A New Framework for Personal Name Disambiguation

L.Georgieva and S. Buatongkue
School of Mathematical and Computer Sciences
Heriot Watt University
Edinburgh, UK, EH14 4AS
Email: L. Georgieva@hw.ac.uk, sb369@hw.ac.uk

***Abstract*— In this paper we study the problem of personal name disambiguation (NED). We develop a framework to address the three challenges in personal name disambiguation: (i) identification of referential ambiguity, (ii) identification of lexical ambiguity, and (iii) predicting the NIL value, that is the value when a named entity cannot be mapped to a knowledge base. Our framework includes extractor, searcher and disambiguator. Experimental results evaluated on real-world data sets, show that our framework and algorithm provide accuracy in personal name linking up to 92%, which is higher than the accuracy of previously developed algorithms.**

Keywords—personal name disambiguation, data cleaning, lexical ambiguity, knowledge bases, context free grammar, similarity metrics.

I. INTRODUCTION

Correct personal name identification of a unique personal name is an important task in many areas including search engines, information retrieval, and machine translation [9, 13, 14]. A large amount of data on the internet refers to names, including not only personal names but also names of locations, books, songs, films. Background knowledge, such as occupation, age, or nationality is usually required to disambiguate and identify a particular person by name.

Personal name disambiguation (NED) is the task of matching the entity in a document to its comparable entry in a large knowledge base (e.g., Wikipedia) and is also known as name entity linking. There are three challenges in personal name disambiguation: (i) *referential ambiguity*, (ii) *lexical ambiguity*, and (iii) *predicting the NIL value*.

We briefly describe the challenges next.

(i) *Referential ambiguity* or name variations means that different names may refer to the same person. Typically, a personal name (in English) consists of three parts: given name, middle name and family name, e.g., George Walker Bush. There are a variety of styles to represent English personal names, including: nicknames, pen names, alias names, short names, or abbreviations. For example the same person George Walker Bush can be referred to as: President Bush, Dubya Bush, George W. Bush, and 43rd President of the United States. Personal alias names can be used as personal names and their syntax can be completely different from the real name such as *The Governor* which can be used to refer to *Arnold Schwarzenegger*.

A single name can be represented in multiple patterns. For example, in the case when the family name is preceded by a preposition (de, da, di, von), an article (le and la) or both (du, des, del, de la, della) the name can be abbreviated in different ways and represented in different formats in different context. It is standardly recommended that de following a first name or title such as Professor, Mr. Dr. is not used with the name alone. When the last name has only one syllable, de is usually retained. The preposition also remains, in the form of d', when it elides with the last name also beginning with a vowel.

For example, the personal name Prof Philippe De Wilde can be represented in different formats in different context, i.e.

1. Prof Philippe D. Wilde
2. Prof P. D. Wilde
3. Prof P. De Wilde
4. Wilde, Philippe De, Professor
5. P D'Wilde, etc.

(ii) *Lexical ambiguity*. The issue of lexical ambiguity means that a single name may refer to multiple persons [8]. For example, the name Chris Martin can refer to any of the following four persons:

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.
5.

(iii) *Predicting the NIL value.* The issue of predicting the NIL value refers to names which cannot be matched to a personal name.

II. OUR APPROACH

In this paper, we propose a new approach that can be used to both identify and disambiguate personal names. We use a combination of context free grammar (CFG) models and Jaro-Winkler similarity metric in order to generate a set of candidate entities. Our model uses CFG to transform the variations in mention name to a unique format. A CFG has the following advantages:
(i) it is flexible for personal name variations because it can capture multiple name formats;
(ii) it can be used to solve the misleading problem in text similarity measurement (including alternatives to Jaro-Winkler, e.g. edit distance or cosine similarity) when the same person is represented using highly different textually (e.g., Bill Gates and William Gates) by transforming the nickname Bill to the given name William;
(iii) CFG is different from other transformation tools because it allows us to understand the internal structure of personal name.

The similarity metric Jaro-Winkler, which we use, was originally designed to deal with typographical errors [21]. Studies which compare the Jaro-Winkler metric to alterative metrics, including Levenshtein, Q-gram, Smith-Waterman, and TF-IDF show that it has a good performance in personal name matching [20].

