Construction and Applications of TeKnowbase – A Knowledge Base of Computer Science Concepts

Prajna Upadhyay
Indian Institute of Technology
New Delhi, Delhi
prajna.upadhyay@cse.iitd.ac.in

Ashutosh Bindal
Indian Institute of Technology
New Delhi, Delhi
ashutoshbindal@gmail.com

Manjeet Kumar
Indian Institute of Technology
New Delhi, Delhi
manjeetk9497@gmail.com

Maya Ramanath
Indian Institute of Technology
New Delhi, Delhi
ramanath@cse.iitd.ac.in

ABSTRACT

In this paper, we make two main contributions. First, we describe the construction and evaluation of TeKnowbase, a knowledge-base of technical concepts in computer science. And second, we show how to use TeKnowbase in a variety of applications, including, generation of pre-requisite concepts for learning a new topic, classification of technical text and querying and ranking computer science articles.

CCS CONCEPTS

• **Information systems** → *Ontologies*; *Information extraction*;

KEYWORDS

knowledge base, technical concept, pre-requisites, ranking, classification

ACM Reference Format:
Prajna Upadhyay, Ashutosh Bindal, Manjeet Kumar, and Maya Ramanath. 2018. Construction and Applications of TeKnowbase – A Knowledge Base of Computer Science Concepts. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3184558.3191532

1 INTRODUCTION

With advances in information extraction research, and the availability of large amounts of structured and unstructured (textual) data, *automatic* construction of knowledge-bases are not only possible, but also desirable because of the coverage they can offer. There are already many such general-purpose knowledge-bases such as Yago [22] and DBPedia [11]. Moreover, projects such as OpenIE [2] and NELL [3] aim to extract information from unstructured textual sources on a large scale. However, there is a paucity of high quality and *specialized* KBs for specific domains. For some domains, for example, for the bio-medical domain, there are well-curated ontologies which partially address this gap (see, for example, the Gene Ontology project [1]). However, for domains such as Computer Science or IT in general, where such curation efforts are hard and the field itself is rapidly growing, it becomes critical to revisit the automatic construction processes that take advantage of domain-specific resources.

In this paper, we describe the construction of a *technical* knowledge-base (TKB) of computer science concepts, called *TeKnowbase* from Wikipedia, technical websites and online textbooks. As with any general-purpose KBs, TeKnowbase can be used in applications such as classification, disambiguation, entity linking, semantic search, etc., of *computer science* resources. However, in addition to these "standard" applications, TeKnowbase can be used in the domain of education for problems which are specific to it. One such problem is the generation of pre-requisite concepts. Briefly, this problem arises when a student wants to learn about a new concept, but is unsure of what pre-requisites are required. The goal is to develop a system that automatically generates the required pre-requisites. As we illustrate in Section 3.1, using TeKnowbase as a resource to generate these pre-requisites results in improved coverage and accuracy. In addition to pre-requisite generation, TeKnowbase can be used as a resource for automatic question generation (to generate, for example, multiple choice or fill-in-the-blanks questions) to test the student's understanding of a subject. The utility of TeKnowbase in these kinds of education-specific applications makes it a valuable resource.

Construction methodology. In order to construct TeKnowbase, we first acquired a dictionary of concepts and entities relevant to computer science (Section 2.1). Using this dictionary, we extracted relationships among them (Section 2.2). We make use of the semantic web standard, RDF, where information is represented as triples of the form ⟨subject⟩⟨predicate⟩⟨object⟩, to represent these relationships. The ⟨subject⟩ and the ⟨object⟩ are entities and the ⟨predicate⟩ represents the relation. In a nutshell, each triple makes a statement about the ⟨subject⟩. Table 1 shows examples of the kind of triples we extract and the number of such triples in our knowledge-base.

Availability of TeKnowbase. TeKnowbase is made available under the Creative Commons Attribution Licence 3.0 and can be downloaded from https://github.com/prajnaupadhyay/TeKnowbase. Some statistics about TeKnowbase are listed in Table 2. Line 1 in Table 2 reports the number of unique entities as well as the variations such as plurals. Entities that we have in common with DBPedia and Freebase are linked using the `owl:sameAs` relation.

Each entity in TeKnowbase is associated with a URI. The URI, consists of the URL from which the entity was extracted. The URL is typically a page dedicated to describing the entity. Further, since

This paper is published under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.
WWW '18 Companion, April 23–27, 2018, Lyon, France
© 2018 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC BY 4.0 License.
ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3191532

Table 1: Statistics for and examples of a subset of relationships extracted. The The first set of 5 relations were extracted from structured sources (see Section 2.2.1) and the second part with 3 relations from unstructured textual sources (see Section 2.2.2).

Relation	Examples of ⟨head_entity, tail_entity⟩	# Triples
type	⟨topological_sorting, graph_algorithm⟩	27,078
concept	⟨nash_equilibrium, game_theory⟩	595
subTopic	⟨hamming_code, algebraic_coding_theory⟩	2,026
application	⟨group_testing, coding_theory⟩	324
terminology	⟨blob_detection, image_Processing⟩	27,018
is_a_high-speed_form_of	⟨gigabig_ethernet, ethernet⟩	n/a
is_an_adaptation_of	⟨ironpython, python⟩	n/a
uses	⟨utc, gregorian_calendar⟩	n/a

Table 2: Statistics from TeKnowbase. Apart from $70,285$ **unique entities, there are** $32,458$ **variations (disambiguated as well as stemmed versions) of them.**

No. of unique entities	70,285
No. of unique relations	2,574
Taxonomic relations (typeOf)	27,078
Total no. of triples	146,657
No. of overlapping entities with DBPedia	17,987
No. of overlapping entities with Freebase	34,785
No. of triples extracted from Wikipedia	99,357
No. of triples extracted from Unstructured sources	3,506

each triple in TeKnowbase is derived through a series of heuristics (described in Sections 2.1 and 2.2), we maintain the entire provenance of the triple. This data is available on request.

