III"
$\langle Symbols \rangle$	".", "-", "'", "

A major criticism in PNTM is the CFG rules or production rules (P). The main purpose of the CFG rules is to generate personal names so, they are not used to recognise personal names.

A grammar rule (R) is a finite production rules. Each rule consists of a head or left hand side (LHS) and a body or right hand side(RHS), similar to a standard CFG rule. The head is a single nonterminal and a body contains a set of nonterminals or terminals and/or variables. A CFG rule is used to define that a nonterminal symbol on the left hand side (head) can be rewritten by other nonterminal or terminal symbols on the right hand side (body). For example, $A \rightarrow \alpha$ is a CFG rule that A can be rewritten by α .

The uppercase letters are used to define variable symbols. The nonterminal symbols and variable symbols we use are described below:

1. Nonterminal symbols include $\langle NAME \rangle$, $\langle PS \rangle$, $\langle PN \rangle$, $\langle GN \rangle$, $\langle FN \rangle$, $\langle Letters \rangle$, $\langle AN \rangle$ and $\langle Letters \rangle$. The start symbol is $\langle NAME \rangle$.
2. Variable symbols include A, G, N, M, F, P, L and S where
 - A is a set of alternative names terminals.
 - G is a set of given name terminals.
 - N is a set of nickname terminals.
 - M is a middle name
 - F is a set of family name terminals
 - P is a set of prefix terminals
 - L is a set of English alphabets
 - S is a set of prefix terminals

Table 4.5 Personal Name Transformation Rules based on CFG.

ID	CFG Rules
R1	$\langle NAME \rangle \rightarrow \langle PS \rangle_1 \langle NAME \rangle_2$
R2	$\langle NAME \rangle \rightarrow \langle GN \rangle_1 \langle MN \rangle_2 \langle FN \rangle_3$
R3	$\langle NAME \rangle \rightarrow \langle GN \rangle_1 \langle FN \rangle_2$
R4	$\langle NAME \rangle \rightarrow \langle FN \rangle_1 \text{''} \text{''} \langle GN \rangle_2 \langle MN \rangle_3$
R5	$\langle NAME \rangle \rightarrow \langle FN \rangle_1 \text{''} \text{''} \langle GN \rangle_2$
R6	$\langle NAME \rangle \rightarrow \langle AN \rangle_1$
R7	$\langle GN \rangle \rightarrow G$
R8	$\langle GN \rangle \rightarrow N$
R9	$\langle MN \rangle \rightarrow M$
R10	$\langle GN \rangle \rightarrow \langle Letters \rangle_1$
R11	$\langle MN \rangle \rightarrow \langle Letters \rangle_1$
R12	$\langle FN \rangle \rightarrow F$
R13	$\langle AN \rangle \rightarrow A$
R14	$\langle PS \rangle \rightarrow P$
R15	$\langle PS \rangle \rightarrow S$
R16	$\langle Letters \rangle \rightarrow L$

The CFG rules is based on Arasu and Kaushik [3]. The sixteen rules are introduced to transform referential ambiguity names into a single format. These rules cover most possible names that are usually used to refer to a person (e.g. full name, nickname, short name). Table 4.5 shows our CFG rules.

1. R1 is used to separate a title from a personal name. For example, given an input *George W. Bush, Jr*, R1 produces two outputs including $\langle PS \rangle = Jr$ and $\langle PN \rangle = George W. Bush$.
2. R2-R6 are used to define the location of $\langle GN \rangle$, $\langle MN \rangle$, $\langle FN \rangle$ and/or $\langle AN \rangle$ in the sequence of personal name tokens and use a space between a token for segmentation. For example, given an input *George W. Bush*, R2 produces three outputs including $\langle GN \rangle = George$, $\langle MN \rangle = W.$ and $\langle FN \rangle = Bush$.
3. R7 is used to transform $\langle GN \rangle$ into the given name that may be matched with a given name in the personal name dictionary. For example, given an input *George*, R7 produces an output $GN = George$.
4. R8 is used to transform $\langle GN \rangle$ where GN is a nickname to a given name that may be matched with a nickname in the personal name dictionary. For example, given an input *Bill*, R8 produces two given names including $GN = William$ and *Willis*.
5. R9 is used to transform $\langle MN \rangle$ into the middle name. We do not match the variable M over our personal name dictionary. The output in this rule is returned to the original token.
6. R10-R11 are used to evaluate that $\langle GN \rangle$ or $\langle MN \rangle$ is an initial letter.
7. R12 is used to transform $\langle FN \rangle$ into the family name that may be matched with a family name in the personal name dictionary. For example, given an input *Bush*, R12 produces an output $\langle FN \rangle = Bush$.
8. R13 is used to transform $\langle AN \rangle$ where $\langle AN \rangle$ is an alternative name into the personal name that may be matched with an alternative name in the personal name dictionary. For example, given an input *43rd President of the United States*, R13 produces an output $\langle AN \rangle = George W. Bush$.
9. R14 is used to remove a prefix that will be matched over the prefixes in our personal name dictionary.

10. R15 is used to remove a suffix that will be matched over the suffixes in our personal name dictionary.
11. R16 is used to transform *Letters* into an initial letter. We do not match this token to our personal name dictionary so the original token is returned.

4.3.2 Predicate

A predicate (P) is a main component in a CFG rule. The variable in an augmented rule $\langle R, P, A \rangle$ is constrained to occur at least once in a personal name dictionary P [3]. However, a token of a personal name can be absent from the dictionary. Hence, in our modules, it may be returned NIL if a token does not match any names in the dictionary.

We now introduce part of predicate (P) or our personal name dictionary. Our personal name dictionary is a collection of titles, nicknames, alternative names, given names and family names. The sources and methods we used to construct the dictionary are described in the subsection below.

Given Names and Family Names

Given names and family names are derived from two sources: The US census [9] and The YAGO knowledge base [40]. The US census website provides lists of given names and family names ("*Census1990*" and "*Census2000*"). The data from the US census is very clean and the given names and family names are distinguished. The second source we use to generate given names and family names is the YAGO knowledge base. YAGO provides full personal names; it does not separate between a given name and a family name. We use rules R1-R5, R7 and R14-R15 given in Table 4.5 to extract given names and family names. As a result, we extract 19,295 given names and 182,661 family names. Table 4.6 shows a part of given names and family names from the dictionary listed in alphabetical order.

Table 4.6 An excerpt from given name and family name dictionary.

Dictionary	List of names
Given name	Aaliyah, Aaron, Aaryn, Aasif, Abaham, ...
Family name	Aabergh, Aaby, Aadland, Aafedt, Aagaard, ...

Alternative Names

Alternative names are derived from the YAGO knowledge base. The alternative name is a name, combined with letters and numbers, a name that consists of only one word (excluding a prefix and suffix) or a name that contains more than four sequences of words (excluding a prefix and suffix). We provide a link between an alternative name and its real name. We use rules R1, R8 and R14-R15 given in Table 4.5 to extract alternative names. Therefore, we extracted 10,122 alternative names from YAGO. Figure 4.4 shows an example of an alternative name dictionary listed in alphabetical order.

id	AliasName	ENid
1	10th President Of The United States	37871
2	11th Lord Kinnaid	5731
3	12th President Of The United States	75324
4	13th President Of The United States	51834
5	14th Earl Of Home	1612
6	14th President Of The United States	23514
7	15th President Of The United States	32185
8	16th President Of The United States	218
9	17th President Of The United States	3745
10	18th President Of The United States	72047

Fig. 4.4 Example of alternative names in a personal name dictionary.

Nicknames

A nickname is an informal name that is used to refer to a person. We always find personal nicknames in a web document. For example, *Bill Clinton* or *Bill Gates* are used for the personal names: *William Jefferson Clinton* and *William Henry Gates*.

Table 4.7 represents common nicknames for people in Anglo-Saxon countries; the countries which use English language as the official language.

Table 4.7 Example of traditional English nicknames

Male Names		Female Names	
Names	Nicknames	Names	Nicknames
Aaron	Erin, Iron, Ron, Ronnie	Amanda	Manda, Mandy
Benjamin	Ben, Bennie, Benjy, Jamie	Barbara	Bab, Babs, Barby, Bobbie
David	Dave, Davey, Day	Dorothy	Dolly, Dot, Dortha, Dotty
Edward	Ed, Ned, Ted, Teddy, Eddie	Emily	Emmy, Millie, Emma, Em

This thesis collected the standard nicknames from the three websites below:

1. <http://www.tngenweb.org/franklin/frannick.htm> [29] has a lot of nicknames from traditional English names.
2. <http://www.censusdiggins.com/nicknames.htm> [12] provides a list of the most common nicknames from A-Z.
3. <https://github.com/carltonnorthern/nickname-and-diminutive-names-lookup> [58] provides a nicknames look-up system. The system was created by Old Dominion University - Web Science and Digital Libraries Research Group. They provide a CSV file that contains US given names and their associated nicknames.

Given a set of nicknames NN then $n \in NN$ is one nickname. Let GN be a set of given names. Given name $g \in GN$ may be an instance of one or more nicknames n , written as $GN(n)$. For example, $GN(Abbie) = \{Abigail, Abner, Absalom\}$. Accordingly, we extract 1,454 surface forms of nickname. A nickname surface form is a nickname dictionary that contains two components: a nickname and a set of its reference given names. Figure 4.5 shows an example of a nickname dictionary listed in alphabetical order.

id	Nickname	GivenName
1	A.b.	Abiah,Abijah
2	Ab	Abel,Abner,Abiah,Abijah,Abiel,Abigail,Absalom
3	Abbie	Abigail,Abner,Absalom
4	Abby	Abigail,Abner
5	Abe	Abraham,Abram,Abel
6	Abertina	Alberta
7	Abram	Abraham
8	Ad	Adam,Adelbert,Adolph,Adolphus,Addy

Fig. 4.5 Example of nicknames in a personal name dictionary.

Prefix and Suffix

The personal name titles (prefix and suffix) are derived from Grace Y.W. Tse [67] who studied "*The grammatical behaviour of personal names in present-day English*". Table 4.8 shows an example of our prefixes and suffixes in our dictionary listed in alphabetical order.

Table 4.8 Example of prefix and suffix

Prefix	Suffix
Baroness, Capt, Cdr, Chief, Col, Count, C.P.D., Dame, Det Chief Insp, Det Insp, Dr, Earl, Emperor	A.B, B.A., B.S., II, III, Jr., Sr.

Personal Name Pattern

We use function *preg_match* in PHP to map a personal name structure over a personal name standard pattern. For example, assign the following PHP function "*CheckPattern(\$name)*" to a personal name which has three sequences of words arranged from a given name, a middle name and a family name (e.g. *George Walker Bush*). Each word has at least two letters and may include symbols (". ", " - " and " ' ") or numbers in the word (e.g. *O'Neill*).

```
function CheckPattern($name){  
  
$pattern = "/^[a-z0-9\ '-]{2,}+( [a-z0-9\ '-]{2,}+)(  
    [a-z0-9\ '-]{2,}+S)"/;  
preg_match($pattern, $name, $match);  
return($match);  
  
}
```

4.3.3 Action

Action is the three step function we use to transform a personal name as an input into personal name standard form as an output. The three steps include removing the prefix and suffix, mapping and splitting a sequence of words and matching and transforming into a uniform representation. The action module takes CFG rules, predicates (prefix & suffix dictionary, personal name dictionary and personal name patterns) to transform an input (personal name) into a uniform format. The action takes a single name as an input and produces 0-n value outputs depending on the rules and predicate that are used in derivation. Table 4.9 describes how we process an input using the CFG rule and predicates.

Given a personal name p , action parses the sequence of words p using the CFG rules that described in Table 4.9. After that the action functions analyse along the sequence of words p to process the standard name. An action has multiple ways to process the output values. Therefore, we developed **Example 4.1.** to summarise the correlation among action, CFG rules and predicate.

1. CFG rules: R1, R14 and R15 illustrate how the prefix and suffix have been removed.
2. CFG rules: R2-R6 are used to map a variety of personal name patterns over the personal name standard patterns and to identify the position of a given name, a middle name, a family name and/or an alternative name in the sequence of words.
3. R7, R8 and R10 describe how a given name can be generated. R7 is used to map a given name over a given name dictionary and generates a matching given name. R8 is

similar to R7 but it handles the case when a given name is a nickname. R10 handles the case when a given name is an initial.

4. CFG rules: R9 describes how a middle name can be generated.
5. CFG rules: R10, R11 and R16 describe how a given name and a middle name can be generated when they are an initial letter.
6. CFG rules: R12 describes how a family name can be generated over a family name dictionary.
7. CFG rules: R13 describes how an alternative name can be generated over an alternative name dictionary.

Table 4.9 The personal name transformation description.

ID	Rules	Predicate	Action
R1	$\langle \text{NAME} \rangle \rightarrow \langle \text{PS} \rangle_1 \langle \text{NAME} \rangle_2$	Prefix and Suffix dictionary	PS = 1.value; Name = 2.value
R2	$\langle \text{NAME} \rangle \rightarrow \langle \text{GN} \rangle_1 \langle \text{MN} \rangle_2 \langle \text{FN} \rangle_3$	Personal Name Pattern	GN = 1.value; MN = 2.value; FN = 3.value
R3	$\langle \text{NAME} \rangle \rightarrow \langle \text{GN} \rangle_1 \langle \text{FN} \rangle_2$	Personal Name Pattern	GN = 1.value; FN = 2.value
R4	$\langle \text{NAME} \rangle \rightarrow \langle \text{FN} \rangle_1 \text{''} \langle \text{GN} \rangle_2 \langle \text{MN} \rangle_3$	Personal Name Pattern	GN = 2.value; MN = 3.value; FN = 1.value
R5	$\langle \text{NAME} \rangle \rightarrow \langle \text{FN} \rangle_1 \text{''} \langle \text{GN} \rangle_2$	Personal Name Pattern	GN = 2.value; FN = 1.value
R6	$\langle \text{NAME} \rangle \rightarrow \langle \text{AN} \rangle_1$	Personal Name Pattern	AN = 1.value
R7	$\langle \text{GN} \rangle \rightarrow \text{G}$	GivenNames (I, G)	value = G
R8	$\langle \text{GN} \rangle \rightarrow \text{N}$	NickNames (I, N, G)	value = G
R9	$\langle \text{MN} \rangle \rightarrow \text{M}$		value = M
R10	$\langle \text{GN} \rangle \rightarrow \langle \text{Letters} \rangle_1$		Letters = 1.value
R11	$\langle \text{MN} \rangle \rightarrow \langle \text{Letters} \rangle_1$		Letters = 1.value
R12	$\langle \text{FN} \rangle \rightarrow \text{F}$	FamilyNames (I, F)	value = F
R13	$\langle \text{AN} \rangle \rightarrow \text{A}$	AlternativeNames (I, A)	value = Personal name
R14	$\langle \text{PS} \rangle \rightarrow \text{P}$	Prefix (I, P)	
R15	$\langle \text{PS} \rangle \rightarrow \text{S}$	Suffix (I, S)	
R16	$\langle \text{Letters} \rangle \rightarrow \text{L}$		value = L

^a A = Alternative name, F = Family name, G = Given name, I = identification number, L = Letter, M = Middle name, N = Nickname, P = Prefix and S = Suffix.

The action functions of PNTM are as follows:

1. Removing prefix and suffix. Firstly, we remove the prefix and suffix from a personal name. A set of rules including R1, R14 and/or R15 are used over the prefix and suffix dictionary to detect prefix and suffix words. Next, we developed an Example 4.1 to explain how we removed the prefix and suffix from a sequence of personal name strings.

Example 4.1. We introduce CFG rules to generate a personal name that contains a prefix and/or suffix. The CFG rules and the steps of transformation are shown in Table 4.10. The transformation is started from starter symbol $\langle NAME \rangle$ and rewritten with the nonterminals or terminals on the LHS until every symbol is transformed into terminals. The sequence of CFG rules: R1, R15, R2, R7, R11, R16 and R12 are used to transform *George W. Bush, Jr* into *George W. Bush*, using the derivation steps described below:

- (a) R1: Segmented name into personal name and title. The nonterminal $\langle NAME \rangle$ is written by nonterminal $\langle PS \rangle$ and $\langle PN \rangle$
- (b) R15: Given the value *Jr* to nonterminal $\langle PS \rangle$ and string *Jr* is removed.
- (c) R2: Segmented personal name into three tokens: $\langle GN \rangle$, $\langle MN \rangle$ and $\langle FN \rangle$.
- (d) R7: Generated the value *George* to nonterminal $\langle GN \rangle$ by matching over a list of given names in the personal name dictionary.
- (e) R11: Written nonterminal $\langle MN \rangle$ by nonterminal $\langle Letters \rangle$.
- (f) R16: Given the value *W.* to nonterminal $\langle Letters \rangle$.
- (g) R12: Given the value *Bush* to nonterminal $\langle FN \rangle$ by matching over a list of family names in the personal names dictionary.
- (h) End of left-most derivation and an action produces one output: *George W. Bush*.

Table 4.10 Expanded framework for processing the personal name *George W. Bush, Jr.*

ID	Derivation	Rules Used
1	$\langle NAME \rangle$	R1: $\langle NAME \rangle \rightarrow \langle PS \rangle_1 \langle PN \rangle_2$
2	$\langle PS \rangle \langle PN \rangle$	R15: $\langle PS \rangle \rightarrow \text{Jr}$
3	$\langle PN \rangle$	R2: $\langle PN \rangle \rightarrow \langle GN \rangle_1 \langle MN \rangle_2 \langle FN \rangle_3$
4	$\langle GN \rangle \langle MN \rangle \langle FN \rangle$	R7: $\langle GN \rangle \rightarrow \text{George}$
5	George $\langle MN \rangle \langle FN \rangle$	R11: $\langle MN \rangle \rightarrow \langle Letters \rangle$
6	George $\langle Letters \rangle \langle FN \rangle$	R16: $\langle Letters \rangle \rightarrow \text{W.}$
7	George W. $\langle FN \rangle$	R11: $\langle FN \rangle \rightarrow \text{Bush}$
8	George W. Bush	Completely left-most derivation

In this case, R1 and R15 are used to remove a suffix from a sequence of words: *George W. Bush, Jr.* R1 is used to distinguish between a personal name and their prefix and/or suffix. A sequence of personal name strings without a prefix and suffix goes to the second step of mapping and splitting.

- Mapping and splitting. We use personal name patterns and CFG rules: R2 - R6 to detect a sequence of personal strings over personal name standard patterns. The sequence of personal name strings split are based on the CFG rules. In [3], the output records have been produced by taking a personal name e.g. *Smith, Andy J.* and then a set of strings is indexed using Aho-Corasick automaton for string matching. The uniform weighting scheme is used to evaluate the output records by calculating the total weighting score in every CFG rule that is used to produce each output record. The lower weights output record indicates higher confidence. To define weight, they assign a non-negative real number weight to each CFG rule. Furthermore, they construct the output records using a bottom-up fashion to evaluate actions along the nodes of a parse tree. Our action process is different from [3] because we detect a sequence of personal name strings over standard patterns in our collections to find a specific pattern in each input personal name. After that, each string splits and identifies what they are: given name, middle name, family name or alternative name. Each string matched directly over the personal name dictionary (e.g. given name maps to a given name dictionary). In example 4.1, CFG R2 is used to segment the personal name *George W. Bush*. We specify that *George* is a given name, *W.* is a middle name and *Bush* is a family name.

3. Matching and transforming a personal name. We use CFG rules: R7 - R13 and R16 to transform a sequence of personal strings when the strings are identified. As shown in example 4.1, we use R7 and a given name dictionary to transform the string *George* into given *George*. The Aho-Corasick similarity metric continues to be used in our work to determine the similarity between the input string and the personal dictionary (given name, family name, nickname and alternative name).

Finally, we develop two examples to explain how different named patterns are transformed into a uniform representation through the sequence of CFG rules.

Example 4.2 We introduce a sequence of CFG rules to generate a personal name when a person uses a nickname instead of a given name and the sequence of personal name tokens begins with a family name with a comma and ends with a given name. The CFG rules we use and left-most derivation are shown in Table 4.11. The transformation is started from starter symbol $\langle PN \rangle$ and rewritten with the nonterminals or terminals on the LHS until every symbol is transformed into terminals. The sequence of CFG rules: R5, R8 and R12 is used to transform personal name '*Gates, Bill*' into *William Gates and Willis Gates*, which is a derivation as described below:

1. R5: Re-arranging the sequence of personal tokens by giving the first value to $\langle FN \rangle$ and the second value to $\langle GN \rangle$.
2. R8: Generating the value *William and Willis* to nonterminal $\langle GN \rangle$ by matching over a list of nicknames in the personal name dictionary.
3. R12: Generating the value *Gates* to nonterminal $\langle FN \rangle$ by matching over a list of family names in the personal dictionary.
4. End of left-most derivation and an action produces two outputs: *William Gates* and *Willis Gates*.

Table 4.11 Expanded framework for processing nickname *Gates, Bill*.

ID	Derivation	Rules Used
1	$\langle PN \rangle$	R5: $\langle PN \rangle \rightarrow \langle FN \rangle_1 ", " \langle GN \rangle_2$
2	$\langle GN \rangle \langle FN \rangle$	R8: $\langle GN \rangle \rightarrow \text{Bill}$
3	William and Willis $\langle FN \rangle$	R12: $\langle FN \rangle \rightarrow \text{Gates}$
4	William Gates and Willis Gates	Completely left-most derivation

Example 4.3 We introduce a sequence of CFG rules to generate a personal name when person use an alternative name instead a real name. The CFG rules we use and left-most derivation are shown in Table 4.12. The transformation is started from starter symbol $\langle NAME \rangle$ and rewritten with the nonterminals or terminals on the LHS until every symbol is transformed into terminals. The sequence of CFG rules: R6 and R13 is used to transform alternative name *Bush* into a set of real names *Billy Green Bush*, *George W. Bush* and *Laura Bush*, which is a derivation as described below:

1. R6: Written nonterminal $\langle NAME \rangle$ by nonterminal $\langle AN \rangle$
2. R13: Generated the value *Bush* into nonterminal $\langle AN \rangle$ by matching over a list of alternative names in the personal name dictionary.
3. End of left-most derivation and an action produces three outputs: *Billy Green Bush*, *George W. Bush* and *Laura Bush*.

Table 4.12 Expanded framework for processing alternative name: *Bush*.

ID	Derivation	Rules Used
1	$\langle NAME \rangle$	R6: $\langle NAME \rangle \rightarrow \langle AN \rangle_1$
2	$\langle AN \rangle$	R13: $\langle AN \rangle \rightarrow \text{Bush}$
3	Billy Green Bush, George W. Bush and Laura Bush	Completely left-most derivation

So far, PNTM which is presented in Figure 4.3, has the ability to transform a personal name which has a referential ambiguity problem into a standard uniform form in three reasons: Firstly, PNTM can produce multiple outputs when a given name is a nickname as represented in example 4.2. Secondly, the framework provides a sequence of CFG rules to transform a personal name that starts with a family name as represented in example 4.2. Finally, it can produce the real name when a document uses an alternative name to mention a person as described in example 4.3.

4.4 Experimental Results and Discussion

In this section, we present the effectiveness of PNTM (personal name transformation modules) for the real world personal name data sets. We evaluate PNTM in Ubuntu 15.04 64-bit operating System (Intel® Core™ i5-2450M CPU @ 2.50GHz × 4 and 8.0 GB RAM). Our

prototype is built in PHP Version 5.6.4-4 ubuntu 6.4, Apache 2.4.10 (Ubuntu) and MySQL 5.6.27. The assessment includes two parts:

1. Evaluating PNTM is real world data sets. Precision is used to measure the performance of PNTM in generating a set of uniform personal name patterns.
2. Comparing the performance of two text similarity matching metrics: Jaro-Winkler and Cosine similarity. Either Jaro-Winkler or Cosine similarity is suitable for PNTM.

4.4.1 PNTM Assessment

We collect personal name data from two knowledge bases: Freebase [30] and Factforge [26]. The two knowledge bases provide multiple mention forms of personal names to evaluate the transformation precision in PNTM. The total number of personal names used in our experiment is 5,243. The 5,243 personal names were collected by us randomly. We wanted to cover a variety of personal name patterns. These personal names will assist in evaluating the accuracy of PNTM. Table 4.13 shows excerpt of the data sets.

Table 4.13 Example of datasets.

ID	Names
1	Aaliyah
2	Alan Brodrick, 2nd Viscount Midleton
3	Thomas R. Marshall
4	Alma De Bretteville Spreckels
5	Pat Benatar
6	The Big Bopper
7	Willis Stephens Jr.
8	Angela Smith, Baroness Smith Of Basildon
9	Allison, Herbert M., Jr.
10	Beecroft, Robert S.

