The Johns Hopkins University

human language technology
center of excellence

Entity recommendations on a Cold Start Knowledge Graph

Pushpendre Rastogi, Vince Lyzinski, Benjamin Van Durme

Technical Report

February 3, 2017

Acknowledgements This material is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA) under the Deep Exploration and Filtering of Text (DEFT) Program, agreement number FA8750-13-2-001. This work was also supported, in part, by the Human Language Technology Center of Excellence. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsor.

HLTCOE
810 Wyman Park Drive
Baltimore, Maryland 21211
http://hltcoe.jhu.edu/

Entity recommendations on a Cold Start Knowledge Graph

HLTCOE, Johns Hopkins University

Pushpendre Rastogi Vince Lyzinski Benjamin Van Durme

1 Introduction

In this report we consider the Adept Knowledge Base (KB), which we will work with as a graph. This graph is composed of a set of vertices (i.e., entities, nodes), a set of edges (i.e., relations, predicates, slot-fills) and vertices and edges may have zero or more properties (i.e., attributes, features).

The predominant way in which analysts interact with and make use of knowledge graphs is by treating knowledge graphs as databases that support evaluation of queries. An analyst may execute two types of queries on a KB:

Logical Queries An analyst may want to retrieve a set of items, that lie in a set exactly described via a logical expression. For example, an analyst may ask for all vertices that are connected to the vertex with the highest degree. A simpler example is static lookup: an analyst may begin with an entity represented as a vertex connected via a "name" attribute to the value "John Smith" and then construct a query which looks for an edge called "worksFor", and then "leaderOf".[1] Logical queries – so called "one hop" and "two hop" – form the basis for evaluation of CS-KBP 2015 task .

Example Queries Here an analyst provides examples of items of interest/relevance, and wants the system to *nominate* or *recommend* items that are similar to the examples. Sometimes the analyst may also provide *negative supervision* in the form of examples that are marked as uninteresting or irrelevant. Databases that can support such example queries and return a sorted list of relevant entities are called *Recommendation Systems* or *Recommenders* in short. They may also be called *Vertex Nomination Systems.* We call algorithms that can rank entities in a dataset based on a set of example entities *Recommendation Algorithms* (RAs).

While the evaluation under DEFT for KB creation has focused on performance under Logical Queries, here *we assume our goal is to enable Example Queries.* I.e., use an automatically created KB as the sole support for performing Vertex Nomination.

Interfaces that enable logical queries view the goal of the Adept KB as one of aggregating Information Extraction (IE) output from raw natural language text into a database that can be queried declaratively by an analyst, similar to the way SQL is used to query a

[1]Specifically, a knowledge graph may be stored inside a graph database, an RDF triplestore or even inside XML files and then queried using a graph query language such as Cypher or Gremlin or an RDF query language like SPARQL or XPath which is the defacto standard for querying XML data.

relational database. The important point is that creating the correct query is left to the analyst who must craft an expression that succinctly and efficiently describes the set of interesting or relevant items. Crafting such a query may be difficult and it may require significant skill on the part of the analyst (See Example 1.1).

Example 1.1

Consider a social network of sportsmen that contains the height and weight of people and the sports that they play as well as their friendship status. Assume that an analyst needs to retrieve adult triathletes with low lung capacities from this database.

Clearly our analyst has her job cut out for her, not only will she have to figure out the various ways in which people might express that they are triathletes, but the attributes of age and lung capacity don't even exist in the dataset. Some triathletes may state explicitly that they are "triathlete" or that their sport of choice is "triathlon" or they may instead state the three or more sports that they play. Because of this variety of expressions individually creating the correct filters by hand can be time consuming. On top of that the analyst will have to manually figure out some rules for guessing the age and lung capacity of a person based on their height and weight.

In such a scenario a *Recommender* can be very useful for achieving quick results. A general purpose RA based on a probabilistic linear model built with third order feature conjunctions can automatically figure out the right entities to retrieve from the database and give useful hints to the analyst for fast prototyping.

