

CONSTRUCTION AND APPLICATIONS OF A TECHNICAL KNOWLEDGE BASE

Prajna Devi Upadhyay



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI

August 2021

©Indian Institute of Technology Delhi - 2021
All rights reserved.

CONSTRUCTION AND APPLICATIONS OF A TECHNICAL KNOWLEDGE BASE

by

Prajna Devi Upadhyay

Department of Computer Science and Engineering

Submitted

in fulfillment of the requirements of the degree of
Doctor of Philosophy

to the



Indian Institute of Technology Delhi

August 2021

Certificate

This is to certify that the thesis titled **Construction and Applications of a Technical Knowledge Base** being submitted by **Ms. Prajna Devi Upadhyay** for the award of **Doctor of Philosophy in Computer Science and Engineering** is a record of bona fide work carried out by her under my guidance and supervision at the Department of Computer Science and Engineering, Indian Institute of Technology Delhi. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma.

Maya Ramanath
Associate Professor
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi- 110016

Acknowledgements

I would like to thank my supervisor, Prof. Maya Ramanath, for her invaluable guidance and feedback that has led to the work in this thesis. She has always maintained very high standards for research and has been extremely critical and objective during our discussions. This has given me the opportunity to improve my research methods. I also thank her for the endless comments she provided on my writing and presentations which have helped me refine them. Her words of encouragement during trying times have boosted me to carry on. Working with her has also helped me being an independent researcher since she gave me the freedom to try new ideas, write and submit them on my own.

I would also like to thank my collaborators, Prof. Srikanta Bedathur and Prof. Tanmoy Chakraborty, for the enriching discussions that led to one of the major components of my Ph.D. I also thank my co-authors—Ashwini Purkar, Tanuma Patra, Ashutosh Bindal, and Manjeet Kumar for their hard work.

I thank my SRC chairpersons, Prof. Saroj Kaushik and Prof. Mausam, and committee members Prof. Parag Singla and Dr. L Venkata Subramaniam for their insightful queries and comments that helped me refine my ideas.

I would like to extend my thanks to the examiners of my thesis for providing me with valuable feedback on my manuscript.

I thank IBM AI Horizons Network and IITD Alumni Association for supporting my travel

to conferences.

I want to thank my friends in the lab—Madhulika, Neha, Chandrika, and Anirban, who were always available for discussions whenever I needed clarity. My friends in the hostel—Swati, Shubhrama, Akanksha, Poonam, and Lopa made this long and difficult journey easier to deal with.

I will forever remain indebted to Dipanjan, my husband, who has been extremely supportive in various stages of my Ph.D. I thank him for putting up with my mood swings, for pulling me out of them, for motivating me, and for celebrating my accomplishments. This thesis would not have been possible without him being by my side.

I have been very fortunate to have extremely loving and caring in-laws, and I thank them wholeheartedly.

I cannot thank my parents enough—they have been supportive throughout my long journey as a Ph.D. student, and have always celebrated my achievements. I thank my sisters—Ambika and Pratima for taking up responsibilities at home when I could not be around because of work commitments. I thank Pushkar—my little brother, who never fails to amaze me with his energy.

All of you inspire me to strive for betterment and aim for higher goals in research.

Prajna Devi Upadhyay

Abstract

Recent years have witnessed an exponential increase in the availability of technical material on the web, possibly due to the efforts to digitize education and open access to scientific publications. Anybody with an internet connection can pull up relevant technical material to study or research. However, it is not necessary that all of this readily available material is comprehensible to most users. A user who is interested to work on a research problem or a student willing to take up a new course may come across multiple topics about which she has little/no idea. She may find a number of learning materials on that topic on the web, however, due to the lack of prerequisite knowledge, she will have to perform multiple searches before obtaining a basic understanding of that topic. To keep up with the latest research, she has to identify and understand different aspects of the topic to conduct a survey. This can be overwhelming for her due to a large amount of technical material available on the web. It would be helpful if there was a system to recommend to her prerequisite concepts for basic understanding and research papers for advanced understanding of the topic.

To build such a system, we first have to store the knowledge from the technical do-

main in the form of entities and relations as a knowledge graph. This will help design applications to consume this information in a systematic way. Knowledge graphs have been crucial for a number of semantic-aware applications. A number of such knowledge graphs, such as Yago, DBPedia, NELL, Wikidata, or Freebase have been constructed in the open domain and have supported tasks such as entity retrieval, question answering, or automatic organization of topics. Although there are multiple open-domain knowledge graphs, there are no known large-scale technical knowledge graphs, and especially in the domain of Computer Science. So, the first contribution of this thesis is TeKnowbase, which is a knowledge base in the domain of Computer Science. We use a combination of information extraction techniques to extract entities and relations from both structured and unstructured sources. TeKnowbase has been evaluated for its quality and is freely available.

The second contribution of this thesis is PreFace, which assists a beginner in the study of a topic in Computer Science by identifying its prerequisites using TeKnowbase. PreFace takes a topic in Computer Science as the query and returns a prerequisite graph, where the nodes represent the prerequisite concepts for the topic and the edges represent the prerequisite relationship. Additionally, it also identifies interesting aspects for the query and returns prerequisites grouped together as facets for each aspect. It achieves this by estimating a language model for the facet as well as the query using TeKnowbase. The facets are then ranked based on their relevance to the query and their coverage. The prerequisite graph, as well as the facets generated by PreFace, have been evaluated to be better than those generated using state-of-the-art prerequisite and facet retrieval

techniques.

Our final contribution is ASK (Aspect-based academic Search using domain-specific KBs), which recommends research papers for the different aspects identified by PreFace for advanced understanding of the queried topic. ASK first assists the user by providing relevant query suggestions for query and the aspect and then returns a ranked list of relevant research papers. The ranking of query suggestions, as well as research papers, is achieved using language models estimated for the query and aspect using TeKnowbase. The evaluation of papers as well as suggestions retrieved by ASK showed that they were superior to those returned by various state-of-the-art pseudo-relevant feedback or diversification techniques.

Overall, this thesis proposes techniques to make technical information more comprehensible for a user.

सारांश

हाल के वर्षों में वेब पर तकनीकी जानकारी की उपलब्धता में घातीय वृद्धि देखी गई है, संभवतः शिक्षा को डिजिटाइज़ करने और वैज्ञानिक प्रकाशनों की खुली पहुंच के प्रयासों के कारण। इंटरनेट कनेक्शन वाला कोई भी व्यक्ति अध्ययन या शोध के लिए प्रासंगिक तकनीकी सामग्री प्राप्त कर सकता है। हालांकि, यह आवश्यक नहीं है कि यह सभी आसानी से उपलब्ध सामग्री अधिकांश उपयोगकर्ताओं के लिए समझ में आ जाए। एक उपयोगकर्ता जो एक शोध समस्या पर काम करने के लिए इच्छुक है या एक नया पाठ्यक्रम लेने के इच्छुक छात्र को ऐसे कई विषयों का सामना करना पड़ सकता है जिनके बारे में उसे बहुत कम/कोई जानकारी नहीं है। उसे वेब पर उस विषय पर कई शिक्षण सामग्री मिल सकती है, हालांकि, पूर्वापेक्षित ज्ञान की कमी के कारण, उस विषय की बुनियादी समझ प्राप्त करने से पहले उसे कई खोज करनी होगी। नवीनतम शोध के साथ बने रहने के लिए, उसे सर्वेक्षण करने के लिए विषय के विभिन्न पहलुओं को पहचानना और समझना होगा। वेब पर बड़ी मात्रा में तकनीकी सामग्री उपलब्ध होने के कारण यह उसके लिए भारी पड़ सकता है। यह सहायक होगा यदि विषय की बुनियादी समझ के लिए पूर्वापेक्षा अवधारणाओं और उन्नत समझ के लिए शोध पत्रों की सिफारिश करने के लिए एक प्रणाली है।

ऐसी प्रणाली बनाने के लिए, हमें पहले तकनीकी डोमेन से ज्ञान को एन्टीटीएस और रिलेशनस के रूप में ज्ञान ग्राफ के रूप में संग्रहीत करना होगा। यह इस जानकारी को व्यवस्थित तरीके से उपभोग करने के लिए अनुप्रयोगों को डिजाइन करने में मदद करेगा। कई अर्थ-जागरूक अनुप्रयोगों के लिए ज्ञान ग्राफ महत्वपूर्ण रहे हैं। ऐसे कई नॉलेज ग्राफ , जैसे यागो, डीबीपीडिया, एनईएल, विकीडाटा, या फ्रीबेस का निर्माण खुले डोमेन में किया गया है और इसमें एंटीटी रिट्रीवल, प्रश्न उत्तर, या विषयों के स्वचालित संगठन जैसे कार्यों का समर्थन किया गया है। हालांकि कई खुले डोमेन नॉलेज ग्राफ हैं, लेकिन बड़े पैमाने पर कोई टेक्निकल नॉलेज ग्राफ नहीं

एन्टीटीएस और रिलेशन्स को निकालने के लिए सूचना निष्कर्षण तकनीकों के संयोजन का उपयोग करते हैं। टेक्नोबैस का मूल्यांकन इसकी गुणवत्ता के लिए किया गया है और यह मुफ्त में उपलब्ध है।

इस थीसिस का दूसरा योगदान प्रीफेस है, जो कंप्यूटर विज्ञान में किसी विषय के अध्ययन में शुरुआती को टेक्नोबैस का उपयोग करके इसकी पूर्वापेक्षाओं की पहचान करने में सहायता करता है। प्रीफेस कंप्यूटर साइंस में एक विषय को क्वेरी के रूप में लेता है और एक पूर्वापेक्षा ग्राफ देता है, जहां नोड्स विषय के लिए पूर्वापेक्षा अवधारणाओं का प्रतिनिधित्व करते हैं और किनारे पूर्वापेक्षा संबंध का प्रतिनिधित्व करते हैं। इसके अतिरिक्त, यह क्वेरी के लिए दिलचस्प पहलुओं की भी पहचान करता है और प्रत्येक पहलू के लिए पहलुओं के रूप में समूहीकृत पूर्वापेक्षाएँ लौटाता है। यह पहलू के लिए लैंग्वेज मॉडल के साथ-साथ टेक्नोबैस का उपयोग करके क्वेरी का अनुमान लगाकर इसे प्राप्त करता है। फिर पहलुओं को उनकी प्रासंगिकता और उनके कवरेज के आधार पर रैंक किया जाता है। पूर्वापेक्षा ग्राफ, साथ ही प्रीफेस द्वारा उत्पन्न पहलुओं का मूल्यांकन अत्याधुनिक पूर्वापेक्षा और पहलू पुनर्प्राप्ति तकनीकों का उपयोग करके उत्पन्न लोगों की तुलना में बेहतर होने के लिए किया गया है।

हमारा अंतिम योगदान आस्क (डोमेन-विशिष्ट केबी का उपयोग कर पहलू-आधारित अकादमिक खोज) है, जो पूछे गए विषय की उन्नत समझ के लिए प्रीफेस द्वारा पहचाने गए विभिन्न पहलुओं के लिए शोध पत्रों की सिफारिश करता है। आस्क पहले क्वेरी और पहलू के लिए प्रासंगिक क्वेरी सुझाव प्रदान करके उपयोगकर्ता की सहायता करता है और फिर प्रासंगिक शोध पत्रों की एक रैंक की गई सूची देता है। क्वेरी सुझावों के साथ-साथ शोध पत्रों की रैंकिंग, टेक्नोबैस का उपयोग करके क्वेरी और पहलू के लिए अनुमानित लैंग्वेज मॉडल का उपयोग करके हासिल की जाती है। शोध पत्रों के मूल्यांकन के साथ-साथ आस्क द्वारा प्राप्त सुझावों से पता चला कि वे विभिन्न अत्याधुनिक सूडो रिलेवेंट फीडबैक या विविधीकरण तकनीकों द्वारा लौटाए गए सुझावों से बेहतर थे।

Contents

Certificate	i
Acknowledgements	iii
Abstract	v
List of Figures	xxiii
List of Tables	xxvii
1 Introduction	1
1.1 Knowledge Graphs	8
1.2 Challenges	11
1.2.1 Construction of a Technical Knowledge Base	11

1.2.2	Recommendation of Prerequisites	12
1.2.3	Aspect-based Academic Search	13
1.3	Contributions	16
1.3.1	TeKnowbase: A Knowledge Base of Computer Science Concepts . .	17
1.3.2	PreFace: Faceted Retrieval of Prerequisites	18
1.3.3	ASK: Aspect based Academic Search using Domain-Specific KBs . .	18
1.4	Organization	19
2	Construction of TeKnowbase	21
2.1	Motivation and Problem	22
2.2	Approach and Contributions	24
2.3	Related Work	24
2.3.1	Manual Creation of Knowledge Graphs	24
2.3.1.1	Manual Curation of Knowledge Graphs by Experts	25
2.3.1.2	Manual Curation of Knowledge Graph by Collaborative Efforts	25
2.3.2	Automatic Creation of Knowledge Graphs	26

2.3.2.1	KG Creation from Semi-Structured Sources	26
2.3.2.2	KG Creation from Unstructured Sources	27
2.4	Construction of TeKnowbase	32
2.4.1	Input and Output of Extraction Process	32
2.4.1.1	Output	32
2.4.1.2	Input	33
2.4.2	Construction Pipeline	34
2.4.2.1	Entity Extraction	35
2.4.2.2	Relation Extraction	37
2.4.2.3	TeKnowbase Completion	46
2.4.2.4	Availability/Statistics of TeKnowbase	47
2.5	Evaluation of Quality of TeKnowbase	50
2.5.1	Setup	50
2.5.2	Results and Analysis	51
2.6	Evaluation of Usability of TeKnowbase	54
2.6.1	Classification Experiment	54

2.6.1.1	Setup	55
2.6.1.2	Features Generation	55
2.6.1.3	Classification Algorithms	55
2.6.1.4	Results	56
2.6.2	Ranking Experiment	56
2.6.2.1	Setup	57
2.6.2.2	Techniques	57
2.6.2.3	Ranking Models	57
2.6.2.4	Results	58
2.7	Conclusions	58
3	ASK: Aspect-based Academic Search	59
3.1	Motivation and Problem	60
3.1.1	Problem Definition	65
3.2	Approach and Contributions	66
3.3	Related Work	68
3.3.1	Aspect-based Document Retrieval	68