We introduce a new algorithm: Simple Partial Tree Matching (SPTM) for personal name disambiguation. SPTM is an entity coherence method, and is developed under the assumption that personal names mentioned in a single web page have the same conceptualization or are related. SPTM performs an ambiguity name evaluation and selects the best personal name entity for each ambiguity occurrence using the following three steps:

1. We first generate an individual concept to each personal name entity base on designed occupation taxonomy architecture.
2. The identified personal names which have the same root node are merged to create the comparison tree. The comparison tree which has the maximum number of nodes is considered first.
3. We rank the candidate entities by comparing the similarity between each candidate entity concept and the comparison tree. The similarity score is calculated from the number of matching nodes and their weights. We give different weights to different hierarchical levels. The nodes which have the same level are ranked as having equal weight. The candidate entity which has the highest score is selected.

The structure of this paper is as follows. In Section 3. we introduce the knowledge bases Wikipedia, YAGO, Freebase and DBPedia, which we have use for experimental evaluation of our results. In Section 4, we design the taxonomy and introduce personal name transformation which uses context free grammar. The algorithm which uses simple partial tree matching is given next in Section 5. The experimental results and conclusion are in Section 6.

III. KNOWLEDGE BASES

We design new ontology architecture of professional categories by integrating web directories and the YAGO ontology [12] to create occupation taxonomy. In the knowledge base, classes and entities are the main components that playing a key role in presenting entity conceptualization. Conventionally, *class* is the hierarchy of the elements that are used for grouping similar entities together.

Consider, for example, Wikipedia [17], which is a free multilingual Web-based encyclopedia. It is built collaboratively by volunteers and each Wikipedia article is usually addressing a single topic only. Each topic has been manually allocated to at least one category in Wikipedia. Categories in Wikipedia are the group of articles that have a similar subject.

We can see a box containing the categories to which an article belongs at the bottom of a page. For example, the page about Elton John is in the categories:

- Elton John,
- 1947 births,
- 20th-century composers,
- 20th-century English male actors,
- 20th-century English singers and 59 more.

The top of Wikipedia categories is: *Portal:Contents/Portals*.

Wikipedia classifies the contents into 12 main category portals including *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 by the following broad categories:

1. By association e.g., by educational institution or by company, where they currently work.
2. By ethnicity, gender, religion, sexuality, disability, medical or psychological conditions.
3. *By the person's name.*
4. By nationality and occupation
5. By place, the place of birth or notable residence. e.g. people living in New York.
6. By year, people are categorized by their year of birth and their year of death.

Wikipedia categories are organized in hierarchical structure. However, categories are not arranged in a strict hierarchy or tree of categories because each article can be assigned to more than one category, and each category can be assigned to more than one parent category. The hierarchy of categories reflects the thematic structure only. For example, the name *Zidane* is in the super-category Football in France, but *Zidane* is a personal name.

Wikipedia deals with personal name ambiguity in categories by using various kinds of attributes to make the names unique. It uses either a single feature or combinations of features to distinguish between people who have the same name: occupation, date of birth, nationality, place of residence.

The significant feature in personal name disambiguation in Wikipedia is *occupation*. e.g.

- Chris Brown (composer).
- Nationality and occupation e.g., Chris Brown (Canadian musician).
- Place of residence and occupation e.g., Chris Brown (California politician).
- Occupation and date of birth e.g., Chris Brown (footballer, born 1992).

YAGO [12] is ontology, which is part of YAGO-NAGA project that developed at Max Planck Institute for Informatics. YAGO stores information in the form of RDF triples *SPO, where S is a subject, P is a property, and O is an object*. A triple of SPO is called a fact. For example,

Arnold_Schwarzenegger(S)
actedIn(P)
The Terminator(O)

is a fact in YAGO. YAGO collects individual entities and their categories from Wikipedia info-boxes and links to the clean taxonomy of WordNet [16]. YAGO contains 365,372 classes, 2,648,387 entities, and 104 relations [12].

Taxonomies in YAGO are well-formed and meaningful. For example, the YAGO instance Zenedide Zidande is a soccer player and he is a person.