In contrast to logical queries, example queries do not require the analyst to create a logical expression by herself. A recommender can use the information stated in the knowledge graph to infer *new* information that is not directly stated in any source document, and recommend relevant items to a user with minimal supervision. This report lays out definitions and pointers to related areas of research from various communities, and then presents the performance of a few canonical VN algorithms (See Table 1.1) when applied to to the Adept KB via a representative set of example queries.

We conclude from these experiments (See Section 5) that the current Adept KB does not support example queries well, owing to its severe sparsity in edges. In short, the "knowledge graph" is not much of a graph, in that the overwhelming majority of vertices form singleton cliques: entities are discovered but no relations connect them to the rest of the KB.

We therefore end the report with a proposal to extend the ontology of the Adept KB, based on conservatively subselecting from an OpenIE ontology just those relations that best support provided example queries. We assume in this that DARPA does not wish to allow a massive (orders of magnitude) increase to the relation types in the Adept ontology, and thus will pursue a strategy for conservatively proposing to a KB-engineer a reduced set of relation types such that if they were included (based on observed performance in their extraction) then we would have a more sufficient graph for performing vertex nomination.

Method	Type
Naive Bayes	Inductive
Modified Adsorption	Transductive
Random Walk	Transductive

Table 1.1: List of Recommendation Algorithms

2 The `BBN2` dataset

The TAC Cold Start Knowledge Base Population (CS-KBP) shared task is organized by NIST to evaluate systems that build a knowledge base (KB) from raw natural language text (TAC-KBP@NIST, 2015). The competing systems extract entities and relations based on textual clues present in natural language documents and populate a KB according to a shared schema of entities and relationships. The set of documents is released at the beginning of the task and the schema is released some time before that. At the end of the shared task the systems submit their automatically generated knowledge bases to NIST, which evaluates their accuracy. BBN participated in the 2015–TAC Cold Start Knowledge Base Population where it was the top performing participant in terms of precision and overall F1 according to the official results as of November 2015 (Bonan Min, 2015). Since BBN's `BBN2` was one of its top performing submissions, to ascertain the feasibility of using an automatically constructed knowledge graph as a recommender, we performed our experiments on `BBN2`.

The `BBN2` dataset is a knowledge graph built according to the Adept Ontology . It contains three standard entity types: PER, ORG, LOC that refer to Person, Organization and Location respectively, as well as four extra types of TITLE, DATE, CRIME, URL (Table 2.1). We will collectively denote entities of these types as named entities. Technically, we define a named entity to be a any entity that has only either a canonical string or a canonical time attached to it and no other attributes. See 2.1 for an example. Because of the errors made during automatic relation extraction there can be multiple named entities with different identifiers that have approximately the same canonical string (See Example 2.2). We call this phenomenon *Denormalization* and we will show later this phenomenon severely hurts the performance of any recommendation system.

Example 2.1

The identifier `9dd1d8d3-9a1f-4ec2-a45b-64cce788eb01` in the dataset has the type `Title` and the canonical string associated with it is `Almighty King`. This identifier has no other attributes attached to it.

Example 2.2

The identifiers `e3ac2b1e-cd3b-4cf8-a1fb-b1f71db123df` and `234f3110-c8ca-49aa-ba7b-a45bb5467c38` have canonical strings *professor* and *Professor* respectively which differ only in their capitalization.

As this example illustrates some of the titles in the `BBN2` dataset lexically differ from each other but mean the same. Matching such fragments can help in reducing sparsity and may improve the performance of the vertex nomination algorithms. We leave such pre-processing for future work, and measure the performance of the VN algorithms on the ADEPT KB as is.

Instantiations	**Type**	**Property**
125022	Person	canonicalString
69544	Organization	canonicalString
23754	GeoPoliticalEntity	canonicalString
1761	Title	canonicalString
819	Date	xsdDate
333	Crime	canonicalString
189	URL	canonicalString
Total = $221,547$		

Table 2.1: The number of instantiations(occurrences) of Named Entities of different types.

