Supporting Employer Name Normalization at both Entity and Cluster Level

Qiaoling Liu and Faizan Javed
CareerBuilder LLC
Norcross, GA 30092
{qiaoling.liu,faizan.javed}
@careerbuilder.com

Vachik S. Dave*
Indiana University - Purdue
University Indianapolis
Indianapolis, IN 46202
vachik.dave25@gmail.com

Ankita Joshi*
University of Georgia
Boyd GSRC, Room 111
Athens, GA 30602
ankita.joshi25@uga.edu

ABSTRACT

In the recruitment domain, the employer name normalization task, which links employer names in job postings or resumes to entities in an employer knowledge base (KB), is important to many business applications. In previous work, we proposed the CompanyDepot system, which used machine learning techniques to address the problem. After applying it to several applications at CareerBuilder, we faced several new challenges: 1) how to avoid duplicate normalization results when the KB is noisy and contains many duplicate entities; 2) how to address the vocabulary gap between query names and entity names in the KB; and 3) how to use the context available in jobs and resumes to improve normalization quality.

To address these challenges, in this paper we extend the previous CompanyDepot system to normalize employer names not only *at entity level*, but also *at cluster level* by mapping a query to a cluster in the KB that best matches the query. We also propose a new metric based on success rate and diversity reduction ratio for evaluating the cluster-level normalization. Moreover, we perform query expansion based on five data sources to address the vocabulary gap challenge and leverage the url context for the employer names in many jobs and resumes to improve normalization quality. We show that the proposed CompanyDepot-V2 system outperforms the previous CompanyDepot system and several other baseline systems over multiple real-world datasets. We also demonstrate the large improvement on normalization quality from entity-level to cluster-level normalization.

1 INTRODUCTION

Entity linking [27] links entity mentions in text to the corresponding entities in a knowledge base (KB) and has many applications such as information extraction and content analysis, in both open domain and specific domains. For example, in the recruitment domain, linking employer names in job postings or resumes to entities in an employer KB, which is defined as the employer name normalization task in [20], is important to many business applications.

In our previous work [20], we proposed the CompanyDepot system, which used machine learning techniques to address the employer name normalization task. After applying the system to several applications at CareerBuilder[1], including RecruitmentEdge[2] [2], Jobfeed[3], and BetterJobs[4], we received many useful feedbacks as well as new requests from these applications, which posed new challenges to the system:

(1) How to avoid duplicate normalization results when the employer KB is noisy and contains many duplicate entities. For example, for the input shown in Figure 1a, the normalization results shown in Figure 1b are not acceptable, because it contains many duplicates.
(2) How to address the vocabulary gap between query names and entity names in the KB. For example, the KB uses "Robert Bosch GmbH" as the entity name but in resumes people may list "Bosch" as an employer name.
(3) How to use the context available in resumes and jobs to improve normalization quality. For example, about half of the jobs in our Jobfeed application have an associated company url, which can be helpful for linking the job to the correct employer entity in the KB.

To address these challenges, in this paper we extend the previous CompanyDepot system to normalize employer names not only *at entity level* by mapping a query to an entity in the KB that best matches the query, but also *at cluster level* by mapping a query to a cluster in the KB that best matches the query. This enables the system to return much better normalization results shown in Figure 1c than the results shown in Figure 1b, for the input shown in Figure 1a. To evaluate the cluster-level normalization, we propose a new metric which measures how likely the system returns a correct result (success rate) and how much result diversity the system reduces correctly via clustering (diversity reduction ratio). Moreover, we perform query expansion based on five data sources to address the vocabulary gap challenge and leverage the url context associated with the employer names in many jobs and resumes to improve normalization quality. Our experiments show that the proposed CompanyDepot-V2 system outperforms the previous CompanyDepot system and several other baseline systems over multiple real-world datasets. We also demonstrate the large improvement on normalization quality from entity-level to cluster-level normalization. Its short query response time allows the system to be applied to online employer name normalization.

*Work done while interning at CareerBuilder.

KDD '17, August 13-17, 2017, Halifax, NS, Canada

DOI: 10.1145/3097983.3098093

[1] http://www.careerbuilder.com/
[2] http://edge.careerbuilder.com/
[3] http://www.jobfeed.com/
[4] http://www.betterjobs.com/

(a) Before normalization (20 values) **(b) Using normalization at entity level (19 values)** **(c) Using normalization at cluster level (12 values)**

Figure 1: The *Previous Employer* facet in Recruitment Edge after searching for US candidates by keywords "price waterhouse".

The main contributions of this paper are as follows:

- Using query expansion and url context to improve normalization quality (Section 5.1).
- Proposing and solving the task of normalization at cluster level (Section 5.2).
- Proposing a new metric for evaluating cluster-level normalization (Section 6.2.2).

The rest of the paper is organized as follows. Section 2 discusses the related work. Next, we provide the problem definition in Section 3 and some preliminaries in Section 4. Then Section 5 describes the proposed system and Section 6 details the experiments. Finally, we conclude the paper in Section 7.

2 RELATED WORK

2.1 Entity Linking with a Knowledge Base

Entity linking [27], also called named entity disambiguation (NED) or named entity normalization (NEN), has attracted much research effort since the availability of large KBs such as Wikipedia and Freebase [3]. It is a key step to understand and annotate the raw and noisy data in many applications. A comprehensive survey of the issues and methods about entity linking is provided in [27]. As summarized by the survey paper, a typical entity linking system has three modules: candidate entity generation, candidate entity ranking, and unlinkable mention prediction.

The employer name normalization task discussed in this paper can be viewed as a general entity linking problem, yet it differs from the traditional entity linking task in three aspects [20]: (1) different data sources; (2) different contexts; (3) different KBs. Therefore, the employer name normalization task has unique challenges such as handling the location and the url context associated with the employer names in jobs and resumes, as well as handling noises and duplicate entities in the KB. The system proposed in this paper adapts the three-module framework used in the entity linking systems. We also propose cluster-level normalization to handle duplicate results, which is not considered in entity linking systems.

2.2 Domain-Specific Name Normalization

The system described in this paper extends the system in [20], with the following contributions: (1) performing query expansion based on external mapping sources and supporting using url in query to improve normalization quality; (2) supporting normalization at cluster level; (3) proposing a new metric for evaluating cluster-level normalization. More details will be described in Section 5.

