“Tell me more" using Ladders in Wikipedia

Siarhei Bykau
Bloomberg LP
sbykau@bloomberg.net

Jihwan Lee
Purdue University
jihwan@purdue.edu

Divesh Srivastava
AT&T Labs-Research
divesh@research.att.com

Yannis Velegrakis
University of Trento
velgias@disi.unitn.eu

ABSTRACT

Knowledge is typically specified using triples in which some object entity is associated with a role, identified by the combination of a subject entity and predicate, e.g., Barack H. Obama is the object entity currently associated with the role of “President of the United States”. Due to the dynamic nature of the real world being modeled, the object entities that are associated with a role naturally change over time, e.g., Barack H. Obama took over from George W. Bush in the role of the President of the United States. Extracting the history of such changes over time is fundamental for building accurate and complete knowledge bases. This paper studies the problem of *role linkage*, which seeks to identify and link together the different object entity occurrences that are used in the same role over time; it differs from entity linkage that seeks to link together occurrences of the same entity. We focus our investigation on Wikipedia articles, where each subject entity has its own page, roles are expressed both in the structured infobox and unstructured text, and all page changes are available.

We formalize the goal of role linkage in Wikipedia as producing a *ladder*, where the rungs capture role linkage within a page revision and the rails capture role linkage in the infobox and the text across revisions. We provide novel algorithms to efficiently construct ladders, making use of supervised learning techniques to account for the different kinds of edits that happen in Wikipedia articles. We empirically evaluate our technique for constructing ladders on multiple Wikipedia datasets against baseline techniques, including one based on a learning-based technique to populate infoboxes and another based on Wordnet, and demonstrate the superiority of our techniques.

1. INTRODUCTION

Knowledge bases are repositories encoding knowledge in structured form, so that they can be better organized, managed, and queried using traditional database languages. Most knowledge bases nowadays are generated from manually curated datasets. A classical example is DBpedia[1] which is generated from Wikipedia data. Wikipedia[2] is one of the largest collections of knowledge and probably one of the most valuable resources of our time.

Despite the fact that the DBpedia content is structured, enabling it to be easily exploited by machines, it is still used mainly by humans to explore structured Wikipedia data. Users who know some interesting piece of information and would like to learn additional facts related to it find DBpedia to be an ideal resource. They may not always know exactly what else they would like to learn, but they can browse the structured data connected to the fact they already know, and decide which of these structured data are of interest to them.

Most knowledge bases, including DBpedia, that extract knowledge from Wikipedia focus only on its structured part. Each Wikipedia page has information about some real-world entity or concept and consists of some free text and some structured part (known as the Infobox). The Infobox is composed of a sequence of attribute name-value pairs, each describing some property of the entity that the page is about. The value of an attribute is often some other entity, and the attribute name describes the role that entity plays for the entity of the page. This structured information is used to generate the knowledge base structures. Consider, for instance, the Wikipedia page about the Cleveland Browns[3], a professional American football team based in Cleveland, Ohio. In its Infobox, one can find an attribute with name `home field` whose value is the entity `FirstEnergy Stadium`, the `owner` whose value is the entity `Jimmy Haslam`, the `president` whose value is the entity `Alec Scheiner`, the `head coach` whose value is the entity `Mike Pettine`, and so on. A knowledge base could, for instance, model these facts as RDF triples connecting the resource `Cleveland Browns` with each one of the aforementioned entities and a property named after the corresponding attribute name.

In this work we deal with the problem of exploiting Wikipedia in order to find related information about a specific given fact (i.e., a “tell me more” service). We observe that in addition to the related information obtained from structural data as recorded in knowledge bases, there is a lot of additional related information that can be found only in the textual part. We do not use knowledge extraction techniques to create structured information from the text; while such approaches exist, they have not yet reached their full potential [18]. Instead we aim at recognizing in the text those parts that contain a specific given fact. Once we have identified all these appearances, we retrieve the textual information related to it. Of course the concept of “related” is broad and may be subject to many different interpretations. Furthermore, since we deal with text data, the associations between different pieces of information that appear in the text may be vague and of differing interest to different users. Nevertheless, since the target audience of a “tell me more" service is a real end user interested in exploratory search, we claim that we can simply retrieve the text that has been identified as containing the given fact, and present it to the user. Hence, once we identify an appearance of the given fact in the text, we return as the related information the text that surrounds it.

We assume that the fact that we have at hand, i.e., the one for which we want to find related information, is already recorded in the Infobox of a Wikipedia page as an attribute name-value pair.