As Table 2.1 shows, the `Person` category is the highest occurring category with 125K instantiations and it is slightly less than double the next largest category of `Organization`. The occurrences of the `URL` type are negligible.

Also we note that the knowledge graph contains a large number of singleton entities. We can infer this from Tables 2.1 and 2.2 since the number of entities is much larger than the total number of edges.

The relations in the `BBN2` dataset comprise of a small number of relation types. Just like named entities every instance of a relation has a unique id. Each instance relates two named entities (See Example 2.3).

Example 2.3

The id `001ae391-3a55-43d1-8d3e-0557bdd943d1` refers to a relation of the type `Resident` which connects its named attribute `person` which must have the type `Person` with its `location` attribute which must be of type `Geopoliticalentity`.

The total number of regular relations specified in the CS-KBP 2015 task was 41. The occurrences for different relations follow a power law (Table 2.2) and the highest occurring `Role` relation type that relates a person to his/her title accounts for 30% of the data out of the total 20 relations that are observed. The reason for this skew is that the natural language text itself has a strong bias about the type of relations that are reported explicitly, and that bias is reflected in the distribution of textual mentions of relations.

Instances	**Relation**	**Attribute 1**	**Attribute 2**
42490	Role	person	role
36631	EmploymentMembership	employeeMember	organization
16288	Resident	person	location
9600	Leadership	leader	affiliatedEntity
8705	Subsidiary†	organization	subOrganization
7242	OrgHeadquarter†	organization	location
2764	Origin	person	affiliatedEntity
1774	ParentChildRelationship‡	parent	child
1612	Die	person	place
1317	SpousalRelationship‡	person	person
1306	StudentAlum	studentAlumni	organization
1281	Founder	founder	organization
1261	BeBorn	person	time
530	InvestorShareholder	investorShareholder	organization
417	SiblingRelationship‡	person	person
387	ChargeIndict	defendant	crime
374	StartOrganization†	organization	time
369	Membership†	organization	member
200	OrganizationWebsite†	organization	url
33	EndOrganization†	organization	time
Total = 134, 581			

Table 2.2: Relations Types and their attributes. †indicates relations for which neither of the arguments are of the type `Person`. ‡indicates relations for which both of the arguments have the type `Person`.

2.1 Relations or Attributes

As shown in table 2.2 the `Role` relation does not relate two people. It only relates a person and its title. In this sense, the `Role` relation act as a feature or an attribute of a person instead of a relation between them, since the `Role` for one person can be completely independent of the Role of another person, unlike the `Sibling, Spousal` and `ParentChild` relationships. The `Sibling, Spousal` and `ParentChild` relationships always affects two people simultaneously in the sense that if a person B is set to be the sibling of A then A must be set to the sibling of B. We note that 11 out of 20 relations including the top four most observed relations, act as attributes of persons instead of relations. So the *person-person* relational information is lower in comparison to *person-attribute* information. Also note that only 6 relations relate organization to other organizations or to locations, so the *attribute-attribute* relational information is also low. However, in order to leverage the information present in the `OrgHeadquarter` relation we use the name of the headquarter of an organization that is related to a person as an attribute of the person.

2.2 Sparsity of the Adept KB

The Adept KB is a graph with more PER entities than edges. To quantify the sparsity and the connectedness of the Adept KB we present a table of the degrees of all the PER entities in the Adept KB in table 2.3. Table 2.4 tabulates the sizes of the components that those PER entities belong to. Note that both of these tables do not count edges that were not incident with at least one PER entity with the exception of edges of type `OrgHeadquarter`. The edges of type `OrgHeadquarter` are included in the counts for the following reasons: 1) The location of the headquarter of the organization that a person is affiliated with can act as an excellent feature for ranking a person entity. 2) Since the number of locations is small extracting these features does not cause an explosion in the number of features.

Degree	0	1	2	3	4	5	6	7	8	9	10	>10
PER Entities	82057	16672	8997	6862	3056	2521	1290	846	615	432	329	1345

Table 2.3: Table of degrees for the PER entities in the Adept KB.