Experimental Results and Discussion

PNTM generated 13,832 output personal names from 5,243 input personal names. The 18 input personal names or 0.13% cannot be generated (NIL values). The overall quality

of PNTM is extremely high and the modules return a NIL value of less than 1%. The 18 NIL values are alternative names. This means that our personal name dictionary does not cover alternative names. We selected 100 sample output names and evaluated them manually. We found that three input personal names are incomplete transformations. One personal name is imperfect in a given name and two personal names are incomplete in a family name. However, when we considered this in detail, we found that these given names and two family names are not English personal names (given name: *Mahatma* and family names: *Kuraishi*, *Kirilenko*). The first part of the letters in each token are matched with our personal name dictionary and the incomplete name is returned. For example, we have the given name *Ma* in our dictionary but *Mahatma* is absent when parsing the given name *Mahatma*, so the system will return *Ma* as an output.

The results are similar to Arasu and Kaushik [3] that has an accuracy of around 97%. We can prove that PNTM has an intuitive capability of transforming personal name variations into a uniform representation.

4.4.2 Text Similarity Metrics Comparison

In this section, we aim to evaluate whether Jaro-Winkler or Cosine similarity provides a good performance in personal name matching under PNTM. Given P is a sequence of personal names (real name e.g. *David Beckham*) and a set of $\{a_1, a_2, \dots, a_n\}$ as his aliases (e.g. {Becks, DB7, David Becham, ..., David Joseph Beckham}). The two personal names P and the alias a_1 are duplicated if their $\text{Sim}(P, a_1)$ is larger than or equal to 0.8. We downloaded 168 aliases of 25 people from Freebase. Freebase uses the property *Also known as* to map a person to their aliases. For example, the aliases for *David Beckham* are

$$\{\textit{Becks}, \textit{DB7}, \dots, \textit{David Joseph Beckham}\}$$

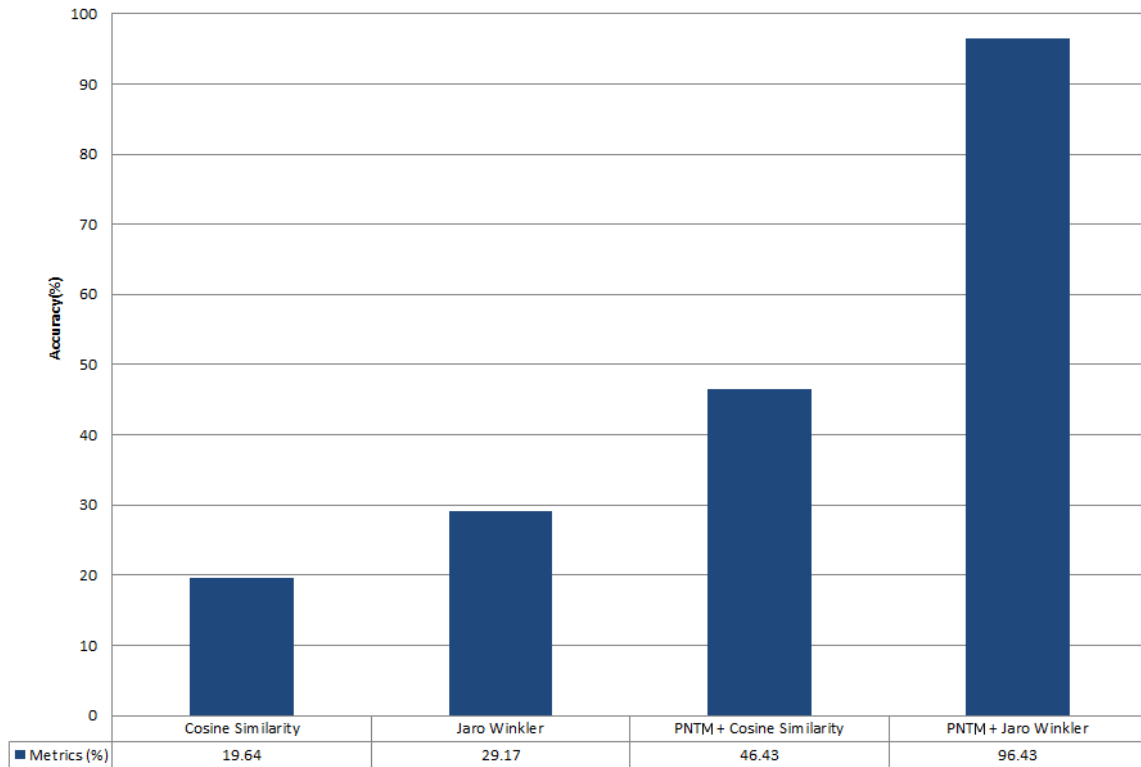


Fig. 4.6 Accuracy comparison of personal name matching among different methods

Figure 4.6 shows percentage of personal name matching performance when using four different similarity metrics (Cosine similarity, Jaro-Winkler, PNTM and Cosine similarity, PNTM and Jaro-Winkler). For an accurate comparison, we calculate the personal name data records that are correctly detected by each metric in comparison. From 168 aliases of 25 people, it can be seen that Cosine similarity returns the lowest accuracy (19.64%) when detecting duplication between a personal name and its alias. The PNTM with Jaro-Winkler provides the highest accuracy (96.43%) but a smaller number of improvements in PNTM with Cosine similarity (46.43%).

Cosine similarity returns the lowest performance because it is a token-based similarity metric [25]. Cosine similarity separates each personal name into words and a pair of words is equal if two words have the same characters in every arrangement (e.g. *Beckham = Beckham* and *Beckham ≠ Beckam*). The Cosine similarity metric only deals with the problem of the rearrangement of words (e.g. *David Beckham versus Beckham, David are equal*). However, it cannot solve the problem of multiple spelling or typographical errors that are usually found in personal name representation.

Jaro-Winkler is a character-based similarity metric [25] that is designed to fill the gap in typographical errors or multiple spelling (e.g. *David Beckham versus David Beckam*

are equal). This is why Jaro-Winkler returns a higher correctly detected personal name duplication than Cosine similarity. However, Jaro-Winkler cannot handle the two main problems in personal name matching, including the rearrangement of words (e.g. *David Beckham* versus *Beckham, David* are equal) and using different words (e.g. nickname, pen name or alias: *Becks* or *DB7* for *David Beckham*).

PNTM with Cosine similarity returns a smaller number of improvement in personal name matching because PNTM handles two main problems: rearrangement of words and using different words to refer to the same person (e.g. *DB7* versus *David Beckham*). As a result, PNTM with Cosine similarity can only provide one enhancement in personal name matching using different words to refer to the same person because the rearrangement of words can be solved with the Cosine similarity metric.

PNTM with Jaro-Winkler returns the highest rate of accuracy in personal name matching because it can solve three main problems: the rearrangement of words, using different words to refer to the same person (PNTM) and typographical errors or multiple spellings (Jaro-Winkler).

4.5 Related Work

One of the greatest challenges in personal name matching is name variations or referential ambiguity. The data standardisation approach transforms multiple forms of names that are used to refer to the same person into a consistent and a uniform representation. The data standardisation makes data matching easier [55].

Raman and Hellerstein [56] introduce the Potter's Wheel, an interactive data cleaning system that allows the user to interact with a system step by step through GUI. The Potter's Wheel provides a good performance in transforming multiple personal name formats into a uniform representation. However, the system does not support personal name variations, e.g. nicknames, aliases and alternative names. Furthermore, Christen et al. [16] argued that the Potter's Wheel is based on domain-specific rules which require a user to configure step-by-step before transformation [16].

Christen et al. [16] proposed the probabilistic hidden Markov modes (HMMs) for personal name and address cleaning and standardisation. The model contains three steps including data cleaning, data tagging and data segmenting. Firstly, the input strings are cleaned by converting to the lower case, removing unwanted strings or replacing them using the corresponding replacement strings. Secondly, the cleaning inputs are split and tagged using look-up tables. Finally, HMMs is used to segment these tagged to produce the appropriate output fields. The accuracy measurement of names using HMMs decreases when

the case of middle names arises. Additionally, this method only solve the different order in personal name arrangement. There is only one problem in name variations.

Arasu et al. [1] propose a context-free transformation framework based on transformation rules. The framework transforms string variations such as synonyms or abbreviations and treats them through a black-box similarity function (e.g. *Bob* \rightarrow *Robert*). However, Arasu and Kaushik [3] concludes two limitations of the black-box similarity function:

1. It does not allow us to understand the internal structure of the context. Therefore, we may miss the rich contextual information necessary to determine if an alias or synonym is meaningful. For example, it is meaningful to transform a nickname into a given name (e.g. *Bob Black* \rightarrow *Robert Black*) but not within a middle name (e.g. *William Bob Black* \rightarrow *William Robert Black*).
2. The similarity of two strings can only increase by adding transformations. This approach is therefore not useful in handling the referential ambiguity (e.g. *The Governor* and *Arnold Schwarzenegger* when two strings are quite different).

Arasu [3] introduced a grammar-based entity representation in pre-processing data using standard Context Free Grammar rules. This research used a sequence of grammar rules for parsing personal names and their affiliations which are represented in different forms in a unique pattern. The researcher claims two points about textual similarity function problems:

1. Two representations of the same entity could be highly dissimilar textually (e.g. George Bush and Dubya Bush).
2. Two representations that are textually very similar could correspond to different entities (e.g. Chris Martin (singer-songwriter) and Chris Martin (football player)).

The framework is based on CFG rules and external domain knowledge for parsing affiliations and data normalisation.

However, a personal name which is mentioned in the web document has various representation than a personal name in a database. The collection of the external knowledge base about a personal first name, last name and nickname that is used to generate personal name normalisation does not cover personal name variations when they occur within a web document. In addition, a limitation uniform weight scheme is used to evaluate the ambiguity of the parse tree (each name component may be mapped to more types of personal name). However, the weight value is useless when a token can qualify both a first name and a last name. For example, the token *Ferguson* can be used as both a first name and a last name.

Chen et al. [14] introduce a probabilistic regular grammar of personal names to represent the personal name structure. This method enhances the precision for extracting Chinese personal names from a web document. Furthermore, the approach reduces the error rate in personal name recognition. Shen et al. [61] combine personal name grammar to solve the problem of personal name classification in web queries. This method provides a simple way to trade-off the precision and recall. The two previous studies gave us the idea to combine regular grammar in personal names with our CFG rules to fix the problem of ambiguity parsing. Therefore, we can remove the uniform weight scheme from our framework because we know which token is used (given name, middle name or family name) and directly maps to a specific name type.

4.6 Conclusion

This chapter introduces Personal Name Transformation Modules (PNTM) which is used to solve the problem of personal name variations. Referential ambiguity or personal name variations is a result of:

1. Multiple spelling or typographical errors(e.g. *Beckham* versus *Beckam*).
2. Rearrangement of words (e.g. *David Beckham* versus *Beckham, David*).
3. Using different words such as an alias, nickname, an alternative name (e.g. *DB7* versus *David Beckham*) to refer to the same person.

PNTM consists of three components: CFG rules, predicate and action that are used to transform personal name variations into a uniform format. CFG rules are a set of productions we use to identify how to generate output from different forms of personal name input. Predicate is an external database that includes a given name dictionary, a family name dictionary, an alternative name dictionary and a prefix and suffix dictionary. Action is a set of functions we use to process the output under CFG rules and predicate.

The experimental results show that PNTM with Jaro-Winkler can handle the results of the three main problems in referential ambiguity: multiple spelling or typographical errors, rearrangement of words and using different words to refer to the same person.

Chapter 5

Personal Name Entity Linking

This chapter introduces the Personal Name Entity Linking Framework (PNELF) that we used to identify mentioned names within a web page to a real world person. Firstly, Section 5.3 describes three main components of PNELF, including personal name extractor, searcher and personal name disambiguator. Secondly, Section 5.4 describes how to predict the NIL value. Finally, Section 5.5 introduces a new algorithm, Simple Partial Tree Matching (SPTM), that we used to handle the lexical ambiguity problem.

5.1 Motivation

Named entity disambiguation (NED) plays an important role in many applications such as entity linking or search engines. NED is the task of linking mentioned names within a web document to the correct real-world entity in the existing knowledge base [45]. NED is also known as entity linking. NED differs from entity resolution (ER) because ER aims to cluster multiple entities or web pages into meaningful sub-groups (a single group represents a unique entity) [45]. Personal Named Entity Linking (PNEL) is the task of mapping the personal named entities that are mentioned within a web page to their corresponding person in a knowledge base.

Dredze et al. [24] described three fundamental challenges in NED: referential ambiguity, lexical ambiguity and the NIL value prediction. Firstly, referential ambiguity or name variation means the different names may refer to the same person. For example, mentioned names: *Becks* and *Db7* refer to *David Beckham*, an English former professional footballer. Secondly, lexical ambiguity means a single name may refer to multiple persons. For example, the mentioned name: *Andrew Scott* may refer to two different people: *Andrew Scott*, an actor and *Andy Scott*, an English footballer. Thirdly, the absent entity means a personal name entity may not appear in a knowledge base.

Man and Yarowsky [48] proposed a new approach using an agglomerative clustering technique based on the vector cosine similarity and biography information to handle lexical ambiguity. The experimental results showed that proper nouns, relevant words within a document and feature sets: birth year and occupation are useful for disambiguation lexical ambiguity. This is the first study to investigate which features in personal information provide a good performance in personal name disambiguation. However, this study deals with an entity resolution problem and is based on specific web documents (providing biography information). However, web pages usually have imperfect or partial information about the person [76].

Bunescu and Pasca [8] solve the referential ambiguity problem using a proper name dictionary derived from Wikipedia title pages, redirect pages and disambiguation pages to generate a set of candidate entities. The context similarity technique and support vector machine (SVM^{light}) are used to deal with the lexical ambiguity problem. The candidate entities are ranked based on the matching score over the query context, the Wikipedia article text and SVM based on a taxonomy kernel of their Wikipedia categories. The candidate entity that has the highest score is considered and linked.

Cucerzan [19] introduced a new technique to recognise the global document-level topical coherence of the entities. His work can both identify and disambiguate names. In the unambiguous process, a vector space model is used to compare the similarity between the contexts in a web document and the contexts in a Wikipedia entity page. Furthermore, the topical coherence score is computed from the referent entity candidate and other entities within the same context based on their overlaps in categories and incoming links in Wikipedia. However, both Bunescu and Pasca [8] and Cucerzan [19] directly derived entity categories from Wikipedia, and as we know, some of the Wikipedia categories are dirty and not well formed [63].

To handle the problems with the Wikipedia categories, the YAGO ontology taxonomy class is used in [62] by combining "*topic coherence*" and "*context similarity*" based on two knowledge bases: Wikipedia and YAGO in the NED task. However, the main process of the three studies above used the context similarity for comparing the pairs entity. The major limitation in the context similarity metric is that it requires very explicit words between the two comparators, which is difficult in natural language because of the flexible use of natural language [45].

Shen et al. [63] propose a new method call LIEGE ("Link the entITies in wEb lists with the knowledGe basE") to link the entities in a web list to the real-world entity. Two features include priority probability (popularity probability in each entity) and entity coherence to metric the similarity conceptual type that is derived from the YAGO taxonomy class and

is used for entity disambiguation. However, the context similarity metric is still used in this work to determine the semantic similarity between a mentioned entity and an entity in a knowledge base. Moreover, the study [22] suggests that the YAGO taxonomy class is not clean and some parts of the taxonomy classes that are derived from Wikipedia do not associate with WorldNet classes.

The previous studies in NED address the precision in named entity disambiguation or the disambiguator process. However, the candidate generating process is one of the most important steps in the NED task. Furthermore, real-world data is often dirty, faulty or invalid formatted information [16]. Therefore, matching the exact term over the proper name dictionary that is used in [8, 19, 62] may not be sufficient for generating a set of candidate entities. To solve the above problems, we proposed the personal name entity linking framework (PNELF) to improve the performance in personal name disambiguation. PNELF differs from the previous approaches above for two reasons:

1. This thesis boosts the performance in the candidate generating process by inserting PNTM to generate a set of uniform personal name representations and uses the Jaro-Winkler function to detect a candidate entity over a personal name surface form. PNTM with Jaro-Winkler is better than an exact-match look up because it can handle three main problems in referential ambiguity: rearrangement of words, multiple spelling or typographical errors and name variations.
2. In the disambiguator process, we introduce the SPTM algorithm to solve the lexical ambiguity problem under our assumption that people who appear within a document have the similar concept. The SPTM only uses a set of mentioned names within a web page and personal name concepts as input for ranking a set of candidate entities so the context similarity metric is unnecessary in our approach.
3. This thesis provides a solution to predict the NIL value when a mentioned name does not appear in a knowledge base. Moreover, this study introduces a method to predict a possible person for an absent entity through BingAPI.

5.2 Background

In this section we present the background to personal name disambiguation. This thesis introduces AlchemyAPI [70], the text mining tool that we use for extracting personal names from web documents.

5.2.1 AlchemyAPI

AlchemyAPI is a text mining tool. It combines natural language processing technology and machine learning algorithms for analysing unstructured data (e.g. web content, emails, blogs, etc.) and to extract semantic meta-data from content, such as named entities (people, locations, companies, etc.), facts and relations, topic keywords, text sentiment, news and blog article authors, taxonomy classifications and scraping structured data.

AlchemyAPI supports three types of content: Internet-accessible web pages, posted HTML or text content for performing content analysis. AlchemyAPI uses web service technology that is useful for a developer to connect via API endpoint with multiple computer languages, e.g. Java, C or C++, C#, Perl, PHP, Python, Ruby, Javascript and Android OS or to access directly through HTTP REST interface.

AlchemyAPI contains twelve APIs including Entity Extraction, Sentiment Analysis, Keyword Extraction, Concept Tagging, Relation Extraction, Taxonomy, Author Extraction, Language Detection, Feed Detection, Text Extraction, Microformats Parsing and Content Scraping. In our work, *Entity Extraction API* is used for extracting proper nouns such as people, organisation and location. The API provides the returning extracted meta-data in multiple formats such as XML, JSON, RDF and Microformats.

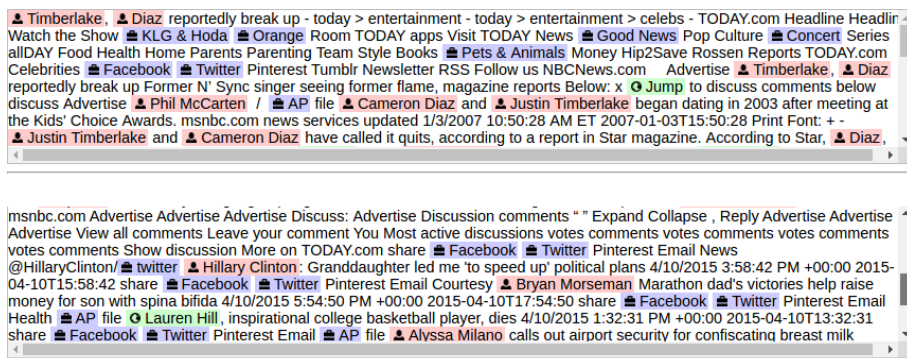


Fig. 5.1 Example results from GATE information extractor

For extracting personal names from a web document, another information extractor is GATE [53], an open source information extractor. However, this tool is limited in two ways. Firstly, it returns the personal names outside the scope of a document. For example, when we input the web document (<http://www.today.com/id/16444023>) shown in Figure 5.1, GATE returns personal names such as *Phil McCarten*, *Hillary Clinton* or *Bryan Morseman*. These names appear as the external links outside the web page. Secondly, Gate cannot recognise a personal name. Even though the names are repeated, it returns every name that occurs in the document. For the above reasons, AlchemyAPI is more effective

than GATE as a personal name extractor.

5.3 Personal Name Entity Linking Framework (PNELF)

Linking a personal name which is extracted from a web document to a real-world entity is a complex task for three reasons: referential ambiguity, lexical ambiguity and absent entity (a personal name may not be represented in a knowledge base). We proposed PNELF as shown in Figure 5.2 to solve these problems. The input is a single web document and the outputs are an identifiable person or the NIL value if an input personal name does not appear in our data catalogue. The rectangles that have the thick line are the processes or the components in PNELF. The personal name extractor is the first component in PNELF. After that a set of mentioned names that are extracted from a web document will go through the searcher component via the thick arrow. The searcher component produces a set of candidate entities. Finally, a set of candidate entities in each mentioned name go to the personal name disambiguator component and return the results: identifiable personal. The dashed rectangles are the data input in each process using the dashed arrow. PNELF consists of three main components: personal name extractor, searcher and personal name disambiguator.

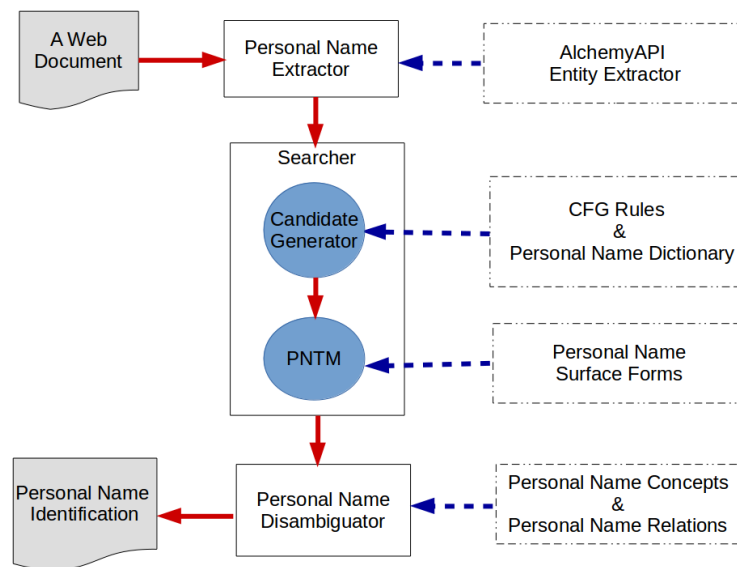


Fig. 5.2 Personal Name Entity Linking Framework

1. The personal name extractor is the component for extracting and constructing personal names that are mentioned in a web document. The Alchemy entity extraction API is

used in this step and the output is a list of mentioned names in a web document.

2. The searcher component aims to generate a set of candidate entities in each mentioned name. This component is separated into two steps: PNTM and candidate generator.
 - PNTM is a step of transforming a variety of different personal names mentioned to a uniform representation. A set of personal names in the first step are the inputs in this step. A non-standard name (e.g. alternative name, nickname or a name that starts with a last name) will be transformed into a uniform format under PNTM modules as described in Chapter 4. We call the personal names that are generated in this step a set of probable personal names.
 - Candidate generator is a step of producing a set of personal name candidates for a mentioned name by matching over a personal name surface form. The input in this step is a set of probable personal names. A set of probable personal names will be matched over our personal name surface form based on Jaro-Winkler, the text similarity metric. The results are a set of candidate entities in each mentioned name.
3. Personal name disambiguator is a component of selecting the best person from ambiguous personal names and returning the NIL value for an unlinkable mentioned name. The disambiguator will be processed if each mentioned name has more than one answer. The component evaluates through a set of candidate entities by calculating a similar score among the candidate entities. We use two features: Simple Partial Tree Matching (SPTM) algorithm and personal name relationships to calculate the similar score. The candidate entity that has the highest score will be selected.

We consider one web document at a time and each document should have more than one mentioned name because we do not have an identifiable personal name concept to create a comparison tree when a mentioned name refers to more than one possible person.

5.3.1 Personal Name Extractor

The "*alchemy API*" is used to detect and construct the personal names that appear in a web document. Given a single web page, the AlchemyAPI returns a set of mentioned names in XML format. Figure 5.3(a) is an input web page and Figure 5.3(b) is an output XML document.

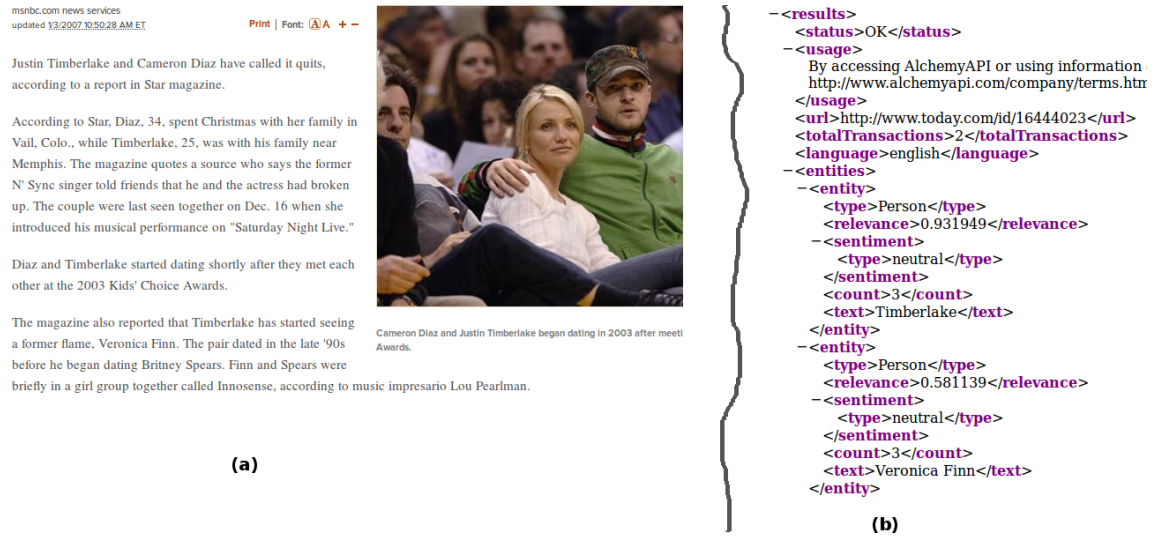


Fig. 5.3 An example of a personal name extractor module (a) input: a web document and (b) output: alchemy API output values in XML format.