[1] https://wiki.dbpedia.org
[2] https://www.wikipedia.org
[3] https://en.wikipedia.org/wiki/Cleveland_Browns

Thus, such information can be fully specified by a triple $\langle s, p, o\rangle$, where s is the entity that the Wikipedia page is about, p is the name of the attribute, and o is the value of the attribute. The challenging problem then is to identify the parts of the text of a page that contain this information and retrieve its surrounding text. To do so, one needs to deal with two dimensions: the heterogeneity within the page and the heterogeneity across time.

The first dimension is the *spatial* dimension and aims at identifying appearances of the given Infobox entry throughout the Wikipedia document of an entity s. Consider a given Infobox entry $\langle p, o\rangle$ which indicates the fact that attribute value o is related to the entity s under a role p. The entity o may appear multiple times in the text of the page s, but not all these appearances play the role p. Consider, for instance, the case of `Mike Pettine` who has the role of `head coach` (as the corresponding entry in the Infobox indicates). As a first step, one can look at the appearances of Mike Pettine in the text. However, not all the sentences mentioning Pettine are relevant to this role. The sentence *"On January 24, 2014, the Browns hired* `Mike Pettine` *as the 15th full-time head coach in team history (and seventh since the team's return in 1999)"*, clearly does so. On the other hand, the sentence *"Previously* `Mike Pettine` *was a defensive coordinator for the NY Jets, then Buffalo"*, refers to Pettine's role as coordinator for different sports teams before becoming the coach. To identify Pettine's role in these sentences, one needs to analyze the text surrounding mentions of Pettine in the text to discover his role there. We refer to this type of linking between some information mentioned in the Infobox, and the same information in the text as spatial linkage.

The second dimension is the *temporal* in which we identify the portions of previous and subsequent Wikipedia page versions of the document, that describe the same kind of information as identified through spatial linkage. The value of the property `head coach` in the Infobox was `Rob Chudzinski`, and in a subsequent version it was changed to `Mike Pettine`, indicating that Pettine assumed the role of head coach replacing Chudzinski. However, once the value of the head coach was updated, no link was maintained to indicate that Pettine succeeded Chudzinski in the role of head coach. Considering the text, in one version of the Cleveland Browns page there is a sentence `Rob Chudzinski is serving as the head coach`, which is in some subsequent version eliminated and a new one is added that reads: *"On January 24, 2014, the Browns hired* `Mike Pettine` *as the 15th full-time head coach in team history (and seventh since the team's return in 1999)."* These two sentences indicate the evolution of the head coach role in the team and they can be used to provide a temporal linkage from Chudzinski to Pettine.

As it has probably become clear, spatial and temporal linkages are not based on the entities alone, as in most linkage mechanisms, but on their semantics as well, i.e., the role the entities are used for. We refer to this novel notion of linkage as *role linkage* to distinguish it from entity linkage. For role linkage we see every piece of data as consisting of two parts: the actual value and the role. In the case of the infobox, the role is the attribute name. In the case of the textual part, we analyze the surrounded text to identify the role. To instantiate the concept of role linkage, we introduce a new data structure called *ladder*. Similar to real-life ladders, our ladder consists of rungs and rails. The rungs span between infobox entries and parts of the text, indicating the spatial links. The rails of the ladder span across different temporal versions, indicating the temporal links.

The specific contributions of our paper are as follows.

1. We develop a new method for answering "tell me more" queries in the DBpedia/Wikipedia context.

2. We introduce and formally define the problem of *role linkage*, which aims at finding and linking references in the structured (infobox) part and the unstructured (text) part that are used under the same role.

3. We introduce a novel data structure called *ladder* which spans across time (revisions) and space (structured and unstructured parts) connecting references to real world entities that are used under the same role.

4. We design and implement an algorithm for efficient ladder extraction from the revision history of a Wikipedia page based on supervised learning and we apply it to role linkage.

5. Using four Wikipedia-based datasets as well as crowds sourced ground truth labels, we report the effectiveness and efficiency of the ladder methods in addressing the challenges of role linkage. The experimental results show that the use of ladders leads to a significant gain in precision without sacrificing recall.

2. PROBLEM STATEMENT

We assume a countable set of names $\mathcal{N}$, entities $\mathcal{E}$ and atomic values $\mathcal{S}$. An *attribute* is a pair $\langle p, o\rangle$ where $p{\in}\mathcal{N}$ and $o{\in}\mathcal{E}$. The n and v are referred to as the attribute *name* and *value*, respectively. The set $\mathcal{A}{=}\mathcal{N}{\times}\mathcal{E}$ is the set of all possible attributes.