1.1 Contributions and Organization

Our contributions are as follows.

(1) We describe the construction of TeKnowbase in Sections 2.1 and 2.2. We conducted a thorough evaluation of the quality of TeKnowbase and we report our results in Section 2.3. TeKnowbase is a freely available online resource.
(2) We describe 3 applications of TeKnowbase: i) Generation of pre-requisite concepts, ii) classification of posts in Stack Overflow (a computer science forum), and, iii) ranking of computer science research articles for keyword search. We show through our evaluations that using TeKnowbase as a resource can improve the accuracy of all 3 applications.

Our paper is organized as follows. We describe the construction of TeKnowbase in Sections 2.1 and 2.2. We present three applications of TeKnowbase in Section 3. We present an evaluation of our knowledge-base in Section 2.3. We briefly describe related work in Section 4 and conclude in Section 5.

2 CONSTRUCTION OF TEKNOWBASE

2.1 Acquiring a list of entities

We extracted nearly $78,000$ entities. We used Wikipedia's article titles as well as it's category system as a source of concepts. Our corpus of Wikipedia articles consists of all articles under the category `Computing`. In all, there were approximately $54,000$ articles. The title of each article was considered an entity. Examples entities we found were `Heap_Sort`, `Naive-Bayes_Classifier`, etc. While Wikipedia has articles on a number of technical entities and concepts, it is not exhaustive. For example, the terms `average_page_depth` (related to Web Analytics) and `fraction_ridge` (related to biometrics) could not be found in Wikipedia, and therefore, we looked for several online resources to augment our entity list.

Our second set of resources were two websites, Webopedia[1] and TechTarget[2]. Each website consists of a number of technical terms and their definitions in a specific format. By writing appropriate wrappers for both these websites, we extracted approximately $24,000$ entities. Finally, we extracted $16,500$ entities from the *indexes* of online textbooks (8 online textbooks were used). We used edit distance to resolve overlapping entities from these sources and ended up retaining over $70,000$ entities.

2.2 Acquiring relationships between entities

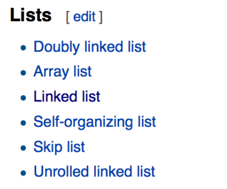
We divided our relationship extraction task into two parts. First, extraction from structured sources, and second, extraction from unstructured, textual sources. Our goal was two-fold: to extract as many different kinds of relationships as possible as well as construct a taxonomy of entities/concepts.

2.2.1 Structured sources. Since our aim was to construct a technical knowledge-base, we manually made a list of relations that our knowledge-base should contain – this is our list of known relations. The relations included the taxonomic relation `typeOf` (as in, ⟨`jpeg typeOf file_format`⟩) and other interesting relationships such as `algorithmFor`, `subTopicOf`, `applicationOf`, `techniqueFor`, etc. In all, we identified 18 relationships that we felt were interesting and formulated techniques to extract them from Wikipedia.

Overview pages. (500 pages): We made use of two kinds of structured pages – "List" pages and "Outline" pages (for example, the pages, `List of machine learning concepts`, `Outline of Cryptography`, etc.). These pages organize lists of entities with headings and sub-headings. Extracting this information gives us the relations `typeOf` and `subTopicOf` with good accuracy (see Section 2.3 for an evaluation of these relations). Figure 1 shows an example of a list page for data structures. We see a list of terms under a heading and can extract triples of the form ⟨`doubly_linked_list typeof list`⟩. Further we were able to extract *taxonomic hierarchies* of two levels by relating the headings to the article title. Continuing the previous example, ⟨`list typeOf data_Structure`⟩ was extracted based on the article title.

Articles on specific topics: These pages refer to discussion on specific topics such as, say, "Coding Theory". These pages consist of many structured pieces of information as follows:

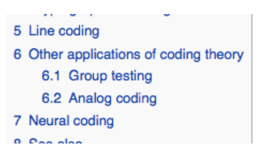
[1]http://www.webopedia.com
[2]http://www.techtarget.com/



Lists [edit]

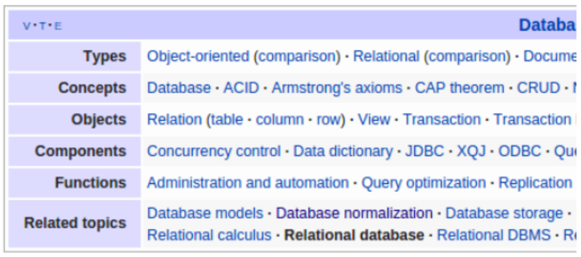
- Doubly linked list
- Array list
- Linked list
- Self-organizing list
- Skip list
- Unrolled linked list

(a) Snippet from "List of Data structures" page to extract `typeOf` relations



5 Line coding
6 Other applications of coding theory
6.1 Group testing
6.2 Analog coding
7 Neural coding

(b) Snippet of the TOC in "Coding theory" page to extract `applicationOf` relations