The extractor travels under tag *<entity>* and consider the element in tag *<type>* first. If the element in tag *<type>* is *Person* then an element that appears in tag *<text>* will be detected e.g. *<text>Timberlake</text>*.

Definition 5.1. Given $X = \{x_1, x_2, \dots, x_n\}$ is a set of mentioned names within a web page w . A web document w will be considered if X has at least two members.

For example, given the web document as shown in Figure 5.3(a) the extractor will return the six mentioned names:

$$X = \{\text{Timberlake, Veronica Finn, Diaz, Britney Spears, Lou Pearlman, Star}\}$$

5.3.2 Searcher

The exact-match look up over the proper name dictionary that is used in [8, 19, 62] may provide incorrect result in personal name matching. This is because personal names have multiple representations. This study improves the searcher performance by adding two steps: PNTM and candidate generator before generating a set of candidate entities.

PNTM

Personal name variations or referential ambiguity is a fundamental problem in personal name matching. Furthermore, this problem reduces the performance in text similarity metrics (e.g. cosine similarity, edit distance) [3]. To solve this problem, we introduced PNTM,

which was explained in Chapter 4 to transform personal name variations into a uniform representation before matching using a text similarity metric.

The PNTM step aims to prepare a set of mentioned names to be ready for generating a set of candidate entities by reducing numerous personal name formats to a uniform representation. The scope of our module is to transform an alternative name, a nickname or a name that starts with a last name into a uniform format.

For example, a set of personal names mentioned in a web document are shown below:

$$X = \{\text{Timberlake, Veronica Finn, Diaz, Britney Spears, Lou Pearlman, Star}\}$$

will be transformed into:

$$X = \begin{cases} x_1 = \textit{Timberlake} = & \{\text{Craig Timberlake, Justin Timberlake}\} \\ x_2 = \textit{Veronica Finn} = & \{\text{Veronica Finn}\} \\ x_3 = \textit{Diaz} = & \{\text{Cameron Diaz}\} \\ x_4 = \textit{Britney Spears} = & \{\text{Britney Spears}\} \\ x_5 = \textit{Lou Pearlman} = & \{\text{Lou Pearlman, Lucille Pearlman, Lucinda Pearlman,} \\ & \text{Louis Pearlman, Louise Pearlman}\} \\ x_6 = \textit{Star} = & \{\text{Jeffree Star, Sunshine Dizon}\} \end{cases}$$

The output values in personal name transformation describe that the module can generate a list of personal names from one input. For example, two personal names: *Craig Timberlake* and *Justin Timberlake* are generated from the short name *Timberlake* or a list of given names: {Lucille, Lucinda, Louis, Louise} are generated from the nickname: *Lou*.

Definition 5.2. Given $G_x = \{g_1, g_2, \dots, g_m\}$ is a set of standard names that are generated from a mentioned name x . For example, the standard names for a mentioned name *Timberlake* are

$$G_{\textit{Timberlake}} = \{\text{Craig Timberlake, Justin Timberlake}\}.$$

The personal name transformation can boost the performance in a text similarity function because it provides a uniform pattern of mentioned names.

Candidate Generator

The candidate generator step is based on a text similarity measurement between the standard name and personal name surface forms that we described in Chapter 3. The Jaro-Winkler

function is used to calculate the similarity score and the person who has a matching score of higher than 97% will be considered to be a candidate entity. The similarity score 97% is an acceptable number from our implementation. This is because the candidate generator is designed to redress the balance between precision and recall in generating a set of candidate entities. We allow a single typographic error to be a candidate entity.

Jaro-Winkler is suitable for first name and last name matching [25, 74] and is faster than other basic edit distance algorithms [60]. Furthermore, the experimental results in Chapter 4 showed that personal name transformation and Jaro-Winkler provide the highest accuracy value in text similarity matching. As a result, we used Jaro-Winkler in our module to search for a set of candidate entities in each mentioned name.

Definition 5.3. Given $P(x) = \{p_1, p_2, \dots, p_m\}$ is a set of candidate entities for a mentioned name x and $p \in P$ is a set of personal names in our data catalogue. For example, a set of candidate entities for a mentioned name *Timberlake* is:

$$P(\textit{Timberlake}) = \{\textit{Craig Timberlake}, \textit{Justin Timberlake}\}.$$

Finally, the NIL value prediction is generated by returning a matching value 1 or 0 to x . It returns 1 if a mentioned name x can be linked to a personal name in our catalogue or it returns 0 if a mentioned name x cannot be linked to any personal name in our catalogue.

$$x = \begin{cases} 0 & \text{if } P_x = \emptyset \\ 1 & \text{otherwise} \end{cases}$$

Finally, from the standard names for a personal name mentioned in the document w , the candidate generator module returns:

$$X = \begin{cases} P(x_1) = P(\textit{Timberlake}) = & \{\textit{Craig Timberlake}, \textit{Justin Timberlake}\} \\ P(x_2) = P(\textit{Veronica Finn}) = & \{0\} \\ P(x_3) = P(\textit{Diaz}) = & \{\textit{Cameron Diaz}\} \\ P(x_4) = P(\textit{Britney Spears}) = & \{\textit{Britney Spears}\} \\ P(x_5) = P(\textit{Lou Pearlman}) = & \{0\} \\ P(x_6) = P(\textit{Star}) = & \{\textit{Jeffree Star}, \textit{Sunshine Dizon}\} \end{cases}$$

As a result, the number of candidate entities in each set is reduced (it returns a set of specific candidate entities). In the next section we propose a new technique to handle the lexical ambiguity problem. In this step, the two mentioned names $x_3 = \textit{Diaz}$ and $x_4 = \textit{Britney Spears}$ are identified as $P(x_3) = \textit{Cameron Diaz}$ and $P(x_4) = \textit{Britney Spears}$. Furthermore,

the two mentioned names $x_1 = \textit{Timberlake}$ and $x_6 = \textit{Star}$ are lexical ambiguity because it returns more than one personal name and the two mentioned names $x_2 = \textit{Veronica Finn}$ and $x_5 = \textit{Lou Pearlman}$ are absent.

To reduce the workload and boost the disambiguator performance by adding a function for proving a set of candidate entities before going to the disambiguator process. This function removes closely similar entities if we find that one of them exactly matches the mentioned name.

For example, the two mentioned names: *Christian Bale* and *Tony Scott*, in the personal name transformation process, the first names *Christian* and *Tony* can be transformed into *Christopher* and *Anthony* because both names can be a nickname. This process returns a set of results as follows:

$$x_1 = \textit{Christian Bale} = \{\textit{Christian Bale}, \textit{Christopher Bale}\}$$

$$x_2 = \textit{Tony Scott} = \{\textit{Tony Scott}, \textit{Anthony Scott}\}$$

The outputs above become inputs in the candidate detecting process. In this process we use the Jaro-Winkler similarity function and minor spelling mistakes can be passed. The two sets of candidate entities for *Christian Bale* and *Tony Scott* are shown below:

$$P(\textit{Christian Bale}) = \{\textit{Christian Bale}, \textit{Christopher Hale}, \textit{Christian Abel}, \textit{Christopher Blake}, \textit{Christopher Gable}\}$$

$$P(\textit{Tony Scott}) = \{\textit{Tony Scott}, \textit{Tony Scotti}, \textit{Tony Scott (footballer)}\}$$

Then these sets of candidate entities will go to the candidate proving function. After this process a set of candidate entities should be:

$$P(\textit{Christian Bale}) = \{\textit{Christian Bale}\}$$

$$P(\textit{Tony Scott}) = \{\textit{Tony Scott}, \textit{Tony Scott (footballer)}\}$$

5.3.3 Personal Name Disambiguator

We now present Simple Partial Tree Matching for ranking candidate entities where a set of $P(x)$ contains more than one element. To disambiguate a personal name entity, we assume that the personal names that appear within a single web page have the same concept and/or have some relations.

To disambiguate a personal name, we use personal name concepts and the personal name entity relations. Both are introduced in Chapter 3 and a simple partial tree matching algorithm is used to calculate the similarity scores. The candidate in each mentioned name x which has the maximum score is selected. The personal name disambiguator will be separated into two steps: building a comparison tree and calculating the similarity score.

1. Building a comparison tree. The mentioned names which have a single candidate entity are considered. This candidate entity is called "precise-entity". The comparison tree is generated from these precise-entity's concepts. For example, in the candidate generator, we can identify two mentioned names x : *Diaz and Britney Spears* are referred to as *Cameron Diaz and Britney Spears*. The concepts from these two people are used to construct the comparison tree.

The concepts which have the same root node are merged and the child nodes are sorted in ascending order. The number of comparison trees depends on the number of different root nodes in a set of clear-entities.

For example, we have one comparison tree for these two clear-entities because *Cameron Diaz and Britney Spears* have the same concepts (*Entertainers and Artists*). Figure 5.4, the conceptual tree of the entity *Cameron Diaz* starts from the root node *Entertainers and Artists* and ends with the *American voice actors* node. *Britney Spears*'s conceptual tree starts from *Entertainers and Artists* like *Cameron Diaz* but ends with the *American music video directors* node. The two concepts are merged and arranged in ascending order as shown in Figure 5.4.

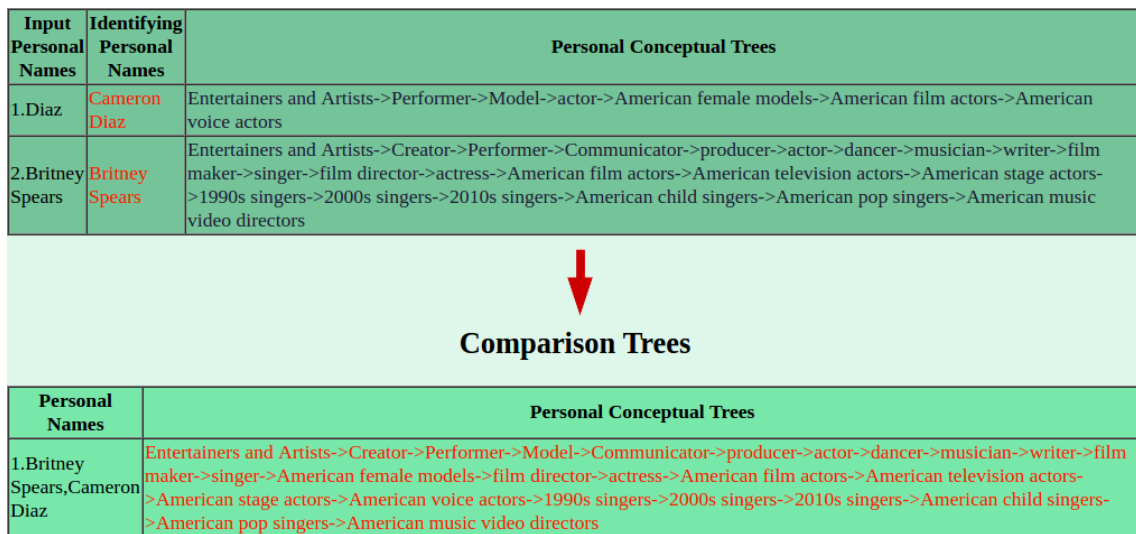


Fig. 5.4 Example of comparison tree

2. Calculating similarity score: The similarity score in each candidate person is computed from two components: the relations values and the matched concept values.

Given $I(p)$ is a set of clear-entities in a web page w . For example, a list of clear-entities in this document is

$$I(p) = \{Cameron\ Diaz, Britney\ Spears\}$$

The SPTM as described in Section 5.5 is used to compute the matched concept value. The matched concept value is the similarity score between a candidate tree and a comparison tree.

The comparison tree which has the maximum number of identifying entities is considered first. If the number of clear-entities is equal, a number of nodes in each comparison tree will be considered. We use function 5.3 to compute the matched concept score between a comparison tree and a candidate tree. Importantly, a candidate tree that has a different root node from the comparison tree will be discarded.

In Figure 5.2, a mentioned name *Star* is a lexical ambiguity because it refers to two different people *Jeffree Star* and *Sunshine Dizon*. Therefore, these two people are the candidate entity for the mentioned name *Star*. For example, to compute the similarity score for the candidate *Sunshine Dizon*:

- (a) The matched concept = $n = \{\text{Entertainers and Artists, Performer, actor}\} = 3$.
- (b) The weighting score = $\sum_{i=1}^n w_i = 1+2+3 = 6$, where w_i is the level of each matched node.
- (c) The similarity score ($T_c, T_{SunshineDizon}$) = $3*6 = 18$.

Simple Partial Tree Matching					
Input Personal Names	Comparison Trees	Disambiguation Personal Name	Personal Conceptual Trees	Matching Trees	Score
1. Timberlake	Entertainers and Artists->Creator->Performer->Model->Communicator->producer->actor->dancer->musician->writer->film maker->singer->American female models->film director->actress->American film actors->American television actors->American stage actors->American voice actors->1990s singers->2000s singers->2010s singers->American child singers->American pop singers->American music video directors	1.1 Justin Timberlake	Entertainers and Artists->Performer->actor->dancer->musician->singer->American film actors->American child actors->American voice actors->1990s singers->2000s singers->2010s singers->American child singers->American male singers->American pop singers	Entertainers and Artists->Performer->actor->dancer->musician->singer->American film actors->American voice actors->1990s singers->2000s singers->2010s singers->American child singers->American pop singers	637.00
		1.2 Craig Timberlake	Entertainers and Artists->Performer->actor->musician->singer->American stage actors->American musical theatre actors->American opera singers	Entertainers and Artists->Performer->actor->musician->singer->American stage actors	102.00
2. Star	Entertainers and Artists->Creator->Performer->Model->Communicator->producer->actor->dancer->musician->writer->film maker->singer->American female models->film director->actress->American film actors->American television actors->American stage actors->American voice actors->1990s singers->2000s singers->2010s singers->American child singers->American pop singers->American music video directors	2.1 Jeffree Star	Entertainers and Artists->Performer->Model->musician->singer->American male models->American male singers	Entertainers and Artists->Performer->Model->musician->singer	60.00
		2.2 Sunshine Dizon	Entertainers and Artists->Performer->actor	Entertainers and Artists->Performer->actor	18.00

Fig. 5.5 An example conceptual matching score

As a result, the mentioned name *Star* is referred to as *Jeffree Star* because the similarity score is higher than that for *Sunshine Dizon*.

5.4 Predicting the NIL Value

A knowledge base contains a large amount of personal name entities. However, in reality many entities do not occur in the knowledge base. Therefore, predicting the NIL value for a mentioned name x is one of the most significant in personal name disambiguation.

The mentioned name x is predicted as NIL when:

1. PNTM cannot generate a standard name for a mentioned name ($x_i = \emptyset$).
2. We cannot generate a set of candidate entities for a mentioned name x ($P(x) = \emptyset$). Due to the mentioned name x does not match any of our personal surface forms (Jaro-Winkler similarity score less than 97%).
3. We can map the mentioned name x to an entity in our data catalogue but it has a different root node from the existing identifying person and it does not have any relations with the existing identifying person. Therefore, we return the NIL value to this mentioned name x . Due to this answer, we reject our assumption that personal names that appear within a single web page have the same concept and/or they are related.

Finally, we propose a technique to query the NIL value of mentioned name x from the search engine. We pass two values to BingAPI including mentioned name x and occupation. An occupation is selected from the most popular occupation in a comparison tree. In addition, the selecting node should have a depth value of more than 1 for *Sportsman and Politician* and 2 for *Entertainer and Artist*. For example, given the NIL mentioned name: *Veronica Finn* with the attached occupation text: *actor* that is extracted from the comparison tree. BingAPI returns ten related links for a mentioned name *Veronica Finn*. Figure 5.6 shows the top ten links returned for *Veronica Finn* with the query *Veronica Finn actor*. Now, we know that *Veronica Finn* is *Justin Timberlake*'s first girlfriend.

1. [Justin Timberlake's first girlfriend Veronica Finn spills ...](#)
2. [Veronica Finn Photo Gallery - AllStarPics.Net](#)
3. [Heathers - Wikipedia, the free encyclopedia](#)
4. [Veronica Finn from Innosense spills the lipton on Justin ...](#)
5. [Justin Timberlake's first girlfriend reveals love letters ...](#)
6. [Justin Timberlake's First Girlfriend Reveals Love Letters](#)
7. [Justin Timberlake & Veronica Finn Dated, Joint Family Tree ...](#)
8. [Veronica Finn | Celebrities lists.](#)
9. [Justin Timberlakes first girlfriend reveals love letters ...](#)
10. [Learn and talk about Veronica Finn, American dance music ...](#)

Fig. 5.6 Example of browsing the NIL value with BingAPI

5.5 Simple Partial Tree Matching Algorithm

The simple partial tree matching (SPTM) technique is based on two algorithms: STM and PTA. Firstly, Simple Tree Matching (STM) gives us the idea of calculating the similarity score between the comparison tree and the candidate tree. We improve the algorithm performance by adding the weighting score at each level of the node tree. Secondly, building a personal conceptual tree and selecting a comparison tree are based on the Partial Tree Alignment (PTA) [81]. SPTM contains two main components: a comparison tree and tree matching, as described below.

Importantly, we proposed two conditions before using SPTM. Firstly, a node in the tree is unique and it can occur once in each tree. Secondly, the multiple levels in the tree hierarchy will be flattened into two levels; the first level is a root node and the second level is a set of child nodes. These child nodes under the root node are arranged flowing down, and start from left to right, one level at a time. Each node contains two important pieces of information: its content and its depth.

Figure 5.7 shows how to remove the level and sort the child nodes. The original tree

(a) has four levels after being flattened, all child nodes are compacted into one level using top-down sorting, from left to right, one level at a time. The resulting flattened tree is shown in Figure 5.7(b).

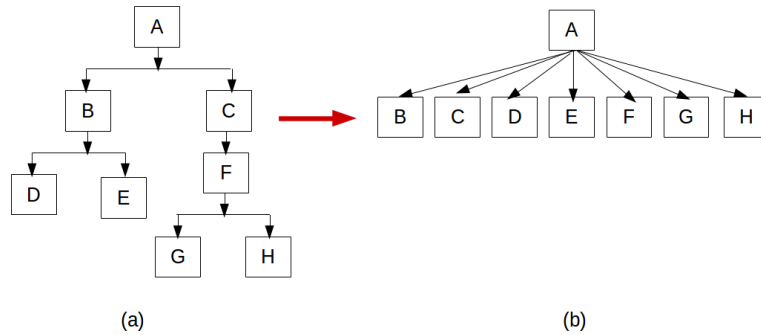


Fig. 5.7 An example of a flattened tree (a) Original tree (b) Flatten tree

5.5.1 Building the Comparison Tree

The personal name concept of identifying mentioned names in each web page is used to construct the comparison tree. Let $t \in T$ is a set of identifying personal name concepts. All nodes in T will be merged into comparison T_c if their root nodes are equal. The duplicate nodes will be removed and the unique node will be arranged in ascending order.

$$T_c = \bigcup_{i=1}^n t_i. \quad (5.1)$$

Where $t_{r1} = t_{r2} = \dots = t_{rn}$.

We develop an example in Figure 5.8 to describe how to create a comparison tree. There are 3 initial trees, and all of them only have two levels. The root node in each tree will be matched first, and the trees can be merged only if their root nodes are equal. Therefore, tree T_1 and T_2 are merged because they have the same root node A and they created the new tree called comparison tree (T_c). All nodes in the two trees are merged and sorted in ascending order. The two comparison trees are shown in Figure 5.8(b).

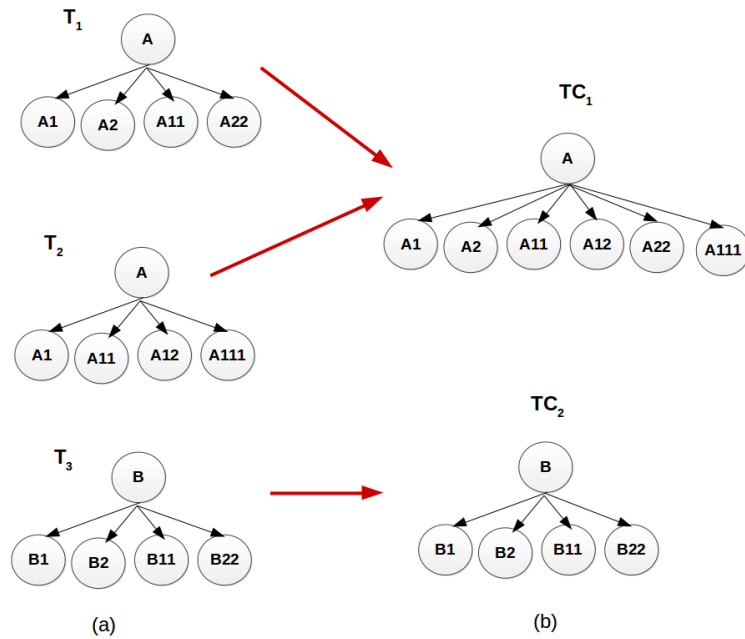


Fig. 5.8 Building the comparison tree (a) initial trees and (b) comparison trees

5.5.2 Simple Partial Tree Matching

The SPTM algorithm consists of three steps: comparison tree selecting, two tree matching and similarity score calculating as described in the following:

1. Comparison tree selecting. The comparison tree is considered from two conditions:
 - 1) The number of members that are joined in the comparison tree.
 - 2) The number of nodes in the comparison tree.

This algorithm considers the comparison tree that has the maximum number of members first. If the number of members is equal, the comparison tree that has the highest number of nodes is selected.

2. Two tree matching. SPTM is a two tree matching between the comparison tree and the candidate tree. The matching tree consists of two steps:
 - 1) Matching the root node. If two root nodes are similar go to the next step, otherwise return unmatched (two trees are different).
 - 2) Matching the child nodes. The child nodes from two trees are matched. A maximum number of matching nodes is the size of the candidate tree and a minimum number of matching nodes is 1.

$$T_m = T_c \cap T_i \quad (5.2)$$

3. Similarity score calculating. The similarity score between the comparison tree and the candidate tree is calculated from the total number of matching nodes and the sum of their depth values. The weighting is assigned to each node and its value is sorted in descending order (1-n) from the root level to the n level. The nodes which are on the same level have equal weight. The leaf nodes have the greater weighting than their parent node because if the leaf nodes are matched, it implied that the two trees are closely related.

$$\text{similarity}(T_c, T_i) = n * \sum_{i=1}^n w_i \quad (5.3)$$

Let n is the number of matching nodes between T_c and T_i and w_i is the weighting score in each matched node. The weighting score is the depth of each node in a tree. All nodes in a tree have a permanent depth value; it never changes because the depth value comes from our occupation taxonomy, as described in Chapter 3.

Figure 5.9 gives an example of the SPTM algorithm. The two trees T_c and T_i have the same root node A , which mean the two trees have the same concept and can be matched. The matching nodes are $\{A, A_1, A_{11}\}$.

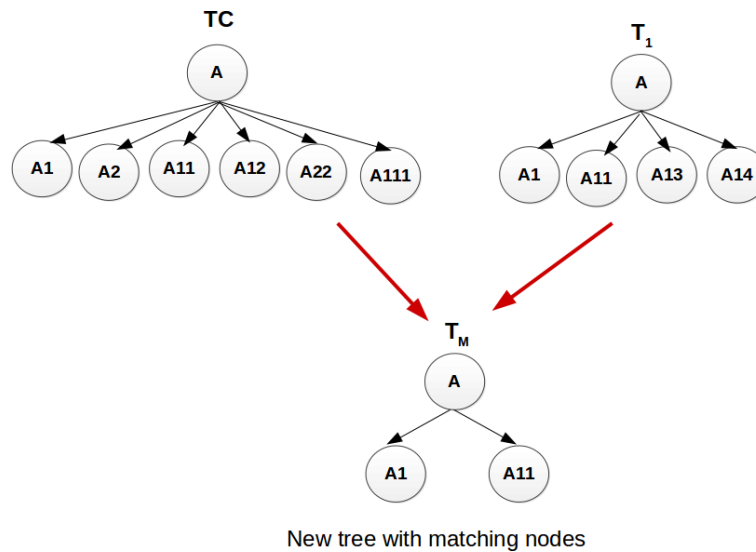


Fig. 5.9 An example of a simple partial tree matching algorithm

5.6 Experimental Results and Discussion

In this section, we evaluate the performance of our personal name disambiguation framework. In order to evaluate the effectiveness of the framework for linking a mentioned name to a real-world entity, the performance of the framework is evaluated using two criteria:

1. The searcher performance in generating a set of candidate entities.
2. The disambiguator performance in personal name entity linking.