A *text* is a finite sequence of entities and atomic values. Let $\mathcal{T}$ be the set of all possible such sequences. We will denote as $txt[i]$ the i-th element of a text txt, and as $pos(w)$ the position i of an element w in a text, i.e., $txt[i] = w$ if and only if $pos(w){=}i$. We also define the *link* of a text txt, and denote it as txt^{link}, to be the sequence of all the entities in the txt in the order they appear in txt. Each entity in a text has its own *neighborhood* which is the sequence of entities or values that surround it. A neighborhood has a radius that determines how many entities or values before or after to consider as neighborhood.

A *page* is a tuple $\langle e, Inf, txt\rangle$ where $e{\in}\mathcal{E}$, referred to as the *page entity*, $Inf{\subset}\mathcal{A}$, referred to as the *infobox* (or the *structured* part) of the page, and $txt{\in}\mathcal{T}$, referred to as the *text* (or the *unstructured* part). We consider the information provided in the infoboxes of the Wikipedia pages as facts. Formally, a *fact* is a triple $\langle s, p, o\rangle$ for which there is a page $\langle s, Inf, txt\rangle$, with $\langle p, o\rangle{\in}Inf$.

Pages are evolving in time through modifications that take place in their text and/or their infobox. Every time a modification takes place, a new *version* of the page is created. The sequence of versions that a page has gone through in time constitutes its *history*. By definition, all the versions have the same page entity. Furthermore, at any given point in time, there can be no more than one page with the same page entity.

Entities in the infobox or in the text of pages indicate some form of relationship between them and the page entity. We refer to this form of relationship as *role*, since it specifies the reason for the entity's appearance in the page. For an entity appearing as value in an attribute in the infobox of a page, the role is the pair of the page entity s and the attribute name p, and in what follows, it will be denoted as $s.p$.

The idea of a "tell me more" service is to identify in the the text of the current and the previous versions of a Wikipedia page all these occurrences of entities that are used under the same role as the role of an entity that is provided as argument. In other words, given a fact $\langle s, p, o\rangle$, we need to identify every entity o' in the text part of the Wikipedia page of the entity s, that appear under the role $s.p$, i.e., the same role as the entity o. We refer to this problem as *role linkage*.

For modeling the results of the role linkage we need a structure that can capture and keep linked together all these entity occurrences in the infobox and in the text across the different versions of

the page in time. In a specific version of a page, under a given role, there can be only one entity in the infobox but many in the text.

A *rung* is a pair that links an entity of a given role in the infobox of a page with an entity in the text with the same role. A sequence of entities in a sequence of versions of a page, all of the same role but with each one in a different version, form a *text rail*. Similarly, infobox entities in different versions of the Wikipedia page but of the same role, are connected together trough a structure called an *infobox rail*. Since there are many entities of the same role in the text of a page, there are many ways they can be combined across different versions, thus, there are many different text rails that can be created. Finding the right way to combine them to form the right text rails is the challenge we solve in the next section. The infobox rail, the text rail and the rungs that have their endpoints in these two respective rails, all of a same role, form a *ladder* structure for that role. More formally:

DEFINITION 2.1. *Given a wikipedia page and role r, a* ladder *for r is the tuple $\langle IR_r, TR_r, Rng_r\rangle$ where IR_r, and TR_r are the infobox and text rails, respectively, of r, and Rng is a set of rungs $\langle b, e\rangle$, for which $b \in IR_n$ and $e \in TR_n$.* ∎

EXAMPLE 2.2. *In the Wikipage of Cleveland Browns the infobox rail $IR_{head\ coach}$ consists of the entities* `Romeo Crennel`, `Eric Mangini`, `Pat Shurmur`, `Rob Chudzinski`, *and* `Mike Pettine`, *that have all served as head coaches, i.e., that have appeared as head coaches at different versions. The abstract section mentions the entity* `Mike Pettine`. *Looking at previous versions of the page one can observe that in the specific sentence, the names* `Pat Shurmur` *and* `Rob Chudzinski` *appear instead, indicating the name of the head coach. These names together also forming a text rail. The name* `Mike Pettine` *is appears also in a later section of the page. At that section but in previous (different) versions the names of* `Pat Shurmur` *and* `Rob Chudzinski` *were mentioned. The latter two with this second appearance of* `Pettine` *form another text rail. Each of the two text rails couples with the infobox rail, and together with rungs that connect the entities in the text and in the infobox rail in the same version form a ladder. Thus, in the specific example, two different ladders will be created, each for the role Cleveland Brown.headcoach*

If from the Wikipedia pages one is able to construct the ladders, then they can be exploited to support any "tell me more" functionalities, by returning the text surrounding the entities in the text rail endpoint of the rungs of the ladder.