3.3.1.1	Regular Search	68
3.3.1.2	Academic Search	69
3.3.2	Query Suggestion	70
3.3.2.1	Query Suggestion using Query Logs	70
3.3.2.2	Query Suggestion without Query Logs	71
3.3.2.3	Query Suggestion in Academic Search	72
3.4	Aspect based Retrieval	72
3.4.1	Language Models	74
3.4.2	Aspect-based Retrieval Model	76
3.4.3	Estimation of Probabilities	78
3.4.3.1	Estimation of Query-Independent Component	78
3.4.3.2	Estimation of Query-Dependent Component	84
3.4.4	Generating Query Suggestions	92
3.4.5	Ranking of Documents	93
3.5	Experiments	94
3.5.1	Experiments for Document Retrieval	94

3.5.1.1	Setup	94
3.5.1.2	Evaluation Methodology	101
3.5.1.3	Results and Discussions	103
3.5.2	Experiments for Query Suggestion	106
3.5.2.1	Setup	107
3.5.2.2	Evaluation Methodology	107
3.5.2.3	Results and Discussion	108
3.6	Conclusion	110
4	PreFace: Faceted Retrieval of Prerequisites	115
4.1	Motivation and Problem	115
4.1.1	Problem Definition	119
4.2	Approach and Contributions	122
4.3	Related Work	125
4.3.1	Facet Extraction	125
4.3.1.1	Facet Extraction for Regular Search	125
4.3.1.2	Faceted Academic Search	126

4.3.2	Prerequisite Determination	127
4.4	PreFace	128
4.4.1	Prerequisite Graph Generation	128
4.4.1.1	Frame Semantics	129
4.4.1.2	Our Approach	131
4.4.2	Generating and Ranking Facets for Prerequisites	138
4.4.2.1	Components of Facet Extraction and Ranking System	138
4.4.2.2	Facet Extraction	139
4.4.2.3	Retrieval Model	141
4.4.2.4	Ranking of Facets	149
4.4.2.5	Item Ranking	149
4.5	Experiments	150
4.5.1	Experiments for Necessary Prerequisites	150
4.5.1.1	Setup	150
4.5.1.2	Baselines	150
4.5.1.3	Benchmarks	151

4.5.1.4	Gold Standard	153
4.5.1.5	Evaluation Methodology and Metrics	153
4.5.1.6	Results	155
4.5.2	Experiments for Faceted Prerequisites	160
4.5.2.1	Setup	160
4.5.2.2	Benchmark Queries	160
4.5.2.3	Baselines	160
4.5.2.4	Evaluation Scheme	162
4.5.2.5	Results and Discussion	165
4.6	Conclusion	167
5	PreFace++: Faceted Retrieval of Prerequisites and Technical Data	169
5.1	System Architecture	169
5.1.1	TeKnowbase.	170
5.1.2	Prerequisite Graph Generation	170
5.1.3	Facet Generation and Ranking	171
5.1.3.1	Candidate Facet Generation	172

5.1.3.2	Retrieval	172
5.1.3.3	Ranking of Facets.	173
5.1.4	Retrieval of Research Papers and Technical Posts	173
5.2	System Implementation	173
5.2.1	Front End	173
5.2.2	Back End	175
5.3	Conclusion	175
6	Conclusion	177
6.1	Future Work Directions	178
6.1.1	Question Answering in the Technical Domain	178
6.1.2	Automatic Generation of Lecture Notes	180
	Appendices	181
A	Agglomerative and Star Clustering	183
A.1	Agglomerative Clustering	183
A.1.1	Calinski-Harabasz Index(CH-Index)	184

A.2	Star Clustering	186
B	QDMiner and QDMKB Approach Towards Facet Extraction	189
B.1	QDMiner	189
B.1.1	List and Context Extraction	191
B.1.1.1	Free Text Patterns	191
B.1.1.2	HTML Tag Patterns	192
B.1.1.3	Repeat Region Patterns	192
B.1.1.4	Post Processing	192
B.1.2	List Weighting	193
B.1.3	List Clustering	195
B.1.4	Facet Ranking	196
B.1.5	Item Ranking	197
B.2	QDMKB	198
B.2.1	Facet Generation	198
B.2.2	Facet Expansion	201
B.2.2.1	Property-based Facet Expansion	201

CONTENTS

xxi

B.2.2.2	Facet Expansion Based on Types	202
B.2.2.3	Facet Grouping	203
B.2.2.4	Facet Weighting	204
	Bibliography	205
	List of Publications	231
	Biography	233

List of Figures

1.1	Examples of some triples in YAGO	9
1.2	Top-2 results retrieved for the query <code>computer vision application</code> by google scholar search engine as of 20 July 2020	14
1.3	Suggestion provided by google scholar for the query <code>computer vision application</code> as on 20 July 2020	15
1.4	Architecture of the system that was built as a part of this thesis	17
2.1	Construction methodology of TeKnowbase	35
2.2	Snippet of the HTML page describing the terms and the corresponding code from <code>Whatis.Techtarget</code>	37
2.3	Snapshot of two overview pages used to extract <code>subtopic</code> and <code>type</code> relation	41

2.4	Snippet of pieces of structured elements found in Wikipedia articles that can be used to extract known relations	42
2.5	Snapshot of template for the topic <code>Database Management System</code>	43
2.6	Snapshot of indexes present in the “Computer Networks” online textbook	44
2.7	Pipeline of post-processing steps on the triples extracted from OpenIE	44
3.1	Top-2 results retrieved for the query <code>computer vision application</code> by google scholar search engine as of 20 July 2020	61
3.2	Suggestions generated by Google Scholar for the query <code>computer vision application</code> as on 20 July 2020	62
3.3	Top-2 results retrieved for query <code>computer vision road vehicle detection method</code> by Google Scholar search engine as of 20 July 2020	63
3.4	Top-2 results retrieved for query <code>use computer vision based methods</code> by Google Scholar search engine as of 20 July 2020	64
3.5	Architecture of ASK	67
3.6	Examples of meta-paths for <code>application relation</code>	86
4.1	Graph of prerequisites for <code>convolutional_neural_network</code>	116

4.2	Snapshot of the first paragraph of Wikipedia page for <code>convolutional_neural_network</code> as on 23 July 2020	117
4.3	Snapshot of the libraries mentioned in the Wikipedia page for <code>convolutional_neural_network</code> as on 23 July 2020	118
4.4	Prerequisites returned for <code>convolutional_neural_network</code> by <i>RefD</i>	119
4.5	Path connecting <code>convolutional_neural_network</code> to <code>medical_imaging</code> and <code>fast_fourier_transform</code>	123
4.6	Architecture of PreFace	129
4.7	A frame for <code>convolutional_neural_network</code>	130
4.8	Prerequisite graph obtained for <code>convolutional_neural_network</code> using different frame representations	136
4.9	Components of the Facet Extraction and Ranking System	138
4.10	Prerequisite graph obtained for <code>OWN_TEXT</code> and <code>OWN_KB</code>	158
5.1	Architecture of PreFace++.	170
5.2	(a) Auto-completion options for the partially completed string <code>artificial_neur</code> , (b) Prerequisite graph returned for <code>artificial_neural_network</code>	174
A.1	Pairwise distances between item in S	184

A.2	Dendrogram for complete-linkage clustering algorithm applied on S	185
B.1	Architecture of QDMiner	190
B.2	Examples of HTML tags and the items they contain	193
B.3	Examples of repeat regions and the information contained in them	194
B.4	QDMKB architecture	199

List of Tables

2.1	Survey of relation extraction techniques	29
2.2	List of known relations identified by domain experts.	39
2.3	4 kinds of IE tasks based on source and types of relations to be extracted .	39
2.4	Some new triples that were added to TeKnowbase using the Neural Tensor Network inferencing model.	47
2.5	Statistics from TeKnowbase	48
2.6	Statistics of TeKnowbase	49
2.7	Evaluation of a subset of triples in the TKB.	52
2.8	Evaluation of triples participating in rarer relations in TeKnowbase.	52
2.9	Accuracy obtained using different models used with NTN	54
2.10	Average classification accuracies.	56

2.11	NDCG@20 values obtained with tf-idf and BM-25 ranking models.	56
3.1	Titles of relevant papers for queries along <i>Application</i> and <i>Algorithm</i> aspect	60
3.2	Example of Language Models for two documents about “Movies” and “Fruits”	74
3.3	Set of rules to determine the scores for papers relevant for the query autoencoder and application aspect	80
3.4	Set of rules to determine the scores for papers relevant for the query <i>minimum</i> spanning tree and algorithm aspect	81
3.5	Set of rules to determine the scores for papers relevant for the query <i>genetic</i> algorithm and implementation aspect	83
3.6	Benchmark queries	97
3.7	Example of queries for various baselines	102
3.8	Results for algorithm, application and implementation aspect	105
3.9	Top-2 papers retrieved for 3 queries along 3 aspects for our method and a competing baseline	106
3.10	Quality of query suggestions	109
3.11	Comparison of the suggestions retrieved by our model and the baseline for query <i>computer vision</i> and <i>algorithm</i> aspect	111

3.12	Comparison of the suggestions retrieved by our model and the baseline for query <code>genetic algorithm</code> and <i>application</i> aspect	111
3.13	Comparison of the suggestions retrieved by our model and the baseline for query <code>autoencoder</code> and <i>implementation</i> aspect	112
3.14	Average time taken for document retrieval and query suggestion for the 3 aspects in seconds	112
4.1	Example of 4 faceted prerequisites retrieved by <i>PreFace</i> for <code>convolutional_neural_network</code>	120
4.2	Similar concepts	137
4.3	Benchmark queries	152
4.4	Precision, Recall and F1 scores after changing frames	155
4.5	Precision, Recall and F1 scores after augmenting prerequisites	155
4.6	Results from user evaluation	157
4.7	The frames generated for <code>convolutional_neural_networks</code> by our technique OWN_KB and the baseline OWN_TEXT	159
4.8	Benchmark queries	161
4.9	Criteria for different relevance judgements for ranking quality	164

4.10 Comparison of PreFace and baselines	165
4.11 Facets containing prerequisites retrieved for <code>convolutional_neural_network</code> by PreFace, QDMKB + RefD, and RefD+TKB for top 3 facets	167

Chapter 1

Introduction

With the growth of Massive Open Online Courses (MOOCS) ¹ and efforts towards digitization of educational content [6], there has been an exponential increase in the availability of study material to anybody with access to the internet. MOOCs are platforms for providing online courses which can be taken by anybody, free of cost. Not only online courses, but a number of technical reading material curated by experts or collaboratively maintained is also available on the web [57, 196]. The availability of free reading material has changed the way education is perceived. It is not necessary for the student to be physically present in the same room as that of the teacher in order to learn a course, and instead, is mostly self-driven [5]. Apart from online courses, the number of research papers in digital form has also exponentially increased over the last decade due to the open access movement [152, 7]. Open access research papers can be publicly accessed without the need for expensive subscription or copyright restrictions which has led to an increase in the popularity of journals that provide open access. Anybody who wishes to study or research on a particular topic can access the required materials from the internet with ease.

However, it is not necessary that all of this readily available material is comprehensible

¹<https://www.mooc.org/>

to most users. Given the choice of studying any topic from the web, a student may take up a course about which she has little or no idea. She may struggle to understand the key concepts because she has limited prerequisite knowledge about those topics [150]. For instance, to understand `artificial_neural_networks`, she has to have prerequisite knowledge of `machine_learning` and `neuron`. The prerequisites of a topic are provided by a teacher in a one-to-one interaction in a classroom but are generally absent from the reading material found on the internet. Even in the case of online courses, it is infeasible for the instructors to provide a tailor-made list of prerequisites of concepts for each student because of varying backgrounds and a large number of the students who take up such courses [149]. So, she has to identify the prerequisites on her own, which is challenging because retrieval systems only return relevant documents that may or may not contain prerequisites in them. Even if they do, she may need to further refer to the prerequisite's prerequisite. This results in more queries, and essentially "knocking around" [189] trying to find appropriate reading material to understand the new concept. Apart from prerequisites, the user may still need to refer to other concepts that help her with an overall understanding of the topic. To understand `artificial_neural_network`, knowledge of `python` or `matlab` is recommended to help the user *implement* `artificial_neural_network`. A suggestion such as `phoneme_classification` would also help the user understand *applications* of `artificial_neural_network`, since `phoneme_classification` is an application of `artificial_neural_network`. So, there exist different *aspects* of understanding a topic which have to be considered to provide an overall understanding.

Apart from students, researchers or academicians also need to have an overall understanding of new topics to stay on top of the developments in their field. A student/researcher who has acquired some basic understanding of a topic would be interested in exploring the latest research in that field. However, the increasing number of technical publications [152] has led to scientific information overload, where researchers find it difficult to keep up with the latest inventions in the field [110]. The number of published journal articles is increasing at the rate of 8–9% over the past few decades, with the period from 2010–2012 alone reporting an increase of over 25% in the number of open access journals in the field of medicine and biology [152]. More than 1 million research articles are pub-

lished each year in the PubMed database in the field of biomedical, which amounts to around 2 papers per minute [110]. In an attempt to systematically consume this flood of information, researchers often spend around 6–8 hours per week on activities such as group discussions or networking. Some researchers use their own curation systems where they manually organize the latest research suggested by Google Scholar or PubMed [110]. With the increase in the number of publications, curation demands more time and effort.

It would be useful if there was a system that would take a topic as input and automatically recommend prerequisites for basic understanding and research papers for advanced understanding of the topic. Given a topic `artificial_neural_network`, the system would first recommend prerequisites such as `machine_learning` or `neuron` in the form of a prerequisite graph. The system would also identify key *aspects* related to the topic and recommend prerequisites for them. One of the aspects for `artificial_neural_network` is `software`, and concepts such as `matlab` or `python` can be recommended as prerequisites. Similarly, concept such as `phoneme_classification` can be recommended as a prerequisite for `application` aspect for `artificial_neural_network`. After the user has acquired some basic understanding, the system would also recommend research papers relevant to different aspects of the queried topic. To illustrate, a paper titled “An Artificial Neural Network for Spatio-Temporal Bipolar Patterns: Application to Phoneme Classification.” can be recommended as being relevant for `application` aspect of `artificial_neural_network` because it describes an application of `artificial_neural_network`, while another paper titled “Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks” can be recommended for `algorithm` aspect of `artificial_neural_network`, since it proposes a new algorithm for training artificial neural networks. Designing such a system requires a systematic organization of the entities and relationships in that domain and storing the information in a structured form. This can be achieved using knowledge graphs [88].