5.6.1 Data Sets

The two criteria above were evaluated in two data sets that provided personal names, their alternative names and their professional categories. The first dataset is YAGO [42] version 2.3.0_{core} as of January 9, 2012. This data sets are used to create our data catalogue and the entity concepts. Our data catalogue is described in Chapter 3. Since there is no familiar publicly available data set for personal name linking tasks and the data sets in [19] are no longer available, we do not have a benchmark for us to compare.

To measure the performance of our framework, we collected 100 web documents from three websites:

1. <http://www.today.com>
2. <https://uk.yahoo.com>
3. <http://www.msn.com/en-gb/>

We collected the content from these websites in two main directories, including football and entertainment to evaluate our framework.

The details of the data sets are shown in Table 5.1. The first five rows are about the data in our data catalogue. The catalogue contains 107,058 people, 332 occupation categories including *Person* (a root categories), 105,604 personal name concepts, 145,638 personal name surface forms and 4,203 personal name entity relations.

In evaluating data sets, we have 992 mentioned names, 119 referential ambiguity names and 114 lexical ambiguity names.

Table 5.1 Summary data sets.

Entity	Total
People	107,058
Occupation categories	322
Personal name concepts	105,604
Personal name surface forms	145,638
Personal name entity relations	4,203
Mentioned names	992
Referential ambiguity names	119
Lexical ambiguity names	114

5.6.2 Evaluation Results

To verify our framework using two criteria: searcher and disambiguator effectiveness as we describe in the following subsection.

Searcher Performance

The searcher component consists of two components: PNTM and candidate generator. We use the function introduced in [34] to evaluate the performance of our searcher component. Figure 5.10 describes the notations we used. We evaluate the effectiveness of the searcher by analysing the precision, recall and NIL value of the candidate entities returning.

Fig. 5.10 Notation for evaluation searcher performance [34]

N	Number of queries in data set
\mathcal{G}	Gold standard annotations for data set ($ \mathcal{G} = N$)
\mathcal{G}_i	Gold standard for query i (KB ID or NIL)
\mathcal{C}	Candidate sets from system output ($ \mathcal{C} = N$)
\mathcal{C}_i	Candidate set for query i
$\mathcal{C}_{i,j}$	Candidate at rank j for query i (where $\mathcal{C}_i \neq \emptyset$)

candidate count ($\langle C \rangle$) is the number of elements in a set of candidate entities. A small number of candidates signifies a lower disambiguation workload.

$$\langle C \rangle = \frac{\sum_i |\mathcal{C}_i|}{N} \quad (5.4)$$

candidate precision (P_c) is the percentage of non-empty candidate sets containing the correct entity.

$$P_c = \frac{|\{C_i | C_i \neq \emptyset \wedge g_i \in C_i\}|}{|\{C_i | C_i \neq \emptyset\}|} \quad (5.5)$$

candidate recall (R_c) is the percentage of non-NIL queries where the candidate set contains the correct candidate.

$$R_c = \frac{|\{C_i | g_i \neq NIL \wedge g_i \in C_i\}|}{|\{g_i | g_i \neq NIL\}|} \quad (5.6)$$

NIL precision (P_\emptyset) is the percentage of entity candidate sets that are correct.

$$P_\emptyset = \frac{|\{C_i | C_i \neq \emptyset \wedge g_i = NIL\}|}{|\{C_i | C_i \neq \emptyset\}|} \quad (5.7)$$

NIL recall (R_\emptyset) is the percentage of NIL queries for which the candidate set is empty. A greater R_\emptyset rate is useful because it reduces the workload of the disambiguator to decide whether queries are NIL-linked when candidates are returned.

$$R_\emptyset = \frac{|\{C_i | g_i = NIL \wedge C_i = \emptyset\}|}{|\{g_i | g_i = NIL\}|} \quad (5.8)$$

We now evaluate the effectiveness of the searcher component by calculating the accuracy in generating a set of candidate entities. The searcher component consists of PNTM, Jaro-Winkler and the personal name surface forms. PNTM is used to transform multiple personal name formats into a uniform representation, and after that, Jaro-Winkler is used for calculating the similarity score over personal surface forms. Finally, a set of matching candidate entities in each are generated.

Figure 5.11 shows the analysis results of the searcher performance. The candidate count is 1.6, which means one mentioned name has an average of 1.6 candidate entities. There is a smaller number of competitors and it reduces the workload in disambiguator component because it contains a smaller number of candidate entities to consider. The precision and recall in generating a set of candidate entities are both over 80%. This result means the searcher has a high performance in generating candidate entities. The performance in predicting the NIL value is nearly 100%.

Disambiguator Performance

Now, we analyse the disambiguator performance by evaluating the precision in personal name entity linking and lexical ambiguity disambiguation. The precision in personal name

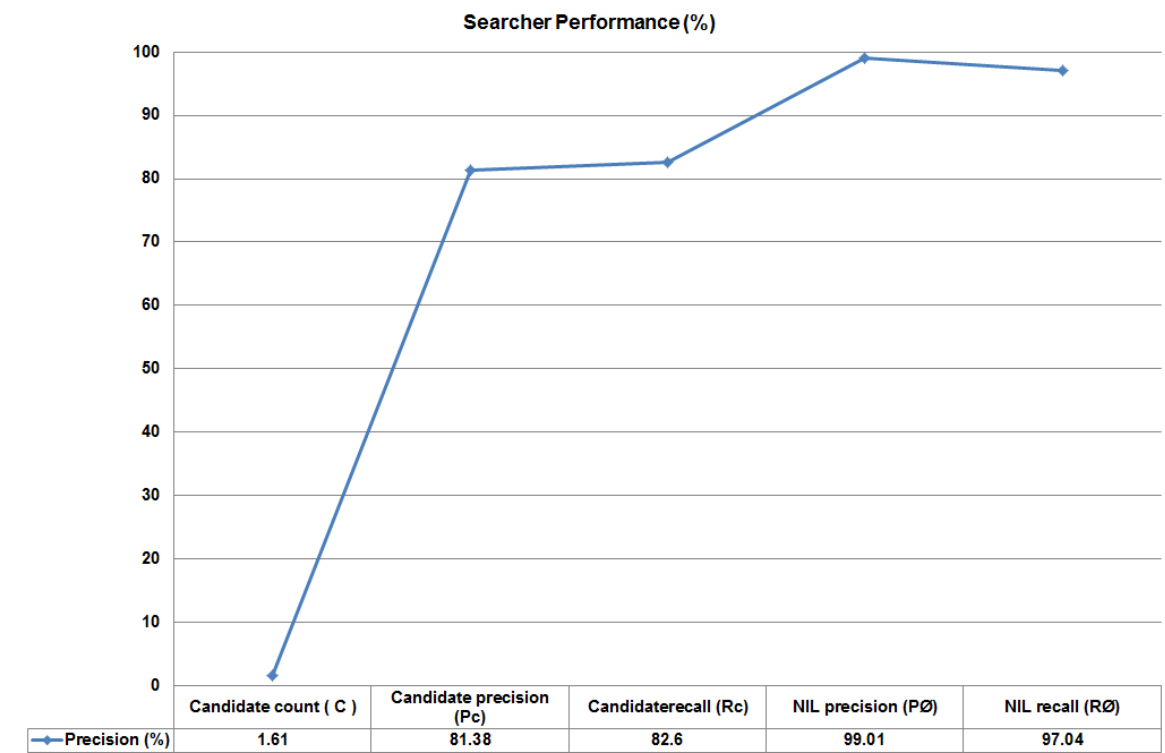


Fig. 5.11 Searcher performance.

linking or lexical ambiguity handling means that the component returns the right person that is mentioned within a document. For example, in Figure 5.3(a), the personal name linking is accurate if it links to *Cameron Diaz* for the mentioned name *Diaz*. In the case of lexical ambiguity, the mentioned name such as *Timberlake* should be linked to *Justin Timberlake* not *Craig Timberlake*.

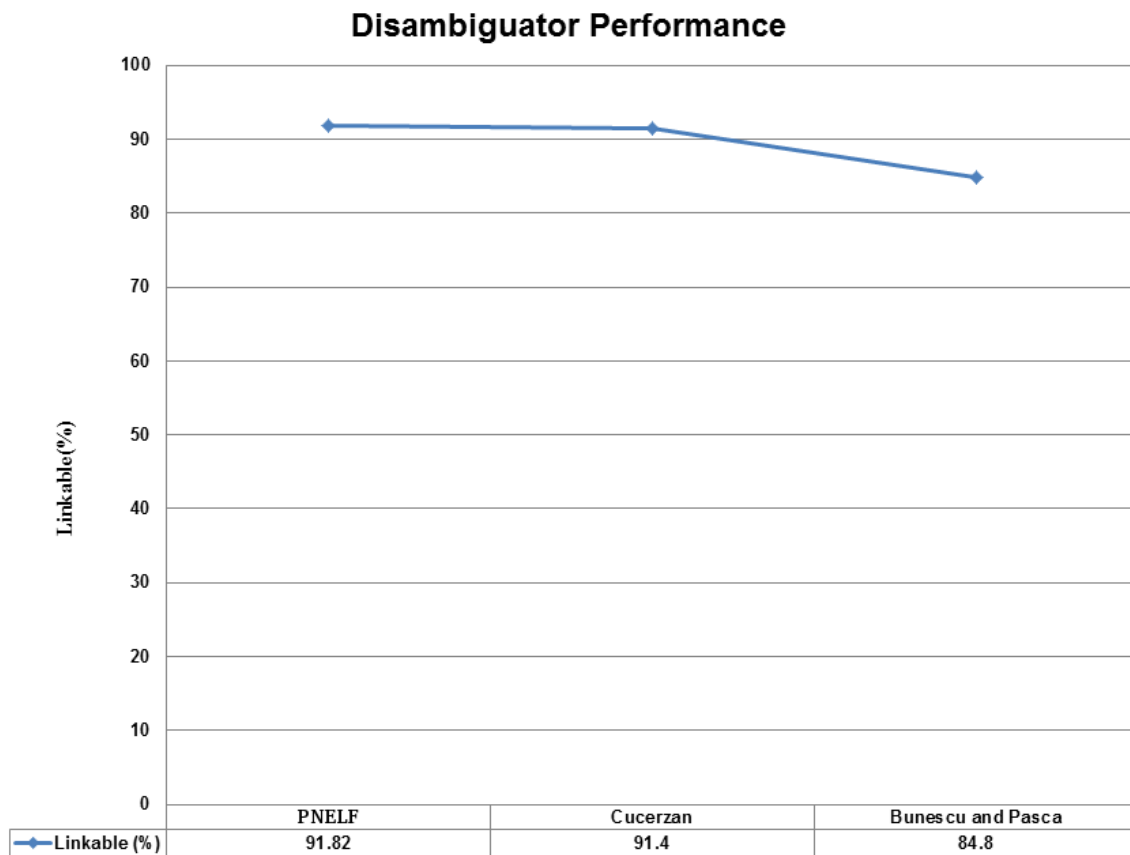


Fig. 5.12 Disambiguator performance.

Figure 5.12 shows the precision of an entity linking comparison using three techniques: PNELF, Cucerzan [19] and Bunescu and Pasca [8]. As can be seen from the graph, PNELF returns the greatest rise in personal name entity linking (91.82%). Extending PNTM in a searcher component and the new algorithm SPTM in a disambiguator component would have improved overall performance, higher than in Cucerzan (40%) and Bunescu and Pasca (7.02%).

The effectiveness in handling lexical ambiguity is 72.07%. The percentage of accuracy in lexical ambiguity disambiguation is not high (less than 80%). However, when we consider the incorrect answers in detail most of them are short names (one word name e.g. *Timberlake*, *Ronaldo* and *Fernando*). The short name usually returns the wrong answer because the searcher generates a large number of candidate entities. Therefore, occupation taxonomy does not sufficient feature to remove an ambiguity in a short name. From our experimental results, we found that the percentage of accuracy moves up to 85.95% if the lexical ambiguity is a full name. Furthermore, we found that personal name entity relations is an unimportant feature; it is rarely used in personal name disambiguation. Finally, we can

remove personal name entity relations from our similarity function.

5.7 Related Work

In recent years, researchers have proposed many methods to deal with three challenges in entity linking, including (i) name variations or referential ambiguity, (ii) entity ambiguity or lexical ambiguity and (iii) NIL value predicting in entity linking [24]. Here we analyse recent solutions which led to our study.

We start with Man and Yarowsky [48], who used the biographical information and applied the agglomerative clustering technique based on vector cosine similarity to distinguish personal name ambiguity. This is primary research which uses biographical data for the disambiguation of personal names and can be applied to our research. The researchers collected biographical information such as birth date, occupation, birth place, nationality and specific types of association such as familial relationships (e.g. son, wife), and employment relationships (e.g. manager of) by generating extraction patterns to extract data from the web. The bottom-up centroid agglomerative clustering is used to cluster the web pages which contain an ambiguous personal name. The researchers used four models: (i) baseline, (ii) most relevant word, (iii) basic biographical features and (iv) extended biographical features to evaluate the clustering methods.

1. Baseline models. They use term vectors composed of either all words (without "stop" words) or only proper nouns and standard cosine similarity to calculate the similarity between vectors. The result shows that using only proper nouns provides a higher accuracy score in clustering more than all words (without feature extracting).
2. Relevant word models. Two methods are used including standard TF-IDF weight and weighting based on the mutual information. The outputs without extracted features show that using proper nouns and weighting based on the mutual information gives the highest accuracy score.
3. Extracted biographical features. Biographical information such as birth year and occupation are used to cluster the web pages.
4. Extended biographical features model. The words which have been seen as filling a pattern are assigned to a higher weight. To evaluate the result, the features are separated into two groups: (i) small features (occupation, birthday (birth year), spouse, birth location and school) and (ii) large features (small features and entity relationships e.g. parent/child). The experiment showed that proper nouns, relevant words

and feature sets are given the highest accuracy score. Additionally, adding extended features is invaluable in this experiment.

However, the study in [48] aims to cluster the entities which are mentioned within the web pages which are imperfect for named entity disambiguation [63]. Furthermore, the approach is a more specific condition because a web page usually does not have information about biological information except a personal profile web page. One interesting finding is the relationship between a proper noun and its biographical features which can distinguish ambiguous personal names. For example, personal names which occur within web pages usually have some connection with each other (e.g. colleague, son, wife or husband). Biographical information such as occupation, familial relationships (e.g. spouse, sibling) is useful in our research for personal name disambiguation.

To solve referential ambiguity, the researchers in [8, 19, 62] used the redirect pages in Wikipedia to construct a personal name dictionary. A redirect page provides an alternative name that is used to make a reference to an entity in Wikipedia. The study [63] combined Wikipedia redirect pages and link probability (popularity score) to evaluate a candidate entity. The popularity score is used to discard little known candidate entities. However, the Wikipedia redirect pages may not cover all possible alternative names in each entity. Moreover, these researchers [8, 19, 62] used exact-match look up to generate a set of candidate entities that cannot handle the word reordering or minor spelling.

Dredze et al. [24] use the string similarity function to measure the similarity between the mentioned text and the entity title to generate candidate entities. To deal with the name variation problem, the researcher not only allows an exact match between the mentioned text and the entity title to be a candidate entity but also:

- 1) The entity title that is wholly contained in or contains the mentioned text.
- 2) The initial letters of the mentioned text match the entity title.
- 3) The entity title matches an alias over an alias dictionary.
- 4) The number of the acceptance similarity scores between the entity title mentioned and the mentioned text is varying depending on text similarity function, including: character Dice score $\text{Dice} > 0.9$, skip bigram Dice $\text{Dice} > 0.6$ and Hamming distance ≤ 0.2 .

Dredze was the first person to focus on names variation. However, the features are largely based on the text similarity function which may reduce the performance when comparing entity names and their alias names.

Our method differs from the method used by these researchers [8, 19, 24, 62]. Firstly, we construct alternative names from YAGO facts that have the properties *means*. Secondly, we add personal name transformation using a set of context free grammar rules to convert a non-standard name (e.g. nickname, word reordering or alternative names) into a uniform

representation. Finally, we use the Jaro-Winkler function to generate a set of candidate entity processes that can handling minor spelling errors.

To deal with lexical ambiguity, Bunescu and Pasca [8] use a bag of word and support vector machine (SVM^{light}) to deal with lexical ambiguity problem. The cosine similarity function is used to measure a similarity between the mentioned name and the related article in Wikipedia. To reduce the error in the bag of words model (Wikipedia article may be too short or incomplete or there may be a synonymous words or phrase problem) this means the correlation is too low between the mentioned name and the Wikipedia article because they do not have relevant words. The SVM kernel which models the magnitude of each word-category correlation based on Wikipedia taxonomy is used for disambiguation. The gap in context similarity is that it requires an exact word overlap between the two compared texts, which may become an over strict constraint due to natural language's usage flexibility [45].

Cucerzan [19] extends [8] and introduces a new technique to meet both identification and disambiguation. This work is the first to recognise the global document-level topical coherence of the entities. In an unambiguous process, a vector space model is used to compare similarity between the mentioned name and the related article in Wikipedia, and the topical coherence between the referent entity candidate and other entities within the same context is calculated based on their overlaps in categories and incoming links in Wikipedia. However, Dredze [24] and Shen [63] argue that the assumption that all entity mentions have the corresponding entities in the knowledge base is wrong because in all reality, some entities may not be stored in knowledge bases. Additionally, Shen [63] claims that the entity disambiguation techniques which are used in Bunescu and Pasca [8] and Cucerzan [19] are dependent on Wikipedia categories while the categories in Wikipedia are not clean and well-formed enough for ontological purposes although they are indeed arranged in a hierarchy.

Shen et al. [63] propose LINDEN for linking entity mentions in the document to the existing knowledge. Two techniques: "*topic coherence*" and "*context similarity*" are based on two knowledge bases: Wikipedia and YAGO are used for named entity disambiguation. The topical coherence is used under the assumption that "*a document largely refers to coherent entities or concepts from one or a few related topics*". Wikipedia-Miner is used to detect a Wikipedia concept in a document. The researcher proposed four steps for named entity disambiguation: (i) Constructing a semantic network among entities in a document by leveraging Wikipedia concepts and the taxonomy of concepts in YAGO. (ii) Measuring the strength of the semantic relation using semantic associativity and semantic similarity. (iii) Calculating global coherence from the average semantic associativity of each candidate

entity to the other mapping entities related to the mentions. (iv) Ranking candidate entities using a future vector and a weight vector. The experiment showed that four features including link probability, semantic association, semantic similarity and global coherence are useful in named entity disambiguation. However, context similarity measurement is used to measure the similarity between a mentioned context and the related context in a knowledge base [62].

Shen et al. [62] proposed a new technique called LIEGE ("Link the entities in web lists with the knowledge base") under the assumption that the entities appearing in a web list have the same concept. The two features: priority probability (popularity probability in each entity) and entity coherence are used to detect the concept similarity. The entity coherence is computed from type hierarchy based similarity that is derived from YAGO and distributional context similarity that evaluates the semantic similarity between entities based on the contexts where they occur in the Wikipedia article. They believe that the entities in each web list have the same type. However, this method still used context similarity measurement but from the external document. Furthermore, both LINDEN [63] and LIEGE [62] are derived type hierarchy from YAGO to measure similarity but Demidova et al. [22] argue that the YAGO ontology is dirty and some of the WordNet classes in YAGO are not associated with Wikipedia categories.

To fix the flaw in the Wikipedia categories and YAGO ontology, we designed a new architecture for creating occupation taxonomy that is based on web directories and YAGO ontology. The new occupation hierarchy architecture is used to generate a personal name concept. Then we used the personal name concept and our algorithm simple partial tree matching to evaluate the ambiguity entity. Our model uses only mentioned names from a web document as the input to measure a set of candidate entities in each mentioned name. As a result, we do not need to compare the relevant contexts between a web document and the related Wikipedia entity page.

Our system predicts the NIL value under two conditions: (i) The searcher cannot generate a candidate entity for a mentioned name. (ii) a linking entity has a different ancestor node and/or is distantly related to the existing identifying entities in a web document. Our method differs from studies [8, 24, 63]. Shen et al. [63] returned the NIL value to the mention text if the size of the candidate entity is equal to zero or below the threshold. Bunescu and Pasca [8] detected the NIL value by creating one entity calling 'out' to predict the NIL value when the similarity score was lower than a fixed threshold. Dredze et al. [24] used a SVM ranker to predict the NIL value. The NIL value is returned if a set of candidate entities is empty.

5.8 Conclusion

We introduce the PNELF framework for personal name entity linking. PNELF consists of three main components: personal name extractor, searcher and personal name disambiguator. The personal name extractor aims to extract a set of mentioned names within a web document. We use AlchemyAP, a text mining tools to extract a set of personal names from a web document.

The searcher component focuses on generating a set of candidate entities in each mentioned name. Personal names that appear on the web are often dirty and lack uniform and multiple representations (e.g. nickname, short name or alias name). So, using only the exact-match look up over the proper name dictionary that is used in [8, 19, 62] is inadequate for generating a set of candidate entities (e.g. *David Beckham vs. David Beckam*). We improve a searcher's performance by including two steps: PNTM and candidate generator. PNTM is used to transform personal name variations such as nicknames is to real names (e.g. *Bill* to *William*), an alternative name to real name (e.g. *DB7* to *David Beckham*) or to reorder a personal name pattern (e.g. *Beckham, David* to *David Beckham*). In the candidate generator steps we use Jaro-Winkler; a text similarity metric to match a set of personal names that are generated from PNTM over the personal name surface form. Jaro-Winkler is a character based similarity metric so it can solve the problem of typographic errors (*David Beckham vs. David Beckam*).

The personal disambiguator aims to select the best person from a set of candidate entities and link to the real-world entity or return the NIL value. In the lexical ambiguity problem, we introduce a new algorithm SPTM for ranking the candidate entity. SPTM is a two tree matching between the comparison tree and the candidate tree. The two trees are considered if the root nodes are equal. We use SPTM to deal with the context similarity problem. Due to SPTM, we do not need any context clues that are extracted from a web page to rank candidate entities. SPTM uses personal name concepts to create a tree under the assumption that people who are mentioned within a single web page have the same concept (the root node is equal e.g. they are an entertainer).

Predicting the NIL value, the NIL value is generated under three conditions:

1. PNTM cannot generate a standard name for a mentioned name ($x_i = \emptyset$).
2. Candidate generator cannot generate a set of candidate entities for a mentioned name ($P(x) = \emptyset$).
3. Disambiguator can link a mentioned name to a real-world entity but the answer person has a different concept from the existing people within a web page.

We evaluate the performance of PNELF using the real-world data sets which show that the accuracy in personal name entity linking is 91.82%, which is better than Cucerzan [19] and Bunescu and Pasca [8].

Chapter 6

Software Specification

6.1 Introduction

The purpose of the chapter is to describe in detail our personal name entity linking software. This chapter gives the following information for personal name entity linking: software overview, data flow and data design, user interface design and software testing.

6.1.1 Goals and Objectives

The main objective of our personal name matching is to identify and disambiguate personal names that are mentioned in a single web document to the real-world entity. Accordingly, the final product must be efficient in terms of precision for personal name linking and easy to use. The system could solve three problems in personal name disambiguation: name variations, personal name ambiguity and predicting the NIL value. Beyond these general design fundamentals, the following concrete functionalities are to be included in the system:

- Simple Partial Tree Matching (SPTM) algorithm for solving the personal name ambiguity.
- The Context Free Grammar (CFG) rules for handling the personal name variations in a mentioned document. The rules are processed over a personal name dictionary (a set of given names, family names, nicknames and alternative names) to generate a unique personal name format.
- Jaro-Winkler text similarity metric for generating a set of candidate entities in each mentioned name. The text similarity function is used over a personal surface form to generate a set of possible personal names for a mentioned.

- Predicting the NIL value function when a mentioned entity is absent from a knowledge base.

6.1.2 System Overview

The personal name matching system is a client-server architecture where a client-side application takes an input from a user and represents the results from the server. The sever-side application produces the output via multiple functions in the system. A system takes a single web page as an input. The system produces the identifiable person for each mentioned name in a web page or returns the NIL value if the mentioned name does not match any personal name in a knowledge base. For a NIL value, the system passes a mentioned name and the occupation that has a maximum number of co-occurrences from the identifiable personal names via BingAPI to produce the top ten links of possible people.