ZenedideZidande→instanceO f →SoccerPlayer→subclassO f →Person

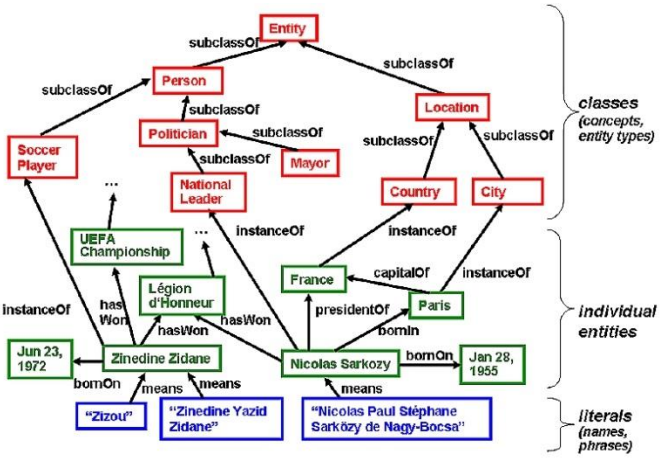

Fig. 1 YAGO structure

The YAGO [12] structure includes of three major parts: classes (concepts, entity types), a set of individual entities, and literals (names, phrases). Figure 1 shows an excerpt of the YAGO knowledge base.

Classes in YAGO are used for grouping similarly entities together and are derived from two main sources: WordNet [16] and Wikipedia [17]. YAGO allows each class to be a subclass of one or multiple classes (YAGO taxonomy) except the root class.

The parent class of YAGO hierarchy is Entity. The relationship *subclassOf* is used to map between superclass and subclass.

Words that have multiple meanings (ambiguous words) could be assigned to several synsets. YAGO considers only nouns and the relationship among synsets (super-subordinate or hyperonymy, hyponymy) to organize taxonomy classes and establish class from the synsets and to link to Wikipedia categories.

The lower classes from Wikipedia categories are mapped to the higher classes from Wordnet by determining the most frequent sense of the head word inWordNet. YAGO allows only the conceptual categories to be a class. The conceptual category is a category that has the head of word in a form of plural. YAGO analyses the head of category name through shallow noun phrase parsing. Most categories in YAGO

are derived from Wikipedia. The depth 4-10 contain 90% of the categories.

A set of individual entities consist of instances such as people, building, or country. YAGO groups entities into six classes:

- 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 Olympics, or world championship tournaments),
- locations, and other.

Each individual entity could be an instance at least one class and connected to its class via relation type.

YAGO deals with ambiguity and synonymy by mapping an alternative name via a relation. The quotes are used to distinguish literals from the entities. The alternative names are derived from Wikipedia redirect pages. The taxonomies in YAGO are merged Wikipedia categories with the concepts from WordNet. In our experiments, we used YAGO, as it provides multiple levels of taxonomy hierarchy.

Freebase [18] is a knowledge base using graph technology to store data. Freebase contains more than 800,000 personal entities and more than 2,000 occupations. Freebase structure includes data describing domains, types, properties, and topics. Freebase contains more than 39 million topics about real-world entities such as people, places, and organizations.

For example, Bob Dylan, Hotel California (song), and love are the topics in Freebase.

Type in Freebase is a set or a category of topics. Each topic can be mapped into one or multiple types. A fit set of properties are used to form a type.

For example, the Football player type may 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., Soccer means the Soccer domain. We can map among topic, type, property, and domain in the form of

subject:predicate:object.

Taxonomy in Freebase has more depths. However, it has thematic categories e.g.,

Milla Jovovich type Film/actor.

This fact means Milla Jovovich is an actor and the actor class is assigned to domain Film but Film is not a person.

DBpedia [19] knowledge base is a multilingual knowledge base by integrating structured data from Wikipedia and 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 [15]. DBpedia classifies a person by occupation and most of occupation classes have only one depth. DBpedia is well-organized but the hierarchy level is not deep enough for crating personal name concepts. We handle this problem by designing a new architecture for occupation taxonomy.

None of the existing knowledge base ontologies is suitable to establish unambiguous personal name concept. Wikipedia categories and Freebase categories are dirty, not well-formed, and bound to a thematic structure. YAGO knowledge base has combined WordNet classes as a backbone and is also connected to Wikipedia categories.