Our work is also related to a set of domain-specific name normalization applications. For example, within the same recruitment domain, Yan et al. described how to normalize the company name on a LinkedIn member's profile position using social graphs based on binary classification [30]. The problem of academic institution name normalization discussed by Jacob et al. [10] is very similar to our task. Instead of an employer KB used in our work, they used a KB of academic institutions. Experiments in [20] showed that CompanyDepot performed better than the sCooL system [10] on five academic institution datasets. NEMO [13] addressed a related task of extracting and normalizing organization names from PubMed articles. The method first uses multi-layered rule matching to extract entity mentions, and then leverages unsupervised clustering to identify entity mentions referring to the same entity, which has a time complexity of $O(k \times N)$, where k is the number of clusters, and N is the number of all entity mentions.

There are also many other domain-specific name normalization applications, e.g., product item name normalization [4], skill name normalization [12], gene name normalization [28], disease name normalization [19], and person name normalization [21]. The main differences of these applications with ours lie in different data sources, contexts, and KBs, which often bring some different challenges.

2.3 Deduplicating Domain-Specific KBs

The task of employer name normalization depends on an employer KB, and a key step in building such domain-specific KBs is deduplication [17]. Kardes et al. proposed graph-based blocking and clustering strategies for organization entity resolution [16]. McNeill et al. proposed a dynamic blocking method to efficiently deduplicate around 5 billion people records [22]. The book by Christen summarizes more methods for general duplicate detection [7].

This paper focuses on the task of employer name normalization, for which we performed simple deduplication of the records in an employer database based on their business names. For cluster-level normalization, we cluster all the entities in the KB using a graph-based clustering method based on external mapping sources. Note

that our clustering is a little different from deduplication, because our clustering can group not only duplicate employer entities but also entities that have corporate linkages (e.g., branches, divisions, and subsidiaries), based on the needs of applications.

2.4 Clustering Methods and Evaluation Metrics

To support employer name normalization at cluster level, we need to solve a clustering problem, i.e., clustering the entities in the employer KB. There are many types of clustering methods available in literature [11, 29], such as clustering based on partition [24, 25], hierarchy [32], density [8, 15], and graph theory [1, 16]. Different methods have different constraints and are suitable for different applications. Some methods depend on designing a feature vector for each data item to calculate the distance between two items [8, 24, 25], or a feature vector for each cluster of items to find the distance between two clusters [32]. Some methods require to prefix the number of clusters k [24, 25], or select a suitable threshold for clustering [8, 15].

Considering the large size of our KB (about 19 million entities) and the number of output clusters (k) is unknown, we choose an efficient graph-based clustering method for our task of cluster-level normalization. This also allows us to easily use the mapping sources described in Section 4.3 to create edges between entities, and avoid a careful choice of k and feature vector for each item or cluster to compute the distance, as required by many other clustering methods.

Evaluation of clustering systems is often done using internal and external evaluation indicators [29, 31]. The internal evaluation indicators usually favor clustering systems with high similarity within a cluster and low similarity between clusters, while the external evaluation indicators try to evaluate the systems based on a gold standard test dataset.

To understand whether the proposed cluster-level normalization system would return satisfactory results in real applications, we prefer external evaluation indicators. We propose a new metric which measures the correctness and diversity of the results returned by the normalization system. By design, getting the ground truth for its computation is much easier than assigning the correct cluster for each input in a test dataset. More details will be discussed in Section 6.2.2.

3 PROBLEM DEFINITION

Our task is to link the employer names in job postings or resumes to entities in an employer KB. Before defining the problem more formally, we first introduce some notations. The entities in the employer KB are denoted by E. The employer names and the associated location contexts and url contexts extracted from job postings and resumes are denoted by $Q = \{q_1, q_2, ..., q_t\}$, where $q_i = (n_i, l_i, u_i)$ is a triple of employer name, the associated location, and the associated url. In the rest of the paper, we call n_i the query name, l_i the query location and u_i the query url. We represent a query location as a triple: $l_i = (City_i, State_i, Country_i)$, where the city, state, country information could be empty. Similarly, the query url could be empty as well.

The problem of employer name normalization can then be summarized as inferring a mapping function $f(q) \Rightarrow e$, where $q \in Q$ and $e \in E \cup \{NIL\}$. Note that NIL (Not-In-Lexicon) means the input query does not refer to any known entity in the KB.

Next, we introduce a clustering function $C(e) \Rightarrow r$, where $e \in E$, $r \in R \subset E$, R is the set of all cluster representative entities (each cluster representative entity corresponds to a cluster), and r is the representative entity of the cluster that e belongs to. One example of a clustering function is based on corporate family tree, i.e., all subsidiaries and divisions of a company r belongs to the same cluster and r is the cluster representative entity.

Finally, we define two levels of employer name normalization:

- Normalization at entity level is to infer a mapping function $f_E(q) \Rightarrow e$, where $q \in Q$ and $e \in E \cup \{NIL\}$, which maps a query to an entity in the KB that best matches the query.
- Normalization at cluster level is to infer a mapping function $f_C(q) \Rightarrow r$, where $q \in Q$ and $r \in R \cup \{NIL\}$, which maps a query to a cluster in the KB that best matches the query.

4 PRELIMINARIES

4.1 Employer Knowledge Base

The employer KB used in this paper is an extension of the one used in [20] with about 1 million more entities. It is built using a third-party employer database, which contains about 26 million employer records. Each record represents an employer with a set of attributes like business name, location, url, industry code, company size, and parent company. Branches and subsidiaries of a company are represented as different records, often with the same business name. The database has a good coverage of employers in US, however, it is noisy and may contain duplicate records for the same employer. For example, it contains different records named "Enterprise Rent A Car", "Enterprise Rentacar", and "Enterprise Rent-A-Car Company" respectively.

To make an employer KB for the employer name normalization system, we performed simple deduplication of the above employer database. We merged all the records with the same business name into a single entity (by assuming that two records with the same business name refer to the same employer) and kept a list of all the original locations. We also computed the number of the original records merged into an entity as the *entity popularity*. This resulted in a set of around 19 million employer entities that serves as our final employer KB. Note that the noises still exist in the KB, which needs to be handled by the employer name normalization system.

Note that the parent company column in the original employer database is very valuable. For example, it tells us the parent company of "Amazon Web Services, Inc." is "Amazon.com, Inc.". Although this data is far from complete as many records have unexpected missing values, we export this data as a mapping source (described in Section 4.3), which is used in both entity- and cluster-level normalization.

4.2 The CompanyDepot-V1 System

The CompanyDepot-V1 system described in [20] only supports employer name normalization at entity level. Its architecture is shown in the dotted box of Figure 2. Before taking any queries, the system needs to index the employer KB using a Lucene[5] indexer.

[5] https://lucene.apache.org/

Table 1: Statistics and examples for mapping sources.