V·T·E	Databa
Types	Object-oriented (comparison) · Relational (comparison) · Docume
Concepts	Database · ACID · Armstrong's axioms · CAP theorem · CRUD ·
Objects	Relation (table · column · row) · View · Transaction · Transaction
Components	Concurrency control · Data dictionary · JDBC · XQJ · ODBC · Qu
Functions	Administration and automation · Query optimization · Replication
Related topics	Database models · Database normalization · Database storage · Relational calculus · **Relational database** · Relational DBMS · R

(c) DBMS template to extract other relations

Figure 1: Snippet of different structured sources

The table of contents (TOC) (1838 TOCs): From our list of known relations, we searched for keywords within the TOC. If the keyword occurred in an item of the TOC, then the sub-items were likely to be related to it. For example, in Figure 1, the Coding Theory page consists of the following item in its TOC: "Other applications of coding theory" and this in turn consists of two sub-items "Group testing" and "Analog coding". Since one of the keywords from our known relations is "application", and the page under consideration is "Coding Theory", we extract the triples ⟨`group_testing applicationOf coding_theory`⟩ and ⟨`analog_coding applicationOf coding_theory`⟩.

Section-List within articles (1909 section lists): Next, there are several sub-headings in articles which consist of links to other topics. For example, the page on "Automated Theorem Proving" consists of a subheading "Popular Techniques" – this section simply consists of a list of techniques which are linked to their wikipedia page. Since "technique" is a keyword from our list of known relations, we identify this section-list pattern and acquire triples such as ⟨`model_checking techniqueFor automated_theorem_proving`⟩.

List hierarchies in articles (113 list hierarchies): As in the case of "List" pages and "Outline" pages, we make use of list hierarchies in articles to extract the `typeOf` relationships.

Templates (1139 templates): Figure 1 shows an example of a template from the "Database Management Systems" page. We extracted row headings as potential relations and added triples such as ⟨`query_optimization functionOf database_management_systems`⟩. However, based on our evaluation (see Section 2.3), we found that templates could have a variety of row headers and they were not always reliable. Therefore, we did not canonicalise these relations, but instead treated them as a generic "relatedTo" relation (therefore, instead of `functionOf`, we had `relatedTo_(functionOf)`).

2.2.2 *Unstructured sources.*

Our unstructured sources include textual description of terms in both Webopedia and Techtarget as well as Wikipedia text of articles corresponding to entities. We limited the text in Wikipedia to the first paragraph. As mentioned previously, we acquired a number of `synonymOf` relations. We were successful in identifying the `synonymOf` relation by using the patterns "is abbreviation for", "X (Y)" and "is short for". We obtained over 1,000 such triples.

Next, we ran the latest version of OpenIE [14] on each of these sources. While the number of extractions was quite large (approximately 400,000 triples), there were a lot of "junk" extractions such as ⟨`as_a_specific example_for file_distribution`⟩. In order to improve the accuracy, we applied the following filters to the extracted triples.

Filter 1: Triples which did not contain entities from our entity dictionary were discarded. This filter alone reduced the number of unuseable triples by nearly 300,000.

Filter 2: We found that entities in the triple could be too long. Therefore, our second filter retained only those triples which had an entity match of at least 50% – that is, 50% of the entity identified by OpenIE was a match for an entity in our dictionary.

Filter 3: Despite these two filters, we found cases where the subject or object referred to conceptual terms such as `the_algorithm` as part of the triple (the algorithm;generates;a path), where the entities are algorithm and path. Therefore, our third filter removed triples where the subject or object started with the word "the" and were less than 3 words in length.

After applying all these filters, we retained 3506 triples. We canonicalised a small percentage of the relations and we omit the details here for lack of space. In general, we were able to identify fine-grained relationships among entities. Table 1 shows a few examples.

2.2.3 *TeKnowbase completion.*

So far we have shown techniques to acquire entities and triples from different web-resources using heuristics. In this section, we focus on acquiring more triples between the entities based on the triples already present in TeKnowbase. This task is well-known as *inferencing in knowledge bases* or *knowledge base completion* [17, 21]. For example, if we have the triple ⟨`knights_tour typeof state_space_search`⟩ and ⟨`state_space_search typeof graph_algorithm`⟩, it can be easily inferred that ⟨`knights_tour typeof graph_algorithm`⟩ using the transitive property, even if this triple is not present in the knowledge base. To extract such rules, [21] have introduced a *neural tensor network* (NTN) that models each relation through its hyperparameters and generalizes several other neural network models. The bilinear tensor product is the primary operation used to relate entities to each other in a neural tensor network, unlike other neural networks. Each entity in the knowledge base is represented as a vector in higher dimension. This vector is obtained by taking a reduced mean of the vectors of the constituent words in the entity name. The vectors for each word is obtained using Word2Vec [15] using the Skip-gram model. Moreover, they have also shown that the accuracy of the model improves when it is initialized with vectors trained on large unstructured corpora. We have used a similar approach and experimented with the following models:

- Word2Vec (with the entities treated as one unit) on wikipedia corpus and textbooks: We trained phrase vectors for each entity using Word2Vec. The training set consisted of unstructured Wikipedia corpus as well as text from online textbooks.

Table 3: Some new triples that were added to TeKnowbase using the Neural Tensor Network inferencing model.

1.	⟨xbasic,typeOf,programming_language⟩
2.	⟨binomial_heap,typeOf,tree⟩
3.	⟨palmdos,typeOf,operating_system⟩
4.	⟨delayed_column_generation,typeOf,convex_programming⟩
5.	⟨levenshtein_coding,typeOf,entropy_coding⟩

These vectors were then used to initialize the neural tensor network model.