Knowledge Graphs. Knowledge graphs [88] represent real-world information in a structured form. They are usually stored as Resource Description Framework [1] (RDF) triples

represented by $\langle s, p, o \rangle$, where s stands for a subject entity, o stands for an object entity, and p stands for the predicate or the relationship describing between the two. Given the fact that “Albert Einstein was born in Ulm”, it can be stored as a triple $\langle \text{albert_einstein}, \text{birthPlace}, \text{ulm} \rangle$, where `albert_einstein` and `Ulm` are the entities, and `birthPlace` describes the relationship between the two. Knowledge graphs can capture the semantics of the domain using entities and relationships due to which they have been used to design semantic-aware systems such as question answering [127, 9, 85], word-sense disambiguation [30], entity summarization [81, 39, 186] or entity retrieval [154, 143, 146, 225]. Over the years, a number of knowledge bases such as Yago [179], DBPedia [114], NELL [32], OpenIE [58], Wikidata [57, 196] and Freebase² have been created. They are also crucial for the functioning of many commercial search engines like Google’s Knowledge Vault [54], Bing’s Satori³, Facebook⁴ and LinkedIn⁵.

There have been attempts at building knowledge bases in the technical domains, such as biomedical, using manual [16, 26] or automatic [43] techniques. However, in the domain of Computer Science, there is a lack of large-scale knowledge graphs. A knowledge base in the domain of Computer Science can help organize the entities and relationships and support applications for basic and advanced understanding of a topic. It can be used to recommend topics to study and scholarly documents in the relevant area for research.

Recommendation of Topics for Basic and Advanced Understanding. In this thesis, we are primarily concerned with developing a system to assist a user with both basic and advanced understanding of a topic. The basic understanding of a topic is provided by recommending its *prerequisites* and identifying different aspects of the queried topic. The advanced understanding is provided by recommending research papers for different aspects of the queried topic. We now define these terminologies as follows.

²Now known as the Google Knowledge Graph

³<https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/>

⁴<https://www.adweek.com/digital/facebook-builds-knowledge-graph-with-info-modules-on-community-pages/>

⁵<https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph>

Definition 1. Prerequisite. A prerequisite of a concept q is another concept p that can be recommended to be studied before q . In other words, having a knowledge of p improves the understanding of q . This is denoted by $q \rightarrow p$. A concept q cannot be a prerequisite of itself, so $q \not\rightarrow q$. Given the concept `artificial_neural_network`, some prerequisites that can be suggested are `machine_learning`, `neuron` or `matlab`.

Definition 2. Necessary Prerequisites. A necessary prerequisite of a concept q is a prerequisite b which **has to** be studied before q . This is denoted by $q \Longrightarrow b$. An absence of such a relationship is denoted by $q \not\Longrightarrow b$. For instance, `machine_learning` or `neuron` are necessary prerequisites for `artificial_neural_network`.

Some properties of a necessary prerequisite are:

1. **Irreflexive:** A concept q cannot be a necessary prerequisite of itself, i.e. $q \not\Longrightarrow q$.
2. **Asymmetric:** If $q \Longrightarrow b$, then $b \not\Longrightarrow q$.
3. $\forall b$ s.t. $q \Longrightarrow b$, $q \rightarrow b$ holds.

To acquire an overall understanding, one has to understand different *aspects* of the queried topic. For instance, a knowledge of a software such as `matlab` can help the user implement `artificial_neural_network`. So, `matlab` can be suggested as a prerequisite for the software aspect of `artificial_neural_network`. If the aspect is application, then a concept such as `phoneme_classification` can be recommended since it is known to be an application of `artificial_neural_network`. A research paper titled “An Artificial Neural Network for Spatio-Temporal Bipolar Patterns: Application to Phoneme Classification.” is relevant for the application aspect of `artificial_neural_network`, since the paper describes an application of `artificial_neural_network`. If the aspect is algorithm, then prerequisites such as `optimization` or `gradient_descent` can be recommended and a paper titled “Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks” can be recommended for further understanding. So, we formally describe an aspect as follows:

Definition 3. Aspect. *An aspect is a keyword that describes some subtopic of q . $q \rightarrow (p, a)$ denotes that the prerequisite p of q can be recommended to understand aspect a of query q .*

Aspects can be used to define *query facets*. According to [56], “a query facet is a set of items which describe and summarize one important aspect of a query. Here a facet item is typically a word or a phrase. A query may have multiple facets that summarize the information about the query from different perspectives.”. We extend this idea of query facets to prerequisites of a query and define a **facet** as follows.

Definition 4. Facet. *A facet refers to a set F of prerequisites of q that can be recommended for an aspect a of a query. This means for all $p \in F$, $q \rightarrow (p, a)$. For example, the set of concepts $\{\text{matlab}, \text{python}\}$ is a facet which can be suggested for software aspect of `artificial_neural_network`. The set of concepts $\{\text{gradient_descent}, \text{optimization}\}$ can be suggested for algorithm aspect of `artificial_neural_network`.*

A knowledge base that stores triples of the form $\langle \text{artificial_neural_network}, \text{subtopic}, \text{machine_learning} \rangle$ can be used to recommend `machine_learning` as a prerequisite for `artificial_neural_network`, since it is a subtopic of machine learning. A triple $\langle \text{phoneme_classification}, \text{application}, \text{artificial_neural_network} \rangle$ can help us recommend `phoneme_classification` as a prerequisite for `application` aspect of `artificial_neural_network`. We can also use the knowledge base to improve the recall of the retrieved prerequisites by identifying similar concepts for a given topic of interest since similar concepts have similar prerequisites, as illustrated by the facet that the prerequisites for `artificial_neural_network` and `convolutional_neural_network` overlap to some extent because both of them are neural networks. A research paper that consists of `phoneme_classification` can be considered highly relevant and recommended for the `application` aspect.

Problems Addressed in this Thesis. In order to make the growing amount of technical content on the web more comprehensible to a user, it is important that we develop a

system that assists users in systematically understanding that content. The first step involves organizing this information from the domain of Computer Science in the form of entities and relationships present in a knowledge graph. While there exist knowledge bases in the other technical domains of biology and medical science, there is a lack of known knowledge bases in the domain of Computer Science. Due to the unavailability of knowledge graphs in the domain of Computer Science, our first problem deals with the construction of a knowledge base in this domain. Since manual curation of a knowledge base requires a great deal of effort, we resort to the automatic construction of knowledge graphs. Moreover, the availability of a large amount of technical information on the web in the form of semi-structured and structured resources allows us to explore different information extraction [166] techniques to extract structured information. It consists of two sub-problems—i) extracting named entities from the domain [109, 226], and ii) extracting relationships between these entities [61, 80, 32].

Once we have constructed a knowledge base in Computer Science, our next problem is to build a system to assist users to achieve a basic understanding of topics using the knowledge base. Such a system will help the user consume the overwhelming amount of technical information in an organized fashion. It should be able to automatically recommend prerequisites for a topic the user is interested in. While there exist techniques that automatically identify prerequisites [119, 184, 120] for a query, they do not consider the different aspects of interest while studying a new topic. So, there is a need to develop a new technique that can automatically determine prerequisites for different aspects of interest of a query.

After the user has gained some basic idea about the topic, she will be interested to explore the latest research in that area. Given the different aspects already identified for the query while determining prerequisites, it would be beneficial if research papers relevant for the query and these aspects are also retrieved for the user. So, our third problem deals with aspect-based academic retrieval of research papers for the query. This problem is different from the problem of generating aspects [94, 56] for a query where the aim is to automatically generate these aspects, which can be provided as suggestions. Our problem,

on the other hand, is to improve the quality of research papers retrieved for a given query and an aspect.

We address these three problems in this thesis by designing a system to recommend topics to study for basic understanding of a queried topic, and then recommending research papers for advanced understanding. The backbone of this system is a technical knowledge base in Computer Science, which organizes the domain knowledge in the form of entities and relations.

Organization. The rest of this chapter is organized as follows. Section 1.1 introduces knowledge graph and the RDF model that is used to store knowledge graphs. In Section 1.2 we describe the key challenges in designing such a system and provide an overview of the techniques to tackle them. We summarize the contributions of this thesis in Section 1.3 and end this chapter describing the organization of the remainder of the thesis.

1.1 Knowledge Graphs

Knowledge graphs are used to store real-world information in a semantic-rich and organized fashion. The information is stored in the form of facts, which are represented as triples, such as $\langle \text{albert_einstein}, \text{bornin}, \text{ulm} \rangle$. It states the fact that *Albert Einstein was born in Ulm*. A popular way of storing such information is in the form of graphs, where the nodes represent the entities, and the edges represent the relation between the two entities. For the same example, `albert_einstein` and `Ulm` are the entities and `bornin` is the relationship between them. A graph of entities and relations that is used to store a set of facts is called a Knowledge Graph. Figure 1.1 shows a set of triples related to `albert_einstein` present in the YAGO [179] knowledge graph.

A popular format for the storage of knowledge graphs is the RDF format (Resource Description Framework) [1]. The Resource Description Framework is a framework for

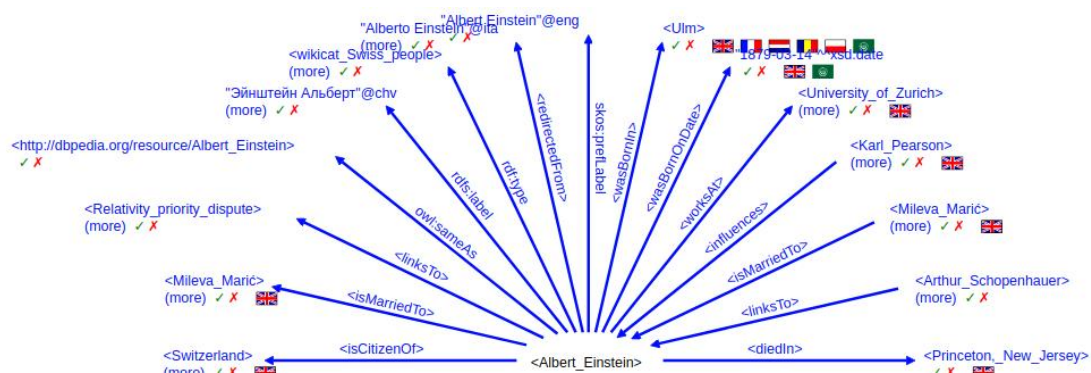


Figure 1.1: Examples of some triples in YAGO that `albert_einstein` participates in. The source of the figure can be found at [209]

representing information on the web recommended by the World Wide Web consortium. It provides a syntax for a graph-based model. RDF is meant to store triple as a graph, where the nodes in the graph represent the subject and the object and the edges represent the predicate or the relationship between the two. The components of the RDF data model are described in the following paragraphs.

Nodes. The nodes in RDF based graph model refer to one of the following:

1. **Entities:** These represent real-world or abstract entities, such as the scientist “Albert Einstein” will appear as `albert_einstein` in the nodes. In the domain of Computer Science, the entity `artificial_neural_network` represents the concept “Artificial neural network”. The set of entities is given by **E**.
2. **Literals:** Literals are constants that represent values such as numbers, descriptions, or dates. They can be used to store facts such as *Albert Einstein was born in 1879*. The birth year “1879” can be stored as a literal. A literal can be the object of an RDF statement, but it cannot be the subject or the predicate. The set of literals is represented by **L**.

Edges. The edges connecting the nodes in a knowledge graph are used to describe the relationship between the subject and the object. These edges are directed. They are called predicates or properties in the RDF model. For instance, `bornIn` is the predicate that connects the subject node `albert_einstein` and the object node `ulm`. The set **P** is used to denote the set of predicates.

Universal Resource Identifier. The entities and relationships of a knowledge graph constitute the resources in an RDF model. RDF uses dedicated web pages to describe these resources. These web pages are represented by a unique identifier, called the Universal Resource Identifier (URI). Each node in an RDF model can point to a URI or a literal. When a node does not consist of either of these, it is called a blank node. Each edge in an RDF model also points to a URI that describes the predicate that the edge represents. The popular knowledge graph YAGO [179] uses the URI https://yago-knowledge.org/resource/Albert_Einstein to describe the entity “Albert Einstein” and the URI <https://yago-knowledge.org/resource/Ulm> to describe the location “Ulm”. The URI <https://yago-knowledge.org/resource/schema:birthPlace> is used to describe the relationship “birthPlace” which connects two entities.

Triples and Knowledge Graphs. A triple refers to a tuple $\langle s \ p \ o \rangle$ consisting of the

subject s , object o and the predicate p . Each triple makes a statement about the subject and the object. The subject can be entities and the objects can be an entity, literal or a blank node. It is formally described as follows:

Definition 5. Triple. *Given a set of entities E , set of literals L and a set of predicates P , a triple t is denoted by a tuple $\langle s \ p \ o \rangle$, such that $t \in E \times P \times \{E \cup L\}$.*

A knowledge graph KG is formally described as follows:

Definition 6. Knowledge Graph. *A knowledge graph KG is a set of triples that can be represented as a graph. $KG \subseteq E \times P \times \{E \cup L\}$*

1.2 Challenges

It is crucial that we develop a system to assist computer science enthusiasts in learning as well as researching a topic. However, doing so comes with its own set of challenges. The first challenge that we need to tackle is the—*automatic construction of a technical knowledge base* to organize the space of entities and relationships in the technical domain. The next challenge deals with *automatically generating prerequisites for a topic of interest* to provide a basic understanding of the topic. The final challenge is the *aspect-based retrieval of research papers* to recommend research papers relevant for the query and an aspect and to provide an advanced understanding of the topic.

1.2.1 Construction of a Technical Knowledge Base

The first problem that we address is the construction of a knowledge base in the domain of Computer Science. This can be done manually [115, 136, 26] or using collaborative efforts [57, 196, 16]. However, manual creation of knowledge bases is expensive and slow, and collaborative techniques are not always reliable [41]. Moreover, domains such

as Computer Science will need annotations more frequently since the field is rapidly growing, which will make the process expensive. A number of techniques to automatically construct knowledge bases from both structured [179, 114] and unstructured sources [32] have been proposed. However, using techniques proposed in [179, 114] does not lead to fine-grained extraction of relationships in the domain of Computer Science, because most of these techniques are designed for the open domain. Learning extractors for different relationships [32] needs a large amount of training data, which is hard to curate. Open Information extraction [58] techniques can be used to extract large number of triples without any supervision, however, when it comes to long and complicated sentence structures which are common in technical documents, these techniques fail [197]. Systems such as DeepDive [204], which claim to automatically construct a knowledge base from structured and unstructured sources, fail to serve our purpose because it needs a curated set of triples as input so that more triples can be inferred. This makes the task of extracting a set of high quality triples in Computer Science challenging. In a nutshell, constructing a knowledge base in the domain of Computer Science involves identifying the right set of resources and then formulating simple, unsupervised techniques to extract the triples.