Source	Size	Example
Wikipedia	135K	IBM Corp. → International Business Machines Corporation
Stock	6K	MSFT → Microsoft Corporation
Hierarchy	272K	Amazon Web Services, Inc. → Amazon.com, Inc.
Legacy	26M	bankofamerica → Bank of America Corporation
Provider	10M	pricewaterhouse coopers → PwC

Once the index is ready, it can take normalization requests. Each request consists of an employer name and its location context (part of or whole location information could be empty). The system then uses a Lucene searcher to retrieve a list of N employer entities. These N candidate entities are then sent to the reranking step, which generates a feature vector for each entity and uses a machine learning based ranking model to rank them. Finally, the top-ranked entity is sent to the validation step to decide whether it is a correct result for the query using a binary classifier. If it says yes, the system outputs this entity to the user; otherwise, it outputs NIL.

4.3 Mapping Sources

Five data sources are used in both our entity-level normalization (to do query expansion) and cluster-level normalization (to do graph-based clustering). Each source contains a set of mappings from surface forms to normalized forms. Table 1 shows the statistics and examples for each source. We describe how each mapping source is obtained below:

- Wikipedia: First, we collected all the Wikipedia pages that use an infobox template type of company[6], e.g., https://en.wikipedia.org/wiki/IBM. Then, we extracted the name field in the infobox as the normalized form, e.g., "International Business Machines Corporation". We then collected all the Wikipedia pages that redirect to the company page, e.g., https://en.wikipedia.org/w/index.php?title=IBM_Corp.&redirect=no, and used their titles as surface forms.
- Stocks: We collected all the public companies listed on NASDAQ, NYSE, and AMEX[7]. Then we used the company name as the normalized form and its stock symbol as the surface form.
- Hierarchy: We collected the parent company column from the employer database described in Section 4.1. We used each parent company as a normalized form and all its child companies as the surface forms.
- Legacy: We collected the mappings from a legacy mapping table used previously at CareerBuilder for employer name normalization.
- Provider: We collected the mappings from a third party data provider.

Note that the mapping sources are not required to have any relations with or dependencies on the KB described in Section 4.1, as long as they are about employer entities and have a reasonable quality. Therefore, more mapping sources can be added in future.

[6]https://en.wikipedia.org/wiki/Template:Infobox_company
[7]http://www.nasdaq.com/screening/company-list.aspx

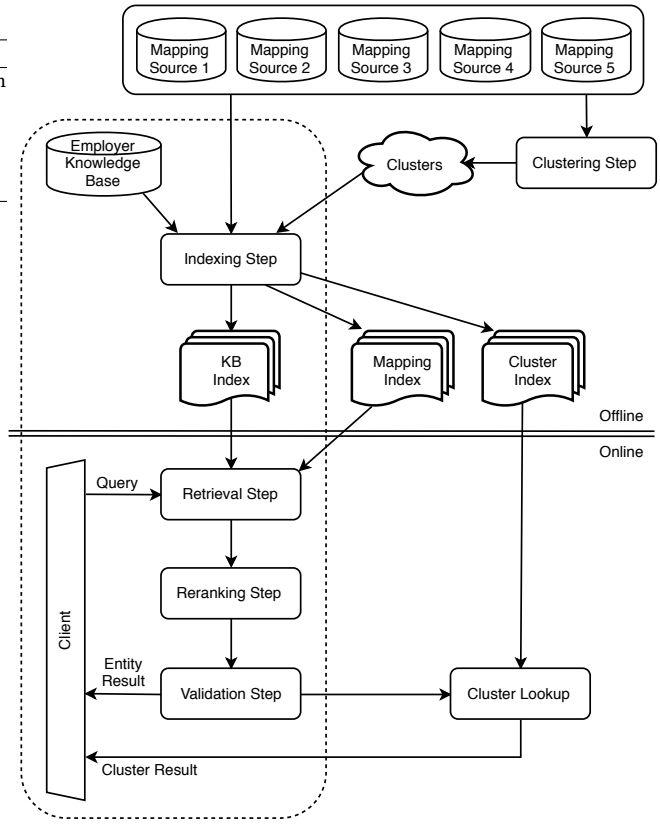

Figure 2: Architecture of the CompanyDepot-V2 system. The dotted box shows architecture of CompanyDepot-V1.

5 THE COMPANYDEPOT-V2 SYSTEM

After applying the CompanyDepot-V1 system described in Section 4.2 to several CareerBuilder applications, new challenges (discussed in Section 1) were posed. Therefore, we extend the system to address them. The architecture of the CompanyDepot-V2 system is shown in Figure 2. Before taking any queries, the system needs to build several indexes offline based on the employer KB and the mapping sources. Once the indexes are ready, the system can take normalization requests. Each request consists of an employer name, its location context and url context (both contexts could be empty). The system then performs normalization at both entity and cluster level, and returns both an entity result and a cluster result.

5.1 Normalization at Entity Level

We solve two challenges on top of CompanyDepot-V1 to improve normalization quality: 1) how to address the vocabulary gap between query names and entity names, and 2) how to use the url context available in many jobs and resumes.

We address the first challenge by using query expansion techniques which have shown their effectiveness on entity linking and information retrieval [5, 9]. We compute a set of alternative names as expansions for a query name based on the five sources of mappings described in Section 4.3. For example, for the query "MSFT",

Table 2: Index structure.

(a) A document in the KB index.

Field	Value
id	15
normalized_form	International Business Machines Corporation
calibrated_name	internationalbusinessmachines
domain	ibm.com
json	{"id": "15", "normalized_form": "International Business Machines Corporation", ...}

(b) A document in the mapping index.

Field	Value
surface_form	IBM
normalized_form	International Business Machines Corporation
mapping_source	wikipedia

(c) A document in the cluster index.

Field	Value
cluster_member_key	internationalbusinessmachines
cluster_representative	International Business Machines Corporation

we can get a list of expansions as {"Microsoft", "Microsoft Corporation", "Msoft Inc"}. Then we use both the original query name and the expansions in retrieving and reranking candidate entities. To address the second challenge, we add a url related field to the KB index and use the query url in retrieving and reranking entities.

As shown in Figure 2, the process of entity-level normalization in CompanyDepot-V2 is similar to CompanyDepot-V1 (shown in dotted box), yet with enhancements in each step which will be described next.