- Word2Vec (with the entities treated as one unit) trained only on wikipedia corpus: Same as above but the vectors were only trained on Wikipedia corpus. Information from textbooks was not included.
- Word2Vec trained on wikipedia corpus and textbooks: We used Word2Vec to train vectors for each word and later obtained vector representation for each entity by the reduced mean of component word vectors. Similar to i), we used text from online textbooks (apart from unstructured text from Wikipedia) to train these vectors and used them to initialize the neural tensor network model.
- Word2Vec trained only on wikipedia corpus: Same as above, except that we excluded textbook information to train vectors used to initialize the neural tensor network model.

We found that i) performed the best, i.e. Word2Vec with entities treated as one unit initialized with textbooks information. A total of 428 triples could be added to TeKnowbase using the inferencing model. Some of them are listed in Table 3. Evaluation of these triples is described in Section 2.3.

2.3 Evaluation of TeKnowbase

2.3.1 Setup. We chose the top-5 frequent relations extracted for evaluation. These were: typeOf, terminology, synonymOf subTopicOf and conceptIn. Together, these five relations constitute about 63% of the triples in our KB. We used stratified sampling to sample from each type of relation. For each relation, we sampled 2% of the triples. Since not every triple extracted from the *unstructured sources* were canonicalised, we evaluated these triples separately by sampling about 2% of the triples. Each triple was evaluated by two evaluators and we marked a triple as correct only if both evaluators agreed. For inferencing, we experimented with 4 different models (as described in Section 2.2.3). To evaluate these models, we created test set in the following way – we generated a list of *true* triples using the transitive property on the typeOf relations. For example, given triples ⟨palmdos,typeOf,dr_dos⟩ and tripledr_dos,typeOf,operating_system in the training set, we returned a triple ⟨palmdos,typeOf,operating_system⟩ in the test set, ensuring that it does not already exist in the training set. To generate negative examples, we shuffled the head entities of the positive set of triples. We evaluated the accuracies of these models on this test set.

2.3.2 Results and Analysis. Table 4 shows the accuracy of triples for each relation. We computed the Wilson interval at 95% confidence for each relation. On closer examination of these results, we found that we achieved the best results for the synonym relation consisting of expansions of abbreviations, such as ALU and Arithmetic Logic Unit as well as alternate terminology such as Photoshop and Adobe Photoshop.

Table 4: Evaluation of a subset of triples in the TKB.

#	**Relation** (rows 1–5)	**# Evaluated triples**	**Accuracy**
1.	typeOf	515	99.0% ± 0.8%
2.	terminologyOf	676	98.9% ± 0.7%
3.	synonymOf	70	100% ± 0.0%
4.	subTopicOf	42	91.3% ± 8.2%
5.	conceptIn	334	95.4% ± 2.1%
6.	*Unstructured sources*	435	63.2% ± 3.7%
7.	*Inferencing with NTN*	428	64.2% ± 4.5%

The best source of extractions are the Wikipedia list pages. In our list of top-5 relations, only 3 were extracted from Wikipedia list pages – typeOf, subtopicOf and synonymOf – and all of them were nearly 100% accurate.

The major source of errors in many of these relations was due to extractions from TOC items (Section 2.2.1). This heuristic did not work well to identify the correct relation. For example, one of the errors was made when “Game types” was an item in the TOC of the page “Game Theory”. It listed “Symmetric/Asymmetric” as a type of game, but we extracted ⟨symmetric/asymmetric typeOf game_theory⟩ which is incorrect.

Unstructured sources. The overall accuracy of triples from unstructured text was found to be 63.2%. Recall that we ensure that the entities are always correct since we filter out triples which do not match an entity in our dictionary. Therefore, the main reason for low accuracy is that extracted relations were incorrect. Some incorrect extractions include: ⟨user requests mail⟩, ⟨packet switching protocol⟩. We are analysing these errors in more detail and improving the accuracy of these extractions in future work.

Inferencing with Neural Tensor Network. The neural tensor network model performed the best when training corpora from textbooks was added. It improved the accuracy of prediction for both Word2Vec models (with words treated separately and with entities treated as a single unit). Additionally, we also observed that Word2Vec with entities treated as one unit outperformed the model where separate vectors are learnt for each word in the technical domain. We were able to add 428 triples to TeKnowbase using this model with an accuracy of 64.25%.

3 APPLICATIONS OF TEKNOWBASE

In this section, we show that TeKnowbase can be useful in 3 application settings – determining pre-requisites for technical concepts, classification of technical documents and improving ranking of academic search.

3.1 Determining pre-requisites for technical concepts

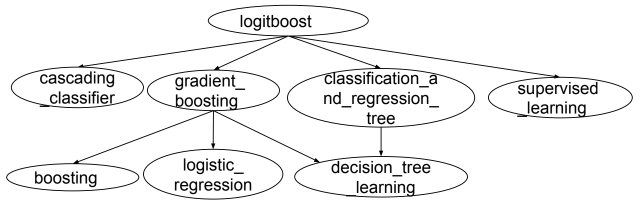
Learning a new technical concept can be challenging because it involves identifying and studying its *pre-requisites.* A pre-requisite is any concept that has to be studied before another for better understanding. Although it is possible to acquire a number of relevant documents by searching for the concept in a search engine, there could be concepts mentioned in those documents that need to be searched again, eventually leading to cascade of searches and wasting a lot of user's time. In such a scenario, a retrieval system that identifies and returns the pre-requisites is extremely useful. For example, given the concept `logitboost`, we would like the system to return pre-requisites as shown in Figure 2 (a). In this section, we show how TeKnowbase can be used to retrieve pre-requisites for technical concepts.