We use simple techniques to construct a technical knowledge base, called TeKnowbase, from various Computer Science specific structured and unstructured sources. The construction of TeKnowbase and its evaluation is described in Chapter 2.

1.2.2 Recommendation of Prerequisites

As already discussed, a computer science enthusiast who wishes to study a new topic will face difficulty understanding it if she does not have the required *prerequisite* knowledge [150]. In a traditional setting, such prerequisites are usually recommended by the teacher. However, in online learning, she has to figure out the prerequisites herself [149]. She can try querying for the topic on the web, but there is no guarantee that the documents returned will contain its prerequisites. Even if it does, she might not be aware of them and

hence, would need to recursively query for those concepts again [189]. To address these issues, there exist techniques that automatically determine prerequisites for a query [119, 184]. However, these techniques do not consider different aspects of interest towards understanding a topic. As a result, prerequisites for different aspects are returned together. Concepts such as `python`, `C`, `machine_learning` and `phoneme_classification` are returned together as prerequisites of `artificial_neural_network`. Out of these, `machine_learning` is a necessary prerequisite, while concepts such as `python` and `C` should be recommended in groups called facets relevant for the `software` aspect of `artificial_neural_network`. So, the first challenge is to identify necessary prerequisites for the query. The next challenge is to group all the prerequisites into multiple facets. We can use existing query-based facet extraction techniques [56, 94] to extract facets from our technical knowledge base. These techniques group concepts reachable via the same sequence of nodes and edges in a knowledge base into the same facet. This may not always lead to good quality facets since concepts reachable via the same sequence of nodes and edges may not be relevant for the same aspect of the query. We discuss these challenges in detail and propose *PreFace* to address them in Chapter 4.

1.2.3 Aspect-based Academic Search

A user who has acquired some basic understanding of a topic would be interested to explore the latest research in that area, covering different aspects of the queried topic. She might be interested in papers describing *applications* of `computer_vision` or *algorithms* of `computer_vision`. In the former case, the query is `computer_vision` and the aspect is *application* and a paper titled *Role of Computer Vision in Automatic Inspection System* should be relevant because this paper describes an application of `computer_vision`. For the latter, *Job-shop scheduling applied to computer vision* would be more relevant because it describes algorithms to perform job scheduling in computer vision. A user interested to retrieve such results will enter the query `computer vision application` on an academic search engine, such as Google Scholar. A document titled *OpenCV Computer Vision Application Programming Cookbook* is retrieved at the top position (shown in Figure 1.2).

The screenshot shows a Google Scholar search interface. At the top, there is a search bar with the text "computer vision application". Below the search bar, it says "Scholar About 36,60,000 results (0.08 sec)" and "YEAR" with a dropdown arrow. The first result is a book titled "[BOOK] OpenCV Computer Vision Application Programming Cookbook Second Edition" by R. Laganière, published in 2014. The second result is a paper titled "A human action recognition system for embedded computer vision application" by H. Meng, N. Pears, and C. Bailey, published in 2007 at the Conference on Computer Vision.

Figure 1.2: Top-2 results retrieved for the query `computer vision application` by google scholar search engine as of 20 July 2020

This document describes OpenCV, which is a library to implement computer vision applications and is less relevant than another paper titled *Role of Computer Vision in Automatic Inspection System*, which describes an application of `computer vision`. It was retrieved at the top position because of the presence of terms `application` and `computer_vision` in the title, without considering the domain-specific relationship between the two. This means that just the presence of the query and the aspect term in the document does not make it the best candidate to be suggested for the query and the aspect.

Consider the document *Role of Computer Vision in Automatic Inspection System*, which is a relevant document, which does not mention the term `application` anywhere in the title. The reason for the relevance of this document is because it mentions the term

`automatic_inspection_system` in the title, which is known to be an application of computer vision. Techniques such as pseudo relevance feedback [112] can be used to identify terms apart from the query and the aspect terms to return relevant documents. However, these terms are not generated by considering the relationship between the query and the aspect, and may not always lead to the best results.

Apart from documents, related suggestions are also of interest to the user. Existing academic search engines, like Google Scholar, provide related suggestions after the query has fetched results so that the user can continue to explore the results to fetch more results. The top-2 suggestions provided by Google Scholar for the query `computer vision application` are `computer vision application opencv` and `computer vision application programming cookbook`, shown in Figure 1.3.

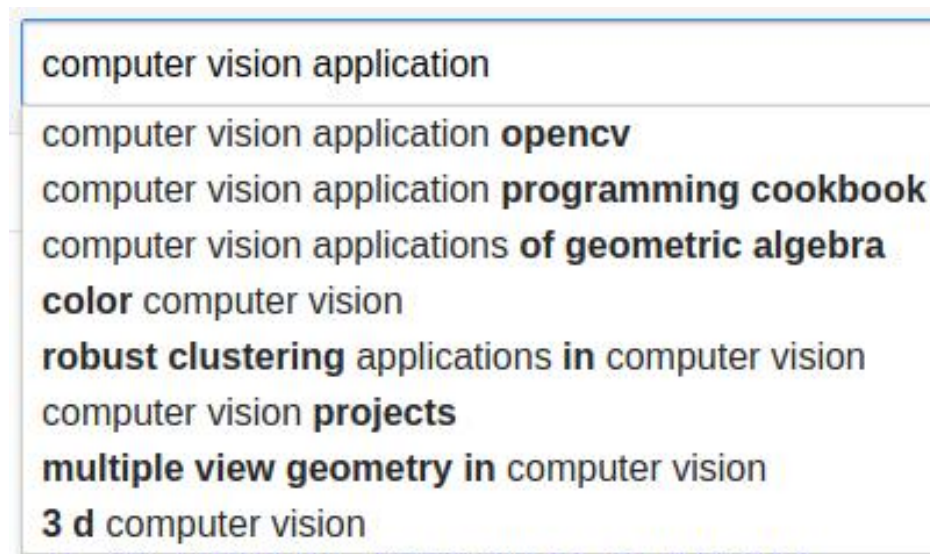


Figure 1.3: Suggestion provided by google scholar for the query `computer vision application` as on 20 July 2020

Both these suggestions retrieve the document *OpenCV Computer Vision Application Programming Cookbook*, which is not the best document to recommend for the application aspect. So, recommending `computer vision application opencv` or `computer vision application programming cookbook` at the top-2 position is not the best idea. Instead, recommending `computer vision application automatic inspection system` would help us retrieve a more relevant document. The procedure for generating suggestions for a query involves looking for the most similar and frequent suggestion in the list of candidate suggestions (generated from query logs or the set of relevant documents for the query) [25, 53]. In our case we have both the query and the aspect as the input, so, existing techniques will look for the query and the aspect terms in the set of candidate suggestions. As a result, these techniques will retrieve suggestions that contain the aspect term, which may not be the best suggestion, such as, `computer vision application programming cookbook`. Moreover, a suggestion not containing the aspect term can be relevant, such as `computer vision automatic inspection system`. So, retrieving the best suggestion for a query and an aspect is challenging, and requires knowledge of the domain.

We address both of these challenges by using language models. The key idea is to estimate a language model for the query and the aspect using TeKnowbase and use it to retrieve documents as well as suggestions, described in detail in Chapter 3.

1.3 Contributions

In this thesis, we have proposed a holistic system that can assist a computer science enthusiast in learning a new topic, as well as performing a literature review on that topic. The system takes a query and returns a reading order of concepts that have to be studied to understand the queried topic. Apart from that, it identifies key aspects towards an understanding of the queried topic and returns prerequisites, as well as research papers relevant for them. Figure 1.4 shows the major components of this system, which are the contributions of this thesis. These are described below:

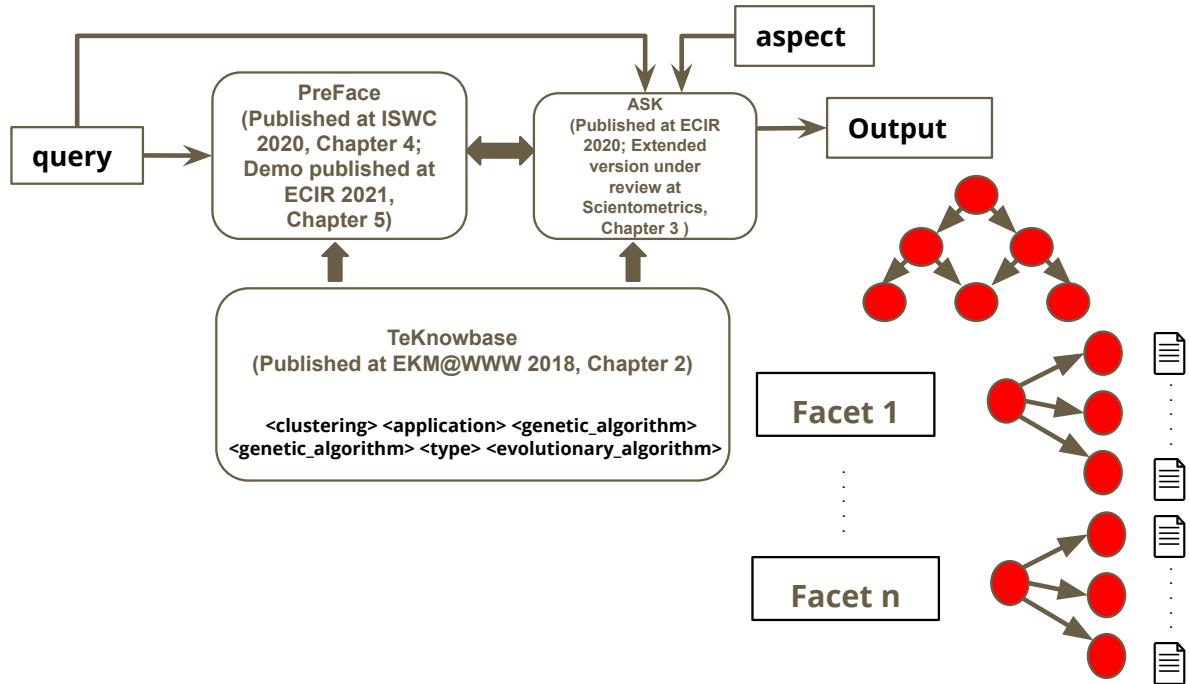


Figure 1.4: Architecture of the system that was built as a part of this thesis. The three main components of the systems are—1) TeKnowbase, which is a domain-specific knowledge base in Computer Science, 2) ASK, that returns relevant documents as well as query suggestions for a given query and an aspect, 3) PreFace, a retrieval system that returns interesting facets as well as prerequisites for a query.

1.3.1 TeKnowbase: A Knowledge Base of Computer Science Concepts

The backbone of our system is *TeKnowbase*⁶, a knowledge base in the domain of Computer Science. The entities in TeKnowbase are concepts in Computer Science, like `genetic_algorithm` or `clustering` and the relationships are both domain-independent relationships, like `typeof` or `synonym` and domain-specific, like `application` or `algorithm`. TeKnowbase stores facts about entities in Computer Science as triples. Some example

⁶Dataset available from <https://github.com/prajnaupadhyay/TeKnowbase>

triples are $\langle \text{clustering, application, memetic_algorithm} \rangle$, $\langle \text{memetic_algorithm, type, evolutionary_algorithm} \rangle$, $\langle \text{genetic_algorithm, type, evolutionary_algorithm} \rangle$, shown in Figure 1.4. These triples were extracted from both structured and unstructured sources, described in Chapter 2. There are two retrieval components that use TeKnowbase. The first component assists an academic search user to retrieve documents relevant to a query and an aspect of the query. The second component takes a query and returns faceted prerequisites for a topic of interest, about which the user has little or no idea. It makes use of the second component to do so. These components are further described in the next subsections.

1.3.2 PreFace: Faceted Retrieval of Prerequisites

We propose *PreFace*, which is a retrieval system to extract prerequisites for queries. PreFace recommends a prerequisite graph of necessary prerequisites, which *have to* be studied by the user, and along with them recommends facets of prerequisites, which are relevant for different aspects of the query. It assists the user in achieving an overall understanding of the queried topic. This is described in Chapter 4. The implementation of a prototype of this system is described in Chapter 5.

1.3.3 ASK: Aspect based Academic Search using Domain-Specific KBs

The second component of our system, *ASK*, takes a query and an *aspect* as input and returns a ranked list of documents as well as query suggestions that are relevant to both the query and the aspect. It makes use of TeKnowbase to formulate a new retrieval model to address the issues of using existing models for aspect-based retrieval of research papers. We compare ASK with various state-of-the-art techniques, such as pseudo-relevance feedback, diversification, and neural models and show that we outperform them by a substantial margin. This is described in Chapter 3.

1.4 Organization

The rest of this thesis is organized as follows. Chapter 2 describes the construction and evaluation of TeKnowbase. Chapter 3 describes ASK, the system that takes a query and an aspect as input and returns a ranked list of documents as well as query suggestions using TeKnowbase. Chapter 4 describes PreFace, a system to extract faceted prerequisites for a query using TeKnowbase. Chapter 5 describes the implementation of its prototype and extension, PreFace++. Finally, Chapter 6 concludes the thesis with future work directions.

Chapter 2

Construction of TeKnowbase

With the exponential growth of structured and unstructured data on the web over the last few decades, techniques to effectively store this data have also gained popularity. Knowledge graphs [88] are means to store such data in a crisp and organized fashion. They can model the semantics of real-world data by storing it as triples in RDF [1] format. As a result, they form the backbone of a number of semantic-aware applications such as question answering [127, 9, 85], entity summarization [81, 39, 186], or entity retrieval [154, 143, 146, 225]. They can also improve ad-hoc document retrieval tasks by providing well-known query expansion and smoothing techniques [19, 20, 24, 206, 104], or using KG relationships to semantically interpret the query and improving results [46, 124, 205, 158]. The success of using entities and relationships from a knowledge graph in supporting various applications has led to the construction of various knowledge graphs in a specific or open domains. There are already many such general-purpose knowledge-bases such as Yago [179], DBPedia [114], NELL [32], OpenIE [58], and Wikidata [57, 196] developed for research purposes. They are also crucial for the functioning of many commercial search

engines like Google’s Knowledge Vault [54], Bing’s Satori ¹, Facebook², and LinkedIn³.