5.1.1 Indexing Step. In this step, we use a mapping index to handle query expansion based on the mapping sources, and revise the KB index to support using url in input query for normalization. Table 2-(a) and Table 2-(b) show the index structure for the KB index and mapping index respectively. Compared to CompanyDepot-V1, the KB index in CompanyDepot-V2 removes the surface forms field and adds a domain field. Considering that urls are often too sparse and noisy, we extract the domains of urls to do domain match instead of direct url match between query and entity. The mapping index in CompanyDepot-V2 enables efficient retrieval of the alternative names in the normalized form field for a query name using the surface form field, which can then be used for query expansion.

Just like in CompanyDepot-V1, we also calibrate entity names and query names to make them more comparable. The calibration of an employer name works as follows: 1) Convert the name to lowercase, and replace "'s" with "s" (e.g., "Macy's" → "macys"); 2) Convert all the non-alphanumeric characters to space; 3) Remove stop-phrases (e.g., "pvt ltd" and "l l c") and stop-words (e.g., "inc", "corporation", "incorporated", and "the"); 4) Expand commonly used abbreviations, e.g., "ctr" → "center", "svc" → "services", "dept" → "department"; 5) remove all spaces in the name. Considering that a query name (or a surface form) is often short while the official name (or normalized form) of an entity could be verbose, comparing two calibrated forms often enables more accurate match than comparing the two original names. Therefore, calibration is used in many steps of both entity-level and cluster-level normalization.

5.1.2 Retrieval Step. In this step, just like in CompanyDepot-V1, we first use Lucene's powerful querying capabilities to retrieve a large set of entities that are possibly relevant to the query name, and then apply several filters to this entity set and keep only the most likely correct results as the final pool of candidate entities.

Specifically, we first search for the query name against all the entity documents in the KB index via an aggregated search in Lucene which combines (1) keyword searches in the normalized form field; (2) fuzzy searches in the normalized form and calibrated name fields; and (3) phrase searches in the normalized form field.

In CompanyDepot-V2, we add new components to the above aggregated search in Lucene when applicable: (a) keyword searches in the calibrated name field based on the calibrated version of each of the query expansion names, and (b) keyword search in the domain field based on the domain of the query url.

After obtaining top N_0 (e.g., N_0=1000) entities from the Lucene searcher, we then generate the pool of candidate entities as follows: (1) From the N_0 results, add to the pool the top N_1 (e.g., N_1=10) entities that have the highest Lucene score. (2) From the N_0 results, use Levenshtein Distance to compute the top N_2 (or N_3) results whose original (or calibrated) normalized form has the minimum distance with the original (or calibrated) query name, and add them to the pool. (3) From the N_0 results, add to the pool the N_4 (or N_5) results for which a mapping from original (or calibrated) query name to original (or calibrated) entity normalized form exists in any mapping source. (4) From the N_0 results, add to the pool the N_6 results for which the domain of the query url matches that of the entity url.

The resulted pool contains $N = \sum_{i=1}^{6} N_i$ candidate entities, which will be reranked in the next step.

5.1.3 Reranking Step. In this step, just like in CompanyDepot-V1, we use a listwise learning-to-rank method, coordinate ascent [23], to rerank the candidate employer entities obtained from the retrieval step. The implementation provided in the RankLib[8] library is used for our experiments. Since only the top-ranked entity matters in our task, we choose P@1 to optimize on the training data.

For each (query, candidate entity) pair, we generate a vector of features, which can be grouped into three categories: (1) **Query features** indicate the complexity of finding a correct result for this query, which include the length of the query name, whether it indicates irrelevant input, and whether the query location is specified. (2) **Query-entity features** indicate the likelihood that the entity is a correct result for the query, which include the score of the Lucene searcher, string comparison of the query name and the normalized form (or the surface forms) of the entity, matching between the query location and the location list of the entity. (3) **Entity features** indicate the prior knowledge about the entity being a correct result for some query, which include the entity popularity, the number of locations of the entity, whether the normalized form of the entity contains a legal word such as "inc" or "llc". A full list of the features can be found in [20].

In CompanyDepot-V2, we generate extra query-entity features to use mappings and to support url matching: (a) whether a mapping from original (or calibrated) query name to original (or calibrated)

[8] http://sourceforge.net/p/lemur/wiki/RankLib/

entity normalized form exists in each (or any) mapping source; (b) whether the domain of the query url matches that of the entity url.

5.1.4 Validation Step. This step is the same as in CompanyDepot-V1. We use a binary classifier to validate the top-ranked result from the reranking step so that either a correct result or NIL is sent to the user. The features used for in this step include all the features used in the reranking step as well as the score output of the learning-to-rank method. We used LibSVM [6] as our binary classifier.

5.2 Normalization at Cluster Level

In this paper, we compute the cluster-level normalization $f_C(q)$ based on the entity-level normalization $f_E(q)$, by $f_C(q) = C(f_E(q))$. In future work, we will try computing $f_C(q)$ independent of $f_E(q)$, based on matching the query q to all entities in a cluster to find the best matched cluster.

The key to computing $f_C(q) = C(f_E(q))$ is to infer the clustering function $C(e) \Rightarrow r$. Once $C(e)$ is computed, we build a cluster index to look up the cluster of an entity efficiently. Table 2-(c) shows the index structure. As will be described in Section 5.2.1, entities in a cluster are identified by their calibrated names, so we use the calibrated name as the key of an entity to locate its cluster representative in the cluster index. We next describe how we compute the clustering function $C(e)$.

5.2.1 Creating Clusters. Considering the number of entities N in the KB is huge (about 19M) and the number of output clusters k is unknown which can be also at million scale, we need a scalable and fast clustering algorithm in terms of both N and k. So we choose an efficient graph-based clustering method.

First, we create an undirected graph based on the mapping sources described in Section 4.3. We take each mapping from a surface form to a normalized form and apply calibration to both parts. Then we add both the calibrated normalized form and the calibrated surface form as nodes in the graph, and create an edge between them. We use calibration to avoid creating a sparse graph.

Next, we remove low-quality edges that only appear in a single mapping source. This ensures that each edge remaining in the graph has been verified by at least two mapping sources. We can increase this threshold to increase the quality of edges when more mapping sources are added in the future.

Based on this graph, we apply transitive closure to find all the connected components in the graph [18, 26]. Each connected component corresponds to a cluster.

Finally, we only keep the clusters that have nodes corresponding to entities in the KB (i.e., nodes that match the calibrated entity normalized forms) and add each entity from the KB that does not appear in the graph (i.e., its calibrated entity normalized form does not match any node) as a separate cluster. This step is necessary because the nodes in the graph are created from the mapping sources which may not exist in the KB, and the entities in the KB may not appear in the graph. After this step, each cluster should contain at least one entity in the KB, and each entity should belong to a cluster. As the mapping sources we used to create the graph include both corporate linkages (from the Hierarchy mapping source) and surface forms (from the other mapping sources), each cluster is a group of duplicate entities and linked entities by corporate linkage.