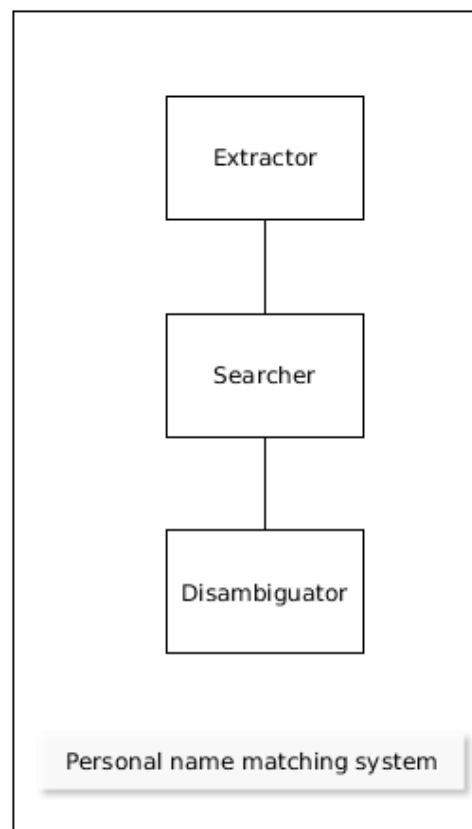


Fig. 6.1 Personal name matching screenshot

The system screenshot is shown in Figure 6.1. The system is developed using Apache2.4.12 (Ubuntu), PHP Version 5.6.11-1 Ubuntu 3.1 and MySQL 5.6.28-0 Ubuntu 0.15.10.1. The

personal name matching has one active actor and two cooperating system APIs. The user accesses the system through the Internet. There are two functions process through the two APIs: AlchemyAPI and BingAPI, as described in Chapter 5. The system components can be split up into three components: extractor, searcher and disambiguator, which we introduced in Chapter 5.

1. **Extractor.** This component is used to extract a set of mentioned names within a single web page. The extractor takes a single web page as an input and produces a set of mentioned names. The component consists of three main tasks:
 - User submits URL via our input form.
 - The input URL passes through the AlchemyAPI.
 - AlchemyAPI produces the output and sends it back to the system in the form of an XML document.
2. **Searcher.** This component is used to produce a set of candidate entities in each personal mentioned name. The component consists of three main tasks:
 - Transforming a set of mentioned names into a uniform representation using a sequence of context free grammar rules and a personal name dictionary.
 - Generating a set of candidate entities in each mentioned name by computing the similarity score between a set of standard names in each mentioned name and the personal name surface form using the Jaro-Winkler similarity function. A personal name which has a similarity score of equal or more than 0.97 can be a candidate entity. This score comes from our experimental results because we want to limit the number of candidate entities in terms of precision and recall. A pair of personal names that has one different letter is equal (e.g. *Tony Scott and Tony Scotti* are similar).
 - Generating the NIL value for a mentioned name that does not have the candidate entity.
3. **Disambiguator.** This component is used when a mentioned personal name is ambiguous (a mentioned name can refer to more than one candidate entity). The component consists of six main tasks:
 - Linking the mentioned name that has one candidate to the real-world entity.
 - Simple partial tree matching is used to calculate the similarity score in each ambiguous mentioned name.

- The candidate entity that has the highest similarity score is selected.
- Verifying the NIL value predicted in each identifiable mentioned name because our assumption is that the identifiable mentioned name should have the same ancestor. Therefore, the identifiable mentioned names that have a different root node will be wrongly identified and the NIL value is generated for this mentioned name.
- The system produces the top ten links for the possible people for a NIL value through BingAPI, as described in Chapter 5.

6.1.3 System Functions

A personal name matching system is a web application where a user will be able to submit a URL via the Internet. The outputs will be processed via three main components. The results in each web page will be linked if they match the real-world entity or returned NIL if they do not map to any person in the knowledge base. The system can predict the top ten links of possible people for a NIL value. The system function is explained using the use case diagram shown in Figure 6.2.

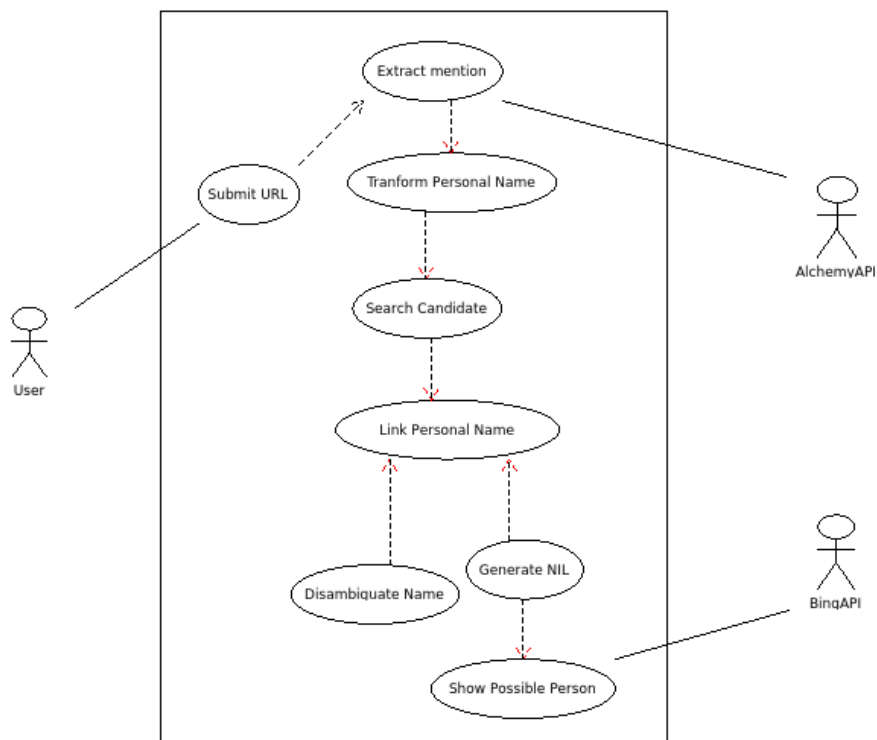


Fig. 6.2 Personal name matching use case diagram

We consider the following use cases:

1. Submit URL use case: the user submits a URL of the web page via the input form. After that, this URL will be processed through the system function.
2. Extract mention use case: The extractor will pass the URL to the AlchemyAPI. The API returns personal name extraction via an XML document. The extractor query a set of personal names from the XML document.
3. Transform personal name use case: The transformation evaluates and transforms set of personal names into a uniform representation using context free grammar rules and a personal name dictionary.
4. Search candidate use case: The personal names which have uniform representation in each mentioned name will be matched with the personal surface form using the Jaro-Winkler similarity function. A candidate entity will be generated if it has a similarity score more than 0.97. A set of candidate entities in each mention will be passed to the linking function.
5. Link personal name use case: The mention will be linked to the real-world person or return the NIL value if a mentioned name cannot map to anyone in the knowledge base.
6. Disambiguate name use case: this function will be process if each mentioned name has more than one candidate entities. A mentioned name refers to multiple people (lexical ambiguity). The SPTM is used for ranking a set of candidate entities. The candidate who has the highest similarity score will be selected.
7. Generate NIL use case: This function will be processed when the searcher cannot assign the candidate to each mention or the linked person has a root node that is different to the collection.
8. Show possible person use case: This function produces the top ten links of possible people for the NIL mentions using the BingAPI.

6.2 Data Design

We describe our data design including the data structures that are used to pass the data through the system and our database design.

6.2.1 Internal Data Structures

We use four types of internal data structure to communicate between components in our system, including HttpPost, XML and JSON format. Firstly, HttpPost is used to communicate between the client and the server. HttpPost protocol is always used to communicate between the client and the server (e.g. HTML input form). Secondly, the XML format is the communication between the extractor component and the AlchemyAPI. Thirdly, the NIL value predictor component and the BingAPI exchange data using JSON. JavaScript Object Notation (JSON) is a lightweight data-interchange format. Finally, the internal data structure is the MySQL database format which is used when we query the information from a data catalogue or personal name dictionary. The performance of the server will be decreased for a large amount of data. Therefore, the indexing and selecting only the data that has the same initial letter are used to reduce the workload in the database.

6.2.2 Database Description

In this section, we introduce our conceptual design using an Entity Relationship (ER) Diagram and our logical design (e.g. tables and keys (constraints)).

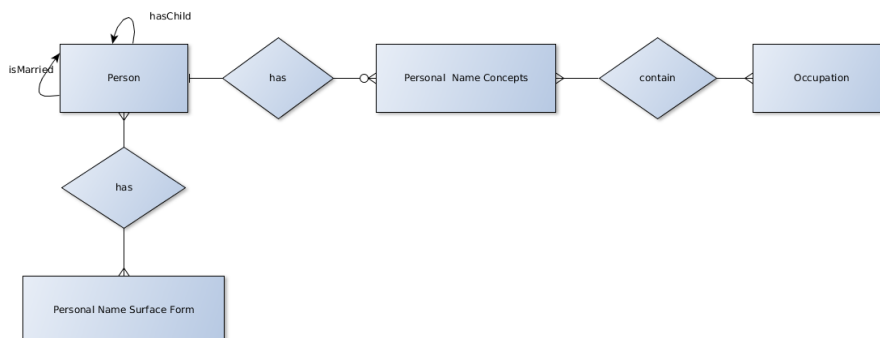


Fig. 6.3 Personal name matching entity relationship diagram

The ER diagram is illustrated in Figure 6.3. The ER diagram contains four entities: *Person*, *Personal Name Concepts*, *Occupation* and *Personal Name Surface Form* as described in the following.

1. *Person*: A person may not have personal name concept because he is not working. Each person has one or more personal surface forms. The entity *Person* has two recursive relationships including *hasChild* and *isMarried*.
2. *Personal Name Concepts*: Each personal name concept can belong to one person. Each personal name concept must have zero or multiple occupations.

3. *Personal Name Occupation*: Each occupation can belong to one or more personal name concepts.
4. *Personal Name Surface Form*: Each personal surface form can belong to one or more person.

The database contains five tables: *Entity_Data*, *EntityTree*, *Category*, *Entity_SurfaceForm* and *Entity_Relations* as described below.

1. *Entity_Data*(*ENid*, *ENname*) is a collection of personal names that are extracted from a knowledge base.
2. *EntityTree*(*ENid*, *root*, *Tree*) is a collection of personal name concept. A tree is a set of category IDs(*Cids*). For example, personal name concepts = (1,0,{0,4,41,44}) which means a personal name id 1 has a root node concept 0 and a set of categories 0,4,41,44.
3. *Category*(*Cid*, *Cname*, *Cparent*, *lft*, *rgt*) is a collection of occupations. *Cid* is a primary key, *Cname* is an occupation's name, *Cparent* is a parent category of this node and *lft* and *rgt* are calculated using the MPTT algorithm.
4. *Entity_SurfaceForm*(*Pname*, *ENids*) is a collection of terms that are used to refer to personal names. *Pname* is a term and *ENids* is a set of people who used this reference. For example, personal surface form = (Aaron Brown, {21, 22, 81754, 90459}) where Aaron Brown is *Pname* and {21, 22, 81754, 90459} is a set of *ENids*.
5. *Relations*(*ENid*, *relation*, *objID*) is a collection of personal name relationships including *hasChild* and *IsMarried*. *objID* is a set of *ENids*.

The database is used to identify and disambiguate personal names mentioned within a web document. Table *Entity_SurfaceForm* and *Entity_Data* are used to generate a set of candidate entities in each mentioned name in the searcher component. The four tables: *Entity_Data*, *EntityTree*, *Category* and *Relations* are used to disambiguate lexical ambiguity in the disambiguator component. The data that are used in the searcher component and the disambiguator component are separated.

6.3 Work Flow of the Process

The flowcharts show in Figure 6.4 and Figure 6.5 explain the sequence of steps and logic to handle our problems. The system starts by taking a URL as an input and then passing

multiple processes and the final result is personal name linking to the real-world entity or the NIL value. After that, the NIL value is processed for browsing possible people using the BingAPI. The system work flow is described below.

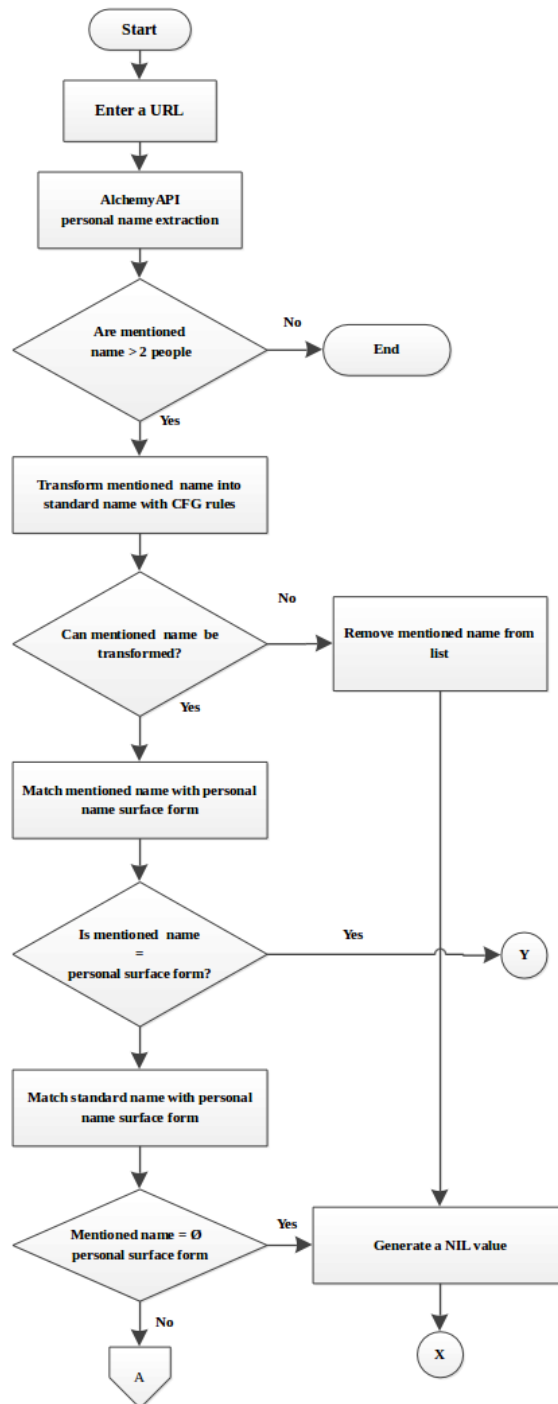


Fig. 6.4 Personal name matching flowchart

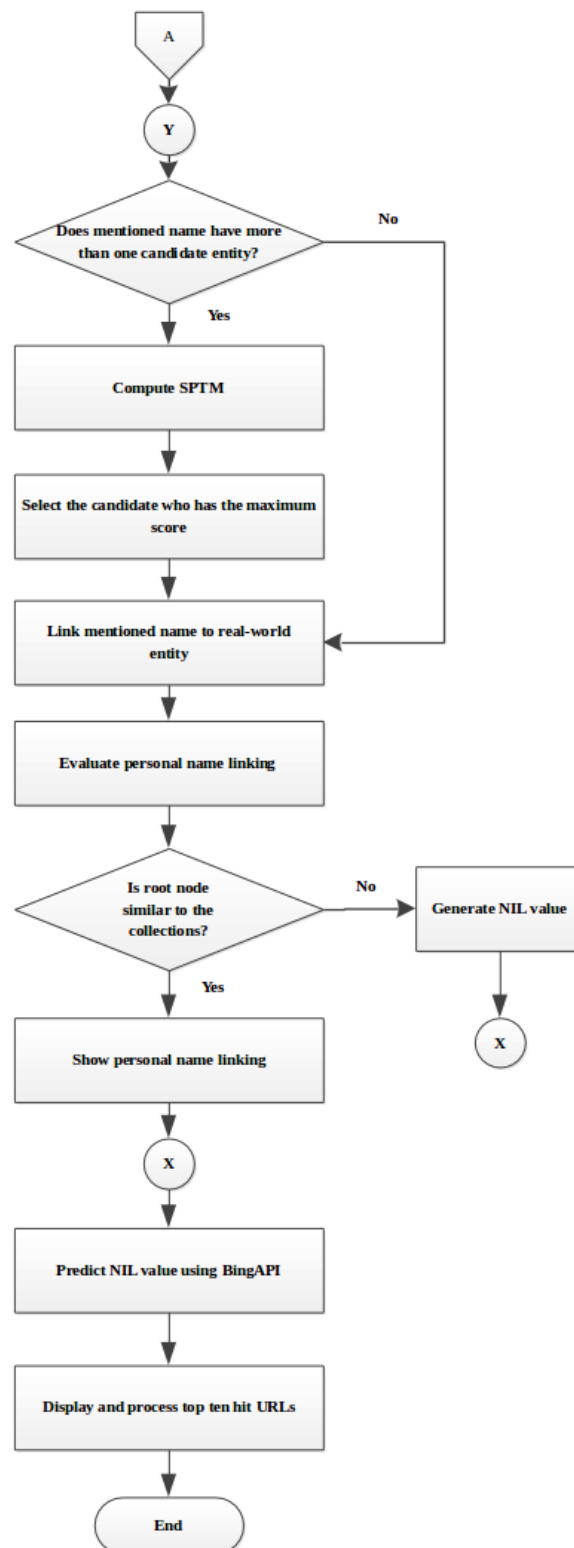


Fig. 6.5 Personal name matching flowchart

1. The process starts when a user submits a URL.
2. The extractor passes the URL to AlchemyAPI through the Internet. AlchemyAPI returns the personal name in XML format. The extractor queries the mentioned names in the XML document. If the extractor returns more than two people, the system goes to the personal name transformation process, but otherwise the process is finished.
3. A set of personal names will be transformed into a uniform format under CFG rules and a personal name dictionary. A set of transformed names are passed to generate a set of candidate entities. The unrecognised mentioned names go through the NIL value predicting process.
4. The searcher matches mentioned names over the personal name surface form under the Jaro-Winkler text similarity function. The process makes two decisions: 1) if a mentioned name matches a personal name surface form go to check the total number of candidate entities process 2) if it does not match personal surface form go to match standard name with a personal name surface form process.
5. The searcher matches a standard name over a personal name surface form by calculating the similarity score between a transformed name and a set of term collections in personal surface form using the Jaro-Winkler function. The candidate that has a similarity score of more than 0.97 is generated for each standard name. The similarity score of 0.97 is from our experimental results because our aim of generating a set of candidate entities is to balance between precision and recall. Therefore, the similarity score 0.97 is the effective point to allow a single letter error in a personal name to be a candidate entity.
6. After the candidate entities are assigned to the mentioned name, the generate candidate process makes two decisions: 1) if a mentioned name does not have a candidate entity going to generate the NIL value 2) if a mentioned name has a set of candidate entities go to check the total number of candidate entities process.
7. In counting the number of candidate entities in each mentioned name, the process makes two decisions: 1) if a candidate entity = 1, go to the entity linking process 2) if the candidate > 1 go to the compute SPTM process.
8. The SPTM process calculates the similarity score using the SPTM algorithm. The candidate who has the highest score is selected and goes to the entity linking process.

9. A set of linking entities are evaluated by considering the root node in each real-world entity. The system returns the NIL value to a mentioned name whose root node is different from the collection people. The system displays the final results.
10. The NIL mentions are processed to prediction possible people using BingAPI and displays the top ten links that are related to possible people.

6.4 User Interface Design

The User Interface (UI) contains the input form for a user to submit a URL and view the results through the Internet. The lists of UIs are described below.

1. Home page. The first time a user accesses the website, the submit form will appear and request the user to input the URL of the web page for personal name linking.

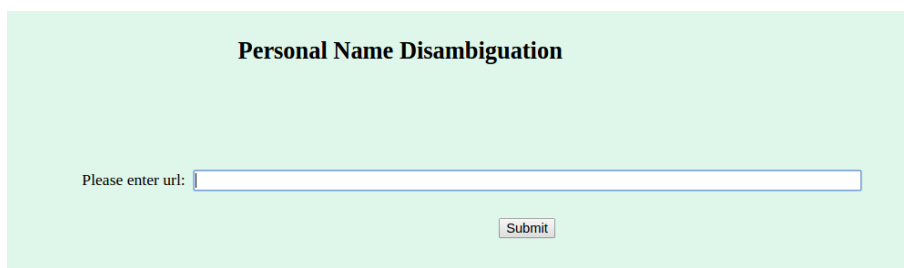


Fig. 6.6 Home page

2. The personal name transformation page will appear when a user submits the form. Figure 6.7 shows the mentions that are extracted from the URL:

http://www.today.com/id/16444023.

The values in the first column are personal names which are extracted from a web page and the values in the second column are the uniform format that are transformed using CFG rules and the personal name dictionary. For example, the mentioned name *Diaz* is transformed into *Cameron Diaz*.

Context Free Grammar Transformation Result

URL: <http://www.today.com/id/16444023>

Input Name	CFG Name
1. Timberlake	1.1 Craig Timberlake 1.2 Justin Timberlake
2. Diaz	2.1 Cameron Diaz
3. Veronica Finn	3.1 Veronica Finn
4. Britney Spears	4.1 Britney Spears 4.2 Sabrina Spears
5. Lou Pearlman	5.1 Lou Pearlman 5.2 Lucille Pearlman 5.3 Lucinda Pearlman 5.4 Louis Pearlman 5.5 Louise Pearlman
6. Star	6.1 Jeffree Star 6.2 Sunshine Dizon

[GenerateCandidate Entity](#)

Fig. 6.7 Personal name transformation page

- Candidate people will appear when user click *GenerateCandidateEntities*. A set of candidate entities in each mention are generated using the Jaro-Winkler text similarity function and the personal surface form. Figure 6.16 demonstrates a set of candidate people in each mention. For example, the mentioned name *Timberlake* has two candidate entities: *Justin Timberlake* and *Craig Timberlake*.

Generate Candidate Entity From Jaro-Winkler Score

URL: <http://www.today.com/id/16444023>

CFG Name	Surface Name	Entity Name	Jaro Winkler Similarity Score
1. EN Input: <i>Timberlake</i>			
1.1 Craig Timberlake	Craig Timberlake	Craig Timberlake	100.00
1.2 Justin Timberlake	Justin Tiberlake	Justin Timberlake	100.00
2. EN Input: <i>Diaz</i>			
2.1 Cameron Diaz	Cameron Diaz	Cameron Diaz	100.00
3. EN Input: <i>Veronica Finn</i>			
4. EN Input: <i>Britney Spears</i>			
4.1 Britney Spears	Birtney Spears	Britney Spears	100.00
5. EN Input: <i>Lou Pearlman</i>			
6. EN Input: <i>Star</i>			
6.1 Jeffree Star	Jeffree Star	Jeffree Star	100.00
6.2 Sunshine Dizon	Sunshine Dizon	Sunshine Dizon	100.00

[Simple Partial Tree Matching](#)

Fig. 6.8 Generating a set of candidate entity pages

- The personal name disambiguation page will appear when a user clicks *Simple Partial Tree Matching*. This page shows the results after solving the lexical ambiguity problem using SPTM algorithm and they are represented in Figure 6.9. For example, the system returns *Justin Timberlake* for the lexical ambiguity *Timberlake*.

Personal Name Matching Result		
URL: http://www.today.com/id/16444023		
Input Entity Name	Answer Entity Name	Category Tree
1. Britney Spears	Britney Spears	Entertainers and Artists->Creator->Performer->Communicator->producer->actor->dancer->musician->writer->film maker->singer->film director->actress->American film actors->American television actors->American stage actors->1990s singers->2000s singers->2010s singers->American child singers->American pop singers->American music video directors
2. Diaz	Cameron Diaz	Entertainers and Artists->Performer->Model->actor->American female models->American film actors->American voice actors
3. Star	Jeffree Star	Entertainers and Artists->Performer->Model->musician->singer->American male models->American male singers
4. Timberlake	Justin Timberlake	Entertainers and Artists->Performer->actor->dancer->musician->singer->American film actors->American child actors->American voice actors->1990s singers->2000s singers->2010s singers->American child singers->American male singers->American pop singers
5. Lou Pearlman	Unmatched	---
6. Veronica Finn	Unmatched	---

[Evaluation Null](#)

Fig. 6.9 The results of Simple Partial Tree Matching algorithm

- The final result page will appear when a user clicks *Evaluate NIL*. This page shows the final results of personal name linking by evaluating all of the collection entities that they have the same root node. The system assigns the NIL value to the linking entity if this entity has a different root node from the collection. Figure 6.10 illustrates the final results. The values in the first column are a set of personal name mentions. The values in the second column are identification names for each mention and we have two mentions including *Lou Pearlman* and *Veronica Finn* which are unlinked to any entity in our collection.