2.1 Motivation and Problem

Although there exist a number of knowledge bases in the open domain, there is a lack of large-scale technical knowledge bases. There are, however, comprehensive and updated taxonomies in various technical domains. Taxonomies are used to define a hierarchy of concepts in a given field. Some examples are MeSH (Medical Subject Headings) [2] in the area of biology and PhySH (Physics Subject Headings) [3] in the area of Physics. Both these taxonomies define fine-grained concepts in their respective fields. However, existing computer science taxonomies such as ACM Computing Classification [4] are coarse-grained, defines only around 2000 concepts, and are manually created and maintained. Because of the expensive manual effort involved, the last two consecutive updates of the ACM Computing Classification System were in 2012 and 1998. Such a slow evolution makes it outdated, so, would be of limited value for a rapidly growing field such as Computer Science. This can be replaced by collaborative efforts, however, it comes with its own set of problems such as trust and reliability [41]. Since these are taxonomies, they are limited to defining hierarchies. It would be more useful to define more fine-grained relationships between the concepts apart from the hierarchical relationships.

In the field of bioinformatics, projects such as GeneOntology [16] have focused on extracting more fine-grained relationships between the concepts to populate the knowledge base. It primarily focuses on the nomenclature of gene and gene products and defining their functions. Another project in the same field is the UMLS (The Unified Medical Language System) [26] which attempts to integrate various terminologies across different biomedical databases. It defines two major types of relationships—i) hierarchical, such

¹<https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/>

²<https://www.adweek.com/digital/facebook-builds-knowledge-graph-with-info-modules-on-community-pages/>

³<https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph>

as ‘is a kind of’ or ‘isa’, ‘part of’, and ii) associative, such as ‘location of’ and ‘caused by’. Both of these projects have relied on collaborative efforts for the population of the knowledge base. Efforts to automate extractions of 6 domain-specific relationships from the unstructured text have been made in [43].

Efforts have also been made to create academic graphs [223, 174, 185, 51] consisting of paper, author, and venue information. These graphs are primarily used for the purpose of building academic dialog and recommendation. Although these graphs also store technical information, they are fundamentally different from the type of graph we want to build. We are interested in building a domain-specific knowledge base where the relationships between the various concepts in Computer Science are defined so that it can help us in recommending topics to study for basic and advanced understanding of a concept. However, there have not been any known efforts in the domain of Computer Science to build such a knowledge graph.

In the open domain, a number of techniques to automatically construct knowledge bases from both structured [179, 114] and unstructured sources [32] have been proposed. However, using the techniques proposed in [179, 114] does not lead to fine-grained extraction of relationships in the domain of Computer Science, because most of these techniques are designed for the open domain. Learning extractors for different relationships [32] needs a large amount of training data, which is hard to curate. Open Information extraction [58] techniques can be used to extract a large number of triples without any supervision, however, when it comes to long and complicated sentence structures which are common in technical documents, these techniques fail [197]. Systems such as DeepDive [204], which claim to automatically construct a knowledge base from structured and unstructured sources, fail to serve our purpose because it needs a curated set of triples as input so that more triples can be inferred. This makes the task of extracting a set of high-quality triples in Computer Science challenging. In a nutshell, constructing a knowledge base in the domain of Computer Science involves identifying the right set of resources and then formulating simple techniques to extract the triples.

2.2 Approach and Contributions

We address this gap by constructing TeKnowbase, which is a knowledge base in the domain of Computer Science. TeKnowbase is constructed by applying a combination of information extraction [166] techniques from semi-structured as well as unstructured sources to increase the coverage of extractions. Our contributions are as follows.

1. We describe the construction of TeKnowbase in Sections 2.4. We demonstrate that by the use of simple techniques, we were able to return high-quality extractions. We made use of a wide variety of IE techniques to improve the recall of extractions.
2. We conduct a thorough evaluation of the quality of TeKnowbase and we report our results in Section 2.5. TeKnowbase is a freely available online resource.

This chapter is organized as follows. Section 2.4 describes the construction of TeKnowbase where the input and output of the extraction process are described in Section 2.4.1 and Section 2.4.2 describes the pipeline, techniques used, and statistics of the constructed knowledge base. Section 2.5 describes experiments to evaluate the quality.

2.3 Related Work

Knowledge bases can be constructed manually, by seeking experts' help to populate the graph, or automatically/semi-automatically using statistical or learning techniques.

2.3.1 Manual Creation of Knowledge Graphs

Manual construction of knowledge graphs relies on the knowledge of experts or crowd. These techniques fall into two major categories: 61

- Manual curation by experts: A set of experts in the given domain are asked to identify triples to be populated. Some of the earliest work on knowledge graph construction relied on experts' knowledge to manually populate the knowledge graph [115]. This makes it a slow and expensive procedure.
- Manual curation by a set of people: Curation of triples is a collaborative process, usually done by a group of people. Although it made it faster, it comes with its own set of issues such as trust or reliability [41].

2.3.1.1 Manual Curation of Knowledge Graphs by Experts

Among the manually created knowledge bases by experts, CYC [115] is a universal schema consisting of around 100000 concepts from human reality. For instance, it stores common sense knowledge such as “You have to be awake to eat” and “You can often see people’s noses, eyes or ears but not their hearts”. The goal of creating such a knowledge graph was to provide a standard ontology for the world wide web or electronic commerce. [136] proposed WordNet, which provides a vocabulary of words in the English language, along with their meanings (called *sense*), and accompanied by synonyms, antonyms, hyponyms, and meronyms. Projects such as UMLS (Unified Medical Language System) [26] aim to define the biomedical vocabulary developed by the US National Library of Medicine. It focuses on around 60 families of biomedical vocabularies, identifying more than 900000 concepts. These concepts are described by almost 2 million names as well as 12 million relations among these concepts.

2.3.1.2 Manual Curation of Knowledge Graph by Collaborative Efforts

These techniques use a network of people to populate triples in a knowledge base. Examples are the projects such as Wikidata [57, 196], Freebase [72] and GeneOntology [16]. Wikidata aims to collect structured information from people all over the world to support Wikipedia. Freebase was a knowledge base in the open domain, that was also collabora-

tively created and maintained. It has been acquired by Google and is now known as the Google Knowledge Graph. The GeneOntology project [16] is an effort to create a knowledge base in the biomedical domain. It was developed by the Gene Ontology Consortium and aims to annotate gene functions in genome databases.

Manual construction of knowledge bases requires a great deal of effort, so a number of techniques have been proposed that aim to construct knowledge bases *automatically*. An overview of these techniques is provided in Section 2.3.2.

2.3.2 Automatic Creation of Knowledge Graphs

The automatic construction of knowledge bases is done from two types of sources—structured/semi-structured and unstructured. Structured sources make use of various structured components of online resources, such as ordered lists, sections, etc. Unstructured sources include sentences used to describe any article. Recently, systems that facilitate knowledge-base construction from heterogeneous sources have been proposed. To illustrate, DeepDive [218, 204, 147] consumes 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 [54] 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 customize our extractions.

2.3.2.1 KG Creation from Semi-Structured Sources

The techniques to extract triples from structured sources involve formulating rules/patterns or learning that utilize the structure of lists, category hierarchy, or info boxes in the Wikipedia web pages. Knowledge bases such as YAGO and its successors [179, 86] as well as DBPedia [114] have been built using the structure of various web components of Wikipedia, such as category hierarchy or infoboxes. YAGO additionally connects Wikipedia pages to concepts from Wordnet. The authors propose CERES [125], which is

a technique to automatically extract triples from semi-structured sources using distance supervision. It uses a seed knowledge base having a bare-minimum ontology, using which it extracts facts from web pages.

2.3.2.2 KG Creation from Unstructured Sources

The techniques that construct knowledge bases from unstructured sources consist of two main components:

- **Entity Extraction:** This problem deals with identifying named entities from the domain of interest. A number of techniques have been proposed to solve this problem which has been summarized in Section 2.3.2.2.
- **Relation Extraction:** This area of research attempts to identify the relationship existing between the entities that have been identified from the previous steps. A review of these techniques is presented in Section 2.3.2.2.

In most cases, these two problems are solved independently of each other. However, techniques have been proposed to jointly extract entities and relationships together, such as [117], [139], [138], [224], which reduces the amount of error introduced due to using the two techniques one after the other.

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 (such as lists of people), but for others, we need techniques to build this dictionary. This task can be divided into two separate problems—i) detection of named entities, similar to segmentation of chunking [106], and ii) classification of named entities into a given set of ontological types, such as $\{person, location\}$, etc.

The most popular techniques for named entity recognition are supervised, which include Hidden Markov Models, Conditional Random Fields, and Support Vector Machines [90, 226, 109]. These techniques learn the disambiguation rules based on features extracted from a large annotated corpus.

Among the semi-supervised approaches [157, 140], bootstrapping is the most popular technique. It reduces the human involvement by a large margin, and has recently been applied to detect named entities in Twitter with minimal supervision [161, 123]. Another technique, called multi-level bootstrapping [160], starts by tagging handful of named entities in a large corpus, acquires its context and grows the set as well as the context by accumulating patterns around the named entities. [33] proposed techniques to generate named entities of the same type by generalizing patterns occurring around the entities using distributional similarity.

Among the unsupervised techniques, [14] assigns a *topic signature* to each Wordnet synset by looking for co-occurring words. Then, it classifies a word given as input into the most similar synset. [83] originally described a method to identify hypernym/hyponyms. This was later used to identify hypernyms/hyponyms of sequences of capitalized words appearing in a document. [60] proposed a PMI-IR technique to classify a named entity into a given type.

Relation Extraction. The problem of relation extraction is not new, and it has been studied for over two decades. These techniques range from domain-specific extraction of relationships to extracting triples not dependent on any ontology. Another dimension that can be used to classify these techniques is based on whether they use hand-coded rules to extract the triples or use sophisticated learning techniques to predict the presence of the relationship. Table 2.1 summarises these techniques based on these dimensions.

1. **Domain Specific Relation Extraction:** The user needs to provide the names of the relationships or the schema of the knowledge base for which the triples have to be extracted. These can be further classified into rule-based/statistical techniques

Techniques	Rule based/Statistical	Unsupervised	Supervised	Semi-supervised
Relations are specified	Hearst Patterns [83], Prospera [141]	URES [164]	feature extraction from sequences/-parse trees [98], kernel based classifiers [29, 181], NELL [32], Neural [216, 176, 145, 138, 221, 222, 122, 71]	relation extraction for biological knowledge base creation [43], bootstrapping [28, 11, 214, 59], distant supervision [137, 159, 87, 183, 162, 78, 54, 220]
Relations are not specified	REVERB [61], KRAKEN [13], EXEMPLAR [133], PROPS [62], Pred-Patt [200]	[80], [36], [188], [49], [210], [213], [130], [173]	[58], [203], [168]	

Table 2.1: Survey of relation extraction techniques

or learning-based techniques such as unsupervised, supervised, or semi-supervised. One of the seminal works that use rules or patterns to extract triples participating in a given relation is by Marti Hearst [83]. She proposed techniques to identify patterns for a relationship expressed by a pair of entities. In [142], the authors developed a scalable system called PROSPERA that extracts n-gram patterns for a given relationship type and uses MaxSat-based constraint reasoning to refine them. This technique forms a part of a scalable architecture implemented using the Map-reduce framework.

In addition to statistical techniques, various learning-based relation extraction systems have been built. The supervised techniques take a set of positive and negative examples for a relationship type and learn classifiers to predict that relationship type. These are the most common and yield high performance. These techniques either extract features from sequences and parse trees and then learn a classifier, such as [98], or use kernel-based classifiers, such as [29, 181]. These techniques rely on expert's intuition for their design. So, neural network classifiers have been proposed which automatically extract features [216, 176, 145, 138, 221, 222, 122, 71]. Although more sophisticated neural networks lead to improvement in the accuracy, it comes at its own cost of annotations and expensive training process.

The aim of unsupervised techniques is to avoid the time and effort spent in annotating the entities with relations. [164] proposed URES, a completely unsupervised technique to extract instances of given relation as input. URES takes the description of the given relation as input and a few seed examples taken from another system called KnowitAll [59] and learns patterns for positive and negative seed instances.

A mix of two techniques (supervised and unsupervised) is employed by semi-supervised techniques. The key idea used by such systems is bootstrapping, where a small number of high-quality training instances are used to create a larger set of training data iteratively [8]. Popular systems are DIPRE [28], SnowBall [11], TextRunner [214] or KnowitAll [59]. Another popular approach, called distant supervision, starts with a few training examples and generates more data and features by applying various types of heuristics [137, 159, 87, 183, 162, 78, 220, 54].

2. **Domain-Independent Relation Extraction:** In such techniques, one does not need to provide names of the relationship or the type of entities that are likely to participate in that relation. Instead, these relations are identified automatically. These techniques fall under the broad category of open information extraction. These can be divided into rule-based, unsupervised, and supervised techniques.

Among the rule-based techniques, REVERB [61] is a shallow extraction system that improves the quality of extractions of previous OpenIE systems by introducing a POS-based syntactic constraint. Another system that uses handwritten rules to extract n-ary relations is KRAKEN [13]. [133] proposed a system called EXEMPLAR, which identifies the connection between the argument and the predicate words in a relationship using the dependency paths between them. PrePatt [200] and PROPS [62] use simple rules to convert the syntactic dependency tree of the predicate in a sentence to a graph-based semantic representation. Both argue that the graph-based representation was able to improve the quality of extractions.

Among the unsupervised techniques, [80] uses hierarchical clustering to group together pairs of named entities likely to participate in the same relation. The similarity between the pairs of entities is measured by taking the context i.e. the terms occurring around the mentions of these entities in the text. The most frequent words in each cluster are then used to select a representative name for the relation expressed by that cluster. However, because it is difficult to determine the similarity threshold and frequency-based naming of the relation is not always accurate, [36] proposed a discriminative category matching (DCM) technique to find discriminative words to describe the clusters. In [188], authors address the inverse relation extraction problem where phrases most likely to express a relation between a pair of entities provided as input are returned in decreasing order of likelihood. A slightly different problem is addressed by [49], where the goal is to automatically extract relationships likely to participate with the input concept of a particular class provided as input. In [213], authors propose generative models to automatically predict pairs of entities participating in different relations. They do so by clustering the dependency paths existing between the entities in the sentences.

2.4 Construction of TeKnowbase

In order to construct TeKnowbase, we first acquired a dictionary of concepts and entities relevant to computer science. Using this dictionary, we extracted relationships among them using a combination of IE techniques [166].

2.4.1 Input and Output of Extraction Process

An information extraction task takes different types of sources as input and returns structured information [166]. So, it is important that we first decide upon the input (sources used for extraction) and output (structures extracted).

2.4.1.1 Output

This chapter focuses on extracting two types of structured data—named entities and relations.

Named Entities. The concepts in the domain of Computer Science make up the named entities to be populated in the knowledge base. For instance, `artificial_neural_network`, `phoneme_classification` and `machine_learning` are entities. These can be useful in annotating computer science text.