3.1.1 Problem. Previous work on finding pre-requisites can be categorized into i) constructing pre-requisite functions to determine if a concept is a pre-requisite of the other [7, 13, 24] ii) generating concept graphs given a set of concepts [25]. All these techniques have typically relied on features from textual sources, structure of textbooks and learning from training examples to construct or learn pre-requisite relationships. To retrieve pre-requisites with better precision and recall, we need to consider relationships between the concepts. We wish to do the same using TeKnowbase. Two key problems that we address using a technical knowledge base are:

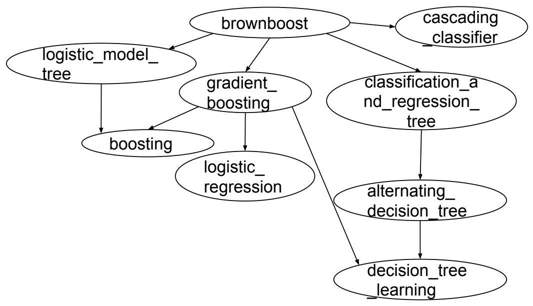
Low Precision. We have observed that all the pre-requisites are not equally "relevant". For example, both `binary_search_tree` and `file_system` are returned as pre-requisites of `b-tree` by the pre-requisite function RefD proposed in [13]. A person having no knowledge of `b-tree` would definitely gain more if `binary_search_tree` is prioritized over `file_system` because `binary_search_tree` helps understand what `b-tree` is better than `file_system`. `file_system` addresses another aspect towards understanding `b-tree` – stating that `b-tree` is *used* in the implementation of `file_system`. RefD models each concept as a "frame" of related concepts. The choice of frame is crucial towards determining relevant pre-requisites. RefD uses concepts that are Wikipedia neighbours to determine the frame. However, not all concepts that are Wikipedia neighbours should be included in the frame. As described in the example previously, `file_system` and `b-tree` are mentioned in the first paragraph of the Wikipedia page of `b-tree`. Due to this, it retrieves non-relevant pre-requisites like `file_system`. To discard such concepts from the frame, we model each concept as a vector in a higher dimensional space. These vector representations or *embeddings* are obtained by running *Node2Vec* [8] on TeKnowbase with an emphasis on `typeOf` relation. As a result, the vector representations of concepts related via `typeOf` will be closer to each other than the rest. For example, the vector representations of `b-tree` and `binary_search_tree` will be closer to each other than `file_system`. We use this intuition to discard non-relevant concepts from the frame. One of the approaches to do so is to cluster the neighbours and retain the cluster that `b-tree` belongs to as the new frame.

Low Recall. Another limitation of previous work is that they focus on improving the precision of the pre-requisites and ignore the problem of low recall. We have observed that pre-requisites of similar concepts are the broadly the same. For example, Figure 2 (a). and (b). shows the pre-requisites for `logitboost` and `brownboost` respectively. Both `logitboost` and `brownboost` are algorithms for `boosting`. It is clear that the pre-requisites of both the concepts are roughly the same and that adding the pre-requisites of `brownboost` (shown in the same figure) improves the recall of `logitboost`. We have exploited this simple observation to improve the recall of pre-requisites. To determine that `brownboost` is most similar to `logitboost`, we make use of entity embeddings trained on our technical knowledge base. Because embeddings have performed very well in determining "similar" words or entities, we have used them to determine the most similar concept to borrow pre-requisites from.

Our technique is briefly described in Algorithm 1. Basically, TeKnowbase is used in 2 ways – i) to improve the precision by determining better frames and ii) to improve the recall by identifying the siblings from TeKnowbase taxonomy whose pre-requisites can be borrowed. Please note that although we have used RefD [13] for our experiments, our technique consisting of determining better frames and improving recall can be used with *any* pre-requisite function.



(a) Pre-requisites for `logitboost`



(b) Pre-requisites for `brownboost`

Figure 2: Pre-requisites for `logitboost` and `brownboost`

3.1.2 Setup. We used the Wikipedia data dump of 2015 for our experiments. Our technique is referred to as OWN_KB, which uses embeddings trained on TeKnowbase to determine the frame. For the pre-requisite function, we used RefD, proposed in [13], which uses all the concepts mentioned in the Wikipedia page as the frame. Apart from RefD, we used 2 other baselines, described as follows:

OWN_TEXT. This uses our technique, but the entity embeddings are generated by training on Wikipedia text. Phrase vectors are trained for each entity using Word2Vec algorithm by treating entities as a single unit.

OWN_COMBINED. This also uses our technique, but the entity embeddings are generated by concatenating the embeddings used in OWN_KB and OWN_TEXT.

3.1.3 Benchmarks. We selected 26 queries spanning different areas of computer science and generated their pre-requisites using RefD, OWN_KB, OWN_TEXT and OWN_COMBINED.

3.1.4 Gold standard. The gold standard list of pre-requisites for each query was generated by experts in that domain. This gold standard list is used to measure the precision and recall.