Personal Name Matching Result		
URL: http://www.today.com/id/16444023		
Input Entity Name	Answer Entity Name	Category Tree
1. Britney Spears	Britney Spears	Entertainers and Artists->Creator->Performer->Communicator->producer->actor->dancer->musician->writer->film maker->singer->film director->actress->American film actors->American television actors->American stage actors->1990s singers->2000s singers->2010s singers->American child singers->American pop singers->American music video directors
2. Diaz	Cameron Diaz	Entertainers and Artists->Performer->Model->actor->American female models->American film actors->American voice actors
3. Star	Jeffree Star	Entertainers and Artists->Performer->Model->musician->singer->American male models->American male singers
4. Timberlake	Justin Timberlake	Entertainers and Artists->Performer->actor->dancer->musician->singer->American film actors->American child actors->American voice actors->1990s singers->2000s singers->2010s singers->American child singers->American male singers->American pop singers
5. Lou Pearlman	Unmatched	---
6. Veronica Finn	Unmatched	---

[Home](#)

Fig. 6.10 Final results in personal name matching page

- The prediction NIL value page will appear when a user clicks the unlinking entity. This page will send the mentioned name and an occupation which has the maximum

number of co-occurrences in the identification of personal name concepts through BingAPI. The top ten links of possible people are shown in Figure 6.11 for the mentioned name *Veronica Finn* and as we know, *Veronica Finn* was *Justin Timberlake*'s first girlfriend.

1. [Justin Timberlake's first girlfriend Veronica Finn spills ...](#)
2. [Veronica Finn Photo Gallery - AllStarPics.Net](#)
3. [Heathers - Wikipedia, the free encyclopedia](#)
4. [Veronica Finn from Innosense spills the lipton on Justin ...](#)
5. [Justin Timberlake's first girlfriend reveals love letters ...](#)
6. [Justin Timberlake's First Girlfriend Reveals Love Letters](#)
7. [Justin Timberlake & Veronica Finn Dated, Joint Family Tree ...](#)
8. [Veronica Finn | Celebrities lists.](#)
9. [Justin Timberlakes first girlfriend reveals love letters ...](#)
10. [Learn and talk about Veronica Finn, American dance music ...](#)

Fig. 6.11 Top ten links of possible people

6.5 Testing Issues

The purpose of testing is to ensure that each component in the personal name matching system is produced using the precision outputs. Then, once the components are joined, the whole system is tested to ensure that all components work together correctly.

6.5.1 Software summary

Personal name matching is a client-server architecture that includes three main components: extractor, searcher and disambiguator. PHP and MySQL database are used to develop our software component. The system components are described in Table 6.1.

Table 6.1 Software Summary

Components	Use cases(#)	Components	
		System	Database
Extractor	<ul style="list-style-type: none"> • Extract mention 	<ul style="list-style-type: none"> • index.php • entity_extractor.php 	data.xml
Searcher	<ul style="list-style-type: none"> • Transform personal name • Search candidate 	<ul style="list-style-type: none"> • CFGmatched.php • AhoCFGmatched.php • PNPpattern.php • JW.php 	<ul style="list-style-type: none"> • Entity_Data • Entity_SurfaceForm
Disambiguator	<ul style="list-style-type: none"> • Link Personal name • Disambiguate name • Generate NIL value • Show possible person 	<ul style="list-style-type: none"> • TreeMatching.php • FindENrelation.php • EndEvaluation.php • Bing.php 	<ul style="list-style-type: none"> • Entity_Data • EntityTree • Category • Relations

6.5.2 System Testing

The system testing starts with three components: an extractor, searcher and disambiguator are integrated to ensure that the system still works properly. We focus on the precision output values in the extractor, searcher and disambiguator. The system could return either personal name linking or the NIL value in each mentioned name. The system is finished if it returns the correct values in every component.

Item to be tested

- The extractor generates correct mentioned names and compares with the XML document that is returned from AlchemyAPI.
 - PNTM transforms a mentioned name into a proper representation format.
 - The searcher generates a set of candidate entities in each mentioned name correctly.
 - The disambiguator calculates and ranks a set of candidate entities in each mentioned name correctly bases on SPTM algorithm.
1. Extractor component: This process connects to AlchemyAPI, an external system for name entity extraction. The extractor processed a URL input and provides a list of personal names to the searcher component.

Test Case ID: UC1.

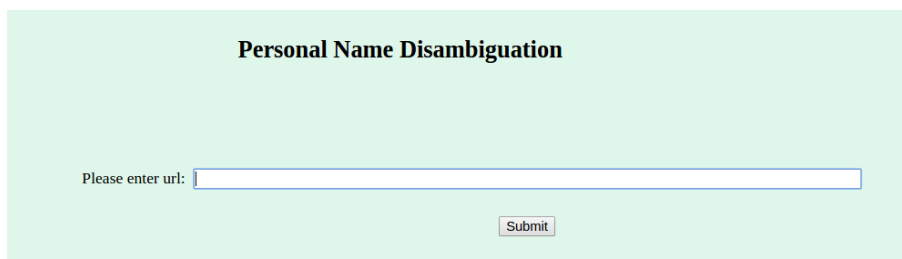
Test name: Extract mentioned name.

Description: To guarantee that the system lists the correct names that are returned from AlchemyAPI.

Prerequisites: NO.

Tests inputs: URL.

Tests outputs: A list of personal names shown in a web browser.



Personal Name Disambiguation

Please enter url:

Submit

Fig. 6.12 The URL input form



Fig. 6.13 A set of mentioned names within a web page

Figure 6.12 shows a URL input form and Figure 6.13 shows a set of mentioned names that are extracted from a web page.

Test procedures:

Step number	Description	Expected Results
1	Open application in a browser	Homepage and input form shown correctly. This can be seen in Figure 6.12
2	Input URL	The personal names that appeared in data.xml are shown in a web browser. This can be seen in Figure 6.13

2. The searcher component is separated into two tasks: PNTM and candidate generator. PNTM is used to generate a uniform personal name representation and a candidate generator provides a set of candidate entities in each mentioned name.

Test Case ID: UC2.

Test name: Personal name transforming.

Description: To guarantee that PNTM returns a set of correct uniform representations in each mentioned personal name.

Prerequisites: A set of personal names that are extracted from a web. page

Tests inputs: A set of personal names

Tests outputs: A list of uniform representation formats and groups by mentioned name are shown in a web browser.

Context Free Grammar Transformation Result

URL: <http://www.today.com/id/16444023>

Input Name	CFG Name
1. Mr. David Robert Joseph Beckham, OBE	1.1 David Robert Joseph Beckham
2. Mr. David Beckham	2.1 David Beckham
3. Mr. Beckham, David	3.1 David Beckham
4. Mr. Beckham, D.	4.1 D. Beckham
5. Mr. DB7	5.1 David Beckham
6. Mr. Dave Beckham	6.1 Dave Beckham 6.2 David Beckham

[GenerateCandidate Entity](#)

Fig. 6.14 A set of mentioned names with prefix and/or suffix transformation under PNTM

Context Free Grammar Transformation Result

URL: <http://www.today.com/id/16444023>

Input Name	CFG Name
1. David Robert Joseph Beckham	1.1 David Robert Joseph Beckham
2. David Beckham	2.1 David Beckham
3. Beckham, David	3.1 David Beckham
4. Beckham, D.	4.1 D. Beckham
5. DB7	5.1 David Beckham
6. Dave Beckham	6.1 Dave Beckham 6.2 David Beckham

[GenerateCandidate Entity](#)

Fig. 6.15 A set of mentioned names without prefix and/or suffix transformation under PNTM

Figure 6.14 shows a list mentioned with a prefix and/or suffix and Figure 6.15 shows a list mentioned without a prefix and/or suffix. The test case aims to evaluate the performance of PNTM tasks in transforming name variations (e.g. rearrangement of name: *David Beckham* vs. *Beckham, David*, nickname: *Dave* vs. *David* or alias name: *DB7* vs. *David*). The PNTM provides a good performance if it returns a correct output.

Test procedures:

Step number	Description	Expected Results
1	Input a list of personal names with titles. For example, {Mr. David Robert Joseph Beckham, OBE, Mr. David Beckham, Mr. Beckham, David, Mr. Beckham, D., Mr. DB7, Mr. Dave Beckham}	The list of personal names is shown correctly in a web browser and their titles are removed. The output values are: {David Robert Joseph Beckham, OBE = David Robert Joseph Beckham, Mr. David Beckham = David Beckham, Mr. Beckham, David = David Beckham, Mr. Beckham, D. = D. Beckham, Mr. DB7 = David Beckham, Mr. Dave Beckham = Dave Beckham and David Beckham}. The results are shown in Figure 6.14.
2	Repeat this step but change the input values: personal name without a title. For example, {David Robert Joseph Beckham, David Beckham, Beckham, David, Beckham, D., DB7, Dave Beckham}	The list of personal name are shown correctly in a web browser. The output values are: {David Robert Joseph Beckham = David Robert Joseph Beckham, David Beckham = David Beckham, Beckham, David = David Beckham, Beckham, D. = D. Beckham, DB7 = David Beckham, Dave Beckham = Dave Beckham and David Beckham}. The results are shown in Figure 6.15.

Test Case ID: UC3.Test name: Search candidate entity.Description: To guarantee that the system returns a set of correct candidate entities in each mentioned personal name.

Prerequisites: A set of standard personal names that are transformed from the personal name transformation component.

Tests inputs: A set of standard personal names.

Tests outputs: A list of candidate entities and groups by mentioned name are shown in a web browser.

Generate Candidate Entity From Jaro-Winkler Score

URL: <http://www.today.com/id/16444023>

CFG Name	Surface Name	Entity Name	Jaro Winkler Similarity Score
1. EN Input: Timberlake			
1.1 Justin Timberlake	Justin Tiberlake	Justin Timberlake	100.00
1.2 Craig Timberlake	Craig Timberlake	Craig Timberlake	100.00
2. EN Input: Diaz			
2.1 Cameron Diaz	Cameron Diaz	Cameron Diaz	100.00
3. EN Input: Veronica Finn			
4. EN Input: Britney Spears			
4.1 Britney Spears	Britney Spears	Britney Spears	100.00
5. EN Input: Brad Pit			
5.1 Brad Pit	Brad Pit	Brad Pitt	100.00
6. EN Input: Tom Cruise			
6.1 Tom Cruise	Tom Cruise	Thomas Cruise (footballer)	100.00
6.2 Tom Cruise	Tom Cruise	Tom Cruise	100.00

[Simple Partial Tree Matching](#)

Fig. 6.16 A set of candidate entities that are generated from the searcher component

Figure 6.16 shows a set of candidate entities. It can be seen that the Jaro-Winkler text similarity metric can solve the problem of misspelling (e.g. *Brad Pit* vs. *Brad Pitt*).

Test procedures:

Step number	Description	Expected Results
1	Input a list of standard personal names. For example, {Timberlake = Craig Timberlake and Justin Timberlake, Diaz = Cameron Diaz, Veronica Finn = Veronica Finn, Britney Spears = Britney Spears, Brad Pit = Brad Pit and Bradford Pit, Tom Cruise = Tom Cruise and Thomas Cruise}	The list of candidate entities are shown correctly in a web browser. A set of candidate entities is: {Timberlake = Craig Timberlake and Justin Timberlake, Diaz = Cameron Diaz, Veronica Finn = \emptyset , Britney Spears = Britney Spears, Brad Pit = Brad Pitt, Tom Cruise = Thomas Cruise (footballer) and Tom Cruise}. A set of candidate entities is shown in Figure 6.16.

3. Disabiguator component. There are three tasks: link a mentioned name to a real-world entity, disambiguate ambiguity name and predict NIL value for a mentioned name that is not represented in the knowledge base.

Test Case ID: UC4.

Test name: Comparison tree.

Description: To guarantee that the system generate the correct comparison tree. The comparison tree is generated from the personal name concepts where a mentioned name has a single candidate entity and the root node is equal. The comparison tree with the maximum number of nodes is considered first.

Prerequisites: A set of candidate entities that are generated from the searcher component.

Tests inputs: A set of candidate entities.

Tests outputs: Comparison tree.

Comparison Trees	
Personal Names	Personal Conceptual Trees
I.Brad Pitt,Britney Spears,Cameron Diaz	Entertainers and Artists->Creator->Performer->Model->Communicator->producer->actor->dancer->musician->writer->film maker->singer->American female models->film director->American film producers->actress->American film actors->American television actors->American stage actors->American voice actors->1990s singers->2000s singers->2010s singers->American child singers->American pop singers->American music video directors

Fig. 6.17 A comparison tree

Figure 6.17 shows a comparison tree that is created from three personal name concepts where a mentioned name has a single candidate including *Cameron Diaz*, *Britney Spears* and *Brad Pitt*.

Test procedures:

Step number	Description	Expected Results
1	Input a list of candidate entities in each mentioned name. For example, {Timberlake = Craig Timberlake and Justin Timberlake, Diaz = Cameron Diaz, Veronica Finn = \emptyset , Britney Spears = Britney Spears, Brad Pit = Brad Pitt, Tom Cruise = Thomas Cruise (footballer) and Tom Cruise}	A set of single candidate entities and comarison trees are shown correctly in a web browser. The output values are: {Cameron Diaz, Britney Spears, Brad Pitt and comparison tree: <i>Entertainers and Artists</i> – > <i>Creator</i> – > <i>Performer</i> – > <i>Model</i> – > ... – > <i>Americanmusicvideodirectors</i> }. The comparison tree is shown in Figure 6.17.

Test Case ID: UC5.

Test name: Link Personal Name.

Description: To guarantee that the system returns a correct answer in each mentioned personal name. The system returns either a linking mentioned name to a real-world entity or represents the NIL value.

Prerequisites: A set of candidate entities that are generated from the searcher component.

Tests inputs: A set of candidate entities.

Tests outputs: Entity linking or NIL value in each mentioned name.

Personal Name Matching Result

URL: <http://www.today.com/id/16444023>

Input Entity Name	Answer Entity Name	Category Tree
1. Brad Pitt	Brad Pitt	1. Entertainers and Artists->Creator->Performer->producer->actor->film maker->American film producers->American film actors->American television actors
2. Britney Spears	Britney Spears	1. Entertainers and Artists->Creator->Performer->Communicator->producer->actor->dancer->musician->writer->film maker->singer->film director->actress->American film actors->American television actors->American stage actors->1990s singers->2000s singers->2010s singers->American child singers->American pop singers->American music video directors
3. Diaz	Cameron Diaz	1. Entertainers and Artists->Performer->Model->actor->American female models->American film actors->American voice actors
4. Timberlake	Justin Timberlake	1. Entertainers and Artists->Performer->actor->dancer->musician->singer->American film actors->American child actors->American voice actors->1990s singers->2000s singers->2010s singers->American child singers->American male singers->American pop singers
5. Tom Cruise	Tom Cruise	1. Entertainers and Artists->Creator->Performer->producer->actor->film maker->American film producers->American film actors
6. Veronica Finn	Unmatched	---

[Home](#)

Fig. 6.18 Personal name entity linking

Figure 6.18 shows the output in personal name entity linking. The system can return either the linking or the NIL value in each mentioned name. Furthermore, the disambiguator ranks a set of candidate entities in each mentioned name correctly and the highest similarity score is selected.

Test procedures:

Step number	Description	Expected Results
1	Input a list of candidate entities in each mentioned name. For example, {Timberlake = Craig Timberlake, Justin Timberlake, Diaz = Cameron Diaz, Tom Cruise = Tom Cruise, Thomas Cruise, Veronica Finn: \emptyset }	The list of candidate entities is shown correctly in a web browser. The output values are {Timberlake = Justin Timberlake, Diaz = Cameron Diaz, Tom Cruise = Tom Cruise, Veronica Finn = \emptyset . The results of personal name entity linking are shown in Figure 6.18.}

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Personal Named Entity Linking is the task of linking the personal named entity that is mentioned within a web page to the corresponding person in a knowledge base. Personal Named Entity Linking can be used as input in many areas, such as search engines, information retrieval, information extraction, named-entity recognition and machine translation. The aim of this thesis is to design a framework and a new algorithm that focuses on two challenges in personal name entity linking:

1. Improving searcher performance. The previous searcher component [8, 19] is based on matching the exact term over personal and surface forms. This method may provide an incorrect result in personal name matching. This is because personal names have various representations (e.g. nickname: *Dave* vs. *David*; shortened name: *Beckham* vs. *David Beckham*; alternative name: *DB7* vs. *David Beckham*; order rearrangement *Beckham, David* vs. *David Beckham*; multiple spellings: *Catherine* vs. *Katherine*; and typographical errors: *Beckham* vs. *Beckam*).
2. Improving disambiguator performance. This thesis focuses on two main problems in the disambiguator component which are usually used for ranking a set of candidate entities when there is lexical ambiguity.
 - Context similarity that requires exactly word overlap between the two compared texts to evaluate whether or not the two people are the same. Context similarity may become an over-strict constraint because natural language is flexible.
 - Topical coherence based on Wikipedia cross-page links or YAGO taxonomy for finding the unity between two persons. However, the category in Wikipedia is

a thematic domain; it is less semantic and not well-formed enough to represent the correct concepts for each person. The taxonomy in YAGO that derives from WordNet and Wikipedia is well-formed and semantically accurate. However, we found that some classes in the YAGO taxonomy use an incorrect correspondence between a class in WordNet and a category in Wikipedia. Therefore, it may reduce the accuracy in candidate ranking.

To solve the problems above, we introduce PNELF, a new framework for personal name entity linking. PNELF consists of three main components: Personal name extractor, Searcher and Personal name disambiguator.

We use Alchemy API for extracting a set of names mentioned within a web document. The searcher component works to generate a set of candidate entities for a mentioned name. To improve the performance in searcher, the thesis has introduced PNTM and Jaro-Winkler text similarity metric to handle the name variation problem. PNTM aims to transform multiple name patterns to a uniform representation. PNTM consists of three modules: CFG rule, predicate and action.

- CFG rule or transformations rule is similar to a standard CFG rule, including head and body. The rules define how an input can be transformed to an output.
- Predicate is an external database consisting of prefix and suffix dictionary, personal name dictionary and a set of personal name patterns.
- Action is a set of functions we use to produce the output under CFG rules and predicate.

PNTM with Jaro-Winkler returns the highest proportion of correct results in personal name matching because it can solve the personal name variation problems as follows:

1. PNTM can handle two problems in personal name matching: the rearrangement of words (e.g. *Beckham, David* vs. *David Beckham*) and the use of different words such as alternative name or nickname to refer to the same person (e.g. *DB7* vs. *David Beckham* or *Dave* vs. *David*).
2. Jaro-Winkler uses character-based text similarity matching, so it can solve the problem of multiple spellings (e.g. *Catherine* vs. *Katherine*) and typographical errors (e.g. *Beckham* vs. *Beckam*).

Experimental results using a large number of web documents show that the new searcher is effective at detecting candidate entities, both in terms of precision and recall.

In the personal name disambiguator component, we propose a new occupation taxonomy: OAPnDis, and new algorithm in personal name disambiguator: SPTM. The design of OAPnDis architecture is based on YAGO taxonomy and the web directory includes four layers:

1. Layer 0: An identifying layer to define that the instance under this layer is human.
2. Layer 1: An overview classifying layer that is derived from web directories.
3. Layer 2: A YAGO-WordNet layer that is derived from YAGO taxonomy.
4. Layer 4: An YAGO-Wikipedia layer that is derived from YAGO taxonomy.

Based on OAPnDis, we can construct a new occupation taxonomy that is clean, well-formed and semantically accurate for generating personal name concepts in each person.

SPTM is a two-tree matching algorithm that compares a comparison tree and a candidate tree to rank a set of candidate entities when lexical ambiguity occurs. The comparison tree is created from a set of personal name concepts in each mentioned name which has a single candidate entity and their root nodes are similar. Candidate tree is the name given to the personal name concepts for each person who share the same personal name surface form. SPTM matches each candidate tree with the comparison tree and the candidate entity who has the highest similarity score is considered and linked to the real-world entity. We assume that persons who are represented within a web document are coherent and that their concepts are similar (i.e. layer 1 or root node is equal). Therefore, context similarity is discarded from PNELF. Experimental results over the real word data show that PNELF provides a higher rate of accuracy in personal name linking than previous work. SPTM and personal name concepts can disambiguate lexical ambiguity.

7.2 Future Work

The empirical results demonstrate that PNELF is effective in personal name linking. However, the effectiveness of short names linking is down. This result indicates that only personal name concept feature is insufficient to identify a short name when a lexical ambiguity problem occurs. Therefore, in the future work we would like to improve the performance of short name ambiguity. We plan to find out a useful evidence across the web documents to calculate the correlation between two people. This information is useful to identify an absent entity or a short name ambiguity.

References

- [1] Arasu, A., Chaudhuri, S., and Kaushik, R. (2008). Transformation-based framework for record matching. In *2008 IEEE 24th International Conference on Data Engineering*, pages 40–49.
- [2] Arasu, A., Götz, M., and Kaushik, R. (2010). On active learning of record matching packages. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 783–794.
- [3] Arasu, A. and Kaushik, R. (2009). A grammar-based entity representation framework for data cleaning. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, pages 233–244.
- [4] Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2670–2676.
- [5] Banko, M. and Etzioni, O. (2008). The tradeoffs between open and traditional relation extraction. In *Proceedings of ACL-08: HLT*, pages 28–36.
- [6] Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., and Fienberg, S. (2003). Adaptive name matching in information integration. *Intelligent Systems, IEEE*, 18(5):16–23.
- [7] Bollegala, D., Matsuo, Y., and Ishizuka, M. (2011). Automatic discovery of personal name aliases from the web. *Knowledge and Data Engineering, IEEE Transactions on*, 23(6):831–844.
- [8] Bunescu, R. and Pasca, M. (2006). Using encyclopedic knowledge for named entity disambiguation. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06), Trento, Italy*, pages 9–16.
- [9] Bureau, U. C. (2014). Genealogy. http://www.census.gov/topics/population/genealogy/data/1990_census/1990_census_namefiles.html/. [Online; accessed 10-Sep-2014].
- [10] Cafarella, M. J., Halevy, A., and Khoussainova, N. (2009). Data integration for the relational web. *Proc. VLDB Endow.*, 2(1):1090–1101.
- [11] Cafarella, M. J., Halevy, A., and Madhavan, J. (2011). Structured data on the web. *Commun. ACM*, 54(2):72–79.
- [12] CensusDiggins (2010). Gen tips. <http://www.censusdiggins.com/nicknames.htm>. [Online; accessed 10-Aug-2012].

- [13] Chanda, J., Sengupta, S., Kanjilal, A., and Bhattacharya, S. (2010). Formalization of the design phase of software lifecycle: A grammar based approach. In *Proceedings of the International Workshop on Formalization of Modeling Languages*, pages 4:1–4:5.
- [14] Chen, Z., Wenyin, L., and Zhang, F. (2002). A new statistical approach to personal name extraction. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 67–74.
- [15] Christen, P. (2006). A comparison of personal name matching: Techniques and practical issues. In *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*, pages 290–294.
- [16] Christen, P., Churches, T., and Zhu, J. X. (2002). Probabilistic name and address cleaning and standardisation. In *Australasian Data Mining Workshop*.
- [17] Churches, T., Christen, P., Lim, K., and Zhu, J. X. (2002). Preparation of name and address data for record linkage using hidden markov models. *BMC Medical Informatics and Decision Making*, 2(1):1–16.
- [18] Cohen, W. W., Ravikumar, P., and Fienberg, S. E. (2003). A comparison of string distance metrics for name-matching tasks. In *Proceedings of International Joint Conference on Artificial Intelligence-03 Workshop on Information Integration*, pages 73–78.
- [19] Cucerzan, S. (2007). Large-scale named entity disambiguation based on Wikipedia data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 708–716.
- [20] DBpedia (2015). Dbpedia. <http://wiki.dbpedia.org/>. [Online; accessed 02-Oct-2015].
- [21] De Melo, G., Suchanek, F., and Pease, A. (2008). Integrating yago into the suggested upper merged ontology. In *Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on*, volume 1, pages 190–193.
- [22] Demidova, E., Oelze, I., and Nejd, W. (2013). Aligning freebase with the yago ontology. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, pages 579–588.
- [23] DMOZ (2016). Dmoz. <https://www.dmoz.org/>. [Online; accessed 02-Jan-2013].
- [24] Dredze, M., McNamee, P., Rao, D., Gerber, A., and Finin, T. (2010). Entity disambiguation for knowledge base population. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 277–285.
- [25] Elmagarmid, A., Ipeirotis, P., and Verykios, V. (2007). Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16.
- [26] Factforge (2015). Factforge fast track to the center of the data web. <http://factforge.net/>. [Online; accessed 02-Oct-2015].
- [27] Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545.