Relationships. They describe the relationships that can exist between the identified entities. This task involves identifying pairs of entities (already extracted) participating in a given relation. This returns a triple of the form `<subject><predicate><object>`, where the `<subject>` and the `<object>` are entities and the `<predicate>` represents the relation. The `<subject>` is the head entity and `<object>` is the tail entity of the triple. To illustrate, the triple `<artificial_neural_network, subtopic, machine_learning>` describes

the subtopic relationship between `artificial_neural_network` and `machine_learning`. `(phoneme_classification, application, artificial_neural_network)` describes the application relationship between `phoneme_classification` and `artificial_neural_network`. The extracted triples are stored in RDF format.

2.4.1.2 Input

We focus on two types of sources—structured/semi-structured and unstructured sources in the domain of Computer Science for our extractions.

Structured/Semi-Structured Sources. We consulted various web resources to extract entities and relationships to be populated in TeKnowbase. These are described as follows:

1. **Wikipedia.** Wikipedia is a free online encyclopedia that is collaboratively maintained. It consists of dedicated articles on various topics. These articles are organized into various categories and sub-categories. The articles from Computer Science can be found in the “Computing” category. So, we make use of the articles in this category to build our dictionary of entities (more details in Section 2.4.2.1). The information in Wikipedia articles is further organized into more granular structures such as table of contents, lists, sections, or templates, which provide some idea about the relationship between topics (described in detail in Section 2.4.2.2)
2. **Domain-Specific Websites.** Our second set of resources were two websites, Webopedia⁴ and TechTarget⁵. Each website consists of a number of technical terms and their definitions in a specific format, which can be extracted by writing crawlers for those websites (described in detail in Section 2.4.2.1).

⁴<http://www.webopedia.com>

⁵<http://www.techtarget.com/>

Unstructured Sources. Our unstructured sources comprised of the following:

1. **Description of Wikipedia and Computer Science Articles.** This consists of raw sentences/descriptions of topics found in Wikipedia, Webopedia, and TechTarget websites.
2. **Online Textbooks.** Apart from these web pages, a number of online textbooks are available for free on the web. The last few pages of such textbooks can be used to refer to some key terms used throughout the textbooks. Such terms can be used as named entities in Computer Science. More details about the extraction process is described in Section 2.4.2.1. These indexes also consisted of (acronym, expansion) pairs which were a good source of **synonym** relations. Since this involves relation extraction, we defer the description of our technique to Section 2.4.2.2.

2.4.2 Construction Pipeline

We divide our construction process into three major parts:

1. **Entity Extraction.** To build a technical knowledge base, the first step is to identify the dictionary or the set of entities. A common filter that can be applied to all triple extraction techniques is to ensure that the entities in the triples are also present in our dictionary. Therefore, apart from being useful in annotating textual resources, the dictionary helps in ensuring that the triples that we acquire are accurate at least so far as the entities are concerned. This is described in detail in Section 2.4.2.1.
2. **Relation Extraction.** After we have identified the entities, the next step is to develop extractors to identify relationships between them. We focus on extracting two types of relations—i) known, domain-specific relations identified prior to the extraction process

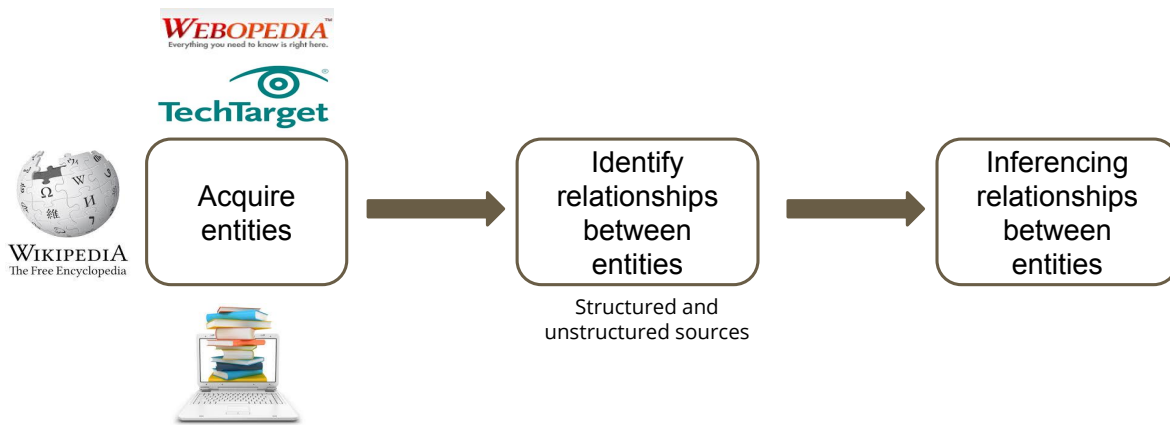


Figure 2.1: Construction methodology of TeKnowbase. The three main parts are i) Acquiring dictionary of entities from sources such as Wikipedia, Webopedia/TechTarget and online textbooks, ii) Identifying relationships between the entities using heuristics applied on structured and unstructured triples iii) inferencing more triples using inferencing models

by domain experts, ii) unknown relations, that were extracted without any information of the domain. The procedure is described in detail in Section 2.4.2.2.

3. Inferencing of Triples. Like other knowledge bases, TeKnowbase is not complete. Given a relation, this component is used to automatically identify pairs of entities (not already participating in that relation) that are most likely to be connected to each other. More details on this procedure are provided in Section 2.4.2.3.

These three parts are described in the Figure 2.1.

2.4.2.1 Entity Extraction

The first step in creating any domain-specific knowledge base is the identification of entities. A number of supervised as well as unsupervised techniques have been proposed

to identify entities from the text (described in detail in Section 2.3). We follow a more straightforward approach—we specifically target technology websites and write wrappers to extract a list of entities related to computer science.

Wikipedia. We used Wikipedia’s article titles as well as its category system as a source of concepts. Our corpus of Wikipedia articles consists of all articles under the category `Computing`. We used the PetScan ⁶ tool to collect articles under 3-hop subcategories of “Computing” category. 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. To illustrate, the terms `average_page_depth` (related to Web Analytics) and `fraction_ridge` (related to biometrics) could not be found in Wikipedia, so we made use of alternative resources to augment our entity list.

Webopedia and TechTarget. Websites such as Webopedia and TechTarget follow a specific format to list the key terms from Computer Science. We analyzed the HTML code for these websites and wrote domain-specific wrappers to extract a list of terms. Figure 2.2 shows a snapshot of the terms and the HTML code from the TechTarget website. We extracted approximately 24,000 entities. Some of the entities that we extracted from these two sources are `average_page_depth`, `fraction_ridge`,

Online Textbooks. The third source of our entities was indexes on online textbooks. We extracted 16,500 entities from the *indexes* of 8 online textbooks. These were on the following topics

1. Computer Networks
2. Database Management Systems
3. Artificial Intelligence

⁶<https://petscan.wmflabs.org/>

Browse Definitions by Alphabet

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z #

BEL - BIN

B - BAR	
BAR - BEL	
BEL - BIN	
BIO - BLA	
BLA - BOO	
BOO - BRA	
BRA - BTU	

bell curve - A bell curve is a form of graph that is used to visualize the distribution of a set of chosen values across a specified group that tend to have a central, normal values, as peak with low and high extremes tapering off relatively symmetrically on either side.

Bell operating company - BOC (Bell operating company) is a term for any of the 22 original companies (or their successors) that were created when AT&T was broken up in 1983 and given the right to provide local telephone service in a given geographic area.

```

<div id="content-center" class="content-center">
  <!-- AlphaDefinitionsController, generated at 21:20:46
  -->
  <section class="section browse-alpha-bridge" id="definitions-listing">
    <ul>
      <li><a href="https://whatis.techtarget.com/definition/bell-curve">bell curve</a> - A bell curve is a form of graph that is used to visualize the distribution of a set of chosen values across a specified group that tend to have a central, normal values, as peak with low and high extremes tapering off relatively symmetrically on either side.</li>
      <li><a href="https://searchnetworking.techtarget.com/definition/BOC">Bell operating company</a> - BOC (Bell operating company) is a term for any of the 22 original companies (or their successors) that were created when AT&#38;T was broken up in 1983 and given the right to provide local telephone service in a given geographic area.</li>
      <li><a href="https://searchnetworking.techtarget.com/definition/Bellcore">Bellcore</a> - Bellcore (Bell Communications Research) provides certain centralized research and standards coordination for the regional Bell operating companies (RBOC)s.</li>
    </ul>
  </div>

```

(a) Snippet of the terms in the webpage for Whatis.Techtarget (b) Snippet of the corresponding HTML code

Figure 2.2: Snippet of the HTML page describing the terms and the corresponding code from Whatis.Techtarget

4. Discrete Mathematics
5. Machine Learning
6. Operating Systems
7. Software Engineering
8. Network Security

Combining Entities from Different Sources. We used edit distance to resolve overlapping entities from these sources and ended up retaining over 70,000 entities. The distance for edit distance was set to 1.

2.4.2.2 Relation Extraction

We divided the set of relationships to be extracted into two parts—i) set of known relations, identified by domain experts, which have to be provided as input to the extraction process, and ii) set of unknown relations, which will be automatically determined by the

extraction process. Table 2.2 lists 24 domain-specific relations that we identified to be extracted.

We further 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.

Based on these two dimensions, we divided our extraction procedure into 4 types, as described in Table 2.3.

1. Extracting Known Relations from Structured Sources. Since our aim is to extract relations between the named entities, 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 24 relationships that we felt were interesting and formulated techniques to extract them from Wikipedia. We focused on two kinds of structured sources—i) Overview pages, comprising of “List of” and “Outline of” pages, and ii) Structured pieces of information present in articles on various topics, such as table of contents, section list within articles, and list hierarchies. Both the sources and the extraction techniques are described in the following section.

- 1. Overview Pages** (500 pages): We made use of two kinds of structured pages—“List” pages and “Outline” pages (such as, the pages, `List of data structure`, or `Outline of Cryptography`, etc.). These pages organize lists of entities with headings and sub-headings. The “Outline” pages were used to extract `subtopic` relation and “List” pages were used to extract `type` relations with good accuracy (see Section 2.5 for an evaluation of these relations). Figure 2.3 shows snapshots of two overview pages—`Outline of Cryptography` and `List of data structures`. From those pages, we were able to extract triples such as `<cipher, subtopic,`

Sr. no	Relation	Examples of ⟨head_entity, tail_entity⟩
1	algorithm	⟨breadth-first_search, graph_traversal⟩
2	application	⟨activity_recognition, hidden_markov_model⟩
3	approach	⟨waterfall_development, software_development_process⟩
4	classification	⟨educational_software, application_software⟩
5	component	⟨user_interface, operating_system⟩
6	concept	⟨declarative_programming, logic_programming⟩
7	data-structure	⟨conceptual_graph, conceptual_schema⟩
8	examples	⟨linked_list, persistent_data_structure⟩
9	implementation	⟨strongswan, internet_key_exchange⟩
10	measures	⟨precision, information_retrieval⟩
11	method	⟨spectral_method, parallel_computing⟩
12	models	⟨entity-relationship_model, conceptual_model⟩
13	problem	⟨covering_problem, graph_theory⟩
14	properties	⟨distributivity, negation⟩
15	research	⟨neural_imaging, neural_engineering⟩
16	software	⟨windows_10_mobile, mobile_operating_system⟩
17	subtopic	⟨exterior_algebra, multilinear_algebra⟩
18	synonym	⟨bfs, breadth-first_search⟩
19	system	⟨google_translate, statistical_machine_translation⟩
20	tasks	⟨feature_extraction, digital_image_processing⟩
21	technique	⟨multilinear_pca, exploratory_data_analysis⟩
22	terminology	⟨suslin_hypothesis, metamathematics⟩
23	theory	⟨de_morgans_law, boolean_algebra_topic⟩
24	type	⟨binary_tree, tree⟩

Table 2.2: List of known relations identified by domain experts.

	Structured	Unstructured
Known Relations	Simple heuristics applied to Structured sources	Pattern finding for synonym relation
Unknown Relations	Simple heuristics applied to Templates	OpenIE applied on unstructured text

Table 2.3: 4 kinds of IE tasks based on source and types of relations to be extracted

cryptography} or {integer,type,data_structure}.

2. **Articles on Specific Topics.** These pages refer to the discussion on specific topics such as, say, “Coding Theory”. These pages consist of many structured pieces of information as follows:

- (a) **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. In particular, in Figure 2.4, 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}.
- (b) **Section-List within Articles** (1909 section lists): Next, there are several sub-headings in articles that consist of links to other topics. For instance, the page on “Document Classification” consists of a subheading “Techniques”—this section simply consists of a list of techniques that 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 {tf-idf,technique,document_classification}. This is shown in Figure 2.4 (d).
- (c) **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. Figure 2.4 (a) shows an example of a list page for data structures. We see a list of terms under the heading “Lists” and can extract triples of the form {doubly_linked_list,type,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,type,data_structure} was extracted based on the article title.

Outline of cryptography

From Wikipedia, the free encyclopedia

Essence of cryptography [\[edit \]](#)

- [Cryptographer](#)
- [Encryption/decryption](#)
- [Cryptographic key](#)
- [Cipher](#)
- [Ciphertext](#)
- [Plaintext](#)
- [Code](#)
- [Tabula recta](#)
- [Alice and Bob](#)

(a) Snapshot of “Outline of Cryptography” page. It lists a number of topics from the field of Cryptography which can be used to extract **subtopic** relation

List of data structures

From Wikipedia, the free encyclopedia

Data types [\[edit \]](#)

Primitive types [\[edit \]](#)

- [Boolean](#), true or false.
- [Character](#)
- [Floating-point numbers](#), limited precision approxir
 - Including [Single precision](#) and [Double precision](#)
- [Fixed-point numbers](#)
- [Integer](#), integral or fixed-precision values.
- [Reference](#) (also called a pointer or handle), a sma
- [Enumerated type](#), a small set of uniquely named v
- [Date Time](#), value referring to Date and Time

(b) Snapshot of “List of data structures” page. It lists a number of data structures which can be used to extract **type** relation with good accuracy

Figure 2.3: Snapshot of two overview pages used to extract **subtopic** and **type** relation

2. Extracting Unknown Relations from Structured Sources. This comprises of structured pieces of information organized in some other form that helps us directly deduce relationships between a pair of concepts. We used Wikipedia templates to deduce new relationships between the entities.