3. LADDER EXTRACTION

To generate the ladders we exploit the fact that, in Wikipedia, synchronized changes, i.e., changes in both the infobox and the text, is the norm rather than the exception. By noticing such changes we are able to identify assistance between entities in the infobox and the text. We have identified two patters of synchronized changes that are typically taking place: the *evolution* pattern is the case in which an update of an entity in the structured part, is accompanied by a same change in the unstructured part and the *adjacency* pattern is the case when the structured part is updated whereas the unstructured part appends a new value.

3.1 Infobox and Text Rail Extraction

The Infobox rail generation is straight forward. For a given role $s.p$ we need to collect all the entities that appear as values in the attribute p in any version of the Wikipedia page of the entity s.

Next, we extract the various text rails for the different roles. Usually, each entity in a role persists for some time, i.e., is replicated in a number of consecutive versions. For example, the history section of Cleveland Browns contains `Pat Shurmur` as a head coach for almost 2 years. To extract such long rail we take advantage of the following observation: If we replay the edit history of a page entity we notice that its versions do not change dramatically from one version to another, but only small gradual changes take place. Thus, we start with an initial version and gradually try to extend our rail towards the future versions of the page.

Algorithm 1: **Rail Extraction Algorithm**

Input: $s.p$
Output: $IR_p, \{TR_p\}, Rng$
(1) $res \leftarrow \emptyset$
(2) $visited \leftarrow \emptyset$
(3) $Rng \leftarrow$ GETRUNGS(s, p)
(4) **foreach** $rung \in Rng$
(5) **if** $rung \notin visited$
(6) $head \leftarrow rung$
(7) $TR_p \leftarrow \{head\}$
(8) **while** $head \neq null$
(9) $candidates \leftarrow \emptyset$
(10) **for** $i = ver(head) + 1$ **to** $ver(head) + n_{ahead}$
(11) **foreach** $j \in [pos(head)-r, pos(head)+r]$
(12) $candidates \leftarrow candidates \cup txt_i^{link}[j]$
(13) $features \leftarrow$ GETFEATURES$(candidates)$
(14) $head \leftarrow$ DECISIONTREECLF$(features)$
(15) $TR_p \leftarrow TR_p \cup head$
(16) $visited \leftarrow visited \cup head$
(17) $res \leftarrow res \cup TR_p$
(18) $IR_p \leftarrow$ GETINFOBOXRAIL(s, p)
(19) $return\ IR_p, res, Rng$

Algorithm 1 presents the extraction process of a set of $\{TR_p\}$ as well as IR_p and a set of rungs Rng for the given role $s.p$ using decision trees. We start by initializing a set of all text rails, res, to an empty set in line (1). We keep track of the entities which we processed with the help of $visited$ which we initialize to an empty set in line (2). Then we obtain an initial set of entities which potentially can belong to the role $s.p$ by calling GETRUNGS(s, p) in line (3). GETRUNGS(s, p) returns all entities in some version of s which are the values of the given attribute $s.p$ in the attributes of s, i.e., $o \in$ GETRUNGS(s, p) if $\exists\langle s, Inf, txt\rangle$, where $\langle p, o\rangle \in Inf_i$ and $e \in txt_i^{link}$.

For each unvisited $rung$ (lines (4-5)) we build a text rail which starts with that $rung$. We set $head$ of the rail to $rung$ in line (6) and we initiate a new text rail TR_p in line (7). Then we grow the rail until we find that it is broken. More specifically, we construct a list of possible candidate entities to which the rail can potentially grow (lines (9-12)). Each candidate entity may come from one of the n_{ahead} successive versions and can have the position within r entities of the original entity position. Note, the function $ver()$ return the revision id of an entity. In line (12) we call the function GETFEATURES() which extracts features associated with every candidate (see Table 1 for the details) and then we apply a decision tree classifier to find the most probable candidate. If that candidate is below a given threshold then DECISIONTREECLF() returns $null$. The most probable candidate becomes a new head of the rail (line (14)). We mark it to the current text rail TR_p in line (15) and mark as visited in line (16). Once a rail is broken (i.e., its $head$ is $null$) we add it to res (line (17)). We repeat the same routine while all $rungs$ are not processed. At the end of the algorithm res contains a set of text subrails which belong to the given role $p.n$. Finally, in line (18) we extract the corresponding infobox rail by calling GETINFOBOXRAIL(s, p) which uses $s.p$ as the role identifier. We return IR_p, res and Rng in line (19).