Component Size	1	2	3	4	5	42350
Number of Components	82057	81	10	10	1	1

Table 2.4: The component sizes of the PER entities in the Adept KB.

Table 2.3 shows that the number of completely disconnected PER entities in the Adept KB constitute 65.6% of the total PER entities and that 13.3% of the PER entities have degree 1 but only 162 PER entities reside in a “dumbbell”, i.e., a component of size 2. The majority of the entities that have degree 1 lie in the giant component that contains 42350 PER entities. This has a lot of significance since it means that more than a third of the PER entities con be disconnected from the giant component by just removing single edges from the graph. This shows that the component has a small densely connected core and a large number of leaves. The above discussion clearly show that the Adept knowledge graph is quite sparse and weakly connected.

3 Evaluation

In order to evaluate RAs we need to simulate the kind of example queries that an analyst might submit to an RA. We represent a user's query as a predicate on vertices, where some positive example of the predicate are provided, and we wish to sort the remaining vertices as to how confident we are that they satisfy the predicate (are of interest to the user).

We chose to use an existing binarized attribute as a synthetic predicate function that acts as the oracle that determines whether an entity is relevant or not. During training and testing we remove this attribute from the data and only present a few examples of entities with the values of this attribute as labels. The algorithm then has to predict whether this hidden attribute is 0 or 1 for the rest of the people on the basis of the training data and then to assign a rank to them such that people with attribute value 1 rank higher than people with attribute value of 0. Two standard evaluation metrics for these tasks are the Precision@K(P@K) metric and the Area Under Precision Recall Curve(AUPR) metric and we would report the performance of the systems on these two metrics and compare them to a random baseline.

Example 3.1

For example, a user of the recommendation system may mark certain employees of the "White House" as people of interest and based on these examples the recommendation system needs to assign a high rank to the remaining employees of the "White House" and a low rank to others. Note that the recommendation system does not receive the attribute that the person was the employee of the "White House" during operation and it has to make a guess on the basis of other persons and their attributes. Consider a second example, where a predicate function may mark people who are "authors" as people of interest and request the recommendation system to find more people who are "authors". Table 3.1 lists the 6 relations types that we use to create a 12 predicate functions which we use for performance evaluation.

To simulate plausible recommendation criterion we assumed that a user of the knowledge graph recommendation system is only interested in ranking the PER entities, therefore the predicate function only needs to be able to label the PER entities.

4 Recommentation Algorithms for the Adept KB

Before we proceed to the discussion and comparison of different vertex nomination algorithms, let us first understand the implications of the sparsity and fragmented nature

Relation	Value 1	Value 2
Role	author	director
EmploymentMembership	Army	White House
Resident	Chinese	Texas
Leadership	Democratic	Parliament
Origin	American	Russia
StudentAlum	Harvard	Stanford

Table 3.1: List of Predicate Functions used as proxies of an analyst's interests. See example 3.1 for an illustration of the usage of the `Role=author` predicate function.

of the Adept KB that we established in Section 2.2. In Section 2.2 we showed that 66% of the vertices were singleton, the rest of the graph had one giant component and few small components. Figure 4.1 is a cartoon representation of a graph with such characteristics. We can see that out of 20 vertices only a small component can be used as input to a vertex nomination algorithm and 14 out of 20 vertices are unusable. In this section we discard such singleton entities and report the result of performing vertex nomination on the connected portion of the data. With the above caveat, let us now describe the

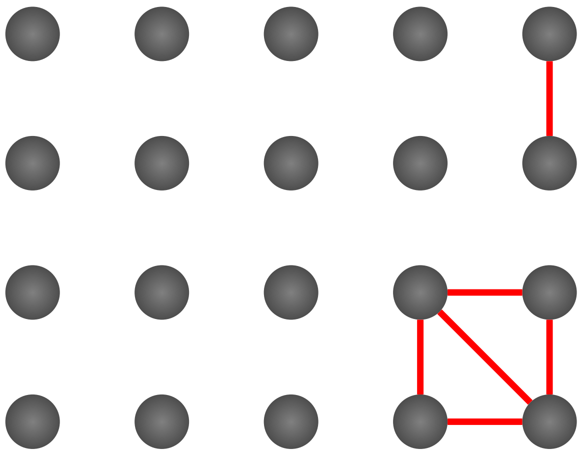

Figure 4.1: Illustration of the relative sparsity in the `BBN2` Knowledge Graph, with the number of connected and unconnected vertices displayed in similar proportion to the full `BBN2` KB.