3.1.5 Evaluation metrics and Results. We evaluated the precision, recall and F1 scores for the pre-requisites. The results are shown in Table 5. Additionally, we also asked two evaluators to rate each edge (a, b) in the pre-requisite graph P_G as follows:

Algorithm 1: Generate pre-requisite graph P_G for C_i

```
**Result:** Pre-requisite Graph $P_G$
$F_i$ = determineFrame($C_i$)();
$P_C$ = Pre-requisites of $C_i$ determined using frame $F_i$ and pre-requisite function $P_F$;
$F_G$ = improveRecall($C_i$, $P_C$)();
Construct a graph $P_G$ where nodes belong to $F_G$;
foreach pair of nodes $(a, b)$ in $P_G$ do
    Determine if $a$ is a pre-requisite of $b$ using $P_F$;
end
From $P_G$, delete edges $(a, b)$ such that $b$ is a descendant in the knowledge base's taxonomy of $a$.;
return $P_G$;
Procedure determineFrame($C_i$)
    Let $S$ = set of Wikipedia neighbours of $C_i$. Let $e_i$ be the $i$-th entity in $S$. $V(e_i)$ is the vector representation of $e_i$;
    Use agglomerative clustering to cluster $S$ using cosine similarity between the vectors as the distance measure;
    Use ch-index to choose the best cluster set and set the frame $F_i$ = the cluster to which $C_i$ belongs;
    return $F_i$;
Procedure improveRecall($C_i$, $P_C$)
    $F_G$ = $P_C$;
    foreach sibling $e_i$ of $C_i$ in technical knowledge base do
        determine the cosine similarity of $e_i$ with $C_i$;
    end
    $L$ = list of siblings of $C_i$ in the technical knowledge base ranked in decreasing order of cosine similarity;
    foreach $e_i$ in $L$ do
        $F_E$ = determineFrame($e_i$)();
        $P_D$ = Determine pre-requisites for $e_i$ using frame $F_E$ and pre-requisite function $P_F$;
        if ($P_D$ - $P_C$) is not empty then
            $F_G = P_C \cup P_D$;
            break;
    end
    return $F_G$
```

Option 1. When b is required and crucial for understanding a
Option 2. When b improves understanding of a
Option 3. When b is a field of study
Option 4. When a is a prerequisite of b, i.e. inversely related
Option 5. None of the above

We computed the *accuracy* of edges as follows:

$$Acc_1 = \frac{\sum_{e_i \in E_1} I(e_i)}{|E_1|}, \text{ where } I(e_i) = 1, \text{ if the edge } e_i \text{ is either marked as Option 1 or Option 2, else } I(e_i) = 0 \tag{1}$$

where E_1 is the set of all edges in P_G. We also evaluated the accuracy for edge-sequences of length 2. The accuracy of edge-sequences of length 2 is computed as follows:

$$Acc_2 = \frac{\sum_{e_i \in E_2} I(e_i)}{|E_2|}, \text{ where } I(e_i) = 1, \text{ if both the edges in edge sequence } e_i \text{ are marked as Option 1 or Option 2, else } I(e_i) = 0 \tag{2}$$

where E_2 is the set of all edge-sequences of length 2 in P_G.

Table 5: Precision, Recall and F1 scores obtained after using different frames for RefD, OWN_KB, OWN_TEXT **and** OWN_COMBINED**. Pruning the frame is used to improve the precision of relevant pre-requisites.**

Representation scheme	Precision	Recall	F1-score
RefD	0.29	**0.42**	0.34
OWN_KB	**0.86**	0.34	**0.49**
OWN_TEXT	0.25	0.08	0.12
OWN_COMBINED	0.50	0.18	0.26

Table 5 compares the precision, recall and F1 scores of RefD, OWN_KB, OWN_TEXT and OWN_COMBINED after improving the frames. OWN_KB returns the best precision of 86% as compared to RefD, OWN_COMBINED and OWN_TEXT which return 29%, 50% and 25% precision respectively. Clearly, this is due to using better frames generated by the technical knowledge base. The recall of OWN_KB is 34% and lower than that of RefD but it still performs better in terms of F1-score while the recall obtained by OWN_TEXT and OWN_COMBINED are even worse.

To improve the scores further, `improveRecall()` function is used. It finds out the most similar sibling from the knowledge base taxonomy and adds its pre-requisites using different techniques – OWN_KB, OWN_COMBINED and OWN_TEXT. Table 6 shows the precision and recall values after this function improves the recall for each of the techniques. The precision of OWN_KB before adding was 86% but the recall was low – only 34%, even lesser than the recall of the baseline RefD. The recall improved to 60%, almost by a factor of 2 at the cost of bringing the precision down by only 14% and improving the F1-score from 0.49 to 0.64. Hence, it performs the best. However, the improvement is not very significant for OWN_TEXT or OWN_COMBINED. The reason is poor quality of frames retrieved by them that returned non-relevant pre-requisites. The recall of OWN_COMBINED improved from 18% to 28% at the cost of bringing precision down from 50% to 43%. The F1- score, however, increased slightly from 0.26 to 0.29. Although its precision (43%) is higher than RefD (29%), it failed to beat RefD in recall or F1-score. OWN_TEXT performs the worst. Its recall increased by a small margin – from 8% to 12% at the cost of 4% reduction in precision – 25% to 21%. The F1-score also increased by a small range from 0.12 to 0.16. The reason for it performing poorly is using concepts that occur near to each other in text in the frame. The poor performance of OWN_TEXT led to the sub-par performance of OWN_COMBINED since the embeddings used in OWN_COMBINED are constructed from OWN_TEXT embeddings. Overall, OWN_KB performs better than each of the other techniques due to the better quality of frames.

Table 7 lists the accuracies – Acc_1 and Acc_2 obtained from user evaluation. The accuracies have been calculated as described in Section 3.1.5. Again, the pre-requisite graph generated using OWN_KB obtains the highest accuracy of 83% for edges and 60% for edge sequences of length 2. OWN_COMBINED obtains an accuracy of

Table 6: Precision, Recall and F1 scores obtained after identifying the most similar sibling using OWN_KB, OWN_TEXT **and** OWN_COMBINED **and adding its pre-requisites. The pre-requisites of siblings from knowledge graph taxonomy are roughly the same and improve recall by significant margin for** OWN_KB**.**

Representation scheme	Precision	Recall	F1-score
RefD	0.29	0.42	0.34
OWN_KB	**0.72**	**0.60**	**0.64**
OWN_TEXT	0.21	0.12	0.16
OWN_COMBINED	0.43	0.28	0.29

Table 7: Results from user evaluation.