- [28] Ferrara, E., Meo, P. D., Fiumara, G., and Baumgartner, R. (2014). Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, 70(0):301 – 323.
- [29] Franklin (2009). Nicknames and naming traditions. <http://www.tngenweb.org/franklin/frannick.htm>. [Online; accessed 10-Aug-2012].
- [30] Freebase (2015). A community-curated database of well-known people, places, and things. <https://www.freebase.com/>. [Online; accessed 02-Oct-2015].
- [31] Google (2013). What did the world search for 2013? <https://www.google.com/trends/topcharts#geo&date=2013>. [Online; accessed 14-August-2014].
- [32] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220.
- [33] Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928.
- [34] Hachey, B., Radford, W., Nothman, J., Honnibal, M., and Curran, J. R. (2013). Evaluating entity linking with wikipedia. *Artificial Intelligence*, 194(0):130 – 150.
- [35] Han and Kamber (2006). *Data mining : concepts and techniques / by Jiawei Han and Micheline Kamber*. Morgan Kaufmann.
- [36] Han, X., Sun, L., and Zhao, J. (2011). Collective entity linking in web text: A graph-based method. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 765–774.
- [37] Han, X. and Zhao, J. (2009). Web personal name disambiguation based on reference entity tables mined from the web. In *Proceedings of the Eleventh International Workshop on Web Information and Data Management*, pages 75–82.
- [38] Hasan, K. M. A., Mondal, A., and Saha, A. (2010). A context free grammar and its predictive parser for bangla grammar recognition. In *Computer and Information Technology (ICCIT), 2010 13th International Conference on*, pages 87–91.
- [39] Hillyer, M. (2012). Managing hierarchical data in mysql. <http://mikehillyer.com/articles/managing-hierarchical-data-in-mysql/>. [Online; accessed 02-Jan-2013].
- [40] Hoffart, J., Suchanek, F. M., Berberich, K., and Weikum, G. (2013). Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194(0):28 – 61.
- [41] Hu, Y., Wang, Z., Wu, W., Guo, J., and Zhang, M. (2010). Recommendation for movies and stars using yago and imdb. In *Web Conference (APWEB), 2010 12th International Asia-Pacific*, pages 123–129.
- [42] Kasneci, G., Ramanath, M., Suchanek, F., and Weikum, G. (2009). The yago-naga approach to knowledge discovery. *SIGMOD Rec.*, 37(4):41–47.
- [43] Kim, Y., Park, J., Kim, T., and Choi, J. (2007). Web information extraction by html tree edit distance matching. In *Convergence Information Technology, 2007. International Conference on*, pages 2455–2460.

- [44] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., et al. (2015). Dbpedia—a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195.
- [45] Li, Y., Wang, C., Han, F., Han, J., Roth, D., and Yan, X. (2013). Mining evidences for named entity disambiguation. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1070–1078.
- [46] Limaye, G., Sarawagi, S., and Chakrabarti, S. (2010). Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, 3(1-2):1338–1347.
- [47] Liu, B. (2011). *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data / by Bing Liu. [eBook]*. Data-Centric Systems and Applications. Springer Berlin Heidelberg.
- [48] Mann, G. S. and Yarowsky, D. (2003). Unsupervised personal name disambiguation. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, pages 33–40.
- [49] Manning, Raghavan, and Schütze (2008). *Introduction to information retrieval / by Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze*. CUP.
- [50] Murphy, C. and Purdum, T. S. (2009). Farewell to all that: An oral history of the bush white house. <http://www.vanityfair.com/news/2009/02/bush-oral-history200902/>. [Online; accessed 14-Sep-2015].
- [51] Nickel, M., Tresp, V., and Kriegel, H.-P. (2012). Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280.
- [52] Nijholt (1980). *Context free grammar: covers, normal forms and parsing / by A. Nijholt*. Lecture notes in computer science: 93. Springer.
- [53] of Sheffield, T. U. (1995). Gate: a full-lifecycle open source solution for text processing. <https://gate.ac.uk/overview.html>. [Online; accessed 10-Jul-2012].
- [54] Peng, T., Li, L., and Kennedy, J. (2014). A comparison of techniques for name matching. *GSTF Journal on Computing (JoC)*, 2(1).
- [55] Rahm, E. and Do, H.-H. (2000). Data cleaning: Problems and current approaches. *IEEE Bulletin of the Technical Committee on Data Engineering*, 23(4):3–13.
- [56] Raman, V. and Hellerstein, J. M. (2001). Potter’s wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 381–390.
- [57] Ren, A., Du, X., and Wang, P. (2009). Ontology-based categorization of web search results using yago. In *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, volume 1, pages 800–804.
- [58] Science, O. D. U. W. and Group, D. L. R. (2014). nickname-and-diminutive-names-lookup. github.com/carltonnorthern/nickname-and-diminutive-names-lookup. [Online; accessed 10-Aug-2014].

- [59] Sen, P. (2012). Collective context-aware topic models for entity disambiguation. In *Proceedings of the 21st International Conference on World Wide Web*, pages 729–738.
- [60] Shaikh, M., Memon, N., and Wiil, U. (2011). Extended approximate string matching algorithms to detect name aliases. In *Intelligence and Security Informatics (ISI), 2011 IEEE International Conference on*, pages 216–219.
- [61] Shen, D., Walkery, T., Zhengy, Z., Yangz, Q., and Li, Y. (2008). Personal name classification in web queries. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 149–158.
- [62] Shen, W., Wang, J., Luo, P., and Wang, M. (2012a). Liege:: Link entities in web lists with knowledge base. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1424–1432.
- [63] Shen, W., Wang, J., Luo, P., and Wang, M. (2012b). Linden: Linking named entities with knowledge base via semantic knowledge. In *Proceedings of the 21st International Conference on World Wide Web*, pages 449–458.
- [64] Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, pages 697–706.
- [65] Suchanek, F. M., Kasneci, G., and Weikum, G. (2008). Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203 – 217.
- [66] Sultana, A., Hasan, Q. M., Biswas, A. K., Das, S., Rahman, H., Ding, C., and Li, C. (2012). Infobox suggestion for wikipedia entities. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pages 2307–2310.
- [67] Tse, G. (2004). A grammatical study of personal names in present-day english: With special reference to the usage of the definite article. *English Studies*, 85(3):241–259.
- [68] Tulder, G. V. (2003). Storing hierarchical data in a database. <https://www.sitepoint.com/hierarchical-data-database/>. [Online; accessed 02-Jan-2013].
- [69] University, P. (2015). Wordnet, a lexical database of english. <https://wordnet.princeton.edu/>. [Online; accessed 02-Oct-2015].
- [70] Watson, I. (2012). Alchemyapi. <http://www.alchemyapi.com/>. [Online; accessed 10-Jul-2012].
- [71] Wikipedia (2015). George bush. https://en.wikipedia.org/wiki/George_Bush/. [Online; accessed 14-Sep-2015].
- [72] Wu, F. and Weld, D. S. (2010). Open information extraction using wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 118–127.

- [73] Xu, H., AbdelRahman, S., Lu, Y., Denny, J. C., and Doan, S. (2011). Applying semantic-based probabilistic context-free grammar to medical language processing - a preliminary study on parsing medication sentences. *Journal of Biomedical Informatics*, 44(6):1068 – 1075.
- [74] Yancey, W. E. (2005). Evaluating String Comparator Performance for Record Linkage. Technical report, Statistical Research Division, U.S. Census Bureau.
- [75] Yang, W. (1991). Identifying syntactic differences between two programs. *Softw. Pract. Exper.*, 21(7):739–755.
- [76] Yerva, S. R., Miklas, Z., and Aberer, K. (2012). Quality-aware similarity assessment for entity matching in web data. *Information Systems*, 37(4):336 – 351.
- [77] Yin, X., Tan, W., Li, X., and Tu, Y.-C. (2010). Automatic extraction of clickable structured web contents for name entity queries. In *Proceedings of the 19th International Conference on World Wide Web*, pages 991–1000.
- [78] Yin, X., Tan, W., and Liu, C. (2011). Facto: A fact lookup engine based on web tables. In *Proceedings of the 20th International Conference on World Wide Web*, pages 507–516.
- [79] Zhai, Y. and Liu, B. (2005). Web data extraction based on partial tree alignment. In *Proceedings of the 14th International Conference on World Wide Web*, pages 76–85.
- [80] Zhai, Y. and Liu, B. (2006). Automatic wrapper generation using tree matching and partial tree alignment. In *proceedings of the 21st national conference on Artificial intelligence-Volume 2*, pages 1687–1690.
- [81] Zhai, Y. and Liu, B. (Dec.). Structured data extraction from the web based on partial tree alignment. *Knowledge and Data Engineering, IEEE Transactions on*, 18(12):1614–1628.

Appendix A

Abbreviation Words Glossary

CFG	Context Free Grammar
DEPTA	Data Extraction based on Partial Tree Alignment
ER	Entity Resolution
LIEGE	Link the entITies in wEb lists with the knowledGe basE
LP	Link probability
LWTM	Layered and Weighted Tree Matching
LHS	LEFT Hand Side
NED	Named Entity Disambiguation
MPTT	Modified Preorder Tree Traversal
OAPnDis	Occupation Architecture for Personal Name Disambiguation
PNDDC	Personal Name Disambiguation Data Catalogue
PNEL	Personal Name Entity Linking
PNELF	Personal Name Entity Linking Framework
PNSFM	Personal Name Surface Form Modules
PNTM	Personal Name Transformation Modules
PTA	Partial Tree Alignment
RHS	Right Hand Side
SPO	Simple Property Object
SPTM	Simple Partial Tree Matching
STM	Simple Tree Matching

Appendix B

Thesis Diagrams

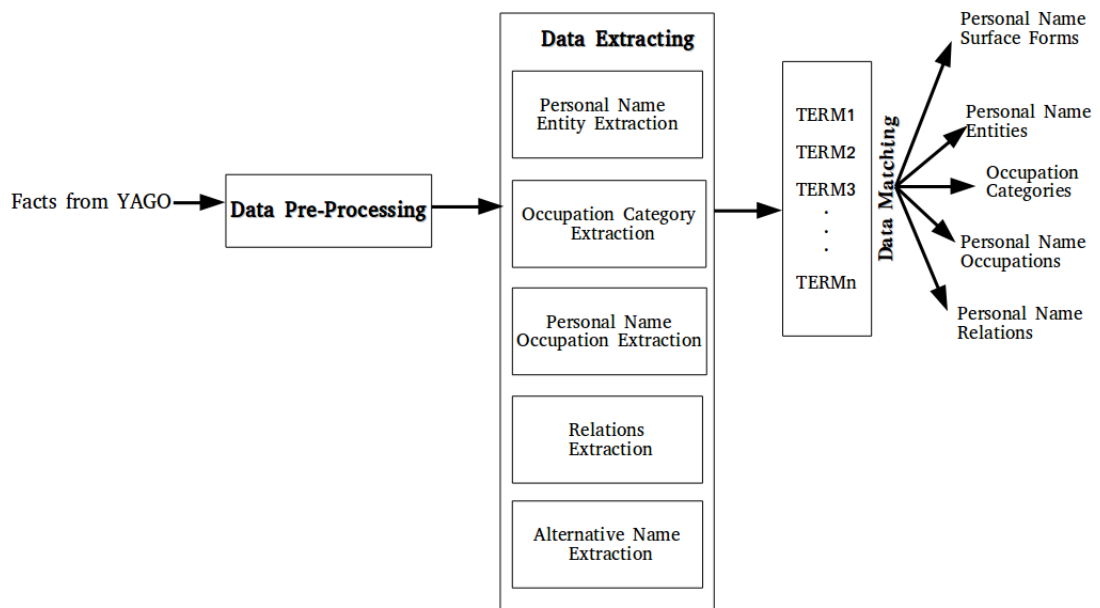


Fig. B.1 Chapter 3: PNSFM modules

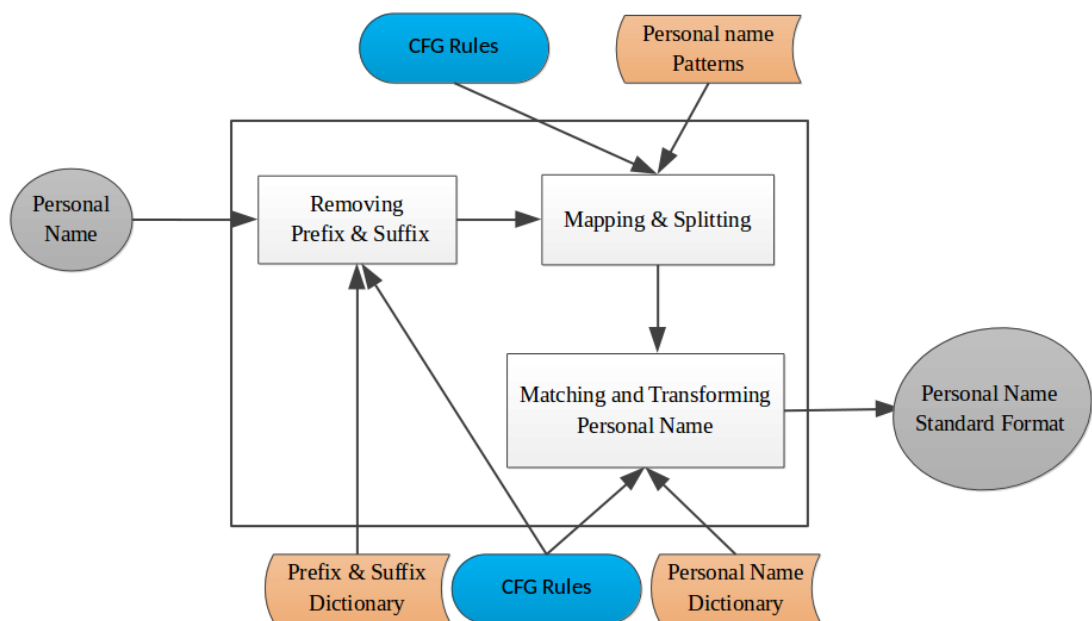


Fig. B.2 Chapter 4: Personal Names Transformation Modules

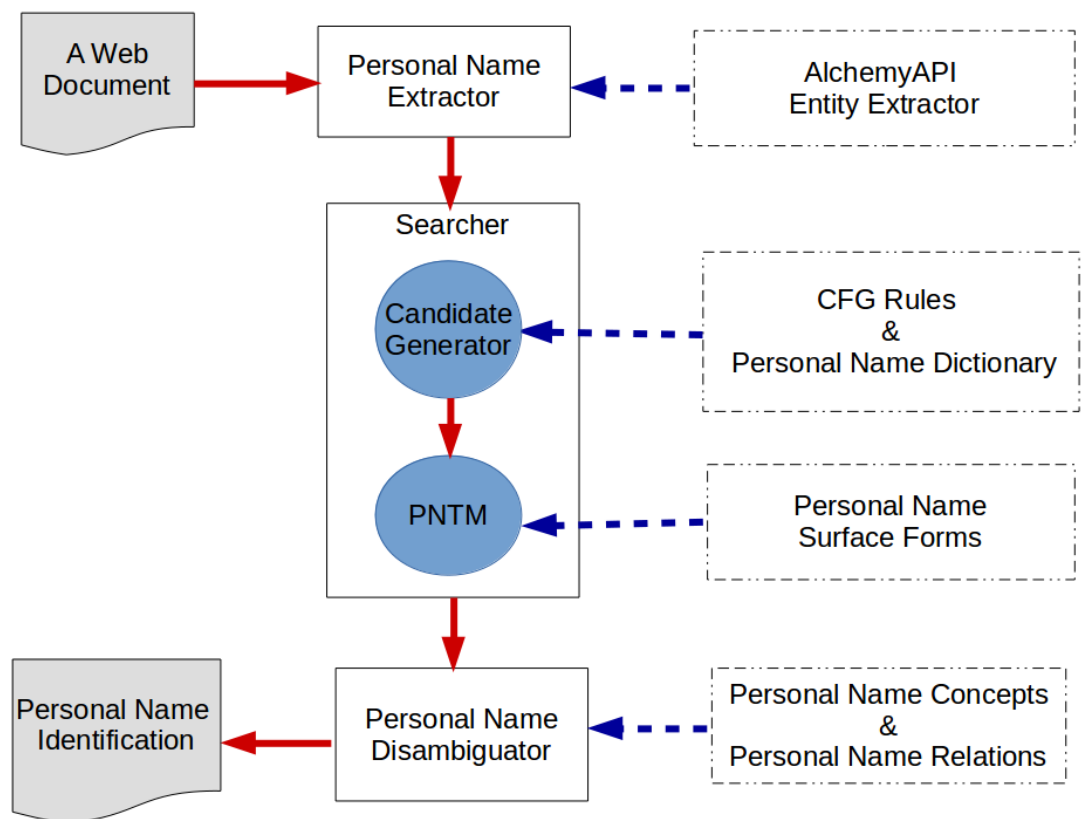


Fig. B.3 Chapter 5: Personal Name Entity Linking Framework

Appendix C

Reviews of Tools in Fact Extracting and Fact Answering

In this section, we introduce the fact extractor and fact answering tools including YAGO, Reverb, Bing and Google.

C.1 YAGO

YAGO [40] is an ontology, which is part of YAGO-NAGA project developed at the Max Planck Institute for Informatics. YAGO2 not only contains entities, facts and events, but also includes times and place. This allows the temporal property of events to be recorded, such as the Olympic Games or the Oscars Academy Awards; that held the facts more significantly. Therefore, YAGO does not only answer facts about what but also when and where. For example, given a question about “*Scientists who were born nearby Ulm and won a Nobel Prize (in any discipline)*”, the results provided are *Albert Einstein, Hans Spemann and Gerhard Ertl*.

Facts in YAGO are automatically extracted from Wikipedia, GeoNames and WordNet. The data structure in YAGO2 consists of two main parts: entities and classes. Entity is the individual instance such as person, river or event which is extracted from Wikipedia articles. The second part is the class or type that is used for specifying and grouping entities.

The YAGO taxonomy framework contains three layers. The highest layer classes are from WordNet, the second layer classes are from Wikipedia categories and the lowest layer is a list of instances. The property *subclassof* is used for mapping between the higher and lower classes. An individual instance should be assigned at least one class, where instance is an individual entity such as a building, person or song. For example, *J. K. Rowling*

is an instance in a class of *English women writers* or *Female billionaires*. These classes are subclasses of the *Writer* and *Person* is a parent of *Writer* class. YAGO links class and subclass with *subclassof* to create a hierarchy of classes. Furthermore, one instance may be represented in numerous ways. For example, *J. K. Rowling* can be represented as *J.k.rolwing*, *Jessica Rowling Arantes*, or *J.K. Rowling*. YAGO uses a property *means* to map between the instance and its reference.

YAGO architecture is divided into two main components. Firstly, facts are a core part of significantly building the knowledge base. Facts are stored in the form of RDF data model of SPO triples: subject (S), predicate (P) and object (O). Each fact has unique ID key. The second component is a representation model. YAGO provides SPOTL (SPO + Time + Location) for browsing and querying the knowledge base via the query language SPARQL.

Browse YAGO2

Entity: case insensitive

Albert_Einstein

Show transitive facts

hasGender	male →
isKnownFor	<ul style="list-style-type: none"> Bose–Einstein statistics → Brownian motion → Einstein field equations → General relativity → Mass–energy equivalence → Photoelectric effect → Special relativity →
influences	<ul style="list-style-type: none"> Boris Podolsky → David Bohm → Paco Ahlgren → Rudolf Carnap → Théophile de Donder →
graduatedFrom	ETH Zurich →
hasAcademicAdvisor	<ul style="list-style-type: none"> Alfred Kleiner → Alfred Kleiner →
	<ul style="list-style-type: none"> 19th-century German people → person → Academics of the Charles University → academician → educ American humanitarians → humanitarian → benefactor → go American pacifists → pacifist → adult → person → American people of Swiss descent → person →

Entity list (left):

- ← A Einstein
- ← Albert Eienstein
- ← Albert einstein
- ← Albert Einstein
- ← Albert Einstein's
- ← Albert Einstien
- ← Albert Einstein
- ← Albert LaFache Einstein
- ← Al Einstein
- ← Compassionate Zionism and Albert Einstein
- ← Einselein
- ← Einstein
- ← Einstein, Albert
- ← Einsteinian
- ← Einstein on socialism
- ← Einstien
- ← Einstien, Albert
- ← Ejnstejno
- ← God does not play dice
- ← Miracle Year
- ...
- ← Albert Einstein
- ← Thomas Stoltz Harvey

Relationships (bottom):

- means
- hasPreferredMeaning
- isKnownFor

Fig. C.1 An example of entity search about *Albert Einstein* in YAGO.

YAGO comes in two versions to downloadable. The core version contains 2.6 million entities and about 124 million facts, while the second full version contains nearly 10 million entities and more than 447 million facts. In the core version, YAGO delivers four formats to download: native (ID and triples of SPO: subject, predicate and object), Resource Description Framework (RDF), Notation3(N3) and the format of the Jena TDB store. Additionally, then full version offers only native and RDF formats. YAGO allows browsing of individual entities via the interface to see each entity details. For example, Figure C.1 shows the entity of *Albert Einstein* and its relationships.

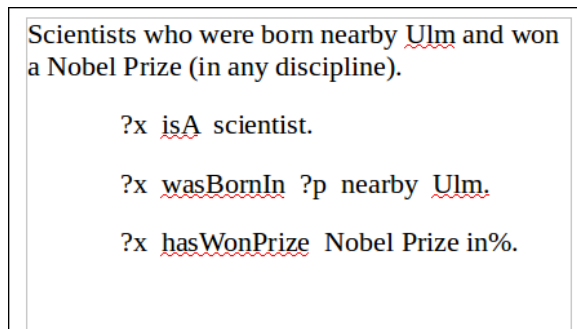


Fig. C.2 An example question and triple patterns with SPARQL syntax.

Moreover, YAGO presents the SPOTL(X) with five joins: subject, predicate, object, time, location and keyword and five triple patterns for searching the YAGO knowledge base. Figure C.2 represents an example of SPARQL syntax query. As the first triple: *?x* is any subject (S) , *isA* is property (P) of the subject and *scientist* is the object(O).

YAGO 2 spotlx

Query

Id	Subject	Property	Object	Time	Location	Keywords
?id0:	?x	isA	scientist			
?id1:	?x	wasBornIn	?p		nearby Ulm	
?id2:	?x	hasWonPrize	Nobel Prize in%			
?id3:						
?id4:						

Fig. C.3 An (a) SPOTL(X) interface.

The interface of SPOTL(X) is demonstrated in Figure C.3. The maximum joining in each query among subject (S), property (P), object (O), time (T), location (L) and keyword (X) is five. The result is revealed in Figure C.4 and Figure C.5. Figure C.4 shows an example of a query result about “*Scientists who were born nearby Ulm and won a Nobel Prize (in any discipline)*”. The information about each entity is detailed in table form with seven columns: ID, subject, property, object, time, location and keyword. Figure C.5 shows the details of each entity using the map format to specify each entity location and a life-line to identify the time period.

Id	Subject	Property	Object	Time	Location	Keywords
1	#569437619 Albert Einstein	type	scientist	1879-03-14 †, 1955-04-18 †	-	Walhalla temple tensor ...
	#508959095 Albert Einstein	wasBornIn	Ulm	1879-03-14 †, 1879-03-14 †	Ulm	Walhalla temple tensor ...
	#508785219 Albert Einstein	hasWonPrize	Nobel Prize in Physics	-	-	Walhalla temple tensor ...
	#492890823 Nobel Prize in physics	means	Nobel Prize in Physics	-	-	Interdisciplinary Science Reviews ...
	#38538330 Ulm	means	Ulm	1181-01-01 †, 1803-12-31 †	Ulm	biomass textiles Novi ...
	#945228 scientist	means	scientist	-	-	-
2	#569437619 Albert Einstein	type	scientist	1879-03-14 †, 1955-04-18 †	-	Walhalla temple tensor ...
	#508959095 Albert Einstein	wasBornIn	Ulm	1879-03-14 †, 1879-03-14 †	Ulm	Walhalla temple tensor ...
	#508785219 Albert Einstein	hasWonPrize	Nobel Prize in Physics	-	-	Walhalla temple tensor ...
	#14195816 Nobel Prize in Physics	means	Nobel Prize in Physics	-	-	Interdisciplinary Science Reviews ...
	#38538330 Ulm	means	Ulm	1181-01-01 †, 1803-12-31 †	Ulm	biomass textiles Novi ...
	#945228 scientist	means	scientist	-	-	-
3	#51106802 Hans Spemann	type	scientist	1869-06-27 †, 1941-09-09 †	-	morphogenesis Gustav Wolff ...
	#508975999 Hans Spemann	wasBornIn	Stuttgart	1869-06-27 †, 1869-06-27 †	Stuttgart	morphogenesis Gustav Wolff ...
	#508802011 Hans Spemann	hasWonPrize	Nobel Prize in Physiology or Medicine	-	-	morphogenesis Gustav Wolff ...
	#492985383 Nobel Prize in Medicine	means	Nobel Prize in Physiology or Medicine	-	-	John Kennedy physiologist ...
	#38538330 Ulm	means	Ulm	1181-01-01 †, 1803-12-31 †	Ulm	biomass textiles Novi ...