Occupation is an important feature for disambiguation in existing approaches. In our work we argue that using the personal name for disambiguation, as well similarity measures, provides better results.

IV. DISAMBIGUATION ALGORITHM

We define extractor as a task for detecting entity name that are mentioned in a document. Searcher is a task for generating a set of candidate knowledge base entities to each mention name in a document. Disambiguator is a task for selecting a best entity to a mention name when the name is ambiguity.

In order to handle the dirty data in Wikipedia and YAGO in our experimental results, we introduce a new occupation taxonomy architecture based on Web directories and YAGO ontology. Our occupation architecture consists of four layers: Person, Web directory classes, YAGO-WordNet classes, and YAGO-Wikipedia classes.

We use two-steps approach to ensure that our occupation taxonomy is clean, well-formed, and semantically sound. Firstly, we handle the thematic domain problem by changing the context in Web directories before mapping them to our occupation taxonomy (e.g. Arts to Artists). Secondly, we evaluate the whole name in Wikipedia category before mapping it into WordNet class.

SPTM performs an ambiguity name test and selects best personal name entity for each ambiguity mention using three steps:

1. We generate an individual concept to each personal name entity base on our occupation taxonomy architecture.
2. The identification of personal names which have the same root node are merged to create the comparison tree. The comparison tree which has the maximum number of nodes will be considered first.
3. We rank the candidate entities 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. We give different weights to different hierarchical levels. The nodes which have the same level are having equal weight. The candidate entity which has the highest score is selected.

Our approach can return NIL value when we cannot generate the candidate entity to a mentioned name or the conceptualization of identification person differs from other identification person in a web page. We can predict the

possible persons for the NIL value by attaching NIL value and an occupation that has maximum occurrence among conceptualization of identification person in a web page using BingAPI.

We improve the searcher performance by transforming the mentions of variation name formats to a unique format. We boost up the the disambiguator performance by solving two problems in NED:

1) Detecting context similarity that is requiring exact words overlap between the two compared documents and

2) Identifying dirty data in Wikipedia categories and YAGO.

II. FRAMEWORK

We use the facts in YAGO knowledge base to design our personal name catalogue, occupation taxonomy, and personal name concepts. Each fact consists of three parts: subject, property, and object. The advantage of this structure is that it completely separates the catalogue into two parts: information for a searcher component and the information for a disambiguator component.

The first part is personal name surfaces form that stores the collection of referent terms for each person. The personal surface form is established from facts that have the property means, where subject is the reference term and object is personal name entity.

Our architecture contains four layers and is based on YAGO ontology and Web directories.

- Layer 0 is a root node to define that an entity is assigned under this layer is a person.
- Layer 1 is derived from Web directories that are used to distinguish person name in a big picture.
- Layer 2 is derived from WordNet in YAGO.
- Layer 3 is derived from Wikipedia categories in YAGO.

The conceptualization in each personal name entity is created from this architecture. Our results show that occupation taxonomy is the useful feature to distinguish people unambiguously, when their names are ambiguous: only 0.06% of different people with the same name, have the same careers, whereas (4.84 %) of people share both name and professional category.

III. PERSONAL NAME TRANSFORMATION WITH CONTEXT FREE GRAMMAR(CFG)

Data on the internet is heterogeneous; it originates from multiple sources and lacks uniform representation. Personal names that appear on the internet can be variations (giving rise to referential ambiguity). Therefore, the exact match lookup over personal surface form that is used, for example in [1] is insufficient to detect candidate entity.

To deal with referential ambiguity problem, we introduce a context free grammar framework to transform multiple formats of personal name to a unique format. This framework is based on [6] and consists of three components: grammar rules, predicates, and actions. We create sixteen CFG rules to handle referential ambiguity problem (e.g., different order, nick name, alternative name).

The rules are given below.