Representation scheme	Acc_1	Acc_2
RefD	0.42	0.21
OWN_KB	**0.83**	**0.60**
OWN_TEXT	0.48	0.12
OWN_COMBINED	0.69	0.40

69% for the edges which is not bad but performs poorly for edge sequences of length 2 (40%). However, it performs better than RefD. This is evident because we earlier found that OWN_COMBINED obtained better precision than RefD. The pre-requisite graph generated using OWN_TEXT performs the worst with 48% and 12% accuracies for edges and edge sequences of length 2. Overall, OWN_KB is the winner.

3.2 Classification experiment

[6] showed that the addition of features from domain specific ontological KBs can improve classification accuracy. Adapting this idea for our setting of a technical KB, a document belonging to the class "databases" may not actually contain the term "database", but simply have terms related to databases. If this relationship is explicitly captured in TeKnowbase, then that is a useful feature to add. We adapted this idea for our setting of a technical KB and classified posts from StackOverflow[3].

3.2.1 Setup. StackOverflow is a forum for technical discussions. A page in the website consists of a question asked by a user followed by several answers to that question. The question itself may be tagged by the user with several hashtags. The administrators of the site classify the question into one of several technical categories. Our task is to classify a given question *automatically* into a specific technical category. We downloaded the StackOverflow data dump and chose questions from 3 different categories: "databases", "networking", and, "data-structures" We created a corpus of 1500 questions including the title (500 for each category). The category into which the questions were manually classified by the StackOverflow site were taken as the ground truth.

3.2.2 Features generation. We generated the following set of features for training.
BOW: Bag of words.
BOW+BOE: Bag of words and bag of entities. Entities are treated as a whole, rather than a bag of words. Note that these entities were identified using the entity list of TeKnowbase.
BOW+BOE+TKB: In addition to the words and entities above, for each *entity*, we added features from the 1-hop neighborhood of the entity. For example, if the entity `run_length_encoding` occurred in the post, then we added as a feature, `data_compression` since we have the triple
⟨`run_length_encoding methodOf data_compression`⟩.

3.2.3 Classification algorithms. We trained both a Naive-bayes classifier as well as SVM with each of the feature sets above.

3.2.4 Results. We performed 5-fold cross validation with each classifier and feature set and report the accuracies in Table 8. Clearly, simply adding new features from TeKnowbase helps in improving the accuracies of the classifiers. This result is encouraging and we expect that optimizing the addition of features (for example, coming up with heuristics to decide which relations to use) will result in further gains.

Table 8: Average classification accuracies.

	BOW	BOW + BOE	BOW + BOE + TKB
SVM	82.1%	87.1%	92.0%
Naive Bayes	86.3%	88.4%	89.6%

Table 9: NDCG@20 values obtained with tf-idf and BM-25 ranking models. The numbers of queries where these models won (W), tied (T) or lost (L) to BOW is listed along with the NDCG scores.

	BOW	BOW + BOE + TKB, W/T/L
tf-idf	0.373	0.380, 32/9/40
BM-25	0.312	0.326, 41/9/31

3.3 Ranking experiment

The use of a bag-of-entities (BOE) model to represent queries and documents for document retrieval is outlined in [26]. We adapt this method to our setting to retrieve research articles in computer science. We note that the use of *knowledge-base embeddings* has been further explored in [27] (though their knowledge-base is different in structure and content to ours). We made use of the data generously provided by [27] to conduct our own ranking experiment with a BOE model.

3.3.1 Setup. The data consists of 100 technical queries (such as `semantic web`, `natural language interface`, etc.) derived from an analysis of the query log of Semantic Scholar[4], and a list of documents that are relevant to each query. We chose 81 of these 100 queries which contained entities from the knowledge bases we used for ranking and ran our experiments on those queries.

[3]stackoverflow.com

[4]https://www.semanticscholar.org

3.3.2 Techniques. We experimented with the following representations of both queries and documents.
BOW: Bag of words.
BOW+BOE+TKB: Bag of words and bag of entities. A pre-processing step identifies all entities occurring in the document and these entities are from the entity list of TeKnowbase. These entities are retained as a whole and not treated as a bag of words. Additionally, we expanded each entity tagged in the query/document by the entities occurring in 1-hop neighborhood in TeKnowbase.

3.3.3 Ranking models. We have used tf-idf [18] and BM-25 [20] ranking models to rank the candidate documents.

3.3.4 Results. We calculated NDCG@20 [10] and report the values in Table 9. We also counted on how many queries the NDCG score won (T), tied (T) or lost (L) to the bag of words model using TeKnowbase. We conclude that identifying and using entities in the document and query representations using TeKnowbase improves the quality of results. We believe that more sophisticated ranking models that use *entity embeddings* such as in [27] can further improve the quality of results, and this is a direction for future work. This establishes the usability of TeKnowbase in the ranking scenario.

4 RELATED WORK

Recently, systems which facilitate knowledge-base construction from heterogeneous sources have been proposed. For example, DeepDive [4] aims to consume a large number of heterogeneous data sources for extraction and combines evidence from different sources to output a probabilistic KB. Similarly, Google's Knowledge Vault [5] also aims to fuse data from multiple resources on the Web to construct a KB. Our effort is similar in that we make use of heterogeneous data sources and customise our extractions.