Fig. C.4 SPOTL(X) query result within table form.

YAGO directly identifies a time point and time span for each entity and fact by classifying entities into four major types: people, artifacts, groups and events and relations are assigned to each group. Firstly, the property: *wasBornOnDate* and *diedOnDate* is used to identify the year of life for the person. The next group is artifacts using property as the *wasCreatedOnDate* and *wasDestroyedOnDate* relationships specify their age. Groups are the third of the entity types, which specify their existence time with property through the *wasCreatedOnDate* and *wasDestroyedOnDate* relations. The last entity type is events that use these properties: *startedOnDate* and *endedOnDate* to define their era. Furthermore, the *happenedOnDate* relation is used for a one day event.

YAGO captures facts that have temporal dimension using *occursSince* and *occursUntil* to explain these facts. Moreover, the property *occursOnDate* is used to define one day or one time facts such as awards. In order to give location base to each entity, the property *happenedIn* is used for an event entity and the *isLocateIn* property is used for artifacts entities. Furthermore, properties such as *wasBornIn*, *diedIn*, *worksAt* or *participatedIn* are used to explain the location in YAGO's facts.

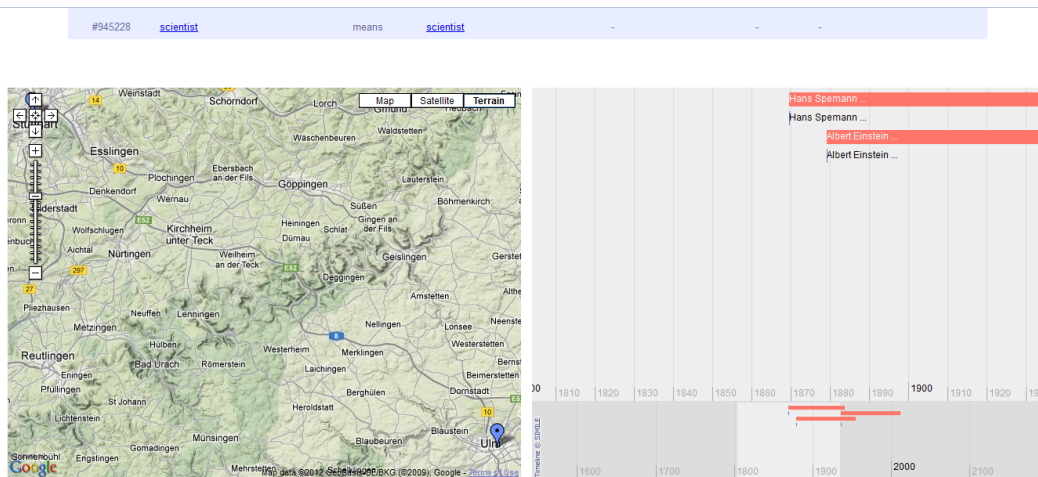


Fig. C.5 The query result from YAGO2 using SPOTLX.

C.2 Facts answering from the Bing and Google search engines

The fact look up engine aims to return the specific answer for a user who inputs a factual query into a search engine system. This section introduces the fact look up engines and compare the ability between the two well-known search engines of: Google and Bing. Figure C.6 and Figure C.7 are the user interface and the result values from Google and Bing using the same query phrase of *jk rowling birthday*.

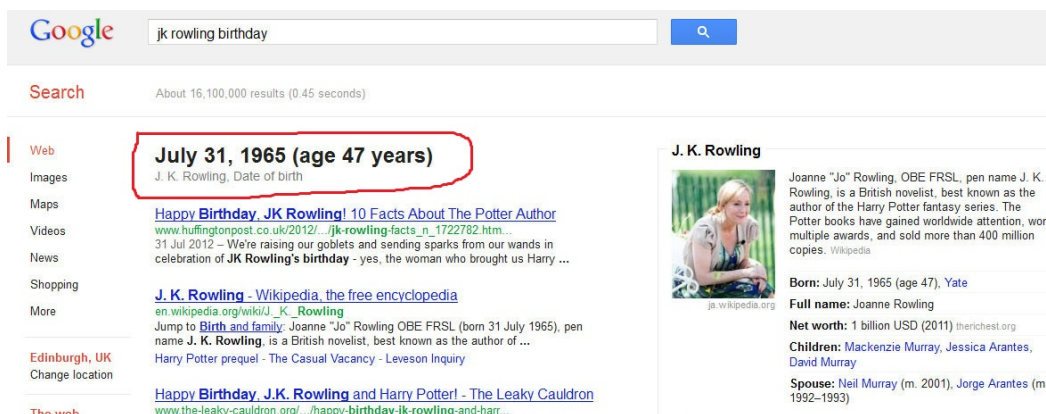


Fig. C.6 The query results about: JK Rowling birthday from Google.

bing WEB IMAGES VIDEOS MAPS MORE

jk rowling birthday

1,930,000 RESULTS

Jk Rowling — Date of Birth: 31 July 1965 Good / Bad
 According to <http://www.celebs101.com/celebrity-Jk+Rowling-707--.html> Show All

J. K. Rowling - date of birth
 July 31, 1965 · age 47 years
 Find out more on: [Freebase](#)

[JK Rowling Birthday Project](#)
[jkrowlingbirthday.com](#)
 JK Rowling Birthday Project: Show your appreciation to the person behind the magic!

RELATED SEARCHES
[Happy Birthday Harry Potter Font](#)
[When Is Draco Malfoy's Birthday](#)
[Harry Potter J.K. Rowling](#)
[J.K. Rowling Information](#)
[J.K. Rowling Birth Date](#)
[J.K. Rowling Facts](#)
[J.K. Rowling Family](#)
[Harry Potter Book Sales Statistics](#)

Fig. C.7 The query results about JK Rowling birthday from Bing.

FACTO [78], a fact look up engine, is one of fact look up engines providing by Bing, which can be accessed at: <http://lepton.research.microsoft.com/facto>. FACTO extracts facts from web tables. FACTO uses the triple form: entity-attribute-value (*JK Rowling - date of birth – 31 July 1965*), *JK Rowling* is an entity, *date of birth* is an attribute and *31 July 1965* is a value to construct fact. Fact values are cleaned before being integrated and stored in a database ready to answer user queries. The key components of FACTO are the data extractor and the query answering engine.

1. The data extractor consists of four major components: table classifier, URL pattern summariser, entity extractor and attribute-value mapping. The table classifier aims to fill out non-data tables (e.g. an empty table or a calendar) from data tables (a table which has an attribute and its value). A Support Vector Machine (SVM) with a linear kernel is used to classify attribute-value tables and training sets are labelled manually. The second component is the URL pattern summariser, which is used in [77] to treat each URL pattern as a data source for learning patterns from some pages and used to extract data from other pages. Yin et al. [77] found that we can categorise web pages into many groups based on their URLs, and that most of web pages in the same group have the same format. Thirdly, the aim of extracting entities is to find the key entity of the web page. FACTO uses two steps for entity extraction: the wrapper builder and the wrapper ranker. The wrapper builder is used for finding key entity in a web page using a user query. It then tries to match this with the HTML tags in a web page until a match is found, and at this point a wrapper is built. The next step is the wrapper ranker using the calculation scores from precision and recall in each wrapper. The highest wrapper score is used to extract data entities in each web page. The final step in the data extractor is mapping the attribute-value to the key entity. To combine

the attribute-value more correctly, the table classifier only consider the contents of an individual table, computing the entropy of all values of this attribute and number of pages containing this attributes from the data source. The attribute which has an entropy of less than 0.5, or which appears in less than five pages, is ignored.

2. The second component is '*query answering*', which consists of five steps: query interpretation, entity-attribute filtering, attribute equivalence, entity equivalence and finally data retrieval and result aggregation. Firstly, query interpretation uses five rules to convert a user query into an entity (e)-attribute (a):

- (a) "e a"
- (b) "e's a"
- (c) "(the)?a of e"
- (d) "(what/who/when/where) (is/are/was/were) (the)? a of (the)? e"
- (e) "(what/who/when/where) (is/are/was/were) e's a".

Additionally, FACTO has four specific rules that are used for directly querying:

- (a) how long
- (b) how old
- (c) when was someone born
- (d) where was someone born

Secondly, entity-attribute filtering is based on two observations:

- (a) Entity-attribute pairs found in many data sources are more likely to be a valid pair.
- (b) Data sources which provide many entity-attribute pairs should have a high hub score.

Thirdly, attribute equivalence is used because the user can input the attribute-value differently from the data store and the web page can represent the same attribute in a variety of forms. The entity-attribute pair value is used to measure the similarity between two attributes. The next step is the entity equivalence, which maps between the user query entity and the data store entity. Finally, FACTO generates the probable entity-attribute pair and returns the best value sets and URLs to a user.

The screenshot shows a Google search for "tom cruise spouse". The search bar contains the text "tom cruise spouse" and a search button. Below the search bar, the results are displayed. On the left, there is a sidebar with navigation options: Web, Images, Maps, Videos, News, Shopping, and More. The main content area is titled "Tom Cruise Spouse" and features a red-bordered box containing three entries: Katie Holmes (m. 2006–2012), Nicole Kidman (m. 1990–2001), and Mimi Rogers (m. 1987–1990). Each entry includes a small portrait image and the name of the spouse with their marriage dates. To the right of this box is a "Feedback" link. Below the box, there is a link to "Tom Cruise - Wikipedia, the free encyclopedia" and a snippet of text: "Cruise married actress **Mimi Rogers** on May 9, 1987; he was 24 and she was 31. ... Cruise met his second **wife**, actress **Nicole Kidman**, on the set of their film Days of ...". On the far right, there is a profile for "Tom Cruise" with a portrait and biographical details: "Thomas Cruise Mapother IV, best known as Tom Cruise, is an American film actor and producer. He has been nominated for three Academy Awards and has won three Golden Globe Awards. He started his career at age 19 in the 1981 film Taps." Below this is a "Wikipedia" link and a list of facts: "Born: July 3, 1962 (age 50), Syracuse", "Height: 1.70 m", "Children: Suri Cruise, Connor Cruise, Isabella Jane Cruise", "Spouse: Katie Holmes (m. 2006–2012), Nicole Kidman (m. 1990–2001), Mimi Rogers (m. 1987–1990)", and "Siblings: Cass Mapother, Lee Ann Mapother, Marian Mapother".

Fig. C.8 The query results about Tom Cruise spouse from Google.

The screenshot shows a Google search for "brad pitt spouse". The search bar contains the text "brad pitt spouse" and a search button. Below the search bar, the results are displayed. On the left, there is a sidebar with navigation options: Web, Images, Maps, Videos, News, Shopping, and More. The main content area is titled "Angelina Jolie (2005–)" and features a red-bordered box containing the text "Brad Pitt, Partner". Below this box, there is a link to "Brad Pitt - Wikipedia, the free encyclopedia" and a snippet of text: "Pitt lives with actress **Angelina Jolie** in a relationship that has attracted wide publicity. ... earlier with **Jennifer Aniston** and **Brad Grey**, CEO of Paramount Pictures. ... between **Pitt** and **Angelina Jolie**, who played his character's **wife** Jane Smith. Angelina Jolie - Legends of the Fall - List of awards - Killing Them Softly". Below this is a link to "Brad Pitt phones ex-wife Jennifer Aniston to congratulate her on the ..." and a snippet of text: "15 Aug 2012 - **Brad Pitt** phones up his **ex-wife Jennifer Aniston** to congratulate her on ... the dark days of 2005 when Brad dumped Jen for **Angelina Jolie**, 37, ...". On the far right, there is a profile for "Angelina Jolie" with a portrait and biographical details: "Angelina Jolie is an American actress and director. She has received an Academy Award, two Screen Actors Guild Awards, and three Golden Globe Awards, and was named Hollywood's highest-paid actress by Forbes in 2009 and 2011." Below this is a "Wikipedia" link and a list of facts: "Born: June 4, 1975 (age 37), Los Angeles", "Partner: Brad Pitt (2005–)", "Children: Shiloh Jolie-Pitt, Zahara Jolie-Pitt, Maddox Jolie-Pitt, Pax Thien Jolie-Pitt, More", "Siblings: James Haven", and "Spouse: Billy Bob Thornton (m. 2000–2003), Jonny Lee Miller (m. 1996–1999)".

Fig. C.9 The query results about Brad Pitt spouse from Google.

Comparing the fact look up engines between Google and FACTO, Google is more accurate, has a better in user interface and has a higher query coverage (as represented in Figures C.8 to C.11). However, Google provides an inconsistent user interface, as the query in Figure C.8 is “*Tom Cruise spouse*” and the answer represents his spouses with the images on the left hand side and his profile on the right hand side, but in Figure C.9, which is a very similar query about “*Brad Pitt spouse*”, the system returns different interface design so the interface does not represented the images in the left hand side and the right hand side is represented his partner profile (*Angelina Jolie*). Finally, the answer about “*Brad Pitt spouse*” is not a complete answer because the former spouse *Jenifer Aniston* is not represented.

WEB IMAGES VIDEOS MAPS NEWS MORE

bing tom cruise spouse

73,900,000 RESULTS

Tom Cruise — Spouse(s): Mimi Rogers (1987–1990) Nicole Kidman (1990–2001) Katie Holmes (2006–present)

According to http://en.wikipedia.org/wiki/Tom_Cruise

Good / Bad Show All

Showing results for tom cruise *wife* too.
Do you want results only for tom cruise spouse?

Tom Cruise - spouse(s)
Katie Holmes · Mimi Rogers · Nicole Kidman
Find out more on: [Freebase](#)

RELATED SEARCHES

- Tom Cruise 3 Wives
- Is Tom Cruise Wife Pregnant
- Tom Cruise Wife Katie Holmes
- Tom Cruise Wife Divorce
- Name of Tom Cruise Wife
- Tom Cruise Next Wife
- Mimi Rogers
- Nicole Kidman

Fig. C.10 The query results about Tom Cruise’s spouse from Bing.

bing brad pitt spouse

13,100,000 RESULTS

Brad Pitt — Spouse(s): Jennifer Aniston (2000–2005)

According to http://en.wikipedia.org/wiki/Brad_Pitt

Good / Bad Show All

RELATED SEARCHES

- Brad Pitt's Ex-Wife
- Brad Pitt Biography

Fig. C.11 The query results about Brad Pitt’s spouse from Bing.

On the other hand, we found some errors in FACTO for answering *Wh* questions. The results of this are shown in Figure C.13. For example, for the *where* question the user expects to see the specific name of place; however, for the query “*Where was Justin Bieber born*”, FACTO return the date of birth but Google returns the result correctly as shown in Figure C.12.

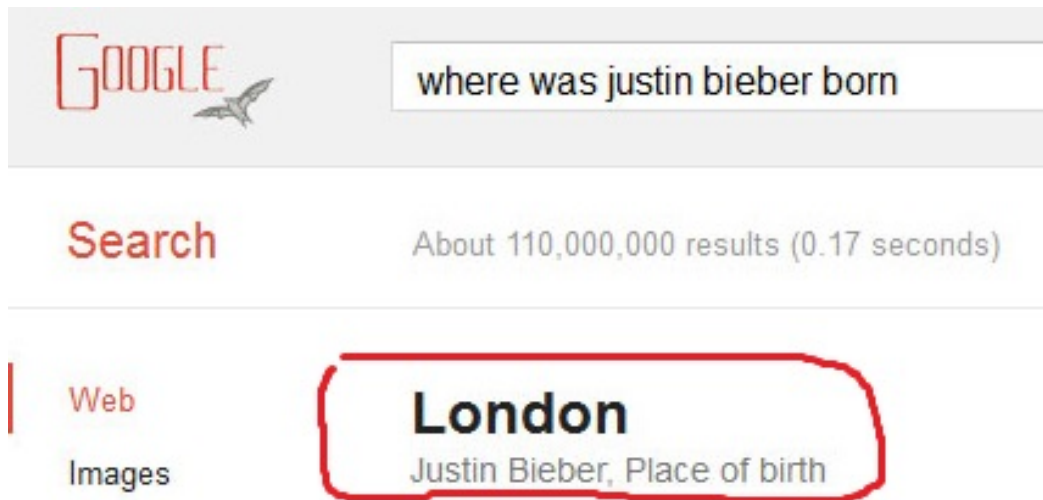


Fig. C.12 The query results for "Where was Justin Bieber born?" from Google.

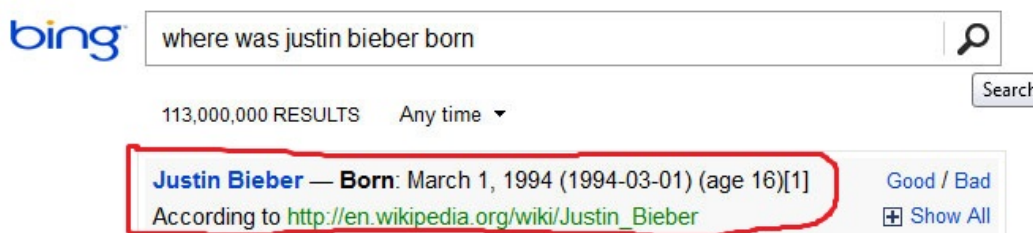


Fig. C.13 The query results for "Where was Justin Bieber born?" from Bing.

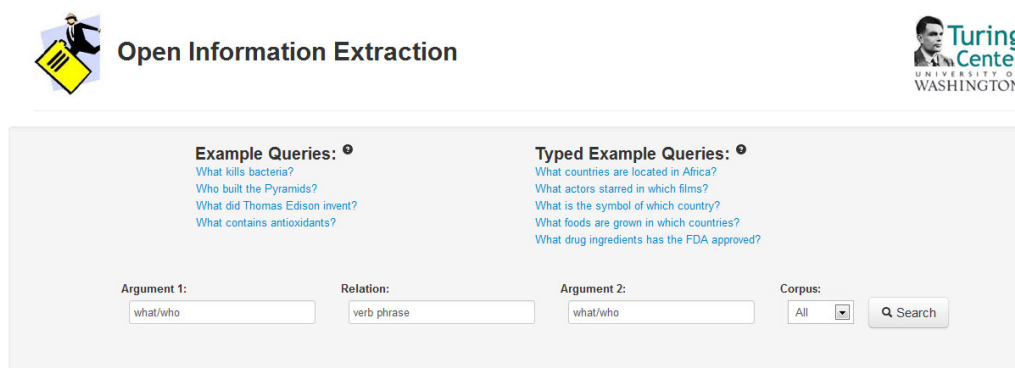
C.3 Open Information Extraction: REVERB

Open Information Extraction (OIE) is an automatic extraction system which extracts English sentences from the web page and stores it in triple form (arg1-relation-arg2) without pre-defined relations (relation-independent extractor) or human intervention [4, 5, 27, 72]. TexRunner [4] was the first Open Information Extraction system that used a self-supervised learner which identified relation phrases in English sentences. TexRunner learns the positive and negative part-of-speech tags and noun phrase chunks using the data set from Penn Treebank for training the Naïve Bayes classifier. WOE [72] improves the precision and recall of TextRunner using Wikipedia infobox as a data set for training the information extractor.


Reverb [27], the new edition of Open Information Extraction, expands the precision and recall of TextRunner and WOE by fixing two common issues that always arise in the output: incoherent extractions and uninformative extraction. Incoherent extractions extract the worthless relation phrase from the sentence, while the uninformative extractions problem

discards the insignificant information.


Reverb presents two constraints that can be used to extract relation phrases in English sentences more correctly: a syntactic constraint and a lexical constraint. The syntactic constraint uses the Part-of-Speech tag pattern: “*V/VP/VW*P*” where “*V*” is “*verb particle? adv?*”, “*W*” is “*noun/adj/adv/pro/det*” and *P* is “*prep/particle/inf.marker*” to match the relation phrases in each sentence first (against TextRunner, which finds the argument (entity) first). A syntactic constraint reduces both incoherent extractions and uninformative extractions. After that, lexical constraint is assigned to permit the relation phrases using the logistic regression classifier for matching relation phrases with relation phrases in the repository.



Open Information Extraction

Example Queries: 

- What kills bacteria?
- Who built the Pyramids?
- What did Thomas Edison invent?
- What contains antioxidants?

Typed Example Queries: 

- What countries are located in Africa?
- What actors starred in which films?
- What is the symbol of which country?
- What foods are grown in which countries?
- What drug ingredients has the FDA approved?

Argument 1:

Relation:

Argument 2:

Corpus:

Fig. C.14 The user interface of the REVERB extraction system

Reverb provides a user interface for querying the facts that are extracted from 500 million web pages that are represented in Figure C.14. Reverb can decrease the errors regarding incoherent extractions and uninformative extractions. However, from our observations, we found that the errors still occur, for example when the query is “*Who is US president?*”. The results of this query are demonstrated in Figure C.15. Reverb returns 25 answers from 107 sentences and reveals some incorrect answers e.g. *Hillary Rodham Clinton, Jeremiah Wright and the top*.



The screenshot shows the REVERB user interface. At the top left is the logo for "Open Information Extraction" and at the top right is the "Hosted by AI2" logo. Below the logos is a search bar with three input fields: "Argument 1:" (empty), "Relation: is", and "Argument 2: US president". To the right of the "Argument 2" field is a dropdown menu set to "All" and a "Search" button. Below the search bar, it says "25 answers from 107 sentences (cached)". Underneath is a horizontal filter bar with the following categories: "all", "person (15)", "us president (13)", "author (12)", "public speaker (9)", "person in fiction (9)", "misc.", and "more types". Below the filter bar is a list of entity names with their respective counts in parentheses: "Hillary Rodham Clinton (2)", "James Buchanan (2)", "Richard Nixon (2)", "the top (2)", and "Jeremiah Wright (2)".

Fig. C.15 REVERB user interface

Reverb provides synonym words for any entities; however, it is insufficient because we find many repetitive entities in the outputs. For example, Reverb cannot specify that *Obama* and *Barack Obama* are the same entity.

C.3.1 Conclusion

The World Wide Web has a large volume of facts; however, it also has incorrect data and ambiguous data. Moreover, web designers can represent the data in a variety of formats as tables, text or graphs, and can use heterogeneous contexts to represent the same meaning (referential ambiguity problem) or use the same words to have a different meaning (lexical ambiguity problem). For example, *New York City* can be represented using different contexts such as *New York* or *Big Apple*; or when we see the word *Apple*, it could be the *fruit* or the name of the computer company *Apple Inc.*

Table C.1 The contrast between YAGO2, Fact look up in Google search engine, FACTO and Reverb.

Items	YAGO2	Google	FACTO	REVERB
Data sources	Wikipedia, WordNet and Geonames	Unknown	web tables	English sentences from WWW
Facts structured	Triple SPO: Subject, Predicate, Object	Unknown	Triple EAV: Entity, Attribute, Value	Triple: Arg1, Relation, Arg2
User query interface	Two major interface: entity query and SPOTL (required experience in SPARQL syntax)	Supported attribute or question such as What, Where, When and How	Supported attribute or question such as What, Where, When and How	Supported Who/What question in the of argument1, relation, argument2
Speed	Modulate	Fastest	Fast	Slow
Data volume	447 million facts and 9.8 million entities(included GeoNames)	Unknown	73 million entity-attribute pairs	7.8million facts
Accuracy	95% precision is approved by human	Higher	High	Moderate
Synonym entity supported	Yes	Yes	Yes	No

Factual information is stored in triple form: (entity, relations and entity) which is based on the RDF model. In our work we focus on the two major problems of: how to handle referential ambiguity and lexical ambiguity. We consider and compare facts which are extracted from the internet, the information for which is shown in Table C.1. YAGO2 is strongest in precision and identifying similarity in each entity using the relations *means*. The property *means* in YAGO2 provides a lot of synonyms in each entity. For example, *Steven George Gerrard* can be represented in other words as (*Gerrard, Gerro, Stephen Gerrard, Steve g, Steve Gerrard*). Furthermore, YAGO2 supports facts with a temporal dimension as position, or events like the US Open. However, YAGO2 has a weakness in representing the results as it is hard for new users who are unfamiliar with SPARQL language to query facts. Google is strongest in representing the results as it is clear and simple for users to submit queries. On the other hand, the output interface is unstable and some facts are incorrect. FACTO is powerful in precision and user interface; however it has errors in wh questions, lacks a number of facts and lacks synonym entities. Reverb is powerful in automatic extraction and unbound relation names. However, the accuracy is poor and redundancy of data is a major concern. Reverb has made a mistake in dealing with the referential ambiguity problem. For example, it cannot recognise that *Obama, Barack Obama* and *Barak Obama* are the same person.

Finally, facts are the most useful things for search engines answering queries. Search engine such as Google and Bing try to return the specific answer in each query. However, the limitation in information extraction and fact answering is disambiguating the personal named entity. That is to say, how to map the fact to the right person and how to map between the querying entity and the knowledge base entity despite variations in the personal name.