R1 NAME → PS 1 NAME 2
PS= 1.value;Name = 2.value
R2 NAME → GN 1 MN 2 FN 3
GN = 1.value; MN = 2.value; FN = 3.value
R3 NAME → GN 1 FN 2
GN = 1.value; FN = 2.value
R4 NAME → FN 1 ” ,” 2 MN 3
GN = 2.value; MN = 3.value; FN = 1.value
R5 NAME → FN 1 ” ,”
GN = 2.value; FN = 1.value
R6 NAME → AN 1 AN = 1.value
R7 GN → G GN(I,G) value = G
R8 GN → N NN(I,N,G) value = G
R9 MN → M value = M
R10 GN → Letters 1 Letters = 1.value
R11 MN → Letters 1 Letters = 1.value
R12 FN → F FN(I,F) value = F
R13 AN → A AN(I,A) value = Personal name
R14 PS → P Prefix(I,P)
R15 PS → S Suffix(I,S)
R16 Letters → L value = L

R1 is used to separate a title from personal name. For example, given an input ’George W. Bush,Jr’, R1 produces two outputs including PS = Jr and PN = George W. Bush.

R2-R6 are used to define the location of GN (given name) , MN (middle name) , FN (family name), and/or AN (alternative name) 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 GN = George, MN = W., and FN = Bush.

R7 is used to transform GN to the given name that may be matched with a given name in the personal name dictionary. For example, given an input George, R7 pro-duces an output GN = George.

R8 is used to transform GN where GN is a nick name to a given name that may be matched with a nickname in the personal name dictionary. For example, given an input Bill and R8 produces two given names including GN = William and Willis.

R9 is used to transform MN to the middle name. We do not match the variable M over our personal name dictionary. The output in this rule is returned the original token.

R10-R11 are used to evaluate that GN or MN is an initial letter.

R12 is used to transform FN to the family name that may be matched with a fam-ily name in the personal name dictionary. For example, given an input Bush, R12 produces an output FN = Bush.

R13 is used to transform AN where AN is an alternative name to 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 produce an output AN = George W. Bush.

R14 is used to remove prefix that will be matched over the prefixes in our personal name dictionary.

R15 is used to remove suffix that will be matched over the suffixes in our personal name dictionary.

R16 is used transform Letters to an initial letter. We do not match this token to our personal name dictionary so, the original token is returned.

A personal name transformation with CFG rules has the advantage by boosting the precision and recall in generating a set of candidate entities in each mention. The experimental results show that the framework can solve the problem of referential ambiguity by transform names variations to a unique form. Furthermore, the CFG framework using with Jaro-Winkler metric can boost up the performance in text similarity measurement. A complete overview of our framework is given in Fig. 2.

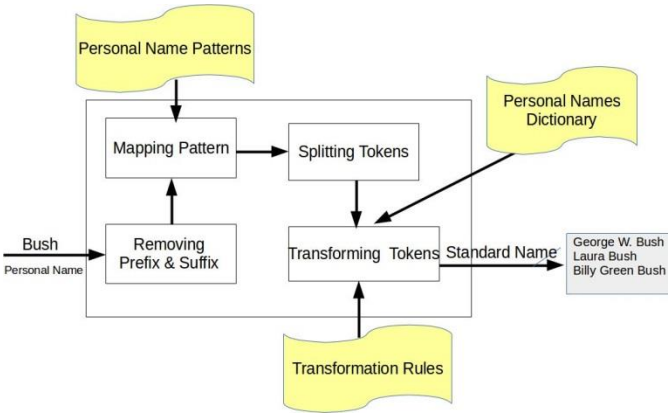

Fig. 2 Personal Name Transformation Framework

IV. New Algorithm for Personal Name Disambiguation with Simple Partial Tree Matching (SPTM)

Our algorithm uses only personal names mention within a web document to disambiguate ambiguity names. We assume that personal names that have appeared within a single web page have the same conceptualization or they are related (spouse or child). We used the layer 1(e.g., Entertainers and Artists, Sportsman, and Politician) to be 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 uses the two tree-matching, 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 higher weight). The comparison tree is created by joining the nodes of identifying the personal name mentions trees which have the same root node. The comparison trees which have the maximum number of nodes will be selected first. The similarity score is calculated from the total matching node multiplied by the total of weighting score plus the related score (the candidate that associated with defining mention will have 100 score). The candidate who has the highest score is selected.