5.2.2 Selecting Cluster Representative Entity. After creating the clusters, we need to select the most appropriate entity to represent each cluster. We consider three types of signals:

- Entity's parent company obtained from the employer database, as described in Section 4.1. The assumption is that a parent company is more appropriate to represent a cluster than its child company.
- Entity popularity computed from the employer KB, as described in Section 4.1. The assumption is that entities with more branches are more appropriate to represent a cluster.
- Entity popularity computed from the mapping sources. The assumption is that entities with more surface forms are more appropriate to represent a cluster.

Table 3 shows some example entities from a cluster and the corresponding signals. After obtaining the above signals for all entities, we compute cluster representatives as follows. For each cluster, we rank the entities in the cluster by entity popularity, and get the entity with the highest popularity. If this entity has a parent company, we use its parent company as the cluster representative; otherwise, we use itself as the cluster representative.

Since two types of entity popularity are available, for each cluster, we compute two versions of cluster representatives: KB version vs. mapping version. We checked the two versions for the largest 300 clusters, and found that the KB version gave overall better results. Therefore, we decide to use this version. Yet, we manually corrected 81 cases when the mapping version gave better representative.

In future work, we plan to use machine learning methods to learn the best cluster representative based on all the signals.

5.2.3 Correcting Clusters. For any clustering method, there are mainly two types of errors that can occur. First, multiple clusters have similar group of entities that need to be combined as a single cluster (merging duplicate clusters). Second, a single cluster has more than one group of entities that need to be put as separate clusters (splitting a mixed cluster).

The clustering errors caused by duplicate clusters would increase the diversity in results returned by the cluster-level normalization system, while the clustering errors caused by mixed clusters would increase the risk of returning a wrong result. Therefore, mixed clusters often have more negative impacts on the applications. We next describe how we split mixed clusters but leave merging duplicate clusters to future work.

Suppose there are only two different companies in a mixed cluster, and the two nodes that represent the two companies are identified as C_i and C_e, where the company that is more likely to connect to other companies is called the *inclusive company* (C_i) and the company that is less likely to connect to other companies is called the *exclusive company* (C_e). Then we try to cut all the paths connecting the two nodes. For every path connecting them, we remove the last edge in the path from the inclusive company to the exclusive company. This can be viewed as detaching the exclusive company from the inclusive company's cluster. It ensures that the entities in the middle of the paths will be put in the cluster of the inclusive company instead of in the cluster of the exclusive company.

For example, in case of a collaboration of a staffing company (e.g., "Kelly Services, Inc.") with another company (e.g., "The Dow Chemical Company"), we always treat the staffing company as

Table 3: An example of selecting the cluster representative entity. The entity in bold is selected as the representative.

Entity	Parent company	Popularity by KB	Popularity by mappings
The Dow Chemical Company	N/A	170	1406
Dow Agrosciences LLC	The Dow Chemical Company	73	162
Dow Chemical Canada ULC	N/A (missing link)	20	0
Dow Pipeline Company	The Dow Chemical Company	8	6
Dow Chemical	N/A	4	0
Dow Hydrocarbons and Resources LLC	The Dow Chemical Company	1	24

Figure 3: An example of splitting a mixed cluster. The dark gray node represents the inclusive company and the light gray node represents the exclusive company.

the inclusive company and the other company as the exclusive company, because the staffing company may have collaborations with many different large companies and these connecting nodes should be part of the staffing company, as shown in Figure 3.

A cluster mixing more than two different companies can be split using the above process iteratively. We currently manually identify the mixed clusters, the inclusive company and exclusive company, from the largest 300 clusters. In future work, we plan to work on automatically identifying them.

5.3 Discussion

An important feedback we received is that our system should be able to quickly fix errors reported by clients. The decoupling of the KB index, mapping index and cluster index enables our system to independently update each index and to easily support ad-hoc correction of errors in KB, mappings, clustering functions, and calibration rules using configuration files.

6 EXPERIMENTS

In this section, we conduct experiments to answer the following questions: (1) How does the proposed CompanyDepot-V2 system compare to CompanyDepot-V1 and other employer name normalization systems used at CareerBuilder? (2) How does normalization at cluster level compare to normalization at entity level?

6.1 Test Suite

6.1.1 Entity-Level Datasets. To evaluate the performance of our method for employer name normalization at entity level, we used the same datasets used in [20], which include two resume datasets and two job datasets from real applications at CareerBuilder.

The two resume datasets (RDB, EDGE) were sampled from two resume databases respectively. The two job datasets were sampled from a database of job postings based on a legacy normalization system: JOB1 (JOB2) was sampled from cases for which the legacy system returned some (no) result at some time. The statistics of the datasets are shown in Table 4. We can see that different datasets have different characteristics in terms of the percentage of queries with location specified and the percentage of international queries.

Table 4: Statistics about the entity-level datasets. %Country (State, URL) means the percentage of queries with country (state, url) specified. %US means the percentage of queries with country=US when country is specified.

Dataset	#Queries	%Country	%US	%State	%URL
RDB	1098	58.5%	96.4%	50.9%	0%
EDGE	1093	97.3%	45.3%	20.8%	0%
JOB1	1100	100%	100%	99.7%	0%
JOB2	500	100%	98.4%	100%	0%
JOBFEED	453	87.5%	100%	87.5%	100%

Note that the original labels for these datasets obtained in [20] were later enriched with the new results from new systems, e.g., CompanyDepot-V2, using the same labeling process.

Since the above datasets do not contain urls, we collected a new JOBFEED dataset to evaluate normalization using urls. The statistics of this dataset is also shown in Table 4. Results and labels are only collected for two systems (CompanyDepot-V2 and WService), because other systems cannot use query urls in normalization.

6.1.2 Cluster-Level Datasets. To evaluate the performance of the proposed method for employer name normalization at cluster level, we obtained a resume dataset and a job dataset from real applications at CareerBuilder.

The resume dataset came from the Recruitment Edge application. From the application's query logs, we got a list of 98 most frequent search queries about companies. Then for each query, we obtained the search results of US candidates and computed the 20 most frequent raw employer names appeared in the profiles of these candidates. As a result, we collected 817 unique raw employer names as the final queries.

The job dataset came from the BetterJobs application. From the application's logs, we got the jobs in the past 5 years. Then we used CompanyDepot-V1.5-C (described in Section 6.3) to normalize at cluster level the raw employer names for these jobs. We chose the top 182 employer entities that post the most jobs and the raw employer names in the jobs posted by these entities. As a result, we collected 6515 unique raw employer names as the final queries.