Algorithm 1 and the entity features described Table 1 are flexible

type	name	description
bool	same	is a substitution
num	jaccard	jaccard distance between neighborhoods
num	pos_diff	position change
num	rev_diff	distance between versions
bool	flink_tngb	$head$ found in candidate's neighborhood
bool	tlink_fngb	candidate found in $head$'s neighborhood

Table 1: Entity features used for text rail extraction

enough to handle various types of errors and edits across versions. (a) *Correction edits* (clarification edits) occur when the authors fix some issue in text (e.g., a new fact added). Usually, those kinds of edits are not massive and our rail extraction handles them effectively since it tolerates small perturbations of the neighborhood (the jaccard feature). (b) *Synonymous or equivalent edits* replace some entity (or words) with a synonym. Our method detects those edits as substitutions and learns them with the help of the same feature. (c) *Adversarial edits* (or vandalism) are intentionally harmful updates which can potentially affect the entire page. We handle them by allowing the rail extraction to skip those revisions by learning the rev_diff parameter (a reverted revision is likely to have similar or identical neighborhood). Overall, our rail extraction effectively handles all other kinds of edits (temporal changes, typos, extension edits, and so on) and is shown to produce high quality robust rails.

3.2 Adjacency

Another pattern which allows us to discriminate between relevant and non-relevant text rails is the adjacency.

EXAMPLE 3.1. *The history section of Cleveland Browns talks about the head coaches of Cleveland Browns and enumerates them in the order they were introduced, i.e.,* Pat Shurmur,Rob Chudzinski, Mike Pettine*. If we replay the version history we will see that first* Pat Shurmur *appeared in the text, then* Rob Chudzinski *was appended as its successor and, finally,* Mike Pettine *was added as the current head coach.*

In other words, an evolution can unfold either in time, i.e., entities substitute each other within the same rail but at different points of time or in space, i.e., they appear one after another in the unstructured part of a page entity. In order to capture adjacent rails we first introduce a number of definitions.

First of all, we need to measure how close two rails are to each other across multiple versions. Note that rails may only partially overlap, i.e., there are versions where one rail exists and another does not. Intuitively, rails are closer to each other if they co-existed for more versions and the total distance between them is small. Formally, we define the adjacency distance, adj, of two rails TR'_p and TR''_p as follows.

$$adj(TR'_p, TR''_p) = \frac{|\{p_i\}|_{e_i,e_j \in txt_i^{link} e_i \in TR'_p, e_j \in TR''_p}}{log \sum_{\substack{p_i | e_i,e_j \in txt_i^{link} \\ e_i \in TR'_p, e_j \in TR''_p}} |pos(e_i) - pos(e_j)|} \quad (1)$$

adj is the ratio between the number of versions where TR'_p and TR''_p co-existed and the logarithm of the cumulative absolute difference in their positions. We say that two rails TR'_p and TR''_p are α-adjacent if $adj(TR'_p, TR''_p) = \alpha$. The higher α the closer two rails to each other.

The fact that two rails are close to each other does not necessarily mean that they should be considered adjacent. We notice that the adjacency implies that (i) there are no repeated entities in the adjacent rails in the same version, e.g., it is not possible to have two close rails which have Pat Shurmur as the value within the same version; and (ii) the order of entity introductions should be the same as in the corresponding infobox rail, e.g., the adjacent text rails should appear as the Pat Shurmur rail, than the Rob Chudzinski rail, and finally the Mike Pettine rail. Those properties are known as the *disjoint rung constraint* (DRC) and *value precedence constraint* (VPC), respectively.

3.3 Ladder Extraction Algorithms

In this section we present two algorithms for ladder extraction, namely the Ladder Extraction Algorithm and Ladder Wordnet.