We observed two conditions before using SPTM. First, every node in the tree is unique ,and it can occur once in each tree. Secondly, the multiple levels in tree hierarchy will be flatten 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, start from left to right, one level at a time. Each node contains two important pieces of information : its content and its depth. Figure 3 shows how to remove the level and sort the child nodes. The original tree has four levels after flattening, all child nodes are compacted into one level using top-down

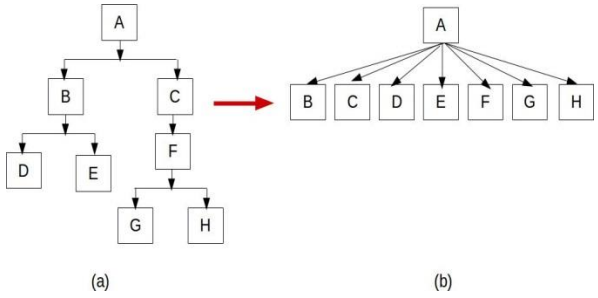

Fig 3. An original tree (a) and a flattened tree (b).

The personal name concept of identifying mentioned names in each web page are next used to construct the comparison tree. Given a set T of identifying personal name concepts, all nodes

in T are be merged into comparison tree if their root nodes are equal. The duplicate nodes are removed and the unique nodes are arranged in ascending order. Fig. 4 shows how to create comparison tree. We start with 3 initial trees; all of them have only two levels. The root node in each tree is matched first; the trees can be merged only if their root nodes are equal. In this example, trees T1 and T2 are merged because they have the same root node A and created the new tree that call comparison tree (Tc). All nodes in two trees are merged and sorted in ascending order.

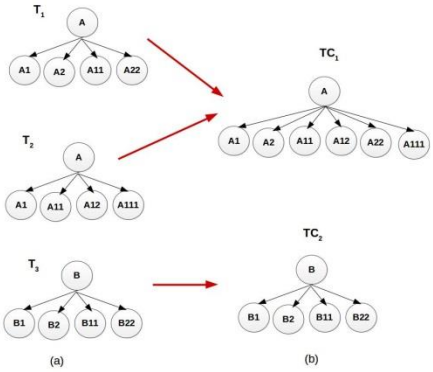

Fig. 4. Building the comparison tree.

SPTM algorithm consists of three steps: comparison tree selecting, two tree matching, and similarity score calculating as follows. First, the comparison tree is selected. The section criterion requires that the comparison tree that has the maximum number of members first. If the member of members of two comparison trees is equal, the comparison tree that has the highest nodes is selected and the algorithm proceeds to Step 2: matching tee. The matching tree consists of two steps: matching the root node and matching the children nodes. In Step 3, the similarity score between the comparison tree and the candidate tree is calculated. Figure 5 gives an example of SPTM algorithm. The two trees Tc and T1 have the same root node A. This means that the two trees have the same concept and can be matched. The matching nodes are {A, A_1,A_{11}}.

Our model predicts the NIL value under two conditions: (i) we cannot generate a candidate entity for a mentioned name. (ii) an mapping entity has a different ancestor node and/or distantly related to the existing identifying entities in a Web document. This is different from existing approaches. For example, in [2] the NIL value is detected by creating one entity calling out to predict NIL value when the similarity score lower than a fix threshold. In [7] a SVM ranker is used to predict the NIL value. The NIL value is then returned if a set of candidate entity is empty or a candidate entity does not in the top of Google returning values.

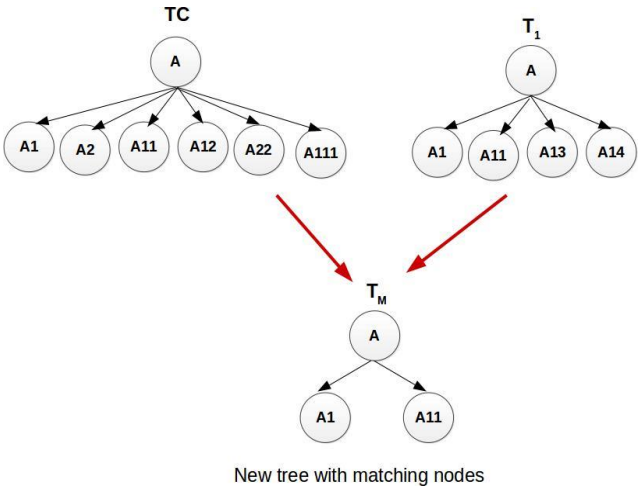

Fig. 5 Matching in SPTM

V. EXPERIMENTAL RESULTS

In order to evaluate the effectiveness of the framework for linking a mentioned name to a real world entity, we measure the performance of the system into two criteria:

1. The searcher performance of generating candidate entities.

2. The disambiguator performance of entity linking.

We used two data sets that provided personal names, their alternative names and their professional categories: YAGO version 2.3.0 and our data catalogue, in which we collected web documents from three websites:

1. http://www.today.com

2. https://uk.yahoo.com

3. http://www.msn.com/en-gb/

Our catalogue contained 107,058 persons, 332 profession categories including Person (a root category), 105,604 personal concepts, 145,638 personal surface forms, and 4,203 personal relations. The evaluating data sets contained 992 mention names, 119 referential ambiguity names, and 114 lexical ambiguity names.

The experimental results demonstrated that our proposed approach achieved excellent performance over real-world data sets. The system takes a single web-page as an input. It 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-occurrence from the identifiable personal names via BingAPI to produce a top ten links of possible persons.

The system is developed using Apache/2.4.12 (Ubuntu), PHP Version 5.6.11-1ubuntu3.1, and mySQL 5.6.28-0ubuntu0.15.10.1. The personal name matching has one active actors and two cooperating system API.

The effectiveness of our proposed model is empirically validated through experimental assessments with real-world data sets. In our data set only 18 personal names are identified as NIL, which amounts to 0.13% of all values. The overall quality of data transformation is extremely high, the framework returned the NIL value in less than 1% of all cases. Subsequent analysis of the returned NIL values revealed that all of the 18 returned NIL values are alternative names.

We evaluated two text similarity functions: Jaro-Winkler and cosine similarity on 168 alias names of 25 personal names. PNTF combined with Jaro-Winkler produced the highest accuracy but shows smaller number of improvement in PNTF with cosine similarity methods.

Searcher's performance: The candidate count is 1.6, i.e one mention name have average 1.6 candidate entities. This is not too high and is reducing the workload for disambiguator. The candidate precision and recall are both over 80%. This means the searcher has high performance in generating candidate entities. The performance in NIL value returning is also very high, it is nealy 100%.

Disambiguator's performance. The percentage score for overall linkable score was 91.82%. The effectiveness in handling lexical ambiguity is 72.07%. This study produced the precision result in entity linking similar to Cucerzan [1] that obtained precision of 91.40%. Most of the incorrect in lexical ambiguity is a result of short name mentioning. From 114 lexical ambiguity mentions, 42 mentions are short names (e.g., Timberlake, Ronaldo, Fernando and the system returns the correct answers only in 42.86% of such cases. Remarkable is that the accuracy improves to 85.95% if the lexical ambiguity is a full name.

VI. RELATED WORK

Frameworks that use extractor, searcher, and disambiguator have been previously proposed. For example studies presented in [1, 2, 3] focused on precision targets for the disambiguator, whereas [4] determined that the searcher component is more important and has a much stronger effect on performance than the disambiguator task because of the potential for referential ambiguity in personal names.

Most real word data is dirty, incomplete, or imprecisely formatted [5] and as a result, we cannot use exact matched lookup over a knowledge base to process a set of candidate entities for a mentioned personal name as it is done in [1, 2]. For example. exact-matched lookup, when the formatting is different will produce dissimilarity: e.g. Barack Obama and *Barak Obama* will be dissimilar, even though they are the same personal name. We use a technique similar to the one introduced in [6] as a pre-processing step, intended to transform personal name variations to a uniform representation. Alternative techniques exist. For example, [4] this problem was solved by using uniform weight scheme score to rank the candidate names. The candidate which has the highest score is selected. However, this method suffers when a context can be both the given name and the last name because the scores are equal. To circumvent this, in our work, a regular grammar in personal name structure is used for segmentation the components in personal name (the location of first name, middle name, or last name) and these components are matched directly to the personal name dictionary.