We manually labeled the normalization results for the above two query sets. For each query, the results of different systems were shown in random order to the labeler. Then the labeler checked the correctness of each pair of query and result.

6.2 Metrics

6.2.1 Metrics for Entity-Level Normalization. To evaluate the performance of different entity-level normalization systems, we used the same precision and coverage metrics as used in [10, 20, 30]. For a set of queries, suppose a normalization system returns I_c correct results, I_w wrong results, and I_n null results (i.e., NIL), then the following metrics are computed:

- Precision is the percentage of correct results out of all non-null results from the system: $Precision = I_c/(I_c + I_w)$.
- Coverage is the percentage of queries that the system returns a non-null result: $Coverage = (I_c + I_w)/(I_c + I_w + I_n)$.

6.2.2 Metrics for Cluster-Level Normalization. An effective cluster-level normalization system needs to produce not only correct results but also as less diverse results as possible via clustering. Therefore, we propose a new metric to evaluate the performance of normalization at cluster level which considers two aspects: 1) Success Rate (SR), indicating how likely the system returns a correct result, and 2) Diversity Reduction Ratio (DRR), indicating how much result diversity the system reduces correctly via clustering.

The only labeling needed for computing this metric is that for each query q, we label whether the result r returned by the system is correct or not. Let Q_S be the set of successful queries, i.e., the queries which receive a correct result, i.e., $Q_S = \{q \in Q \mid f_C(q)$ is a correct result for $q\}$. We define Success Rate (**SR**) of the system as

$$\mathbf{SR} = \frac{|Q_S|}{|Q|} \tag{1}$$

To measure the diversity in results returned by a system, we adapted the true diversity metric [14] which is defined based on entropy. As it does not matter how diverse the wrong results are, we only compute the diversity in the correct results. Let $Q_{S|r}$ be the set of successful queries that are mapped to the cluster of r, i.e., $Q_{S|r} = \{q \in Q_S \mid f_C(q) = r\}$. We first compute the entropy of the correct results as

$$\mathbf{H} = -\sum_{r \in R} \frac{|Q_{S|r}|}{|Q_S|} \cdot ln\left(\frac{|Q_{S|r}|}{|Q_S|}\right) \tag{2}$$

The above entropy $H \in [0, ln(|Q_S|)]$ is not linear to $|Q_S|$, which makes it a little hard to understand and interpret. So True Diversity [14] is proposed as $TD = exp(\mathbf{H})$. It gives the effective number of correct clusters returned by the system, and is linear to $|Q_S|$.

Based on the above True Diversity, we can compute how much result diversity the system reduces correctly, i.e., Diversity Reduction Ratio (**DRR**), which is in range $[0, 1]$:

$$\mathbf{DRR} = 1 - \frac{exp(\mathbf{H}) - 1}{|Q_S| - 1} \tag{3}$$

Finally, we compute the f-score (or the harmonic mean) of Success Rate and Diversity Reduction Ratio to measure the normalization quality:

$$\textbf{F-score} = \frac{2 \cdot \mathbf{SR} \cdot \mathbf{DRR}}{\mathbf{SR} + \mathbf{DRR}} \tag{4}$$

The proposed metric has three merits. First, it is easy to interpret, showing the correctness and diversity of the results returned by a cluster-level normalization system. Second, it only needs minimum labeling effort, i.e., labeling for each (query, result) pair whether the result is correct for the query or not. The labels can be reused and shared across systems. This is much easier than assigning the correct cluster for each input as required by other external cluster evaluation indicators. Third, it directly relates to the metrics for entity-level normalization: $SuccessRate = Precision \times Coverage$.

Table 5: Systems in comparison.

Legacy: a system previously used at CareerBuilder for employer name normalization. It is based on a mapping table which contains a large number of mappings from surface forms to entities, generated via some black-box algorithm with manual corrections.
WService: a third-party web service that supports employer name normalization, with two characteristics: (1) It requires an employer name and a specified country to return any result; (2) It uses its own KB, which is a superset of the KB used by other systems, with a much better coverage of international employers.
CompanyDepot-V1 (CD-V1): the system described in Section 4.2. In this experiment it used the same KB as CD-V2-E. The training and test datasets are shown in Table 6.
CompanyDepot-V2-E (CD-V2-E): the system described in Section 5 that does normalization at entity level. The training and test datasets are shown in Table 6. We manually specified the weight of the url domain matching feature when testing on the JOBFEED dataset.
CompanyDepot-V1.5-C (CD-V1.5-C): a preliminary version before CD-V2-C. It used the old KB in [20] and CD-V1 for entity-level normalization, and did not apply the method of splitting mixed clusters.

Table 6: Training and test datasets.

Train	EDGE	RDB					
Test	RDB	EDGE	JOB1	JOB2	JOBFEED	Resume	Job

6.3 Systems and Results

Table 5 summarizes the systems compared in the experiments.

6.3.1 Results of Entity-Level Normalization. On the entity-level normalization datasets, we compared the results of four methods, CD-V1, CD-V2-E, Legacy, and WService. For CD-V1 and CD-V2-E, the output contains a confidence score along with the result. By varying the threshold on the confidence score, we can plot a precision-coverage curve. For Legacy and WService, however, the confidence score is not available, so we can only derive a single precision and coverage value.

Figure 4 shows the precision-coverage values of the four methods on the four datasets that do not have a url in queries. We can see that CD-V2-E performs the best on all these datasets, followed by CD-V1. This shows that the query expansion approach based on various mapping sources used by CD-V2-E is helpful for improving the normalization quality, compared to CD-V1. The superior results of CD-V2-E and CD-V1 over those of Legacy and WService shows the effectiveness of using machine learning based approach to learn the importance of various features and to combine them for solving the normalization problem.