Ladder Extraction Algorithm. First, we initialize the text rail set (which consists of all relevant text rails) to an empty set. Then we extract all relevant text rails to the given role, its infobox rail and a set of rungs by calling Algorithm 1. For each pair of text rails we compute adj distance and use it to cluster text rails into bundles. For that we use correlation clustering [1] since the distance may not exist for all pairs of text rails (e.g., for those text rails which don't have overlapping versions). As a result we produce groups of rails which are adjacent based on the adj distance. As we discussed in Section 3.2 not all adjacent text rails are relevant to the corresponding infobox rail and therefore we need to make sure that *DRC* and *VPC* are satisfied for each bundle. So we enforce *DRC* and *VPC* by checking if bundle's text rails satisfy the corresponding constraint and if not then the bundle is split back into individual text rails (i.e., each split rail become a new bundle). Finally, for each bundle we check if it contains more than one distinct entity and if so than we consider that bundle relevant and we add it to the result set. Those relevant bundles are the output of Ladder Extraction Algorithm.

Ladder Wordnet. Ladders use only the temporal dynamics of entities in the space and time dimensions to do role linkage. We can further enhance the ladder method by considering semantic aspects of entities and a given attribute. Since an attribute name describes what an entity actually talks about, if the attribute name is found in the neighborhood of the entity, then it brings us a strong signal that indicates the entity is highly likely to be relevant to the role of the attribute [6]. Due to the fact that different words may have the same concept and different concepts may be interconnected in terms of semantic and lexical relations, it would not be meaningful to seek the exact match of the attribute in the neighborhood of the entity. To capture such semantic binding between an attribute name and words in the neighborhood of an entity, we leverage Wordnet [15] by computing the entity relevance score as follows:

$$S(s.p, e) = \sum_{w_i \in ngb(e)} (\rho(w_i, e) * \tau(w_i, s.p))$$

where $\rho(w_i, w_j) = e^{-0.1 * d_{ij}}$ and d_{ij} is the word distance between those two words in a sequence of words and $\tau(w_i, w_j) = 1/s_{ij}$ and s_{ij} is the distance of shortest path between w_i and w_j in the Wordnet graph.

4. EXPERIMENTS

4.1 Experimental Setup

Datasets. In the experimental evaluation we use the Wikipedia real world datasets. Initially, we collected thousands of Wikipedia pages and extract 19,511 distinct page-role pairs from the set of pages. We then partitioned them into 8 subsets according to the following three parameters: n_trans - the number of transitions within a role, ambiguity - the level of ambiguity of a role and

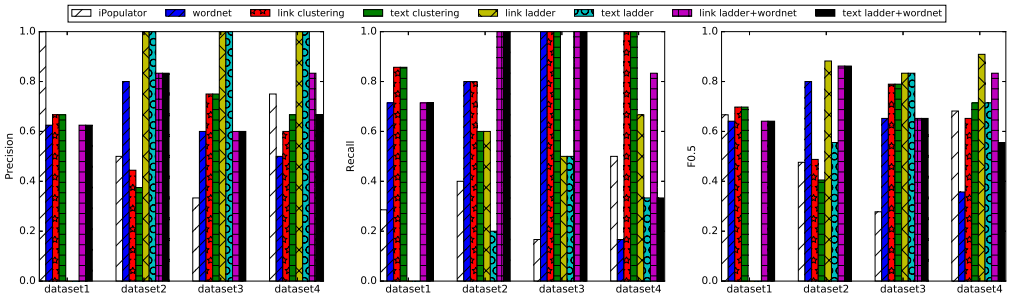

Figure 1: Effectiveness of the `iPopulator`, `wordnet`, `clustering`, and `ladder` methods

dataset	n_rev	ambiguity	n_trans	no. of pages
dataset1	low	low	low	9,241
dataset2	high	low	low	2,708
dataset3	high	low	high	597
dataset4	high	high	high	218

Table 2: Wikipedia datasets

`nrev` - the number of revisions of a page. After discarding subsets that do not contain any pages in the population of Wikipedia pages, we finally used 4 subsets presented in Table 2.

Crowdsourced Ground Truth. Answering "tell me more" queries requires us to correctly identify relevant and non-relevant entities in the text for a given pair of page and role. We thus collected ground truth using Amazon Mechanical Turk (AMT)[4]. Since we have a large number of input entities for role linkage (e.g., a typical page consists of thousands of entities) we need to carefully choose which entities to validate. We first randomly sample pages from each of the datasets and run all the methods over the sampled pages. Then each method returns a set of entities that are considered as relevant to given a pair of page and role. We took the union of all the sets of entities and extracted entities on which there is at least one disagreement among all methods. We used AMT to collect ground truth labels on those entities and provided crowds with the following information. The description explains what they are expected to do along with a page title and timestamp when it was created. The paragraph is one that actually appears in the Wikipedia page and includes a highlighted entity to validate. For each of the entities to validate we ask three crowd members to determine whether it is referring to a given role in the presented paragraph. We build the ground truth for the entities based on the majority voting of the collected answers for the effective analysis in Section 4.