experimental setup that remains common across the application of all of the following algorithms. Let the set of named entities in the Adept KB that have the type PER be $\mathcal{P}$. Let us consider the first row of table 3.1. It can define two predicate function / criteria for relevance over the elements of $\mathcal{P}$. The first predicate selects $\mathcal{Q}_\infty \subset \mathcal{P}$ such that all elements of $\mathcal{Q}_\infty$ have an attribute called `role` with the value of `author`. Analogously, the second predicate selects $\mathcal{Q}_2$ such that all elements of $\mathcal{Q}_2$ have `role=director`.

Table 3.1 specifies a total of 12 such predicate functions and we evaluate the performance of a VN algorithm over all of the 12 predicate functions. For each predicate function we perform 5 trials and measure the performance in terms of the Area Under Precision-Recall(AUPR) and Precision@K(P@K).[2] After performing five trials we average the performance of the VN algorithm across the trials and calculate the 90% confidence interval assuming the t-distribution for the standard error(SE). Finally we aggregate the confidence intervals and the average performance of the algorithms over all the predicate functions and create a single average performance and performance interval for each algorithm. We report these metrics in the later sections for different hyper-parameter settings and most notably for different feature sets.

During each trial for a predicate, we select 10 entities that satisfy the criterion and which we assume as the positive input that the user provided to the VN algorithms, similarly we chose 10 PER entities that do not satisfy the criteria and assume that the user provided those as negative input to the VN algorithm. During each trial we also remove the edges/attributes that correspond to the predicate function from the Adept KB before inputting the KB to the VN algorithm. We vary the entities used as training data across trials but ensure that the same entities are used as training input across different algorithms for a fixed predicate function and trial index. We denote the number of training instances including both the negative and the positive labeled instances as N and the remaining instances as M. We denote the set of PER entities whose relevance is observed with the symbol $\mathcal{O}$. The subset of $\mathcal{O}$ that satisfies the predicate function is called $\mathcal{O}^+$ and we define $\mathcal{O}^- = \mathcal{O} \setminus \mathcal{O}^+$. Clearly $|O| = N = 20$ for all trials. Note that M can hypothetically have zero entities that satisfy the predicate function, however in practice since there are more than 10 labeled instances this does not happen for any of the predicates.

4.1 Entity Recommendation Through the Naive Bayes Model

Let $v \in \mathcal{P}$. We can represent v through a feature vector by enumerating all edges that are incident on v and representing the type of edge along with the neighboring vertex as the occurrence of a binary feature. For example, let $\textcircled{v}\xrightarrow{r}\textcircled{q}$ be an arc of type r connecting v to vertex q. This arc can be represented as the occurrence of the binary feature $\xrightarrow{r}\textcircled{q}$. All such binary features formed by all the arcs that are incident on v can be used to create a feature vector f that represents v. Given this construction of a feature representation of v we can represent $\mathcal{O}$ as the set $\{(f^i, y^i) | i \in \{1, \dots, N\}\}$. Here $y^i \in \mathcal{Y} = \{-1, 1\}$ is the label assigned to the v^i and f^i is its binary feature representation. $y^i = 1$ means that the feature f^i represents a vertex of interest and $y^i = -1$ means the opposite. Let K be the length of f^i. We denote the kth element of f^i as $f^i[k]$ and we drop the superscript i