Templates (1139 templates): Wikipedia has various pieces of structured information embedded in the form of templates. Templates are used to provide an overview of topics related to the Wikipedia article. Figure 2.5 shows a template from the “Database Management Systems” page. It organizes all the topics related to **Database Management System** into **Types**, **Concepts** and **Objects** etc. This allows us to extract the row headings as potential relations and triples such as $\langle \text{query_optimization}, \text{functionOf}, \text{database_management_systems} \rangle$. However, based on our evaluation (see Section 2.5), we found that templates could have a variety of row headers and they were not always reliable. Therefore, we did

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 using the two level hierarchy.

5 Line coding

6 Other applications of coding theory

 6.1 Group testing

 6.2 Analog coding

7 Neural coding

8 See also

(b) Snippet of the TOC in “Coding theory” page to extract applicationOf relations

Derivative-free optimization

Algorithms [edit]

- DONE
- Bayesian optimization
- Coordinate descent and adaptive coordinate d
- Cuckoo search
- Evolution strategies, Natural evolution strategi
SNES)
- Genetic algorithms
- LIPO algorithm
- MCS algorithm
- Nelder-Mead method

(c) Snippet of section list to extract algorithm relation

Document classification

Techniques [edit]

Automatic document classification techniques include:

- Expectation maximization (EM)
- Naive Bayes classifier
- tf-idf
- Instantaneously trained neural networks
- Latent semantic indexing
- Support vector machines (SVM)
- Artificial neural network
- K-nearest neighbour algorithms
- Decision trees such as ID3 or C4.5

(d) Snippet of section list to extract technique relation

Figure 2.4: Snippet of pieces of structured elements found in Wikipedia articles that can be used to extract known relations

V·T·E		Database Management System
Types		Object-oriented (comparison) · Relational (comparison) · Document
Concepts		Database · ACID · Armstrong's axioms · CAP theorem · CRUD · I
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

Figure 2.5: Snapshot of template for the topic Database Management System

not canonicalise these relations, but instead treated them as a generic “relatedTo” relation (therefore, instead of `functionOf`, we had `functionOf_(relatedTo)`).

3. Known Relations from Unstructured Sources. As already described in Section 2.4.1, our unstructured sources include a textual description of terms in both Webopedia and TechTarget as well as Wikipedia text of articles corresponding to entities, and text extracted from indexes of online textbooks. We limited the text in Wikipedia to the first paragraph. Our known relation, in this case, was `synonym` relation. The two types of sources we used to extract this relation are described as follows:

1. **Indexes of online textbooks.** The simple way in which the abbreviations along with their expansions were organized in the indexes allowed us to extract `synonym` relation with high accuracy. Figure 2.6 shows a few indexes present in a “Computer Networks” textbook. It is clear that the pattern CBC (cipher-block chaining) indicates that cipher-block chaining is an expansion for CBC. So, we looked for patterns “X (Y)” where X and Y were entities already extracted and deduced the triple

$\langle X, \text{synonym}, Y \rangle$.

2. **Patterns from Wikipedia and Webopedia text.** We were also 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.

carrier sensing, 454–456
 CBC (cipher-block chaining), 681–682
 CBR (constant bit rate) ATM network service, 312
 CDMA (code division multiple access), 522–526
 CDNs (Content Distribution Networks), 588, 603–608

Figure 2.6: Snapshot of indexes present in the “Computer Networks” online textbook. The simple pattern in which the abbreviations and their expansions are arranged allowed us to extract `synonymOf` relation with high accuracy

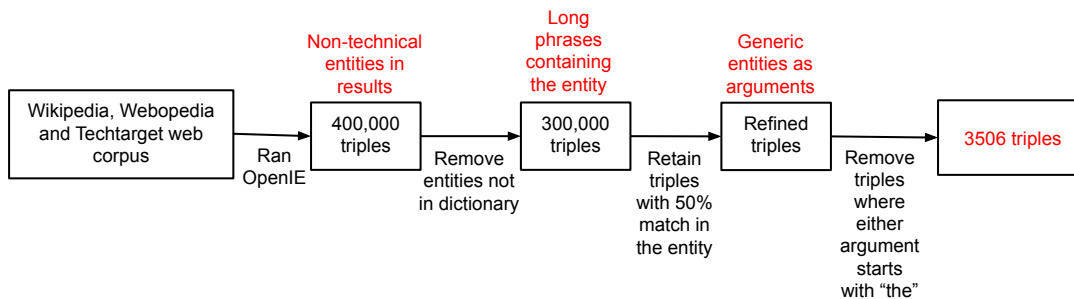


Figure 2.7: Pipeline of post-processing steps on the triples extracted from OpenIE

4. Unknown Relations from Unstructured Sources. Our next set of extraction involved identifying unknown relations from unstructured text. We ran the latest version of OpenIE [169] 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. Figure 2.7 shows the set of post-processing steps we performed on the extractions from OpenIE.

Filter 1: Triples which did not contain entities from our entity dictionary were discarded. This filter alone reduced the number of unusable triples by nearly 300,000. We searched for the longest matching entity in both the subject as well as the object of the triple. As long as there was a match we retained the triple.

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. For instance, `<huffman coding, is, just one of many algorithms for deriving prefix codes>` is a triple extracted by OpenIE. Since OpenIE sometimes assigns pieces of text (as in the example above) rather than crisp entities, we found that despite matching the longest entity, the triple still did not make sense. The entities extracted from this triple after retaining longest matching entity are `huffman_coding` and `prefix_code`. Therefore, our second filter retained only those triples where the entity match was at least 50% of the words in the subject or object.

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. In general, we were able to identify fine-grained relationships among entities. Table 2.6 shows a few examples.

2.4.2.3 TeKnowbase Completion

So far we were successful in acquiring entities and triples from different web resources using a combination of IE techniques. 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* [175, 144, 108]. Consider the triple $\langle \text{knights_tour}, \text{typeof}, \text{state_space_search} \rangle$ and $\langle \text{state_space_search}, \text{typeof}, \text{graph_algorithm} \rangle$, it can be easily inferred that $\langle \text{knights_tour}, \text{typeof}, \text{graph_algorithm} \rangle$ using the transitive property, even if this triple is not present in the knowledge base. To extract such rules, a number of inferencing models have been proposed. One such model that we experimented with is described as follows:

1. Neural Tensor Network. [175] 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 a 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 [134] 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 an unstructured Wikipedia corpus as well as text from online textbooks. 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.

1.	<code><xbasic,typeOf,programming_language></code>
2.	<code><binomial_heap,typeOf,tree></code>
3.	<code><palmdos,typeOf,operating_system></code>
4.	<code><delayed_column_generation,typeOf,convex_programming></code>
5.	<code><levenshtein_coding,typeOf,entropy_coding></code>

Table 2.4: Some new triples that were added to TeKnowbase using the Neural Tensor Network inferencing model.

- 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 2.4. Evaluation of these triples is described in Section 2.5.

2.4.2.4 Availability/Statistics of TeKnowbase

TeKnowbase is made available under the Creative Commons Attribution Licence 3.0 and can be downloaded from <https://github.com/prajnaupadhyay/TeKnowbase>. 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. The source code and the files used to generate TeKnowbase can be found at the following

No. of unique entities	70,285
No. of unique relations	2,574
Taxonomic relations (<code>typeOf</code>)	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	4,506

Table 2.5: Statistics from TeKnowbase. Apart from 70,285 unique entities, there are 32,458 variations (disambiguated as well as stemmed versions) of them.

location:

<https://bitbucket.org/prajnaupadhyay/teknowbasecode/src>

Some statistics about TeKnowbase are listed in Table 2.5 and described as follows:

1. Number of Entities. Line 1 in Table 2.5 reports the number of unique entities as 70,285. Apart from that, we extracted around 32,458 variations of them in the form of disambiguations and stemmed versions.

2. Number of Relations. We extracted 2574 unique relations. We were able to extract a large number of arbitrary relations apart from the 24 identified types by applying appropriate filters on the extractions from OpenIE. Examples of relations extracted by OpenIE include `is_a_high-speed_form_of`. The triple participating in this relation is `<gigabig_ethernet, is_a_high-speed_form_of, ethernet>`. This is given in table 2.6.

3. Number of Triples. We extracted 146,657 triples. Out of them, 27,078 were triples that participated in `type` relation and formed the taxonomy of concepts. More than

Relation	Examples of $\langle \text{head_entity}, \text{tail_entity} \rangle$	# Triples
type	$\langle \text{topological_sorting}, \text{graph_algorithm} \rangle$	27,078
concept	$\langle \text{nash_equilibrium}, \text{game_theory} \rangle$	595
subTopic	$\langle \text{hamming_code}, \text{algebraic_coding_theory} \rangle$	2,026
application	$\langle \text{group_testing}, \text{coding_theory} \rangle$	324
terminology	$\langle \text{blob_detection}, \text{image_Processing} \rangle$	27,018
is_a_high-speed_form_of	$\langle \text{gigabit_ethernet}, \text{ethernet} \rangle$	n/a
is_an_adaptation_of	$\langle \text{ironpython}, \text{python} \rangle$	n/a
uses	$\langle \text{utc}, \text{gregorian_calendar} \rangle$	n/a

Table 2.6: Statistics for and examples of a subset of relationships extracted. The first set of 5 relations were extracted from structured sources and the second part with 3 relations from unstructured textual sources (see Section 2.4.2.2)

99,000 triples were extracted from Wikipedia. The remaining triples were extracted from unstructured sources, consisting of filtered OpenIE extractions and triples participating in synonym relations.

4. Overlap with Other Knowledge Bases. One of the goals of constructing TeKnowbase was to have a knowledge base in Computer Science that defined fine-grained entities and relationships which could not be found in other existing knowledge bases. So, we also measured the overlap of TeKnowbase entities with that of DBPedia and Freebase. We did this using exact string matching of entities i.e. there is an overlap if the exact entity is present in both the sources. 34,000 entities were in common with Freebase and 17,000 entities were found in common with DBPedia. This means that more than half of the entities in TeKnowbase are unique. This information is present in Line 5 and 6 of Table 2.5. Entities that we have in common with DBPedia and Freebase are linked using the `owl:sameAs` relation.

Table 2.6 shows examples of the kind of triples we extract and the number of such triples in our knowledge base.

2.5 Evaluation of Quality of TeKnowbase

2.5.1 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 canonicalized, 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.

We also evaluated the class of rarer relations in the following way. We first created 5 classes of triples consisting of relations participating exactly in 1, 2, 3, 4 and 5 triples. Then, for each class, we sampled 10% of the triples. Following a similar approach, each triple was evaluated by two evaluators and we marked a triple as correct only if both evaluators agreed. Table 2.8 shows the accuracy values for the relations from this class.

For inferencing, we experimented with 4 different models (as described in Section 2.4.2.3). We first obtained a list of entities that were tagged in the textbooks and constructed an induced TeKnowbase on those entities. We only considered triples participating in those entities to be added to the training set and the test set. To create the training set, we considered the following relations:

1. `terminology(relatedTo)`
2. `typeOf`
3. `conceptIn`
4. `subTopicOf`
5. `applicationOf`

6. system(relatedTo)
7. componentIn
8. subField(relatedTo)
9. algorithmFor
10. isSoftware(relatedTo)
11. examples(relatedTo)
12. is a form of

We then sampled triples for each of the relations above. For `terminology(relatedTo)` and `typeOf` relations, we added all the triples in induced TeKnowbase to the training set. We generated a list of *true* triples using the transitive property on the `typeOf` relations and added it to the test set. For instance, given triples $\langle \text{palmdos}, \text{typeOf}, \text{dr_dos} \rangle$ and $\langle \text{dr_dos}, \text{typeOf}, \text{operating_system} \rangle$ in the training set, we returned a triple $\langle \text{palmdos}, \text{typeOf}, \text{operating_system} \rangle$ in the test set, ensuring that it does not already exist in the training set. For the rest of the relations, we added 80% of the triples to the training set and 20% to the test set. To generate negative examples, we shuffled the head entities of the positive set of triples.

2.5.2 Results and Analysis

Table 2.7 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 terminologies such as Photoshop and Adobe Photoshop. Table 2.8 shows the evaluation and confidence intervals for the class of rarer relations. The triples in this category are extracted using

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

Table 2.7: Evaluation of a subset of triples in the TKB.

#	Category (rows 1–5)	# Evalu- ated triples	Accuracy
1.	Relations that participate in 1 triple	152	71.7% \pm 3.48%
2.	Relations that participate in 2 triples	55	74.5% \pm 5.63%
3.	Relations that participate in 3 triples	35	60% \pm 7.98%
4.	Relations that participate in 4 triples	25	76.0% \pm 8.28%
5.	Relations that participate in 5 triples	31	70.9% \pm 7.87%

Table 2.8: Evaluation of triples participating in rarer relations in TeKnowbase.

OpenIE, Table of Contents and Templates. The accuracy is lower than the accuracy of the top-5 relations because of errors in the extraction using OpenIE and Table of Contents (described later in this section).

The best source of extractions is 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. This heuristic did not work well to identify the correct relation. To illustrate, 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 the unstructured text was found to be 63.2%. Recall that we ensure that the entities are always correct since we filter out triples that 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 analyzing these errors in more detail and improving the accuracy of these extractions in future work.

Taxonomy. We specifically analyzed the taxonomy (all triples consisting of the `typeOf` relation) since this is an important subset of any KB as well as the largest subset in our TKB. As previously mentioned, our taxonomy consists of over 25,000 triples, and our evaluation yielded an accuracy of $99.0\% \pm 0.8\%$ at 95% confidence. The top two sources of these triples were Wikipedia lists and List Hierarchies. Around 2000 distinct classes were identified, including, `file formats` (nearly 800 triples), `programming languages` (nearly 700 triples), etc.

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

Training Set	Word2Vec	Phrase2Vec
Trained without textbooks	57%	61%
Trained with Textbooks	58%	62%

Table 2.9: Accuracy obtained using different models used with NTN

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 learned 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%.

2.6 Evaluation of Usability of TeKnowbase

In this section, we show that TeKnowbase entities and relations can be useful in 2 application settings—classification of technical documents and improving the ranking of academic search. We hasten to note that we are not claiming new algorithms or techniques here, nor are we comparing our methods to the state-of-the-art. We simply wish to demonstrate that using TeKnowbase can result in substantial gains over baselines.

2.6.1 Classification Experiment

[67] 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 StackOverflow⁷.

⁷ stackoverflow.com

2.6.1.1 Setup

StackOverflow is a forum for technical discussions. A page on 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 was taken as the ground truth.

2.6.1.2 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>`.

2.6.1.3 Classification Algorithms

We trained both a Naive-Bayes classifier as well as SVM with each of the feature sets above.