Competitive Methods. We compare the following methods in the experimental evaluation. `iPopulator` [11] is a system that automatically populates infoboxes of Wikipedia articles by extracting attribute values from the article's text. Although its objective is different from ours, we adopt the idea of `iPopulator` to determine whether a given entity is relevant to a certain role which is specified by an attribute. `wordnet` leverages entity's relevance score which is determined by the distance from the entity to an attribute name in the text and the similarity between the attribute name and neighboring words of the entity. `link/text clustering` is based on the clustering of input entities based on their link/text neighborhood (i.e., surrounding entities). We use the jaccard distance of neighborhoods to measure the similarity between entities. As an implementation we use `DBSCAN` [5] because the algorithm is naturally able to capture the gradually changing neighborhoods. `link/text ladder` uses the ladder structure based on the link/text neighborhood. `link/text ladder wordnet` determines an entity's relevance based on disjunction of the results of the `link/text ladder` and the `wordnet` methods.

[4]https://www.mturk.com/mturk/

4.2 Effectiveness

The goal of this experiment is to see how effectively each of the considered methods is able to identity which entities are relevant to a given page and role on the real datasets presented in Section 4.1. We report the effectiveness using precision, recall and $F_{0.5}$ on the ground truth collected from AMT.

In dataset1, all methods perform equally except `ladders` with respect to $F_{0.5}$ and `ladders` have the precision and recall of 0. That is explained by the fact that `ladder` works only for pages which contain value evolution, i.e., they either capture substitutions or adjacent rails. However, dataset1 is sampled in a way that it has pages with few revisions, no ambiguity and no evolution and thus there cannot exist relevant ladders.

Looking at the `text ladder` and `link ladder`, we observe that they have very high precision but relatively low recall. That suggests that `ladders` have low false positive rate, i.e., if ladders captured evolution and/or adjacency then it is highly likely that the entities which belong to that ladder are relevant. From the `ladder wordnet` results we observe that the method successfully combines two different (and complementary) ideas that exhibit a good precision/recall trade-off which results in high $F_{0.5}$ scores. The best results are achieved with the `link` neighborhood. Regarding the clustering baseline, we notice that it has a high false positive rate because it groups together many irrelevant entities. `iPopulator` does not show a clear trend of its performance over different datasets. That is because it does not employ any temporal dynamics or value evolution of entities in Wikipedia articles and it rather relies on syntactical features of entities which are not related to any characteristics presented in Table 2. Also, `iPopulator` falls behind our ladder based methods in overall performance.

Takeaways. In the presence of evolution, ladders show significant improvements in role linkage precision and if combined with `wordnet` achieve a good trade-off between precision and recall.

4.3 Efficiency

In this section, we study the efficiency of the ladder methods. As described in Section 4.1, the Wikipedia pages are categorized into four different sets depending on their statistical characteristics of `n_rev`, `ambiguity`, and `n_trans`. However, the most important factor that affects running times is the size of *Rng* denoted

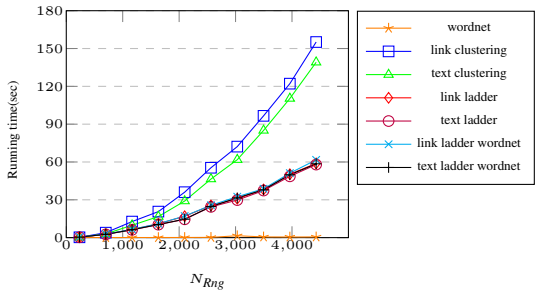

Figure 2: Efficiency with varying number of rungs

by N_{Rng}. Note that N_{Rng} determines the input size: `wordnet` and `clustering` work on only *Rng* and `ladder` performs the rail extraction algorithm that is initiated with *Rng*. In order to see the correlation between N_{Rng} and the running time, we partition all the pages into 10 bins, each of which is associated with one of the intervals that equally divide the range of [`min`(N_{Rng}), `max`(N_{Rng})]. Then we sample 10 pages from each bin and take the average of the running times of all the methods for the sampled pages. As shown in Figure 2, the running times increase as N_{Rng} increases. The running time of `wordnet` stays almost the same. That is because the relevance computation in `wordnet` takes constant time for each entity and we cache the computed relevance scores for entities having the same role to leverage the fact that most of the entities appear repeatedly in the entire revision history. The running time of `ladder` grows linearly with N_{Rng} and is much more efficient than `clustering` that has $O(n^2)$ time complexity.