[2]We assume that the annotations present in the Adept KB are complete and correct and use those as the gold standard. This assumption sometimes turns out to be patently wrong. For example in the case of the predicate `Employer=Army` a number of time the Adept KB may have the annotation that `Employer=US Army` or `Employer=Pentagon` but not the annotation that `Employer=Army`. In our experiments we side-step these issues and assume that our task is to faithfully predict the predicate function as present in the Adept KB.

when the exact data instance is immaterial. Let $\hat{y}_{\text{NB}}$ denote the estimate produced by the Naive Bayes model. The Naive Bayes models model the probability of (x, y) and it estimates $\hat{y}_{\text{NB}}$ as follows:

$$p(x, y) = p(f, y) = p(y)p(f \mid y) = p(y) \prod_{k=1}^{K} p(f[k] \mid y) \tag{4.1}$$

$$\hat{y}_{\text{NB}} = \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|f) = \operatorname*{argmax}_{y \in \mathcal{Y}} p(y)p(f|y) \tag{4.2}$$

The Naive Bayes model can be trained by optimizing the log probability of the entire corpus, $\{(f^i, y^i) | i \in \{1, \ldots, N\}\}$ as specified by the following expression:

$$\sum_{i=1}^{N} \log(p(v^i, f^i, y^i)) = \sum_{i=1}^{N} \log(p(y^i)) + \sum_{i=1}^{N} \log(p(v^i, f^i | y^i)) \tag{4.3}$$

Once the Naive Bayes model is trained on $\mathcal{O}$ then we can perform vertex nomination by ranking the an unlabeled instance v according to the posterior probability $p(y = 1|f)$.

Feature Design: Let us describe the features that we use for the Naive Bayes model with an example. Consider a an entity named "Shaikh Ahmed" that is known to have three properties: 1) It has a Role of "Gen."; 2) It has a second Role of "chief"; and 3) The name of its Residence is "Afghanistan". We begin by concatenating the relation type with the attribute value to get 3 distinct feature types "Role=Gen.", "Role=chief" and "Residence=Afghanistan". We call these features the "Concatenative" features. Note that this featurization does not share parameters between two different feature even though they may have the same attribute value. For example, it seems plausible that sharing the parameters between "Residence=Baltimore" and "Origin=Baltimore" could be useful. We can alleviate this feature sparsity through the addition of "Backoff" features that are based only on the attribute value and not the relation type. For example we can add the following three features to "Shaikh Ahmed". i) "Gen." ii) "chief" and iii) "Afghanistan". We call this augmented featurization "Concatenation w/ Backoff". We call the feature set that contains only features like i, ii and iii the "Only Backoff" feature set. Finally when we add the documents that a particular entity was mentioned in as features to "Concatenative" features then we get the "Concatenative w/ Doc" feature set. For all of the above methods we prune the featureset used to only those that features f that occurred in $\mathcal{O}^+$ separately for each trial.

Results We present the aggregate results of the naive bayes classifier's performance in table 4.1. We can see that the performance of the Naive Bayes method varies a lot with the feature set and the "Concatenative" and "Concatenative w/ Doc" feature sets perform the best. Using backoff features seems to hurt the performance and the variance in the values of AUPR and P@10 indicates that the method is very sensitive to the examples that are chosen as inputs. This result is expected since the method only receives 10 data points as input. However even with 10 data points the Naive Bayes method is able to achieve 30% precision amongst its top 10 recommendations on average which indicates that there are correlations between the values of different attributes that can be exploited for performing recommendations.

	Only Backoff	Concatenative	Concatenative w/ Doc	Concatenative w/ Backoff	Random
AUPR	8.8 ± 3.9	12.6 ± 5.9	12.5 ± 5.8	10.9 ± 5.0	0.9 ± 0.1
P@10	20.4 ± 13.0	29.6 ± 16.7	29.8 ± 18.1	25.3 ± 12.7	0.2 ± 0.5

Table 4.1: Performance(%) with NB with 90% confidence intervals.