Figure 5 shows the precision-coverage values of the applicable methods on the JOBFEED dataset that contains a url in each query. Note that since only CD-V2-E and WService can handle url in query, so we only compared their performance on this dataset. For each system, we ran two versions: one using the query url and the other ignoring the query url. We surprisingly found that

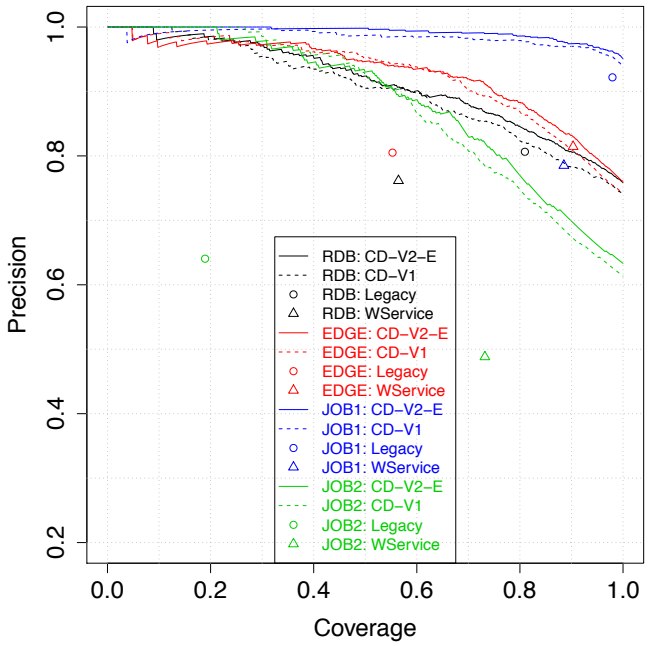

Figure 4: Results on 4 entity-level normalization datasets.

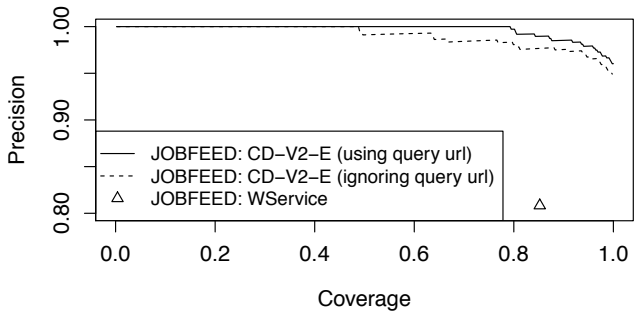

Figure 5: Results on JOBFEED (entity-level normalization).

Table 7: Results on cluster-level normalization datasets.

(a) Resume dataset.

System	SuccessRate	DiversityReductionRatio	F-score
CD-V2-C	0.963	0.704	**0.814**
CD-V1.5-C	0.897	0.688	0.779
CD-V2-E	0.958	0.416	0.580

(b) Job dataset.

System	SuccessRate	DiversityReductionRatio	F-score
CD-V2-C	0.904	0.979	**0.940**
CD-V1.5-C	0.778	0.981	0.868
CD-V2-E	0.905	0.926	0.915

WService returns the same results for both versions, which means that although the system takes the query url in input, it does not take the advantage of it for normalization. In contrast, we see that CD-V2-E performs better when using the query urls than ignoring them. Yet, we expected a larger difference. With a deeper analysis, we found that using query urls actually corrected 20 out of the 24 errors made by CD-V2-E ignoring query urls, which shows the usefulness of the url context. However, among the 18 errors made by CD-V2-E using query urls, 14 of them were caused by the url matching, and ignoring query urls corrected them. The main reason is that either the query url in the input or the entity url in the KB is wrong, which causes wrong results to be matched on the url field. After correcting these url errors in input or KB, the performance of CD-V2-E using the query urls increases from 96.0% to 99.1%. Therefore, we believe that CD-V2-E can effectively make use of the query url for improving normalization quality.

Note that Figure 4 and the one in [20] look slightly different because ambiguous queries are excluded and more labels for results are added. Also the CompanyDepot systems perform much better on the EDGE dataset, as the KB used in this paper has been enriched.

6.3.2 Results of Cluster-Level Normalization. On the cluster-level normalization datasets, we compared the results of three methods, CD-V1.5-C, CD-V2-C, and CD-V2-E. The first two systems perform normalization at cluster level and the last one provides the best normalization at entity level, based on the results in Figure 4.

Table 7 shows the normalization quality of the three methods. We can see that CD-V2-C achieves higher f-score than CD-V2-E on both datasets. On one hand, CD-V2-C has similar success rate as CD-V2-E. This means that CD-V2-C did not introduce many errors during creating clusters and selecting cluster representatives. On the other hand, CD-V2-C has a much higher diversity reduction ratio, i.e., a much less diversity in its correct results, than CD-V2-E. Altogether, the results indicate that CD-V2-C successfully clusters entities with corporate linkage or duplicate entities and chooses good cluster representatives.

We can also see that CD-V2-C achieves higher f-score than CD-V1.5-C on both datasets. On the resume dataset, CD-V2-C is superior in both success rate and diversity reduction ratio. On the job dataset, we see the success rate increases much more (0.778 vs. 0.904) from CD-V1.5-C to CD-V2-C. This is because of improvement in both the entity-level normalization result $f_E(q)$ (as a result of KB enrichment and using CD-V2-E for entity-level normalization instead of CD-V1) and the clustering function $C(e)$ (as a result of splitting mixed clusters). The enhancement of $f_E(q)$ and $C(e)$ can reduce the errors of the cluster-level normalization system. However, this enhancement resulted in a little decrease in diversity reduction ratio (0.981 vs. 0.979) from CD-V1.5-C to CD-V2-C, because KB enrichment and cluster splitting created more clusters and led to larger diversity in correct results. Overall, CD-V2-C shows better performance.

We notice CD-V2-C shows quite different performances on the two datasets. From Section 6.1.2, we know that compared to the job dataset the resume dataset is cleaner because its queries come from the 20 most frequent raw employer names appearing in searched user profiles. Therefore, CD-V2-C achieves higher success rate on the resume dataset (0.963 vs. 0.904). On the other hand, the job dataset is less diverse because its queries are collected based on a cluster-level normalization system (CD-V1.5-C), so CD-V2-C achieves higher diversity reduction ratio on the job dataset (0.979 vs. 0.704).

6.4 Discussion

From the experiments, we notice that the clustering function $C(e)$ should be inferred based on the needs of applications, because different applications may need different $C(e)$, e.g., clusters of different granularity level. One advantage of our system is that it can be easily extended to handle multiple clustering functions for multiple applications by building multiple clustering indexes.

7 CONCLUSION AND FUTURE WORK

In this paper, we focused on the employer name normalization task which takes an employer name and its associated location and url from job postings or resumes as input and normalizes it into entities in an employer KB. We extended a previous system called CompanyDepot. First, we proposed methods to normalize employer names at cluster level, which maps a query to a cluster in the KB that best matches the query. Second, we proposed a new metric based on success rate and diversity reduction ratio for evaluating the cluster-level normalization. Third, we used query expansion based on five mapping sources and leveraged the url context available with the employer names in jobs and resumes to improve normalization quality. We compared the performance of the proposed CompanyDepot-V2 system with CompanyDepot-V1 and other normalization systems used at CareerBuilder. The experimental results over multiple real-world datasets showed the effectiveness of the proposed system.