Entity extraction: One of the important aspects of building domain-specific knowledge-bases is that a dictionary of terms that are relevant to the domain should be acquired. It is possible that such dictionaries are already available (for example, lists of people), but for others, we need techniques to build this dictionary. [19] gives an overview of supervised and unsupervised methods to recognize entities from text. We follow a more straightforward approach – we specifically target technology websites and write wrappers to extract a list of entities related to computer science.

Information Extraction: Research in information extraction to build knowledge-bases make use of a variety of techniques (see [23] for an overview). In general, information extraction can be done from mostly structured resources such as Wikipedia (see, for example, YAGO [22]) or from unstructured sources (for example, OpenIE [2]) where the relations are not known ahead of time. Moreover, there are rule-based systems such as SystemT [12], using surface patterns and supervised techniques for known relations [9], distant supervision [16], etc. We use a mix of these approaches – we formulate different ways to exploit the structured information sources in Wikipedia, and use surface patterns and OpenIE to extract relationships from unstructured sources.

5 CONCLUSIONS

In this paper, we described the construction of a knowledge-base of technical concepts related to computer science from both structured and unstructured sources. Our approach consisted of two steps – constructing a dictionary of terms related to computer science and to extract relationships among them. Our experiments showed that our triples are accurate. Apart from using heuristics to extract triples, we also experimented with inferencing models and showed that all models improve after adding information from online textbooks. We have also demonstrated the usability of TeKnowbase in 3 application settings. Other interesting applications that can benefit from TeKnowbase are question answer generation and summarization of technical concepts. Our future work consists of developing such applications.

6 ACKNOWLEDGEMENTS

We want to thank Ashwini Purkar, Tanuma Patra and Akanksha Bansal for their contributions in building TeKnowbase.

REFERENCES

[1] Michael Ashburner et al. 2000. Gene Ontology: tool for the unification of biology. *Nature Genetics* 25, 1 (2000), 25–29.
[2] Michele Banko et al. 2007. Open Information Extraction from the Web. *IJCAI* (2007), 2670–2676.
[3] Andrew Carlson et al. 2010. Toward an Architecture for Never-Ending Language Learning. *AAAI* (2010), 1306–1313.
[4] Christopher De Sa et al. 2016. DeepDive - Declarative Knowledge Base Construction. *SIGMOD Record* 45, 1 (2016), 60–67.
[5] Xin Dong et al. 2014. Knowledge vault - a web-scale approach to probabilistic knowledge fusion. *KDD* (2014), 601–610.
[6] Evgeniy Gabrilovich and Shaul Markovitch. 2005. Feature Generation for Text Categorization Using World Knowledge. *IJCAI* (2005), 1048–1053.
[7] Jonathan Gordon et al. 2016. Modeling Concept Dependencies in a Scientific Corpus. *ACL* (2016), 866–875.
[8] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks. *KDD* (2016), 855–864.
[9] Marti A Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. *COLING* (1992), 539–545.
[10] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (2002), 422–446.
[11] Jens Lehmann et al. 2015. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.
[12] Yunyao Li, Frederick Reiss, and Laura Chiticariu. 2011. SystemT - A Declarative Information Extraction System. *ACL* (2011), 109–114.
[13] Chen Liang et al. 2015. Measuring prerequisite relations among concepts. *EMNLP* (2015), 1668–1674.
[14] Mausam et al. 2012. Open Language Learning for Information Extraction. *EMNLP-CoNLL* (2012), 523–534.
[15] Tomas Mikolov et al. 2013. Efficient Estimation of Word Representations in Vector Space. arxiv:1301.3781. Technical report available from: http://arxiv.org/abs/1301.3781. (2013).
[16] Mike Mintz et al. 2009. Distant supervision for relation extraction without labeled data. *ACL/IJCNLP* (2009), 1003–1011.
[17] Dat Quoc Nguyen. 2017. An overview of embedding models of entities and relationships for knowledge base completion. Technical report available from: http://arxiv.org/abs/1703.08098. (2017).
[18] Juan Ramos. 1999. Using TF-IDF to Determine Word Relevance in Document Queries. Technical report available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424&rep=rep1&type=pdf. (1999).
[19] Xiang Ren et al. 2016. Automatic Entity Recognition and Typing in Massive Text Corpora. *WWW* (2016), 1025–1028.
[20] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (2009), 333–389.
[21] Richard Socher et al. 2013. Reasoning with Neural Tensor Networks for Knowledge Base Completion. *NIPS* (2013), 926–934.
[22] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago - a core of semantic knowledge. *WWW* (2007), 697–706.
[23] Fabian M Suchanek and Gerhard Weikum. 2014. Knowledge Bases in the Age of Big Data Analytics. *PVLDB* 7, 13 (2014), 1713–1714.
[24] Partha Pratim Talukdar and William W. Cohen. 2012. Crowdsourced Comprehension: Predicting Prerequisite Structure in Wikipedia. *BEA@NAACL-HLT* (2012), 307–315.
[25] Shuting Wang et al. 2015. Concept Hierarchy Extraction from Textbooks. *DocEng* (2015), 147–156.
[26] Chenyan Xiong, Jamie Callan, and Tie-Yan Liu. 2016. Bag-of-Entities Representation for Ranking. *ICTIR* (2016), 181–184.
[27] Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit Semantic Ranking for Academic Search via Knowledge Graph Embedding. *WWW* (2017), 1271–1279.