Error Analysis We analyzed the coefficients of the trained Naive Bayes models and the predictions made by the system, specifically we tried to understand why the Naive Bayes model sometimes gave a very low rank to relevant entities. For the first part of the analysis we checked the most important features selected by the method for a few trials of the algorithm for the predicate "EmploymentMembership=Army". We found that the NB model gave a high score to intuitive feature values that correlated with the true predicate, for example it gave high weight to the "EmploymentMembership=Pentagon" feature when trained to recommend entities that satisfied "EmploymentMembership=Army". Such an assignment of high scores to related entities is an example of the kind of rules that we wish to learn.[3]

Useful features for recommending entities are tabulated in table 4.1.

Relation	**Value**	**Features**	
Role	author	Role=Director	Role=Executive
Role	director	EmploymentMembership=U.S.	StudentAlum-HQ=U.S.
EmploymentMembership	Army	EmploymentMembership=Pentagon	Role=spokesman
EmploymentMembership	White House	EmploymentMembership=United States	Role=representative
Resident	Chinese	Leadership-HQ=Chinese	Leadership=Communist Party
Resident	Texas	EmploymentMembership=Texas	Origin=American
Leadership	Democratic	EmploymentMembership=Democratic	Leadership=Democrats
Leadership	Parliament	Role=representative	Role=president
Origin	American	Resident=American	Role=president
Origin	Russia	Resident=Russian	EmploymentMembership=Russian
StudentAlum	Harvard	StudentAlum=Harvard Law School	EmploymentMembership=Harvard
StudentAlum	Stanford	Resident=California	Origin=American

Table 4.2: Highly weighted features for recommending PER entities according to relationship criterion. For example, we found that if the graph claimed that someone was a resident of "Russian" or employed by "Russian", then their origin might likely be "Russia".

One reason for measured "error" is that in some cases the NB algorithm made correct recommendations in the top 10, but were counted as incorrect since they were not part of

[3]Keeping in mind that such rules can only be applied to those PER vertices that have features under the graph, and that most do not.

the Adept KB. This highlights a known challenge associated with evaluating any such recommendation task (where the entire truth set is not known even at test time).

For the second part of the analysis we checked why the Naive Bayes model did not assign a high rank to the true answers: for half of the cases we found that the PER entities that were misclassified did not have any feature that occurred in $\mathcal{O}^+$. This further evidence of data sparsity in the graph: even for vertices sub-selected from the graph for purposes of these synthetic experiments, chosen in order that they had non-zero features, it was still often the case that there was no feature overlap between the test vertex and the training data (except for the single "feature" that represented the predicate we were directly trying to predict). We also observed on manual inspection a fair amount of lexical diversity in feature values: while we do not expect an order of magnitude impact, it did appear that further improvements to canonicalization of values in the KB might offer some limited improvement (e.g., a hypothetical example would be the difference between "Russia" and "Russian" as values in the graph, for the same relation argument across two examples).

4.2 Entity Recommendation Through Modified Adsorption

The MAD algorithm was introduced by (Talukdar and Crammer, 2009) as an algorithm for graph-based semi-supervised learning. The MAD algorithm receives a graph along with a labeled set $\mathcal{O}$ of entities in the graph and binary labels indicating their relevance or irrelevance to the predicate as input. As output, the MAD algorithm produces a soft assignment from the unlabeled nodes to the binary label set.[4]

The MAD algorithm produces a soft assignment by solving an unconstrained optimization problem over the space of all labelings where the objective is defined such that the labeling it produces are "smooth", i.e., the labels of nodes that share an edge tend to be similar, and it tends to agree with the input partial labeling and that it should labeling should be close to an eigenvector of the graph laplacian of the input graph. Ideally, the soft assignment that MAD produces should match the true unknown labels of the unlabeled vertices.