Takeaways. `ladder` can efficiently find relevant entities for a given attribute while running in linear time and its running time mostly depends only on N_{Rng} that is usually small. Also, `ladder wordnet` does not have an overhead coming from leveraging `wordnet`, due to its constantly good efficiency.

5. RELATED WORK

Since we link together entities, a related topic is the problem of entity linkage [7, 4]. However, we do not link entities that represent the same real world entity, but entities that are used under the same role (role linkage) in the infobox and the text and across different versions. Thus, the work on temporal linkage [13, 3, 12] is somehow closer to us, yet it still identifies structures representing the same real world entity. Role linkage required novel techniques since it cannot take advantage of the ideas from collective entity linkage [2] since occurrences of entities under the same role are independent (i.e., if we found that one occurrence of entity belongs to some role it does not help with others).

The problem of *populating Wikipedia infobox* with values from text [11, 17] is related to ours since in principle they also need to recognize the entities in the text. However, we solve an inverse problem, i.e. we find which entity occurrences in text are the values of an infobox attribute.

Information extraction [9, 10, 16, 19] aims to extract entities and their relations from text. Since we deal with temporal data the closest related work is in temporal information extraction [14, 8], i.e., finding entities when the data evolve in time. The above approaches, however, are not applicable to the problem of role linkage since role linkage uses the entities which are already extracted from text.

6. CONCLUSION

We studied the problem of using the text of wikipedia pages to provide related information to a fact. A fact is an attribute in the infobox. The related information is the surrounded text in the appearances of the attribute value in the text but under the same role. So we define the novel concept of role linkage and we materialize the solution through the introduction of a "ladder" structure that keeps links entities of the same role between the infobox and the text and across versions in time. The challenging task is how to formulate these ladders. We present some solutions and an extensive set of experimental results.

REFERENCES

[1] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, June 2004.

[2] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *TKDD*, 1(1), 2007.

[3] Y. Chiang, A. Doan, and J. F. Naughton. Modeling entity evolution for temporal record matching. In *SIGMOD*, pages 1175–1186, 2014.

[4] D. Dey, S. Sarkar, and P. De. Entity matching in heterogeneous databases: A distance based decision model. In *HICSS*, pages 305–313, 1998.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[6] A. Gliozzo, B. Magnini, and C. Strapparava. Unsupervised domain relevance estimation for word sense disambiguation. In *EMNLP*, pages 380–387, July 2004.

[7] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[8] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[9] J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust disambiguation of named entities in text. In *EMNLP*, pages 782–792, 2011.

[10] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti. Collective annotation of wikipedia entities in web text. In *KDD*, pages 457–466, 2009.

[11] D. Lange, C. Böhm, and F. Naumann. Extracting structured information from wikipedia articles to populate infoboxes. In *CIKM*, pages 1661–1664, 2010.

[12] F. Li, M.-L. Lee, W. Hsu, and W.-C. Tan. Linking Temporal Records for Profiling Entities. In *SIGMOD*, pages 593–605, 2015.

[13] P. Li, X. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *VLDB*, 4(11):956–967, 2011.

[14] X. Ling and D. S. Weld. Temporal information extraction. In *AAAI*, 2010.

[15] G. A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, Nov. 1995.

[16] D. Milne and I. H. Witten. Learning to link with wikipedia. In *CIKM*, pages 509–518, 2008.

[17] A. Sultana, Q. M. Hasan, A. K. Biswas, S. Das, H. Rahman, C. H. Q. Ding, and C. Li. Infobox suggestion for wikipedia entities. In *CIKM*, pages 2307–2310, 2012.

[18] G. Weikum, J. Hoffart, and F. M. Suchanek. Ten years of knowledge harvesting: Lessons and challenges. *IEEE Data Eng. Bull.*, 39(3):41–50, 2016.

[19] M. A. Yosef, J. Hoffart, I. Bordino, M. Spaniol, and G. Weikum. AIDA: an online tool for accurate disambiguation of named entities in text and tables. *PVLDB*, 4(12):1450–1453, 2011.