Methods proposed for the disambiguator task in [1, 2, 3, 7, 9, 10, 11, 13, 14] handle lexical ambiguity in different ways. Those methods can be divided broadly into two types: context similarity methods and combining methods. The first one is textual similarity method called a bag of word or context similarity. Context similarity uses the terms around the entity mention and the Wikipedia page that related with entity to measure similarity between two entities. The limitation in context similarity method is requiring exact word overlap between the two compared texts, which may become an over strict constraint because of flexible usage in natural language.

The empirical results demonstrated that our framework is effectiveness in personal name linking. Hence the effective in short name linking indicates that only personal name concepts are insufficient for mapping a short name ambiguity to the real-world entity. There-fore, in our future work we would like to improve the performance in short name ambiguity. We plan to mine the evidences across the document that introduced in [21] using the connection between persons.
.

VII. CONCLUSION

In this paper we describe a framework for personal name disambiguation which consists of extractor, searcher and disambiguator and achieves a high percentage of correct disambiguation. The extractor component uses Alchemy API that uses to extract personal name from a web document. This tool has overall good ability to extract the personal names that are mentioned within a web-page. The searcher component contains two main functions: personal name transformation and candidate generator and combines three components: personal name transformation with context free grammar rules, personal surface form, and text similarity function.

REFERENCES

[1] Curcezan, S. Large-scale named entity disambiguation based on Wikipedia data. In Proceedings EMNLP-CoNLL, pages 708–716, 2007.

[2] Bunescu, R. and Pasca, M. Using encyclopedic knowledge for named entity disambiguation. In Proceedings of EACL, pages 9–16, 2006.

[3] Shen, W., Wang, J., Luo, P., and Wang, M. Linden: Linking named entities with knowledge base via semantic knowledge. In Proceedings of WWW '12, pages 449–458, USA. ACM, 2012.

[4] Hachey, B., Radford, W., Nothman, J., Honnibal, M., and Curran, J. R. Evaluating entity linking with wik-ipedia. Artificial Intelligence, 194(0):130 – 150, 2013

[5] Christen, P., Churches, T., and Zhu, J. X.. Probabilistic name and address cleaning and standardisation, 2002.

[6] Arasu, A. and Kaushik, R. A grammar-based entity representation framework for data cleaning. In Proceed-ings of SIGMOD, pages 233–244, ACM, 2009.

[7] Dredze, M., McNamee, P., Rao, D., Gerber, A., and Finin, T. Entity disambiguation for knowledge base pop-ulation. In Proceedings of COLING , pages 277–285, 2010.

[8] Li, Y., Wang, C., Han, F., Han, J., Roth, D., and Yan, X.. Mining evidences for named entity disambiguation. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, pages 1070–1078, 2013.

[9] Yang, W. Identifying syntactic differences between two programs. Softw. Pract. Exper., 21(7):739–755, 1991.

[10] 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, 2006.

[11] Mann, G. S. and Yarowsky, D. Unsupervised personal name disambiguation. In Proceedings of Conference on Natural Language Learning at HLT-NAACL- Volume 4, CONLL, pages 33–40, 2003.

[12] Demidova, E., Oelze, I., and Nejdl, W. Aligning freebase with the YAGO ontology. In Proceedings of ACM International Conference on Information & Knowledge Management, CIKM, pages 579–588, 2013.

[13] Han, X. and Zhao, J. Web personal name disambiguation based on reference entity tables mined from the web. In Proceedings of International Workshop on Web Information and Data Management, WIDM, pages 75–82, 2009.

[14] Raman, V. and Hellerstein, J. M. Potter's wheel: An interactive data cleaning system. In Proceedings of the International Conference on Very Large Data Bases, VLDB, pages 381–390, Morgan Kaufmann Publishers, 2001.

[15] Lehman, J et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web, 6(2):167–195, 2013.

[16] WordNet https://wordnet.princeton.edu/

[17] Wikipedia https://en.wikipedia.org/wiki/Main_Page

[18] FreeBase https://en.wikipedia.org/wiki/Freebase

[19] DBPedia http://wiki.dbpedia.org/

[20] Bilenko Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., and Fienberg, S.. Adaptive name matching in information integration. Intelligent Systems, IEEE, 18(5):16–23, 2003.

[21] Elmagarmid, A., Ipeirotis, P., and Verykios, V. Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering 19(1):1–16, 2007.