In order to apply MAD which is only defined for graphs with untyped edges to the Adept KB where the edges have types we use the following trick: for each edge of the form $\textcircled{v}\xrightarrow{r}\textcircled{q}$ where v is a PER entity, we change the edge from its original form to $\textcircled{v}\longrightarrow\textcircled{rq}$. I.e., for each edge incident on a PER entity we fuse the edge type with the neighboring vertex and leave the PER entity as is. This creates many new neighboring vertices and removes typing information from the edges. As explained in section 4.1 there are two variants of the Adept KB that we consider, the first variant does not contain document co-occurrence information but the second variant contains it. In order to add document co-occurrence information to the Adept KB graph we create "document" nodes and add edges between PER entities and the documents that they occurred in. For both

[4]The MAD algorithm can be stated in more general terms but the above treatment suffices for our purposes. We refer the reader to (Talukdar and Crammer, 2009) for details of the MAD algorithm and its analysis.

of the variants we created untyped versions of the original graph. We present the results of applying the MAD algorithm to the Adept KB in table 4.3.

Results The results indicate that adding the document co-occurrence information hurts the performance of the MAD algorithm although the decrease is not statistically significant for either of the metrics. The overall performance of the MAD algorithms is much worse than the performance of the NB model especially on the P@10 metric.

	Concatenative	Concatenative w/ doc
AUPR	7.0 ± 2.8	5.4 ± 1.8
P@10	9.8 ± 11.2	9.6 ± 7.5

Table 4.3: Performance(%) with MAD with 90% confidence intervals.

4.3 Entity Recommendation with Unweighted Random Walks

The Unweighted Random Walk method uses statistics about the commute time of unweighted random walks to perform vertex nomination as follows: Given $\mathcal{O}^+$, for each $v \in \mathcal{O}^+$ we perform w random walks of some fixed length l that start from v. w and l are hyper-parameters which we set to 10 and 3 after some initial tuning. During each random walk we keep track of the vertices visited by that random walk and we aggregate the counts of all the vertices visited by a random walk, over all the random walks performed, for the set $\mathcal{O}^+$ corresponding to a given predicate and trial. This vector of counts is used to arrange the unlabeled vertices by assigning a high ranking to those vertices that were visited more often. We use this method on the two variants of graphs, "Concatenative" and "Concatenative w/ Doc" which have the same meaning as before.

Results We present the results in table 4.4. The results indicate that unweighted random walks perform worse than the MAD algorithm on the AUPR metric, but they outperform MAD in terms of precision. The performance of Unweighted Random Walks is worse than the Naive Bayes models.

	Concatenative	Concatenative w/ Doc
AUPR	2.6 ± 1.8	1.6 ± 0.7
P@10	12.0 ± 9.0	7.6 ± 7.3

Table 4.4: Performance(%) with Random Walks with 90% confidence intervals.

5 Conclusions

Our experiments with vertex nomination on the Adept KB indicate that the the closed ontology under which the KB was generated was too restrictive. Because of the small number of relations in the ontology, a large number of named entities that were extracted from the text corpus remained disconnected from other entities, and clearly it is impossible to use such disconnected entities in a vertex nomination algorithm.

Amongst the small subset of the graph that was not disconnected, it is possible to perform vertex nomination using the Naive Bayes model however the performance was low. It is important to note that the Adept "Knowledge Graph" behaved more like a labeled set of features, or a bipartite graph, instead of a communication graph which the original vertex nomination methods were designed for and applied to. The original thought behind applying Vertex Nomination methods to Knowledge Graphs was the assumption that the Knowledge Graph is rich in entity to entity relations, much like in a social network. What we observe is that the KG is not rich in Entity to Entity relationships, e.g., out of a total of $134,581$ relations only $3,508$ relations connect one person entity to another.

Proposed Next Steps The results indicated that the current Adept KB does not support methods for vertex nomination since a majority of the entities in the KB are disconnected, therefore, we propose to investigate methods to "expand" the ontology and introduce more relations in the ontology.

References

Marjorie Freedman Bonan Min, Constantine Lignos. Bbn's 2015 system for cold start knowledge base population. 2015.

TAC-KBP@NIST. *Cold Start Knowledge Base Population at TAC 2015 Task Description.* NIST, 1.1 edition, July 2015.

Partha Pratim Talukdar and Koby Crammer. New regularized algorithms for transductive learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 442–457. Springer, 2009.