In our future work, we plan to do cluster-level normalization based on matching the query with all entities in a cluster instead of depending purely on normalization at entity level. In the clustering step, we currently use only mapping sources and could also use the attributes in the KB in future. To select better cluster representatives, we would like to use machine learning methods for combining multiple signals. Finally, we also intend to develop purely automatic ways to split mixed clusters and merge duplicate clusters.

ACKNOWLEDGMENTS

We would like to thank the TextKernel team, the Recruitment Edge team, and Hunter Burk at CareerBuilder for their great help with preparing for the datasets used in this paper. We would also like to thank Phuong Hoang and Janani Balaji for their helpful discussions. Finally, we thank the reviewers for their valuable comments.

REFERENCES

[1] J. Gary Augustson and Jack Minker. 1970. An Analysis of Some Graph Theoretical Cluster Techniques. *J. ACM* 17, 4 (Oct. 1970), 571–588.
[2] Janani Balaji, Faizan Javed, Chris Min, and Sam Sander. 2017. An Ensemble Blocking Approach for Entity Resolution of Heterogeneous Datasets. In *Proceedings of the 30th International FLAIRS (Florida Artificial Intelligence Research Society) Conference.*
[3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proc. of the 2008 ACM SIGMOD International Conference.*
[4] A. Borkovsky. 2003. Item name normalization. (April 29 2003). US Patent 6,556,991.
[5] Claudio Carpineto and Giovanni Romano. 2012. A Survey of Automatic Query Expansion in Information Retrieval. *ACM Comput. Surv.* 44, 1, Article 1 (Jan. 2012), 50 pages.
[6] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2 (2011), 27:1–27:27. Issue 3. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[7] Peter Christen. 2012. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer Publishing Company, Incorporated.
[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96).* 226–231.
[9] Swapna Gottipati and Jing Jiang. 2011. Linking Entities to a Knowledge Base with Query Expansion. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11).* 804–813.
[10] Ferosh Jacob, Faizan Javed, Meng Zhao, and Matt Mcnair. 2014. sCooL: A system for academic institution name normalization. In *Collaboration Technologies and Systems (CTS), 2014 International Conference on.* 86–93.
[11] A. K. Jain, M. N. Murty, and P. J. Flynn. 1999. Data Clustering: A Review. *ACM Comput. Surv.* 31, 3 (Sept. 1999), 264–323.
[12] Faizan Javed, Phuong Hoang, Thomas Mahoney, and Matt McNair. 2017. Large-Scale Occupational Skills Normalization for Online Recruitment. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.* 4627–4634.
[13] Siddhartha Jonnalagadda and Philip Topham. 2010. NEMO: Extraction and normalization of organization names from PubMed affiliation strings. *Journal of Biomedical Discovery and Collaboration* 5 (2010), 50.
[14] Lou Jost. 2006. Entropy and diversity. *Oikos* 113, 2 (2006), 363–375.
[15] Amin Karami and Ronnie Johansson. 2014. Choosing dbscan parameters automatically using differential evolution. *International Journal of Computer Applications* 91, 7 (2014).
[16] Hakan Kardes, Deepak Konidena, Siddharth Agrawal, Micah Huff, and Ang Sun. 2013. Graph-based Approaches for Organization Entity Resolution in MapReduce. In *Proceedings of TextGraphs-8 Graph-based Methods for Natural Language Processing Workshop at EMNLP.*
[17] Mayank Kejriwal, Qiaoling Liu, Ferosh Jacob, and Faizan Javed. 2015. A Pipeline for Extracting and Deduplicating Domain-Specific Knowledge Bases. In *Proceedings of 2015 IEEE International Conference on Big Data.*
[18] Lars Kolb, Ziad Sehili, and Erhard Rahm. 2014. Iterative Computation of Connected Graph Components with MapReduce. *Datenbank-Spektrum* 14, 2 (2014), 107–117.
[19] Robert Leaman, Rezarta Islamaj Dogan, and Zhiyong Lu. 2013. DNorm: disease name normalization with pairwise learning to rank. *Bioinformatics* 29, 22 (2013), 2909–2917.
[20] Qiaoling Liu, Faizan Javed, and Matt McNair. 2016. CompanyDepot: Employer Name Normalization in the Online Recruitment Industry. In *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16).* 521–530.
[21] Walid Magdy, Kareem Darwish, Ossama Emam, and Hany Hassan. 2007. Arabic Cross-document Person Name Normalization. In *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources (Semitic '07).* 25–32.
[22] Wiliam P. McNeill, Hakan Kardes, and Andrew Borthwick. 2012. Dynamic Record Blocking: Efficient Linking of Massive Databases in MapReduce. In *Proceedings of 10th International Workshop on Quality in Databases (QDB) at VLDB.*
[23] Donald Metzler and W. Bruce Croft. 2007. Linear Feature-based Models for Information Retrieval. *Inf. Retr.* 10, 3 (June 2007), 257–274. DOI:http://dx.doi.org/10.1007/s10791-006-9019-z
[24] Malay K Pakhira. 2014. A Linear Time-Complexity k-Means Algorithm Using Cluster Shifting. In *Computational Intelligence and Communication Networks (CICN), 2014 International Conference on.* IEEE, 1047–1051.
[25] Hae-Sang Park and Chi-Hyuck Jun. 2009. A Simple and Fast Algorithm for K-medoids Clustering. *Expert Syst. Appl.* 36, 2 (March 2009), 3336–3341.
[26] Thomas Seidl, Brigitte Boden, and Sergej Fries. 2012. CC-MR — Finding Connected Components in Huge Graphs with Mapreduce. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I (ECML PKDD'12).* 458–473.
[27] Wei Shen, Jianyong Wang, and Jiawei Han. 2015. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Trans. Knowl. Data Eng.* 27, 2 (2015), 443–460.
[28] Joachim Wermter, Katrin Tomanek, and Udo Hahn. 2009. High-performance gene name normalization with GENO. *Bioinformatics* 25, 6 (2009), 815–821.
[29] Dongkuan Xu and Yingjie Tian. 2015. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science* 2, 2 (2015), 165–193.
[30] Baoshi Yan, Lokesh Bajaj, and Anmol Bhasin. 2011. Entity Resolution Using Social Graphs for Business Applications. In *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2011.* 220–227.
[31] Mohammed J Zaki and Wagner Meira Jr. 2014. *Data mining and analysis: fundamental concepts and algorithms.* Cambridge University Press.
[32] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *SIGMOD Rec.* 25, 2 (June 1996), 103–114.