2.6.1.4 Results

We performed 5-fold cross-validation with each classifier and feature set and report the accuracies in Table 2.10. 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 (such as, coming up with heuristics to decide which relations to use) will result in further gains.

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

Table 2.10: Average classification accuracies.

	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

Table 2.11: 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.

2.6.2 Ranking Experiment

The use of a bag-of-entities (BOE) model to represent queries and documents for document retrieval is outlined in [207]. 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 [208] (though their knowledge-base is different in structure

and content to ours). We made use of the data generously provided by [208] to conduct our own ranking experiment with a BOE model.

2.6.2.1 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 Scholar⁸ 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.

2.6.2.2 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 the 1-hop neighborhood in TeKnowbase.

2.6.2.3 Ranking Models

We have used tf-idf [156] and BM-25 [163] ranking models to rank the candidate documents.

⁸<https://www.semanticscholar.org>

2.6.2.4 Results

We calculated NDCG@20 [92] and report the values in Table 2.11. 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 [208] 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.

2.7 Conclusions

In this chapter, we described the construction of TeKnowbase, which is 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 extracting 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 they showed improvement after adding information from online textbooks. There are a lot of improvements that can be made to our system, purely to increase coverage. We used simple techniques, such as surface patterns, to extract relationships from textual sources. We can try more complex, supervised techniques to do the same. In order to extract unknown relationships, we are interested in exploring Open IE and inferencing techniques in more detail, particularly in identifying interesting and uninteresting relationships.

Chapter 3

ASK: Aspect-based Academic Search

The rapid growth of research papers in digital form has motivated the need for academic search engines, such as Google Scholar¹, PubMed², and Semantic Scholar³. These search engines allow researchers across the world to quickly filter through a large corpus of articles. Apart from retrieving documents relevant for the query, these search engines also perform more complex tasks such as recommending new articles based on a query article [34] or providing suggestions that help guide users retrieve relevant documents [100].

A researcher who has been introduced to multiple aspects of a topic would also be willing to explore the latest research on that topic. More specifically, she would be interested in an aspect-based retrieval of relevant documents for a query. An aspect describes some subtopic of the query the user is interested in. In this chapter, we focus on this problem and highlight the challenges faced by existing systems. We first illustrate the problem using motivating examples and then describe our technique.

¹<https://scholar.google.com>

²<https://ncbi.nlm.nih.gov/pubmed/>

³<https://semanticscholar.org>

Table 3.1: Titles of relevant papers for queries along *Application* and *Algorithm* aspect

Query	Aspect	Corresponding Relevant Document
computer_vision	application	On-road vehicle detection using optical sensors: A review
computer_vision	algorithm	Semi-Supervised Ensemble Methods for Computer Vision
genetic_algorithm	application	Multi-Objective Genetic Algorithm for Robust Clustering with Unknown Number of Clusters
genetic_algorithm	algorithm	Novel Hybrid Approaches For Real Coded Genetic Algorithm to Compute the Optimal Control of a Single Stage Hybrid Manufacturing Systems

3.1 Motivation and Problem

An academic search user often has an *aspect* of interest in mind while retrieving documents relevant to a query. An aspect describes a subtopic of the query for which the user wishes to retrieve information. To give an idea, consider the query `computer_vision` and an aspect of interest, say, *application*, as opposed to another aspect of interest, such as *algorithm*. This means that the user is interested in retrieving papers that describe applications of `computer vision`. Therefore, a paper titled “On-road vehicle detection using optical sensors: A review” should be ranked higher by the retrieval system than “Semi-Supervised Ensemble Methods for Computer Vision”, because the former describes an *application* of computer vision, while the latter describes semi-supervised algorithms for computer vision, so would be more relevant for the *algorithm* aspect. Table 3.1 shows some examples of relevant papers for two queries and two aspects.

In the following paragraphs, we demonstrate with examples why this problem is challenging.

Scenario 1: One way of retrieving research papers relevant for the query and the aspect is to enter the terms for the query and the aspect as a single query in an academic search engine. On entering the query `computer vision application` on Google Scholar, a ranked list of results is returned as shown in Figure 3.1.

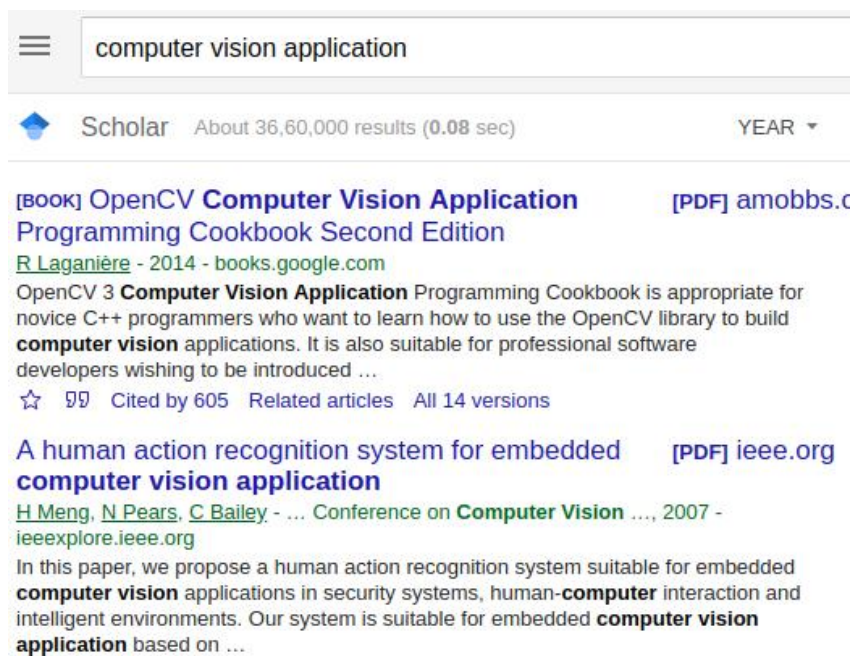


Figure 3.1: Top-2 results retrieved for the query `computer vision application` by google scholar search engine as of 20 July 2020

The result returned at the top position is “OpenCV Computer Vision Application Programming Cookbook Second Edition”. This is a paper that describes a library to implement computer vision applications. However, the document retrieved at the second position “A human action recognition system for embedded computer vision application” is more relevant because it is wholly dedicated to describing an application of computer vision, instead of also talking about its implementation. So the topmost document is not the best document to be suggested for *application* aspect. It was retrieved at the top position because it consists of the term `application` in the title and the abstract. On the

other hand, the document retrieved at the second position is more relevant because of the presence of the term `action recognition system` along with `application`, because `action recognition system` is known to be an application of computer vision. This signifies that the relevance is not determined only by the presence of the aspect term in the document. A number of other terms apart from the aspect term determine the relevance.

Scenario 2: On being unable to retrieve enough relevant documents, the user will look for other options to retrieve relevant documents. She may go through the related queries suggested by the search engine. Figure 3.2 shows the suggestions provided by google scholar for the query `computer vision application`.

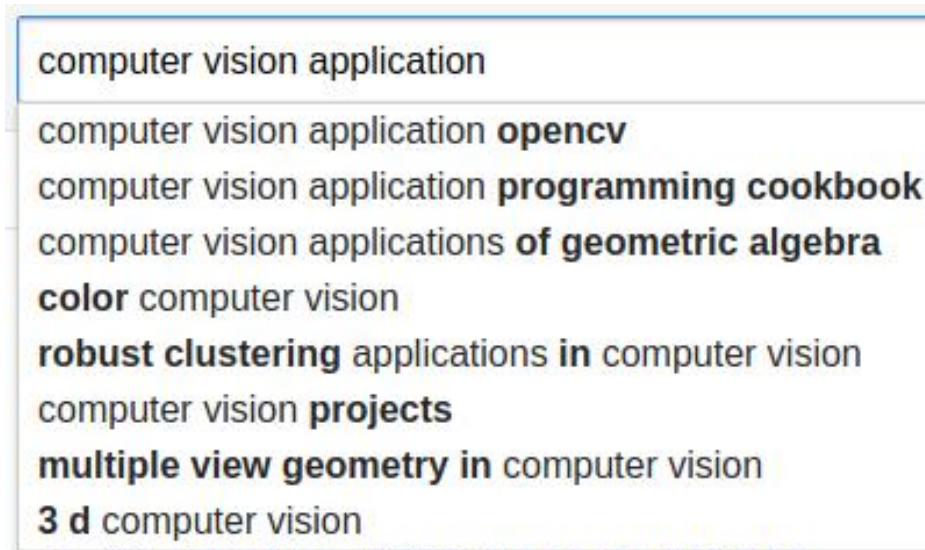
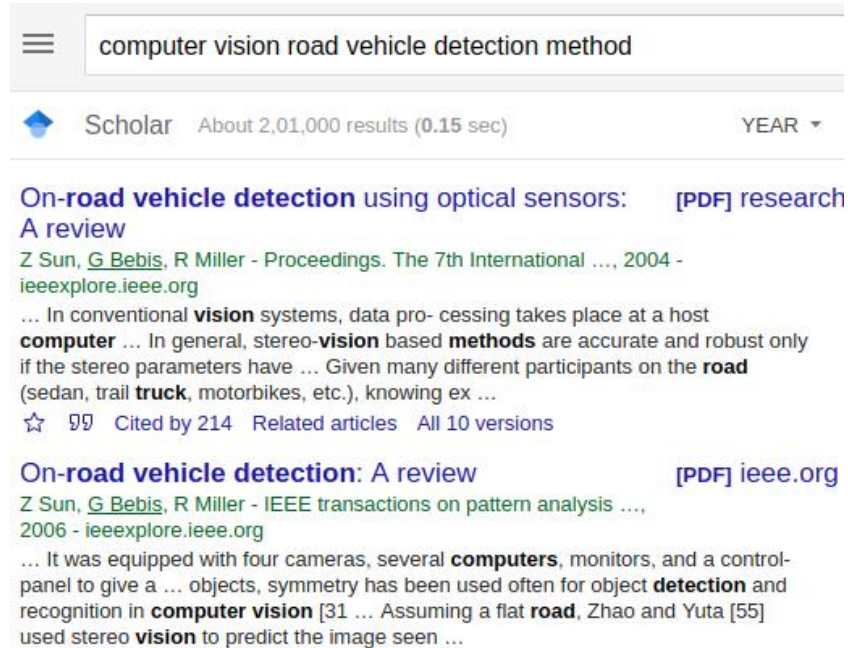


Figure 3.2: Suggestions generated by Google Scholar for the query `computer vision application` as on 20 July 2020

The suggested queries also are unable to retrieve relevant results because they are gener-



The screenshot shows a Google Scholar search interface. At the top, a search bar contains the query "computer vision road vehicle detection method". Below the search bar, it indicates "Scholar About 2,01,000 results (0.15 sec)" and a "YEAR" dropdown menu. The first result is titled "On-road vehicle detection using optical sensors: A review" with a "[PDF] research" link. The authors are listed as "Z Sun, G Bebis, R Miller - Proceedings. The 7th International ... 2004 - ieeexplore.ieee.org". A snippet of the abstract follows: "... In conventional **vision** systems, data processing takes place at a host **computer** ... In general, stereo-**vision** based **methods** are accurate and robust only if the stereo parameters have ... Given many different participants on the **road** (sedan, trail **truck**, motorbikes, etc.), knowing ex ...". Below the snippet are icons for a star, a document, and the text "Cited by 214 Related articles All 10 versions". The second result is titled "On-road vehicle detection: A review" with a "[PDF] ieee.org" link. The authors are "Z Sun, G Bebis, R Miller - IEEE transactions on pattern analysis ..., 2006 - ieeexplore.ieee.org". A snippet of the abstract follows: "... It was equipped with four cameras, several **computers**, monitors, and a control-panel to give a ... objects, symmetry has been used often for object **detection** and recognition in **computer vision** [31 ... Assuming a flat **road**, Zhao and Yuta [55] used stereo **vision** to predict the image seen ...".

Figure 3.3: Top-2 results retrieved for query `computer vision road vehicle detection method` by Google Scholar search engine as of 20 July 2020

ated from the list of documents retrieved for the query `computer vision application`. The first two suggestions—`computer vision application opencv` and `computer vision application programming cookbook` are not the best suggestions for the application aspect. This is because no application of `computer_vision` is mentioned in the suggestion, and expanding the query with this suggestion retrieves “OpenCV Computer Vision Application Programming Cookbook Second Edition”, which is not the best document for the query and the aspect.

Scenario 3: Consider the query `computer vision road vehicle detection method` instead of `computer vision application`. The results for this query retrieved by the Google Scholar search engine are shown in Figure 3.3. The paper retrieved at the top position is

The screenshot shows a Google Scholar search interface. At the top, there is a search bar containing the query "use computer vision based methods". Below the search bar, it indicates "Scholar About 35,00,000 results (0.08 sec)" and a "YEAR" filter. The top two search results are displayed:

- Computer vision based methods for detecting weeds in lawns** [PDF] springer. Authors: U Watchareeruetai, Y Takeuchi, T Matsumoto... - Machine Vision and ..., 2006 - Springer. The abstract snippet reads: "... Based on this assumption, we use only edge images and ignore color information. Consequently, the methods should be robust against the effect of changing light conditions, etc ... Page 3. Computer vision based methods for detecting weeds in lawns 289 Fig ...". It also shows "Cited by 48", "Related articles", and "All 14 versions".
- Vision-based methods for driver monitoring** [PDF] resea. Authors: E Wahlstrom, O Masoud... - Proceedings of the ..., 2003 - ieeexplore.ieee.org. The abstract snippet reads: "Page 1. VISION-BASED METHODS FOR DRIVER MONITORING Eric Wahlstrom, Osama Masoud, and Nikos Papanikolopoulos, Senior Member, IEEE Department of Computer Science and Engineering, University of Minnesota, USA Abstract- This project involves the use of a ...".

Figure 3.4: Top-2 results retrieved for query use computer vision based methods by Google Scholar search engine as of 20 July 2020

“On-road vehicle detection using optical sensors: A review”. Both the results retrieved at the top-2 positions are relevant for *computer vision* and *application* aspect. A common observation for both the documents is that they do not mention the term *application* anywhere in the title, or the abstract. The relevance is determined by the presence of the term *vehicle_detection*, because vehicle detection is a well-known application of computer vision.

Scenario 4: Consider the query use computer vision based methods instead of computer vision application. The results for this query retrieved by Google Scholar search engine is shown in Figure 3.4. Both the results retrieved at the top-2 positions are relevant for *computer vision* and *application* aspect. The term *application* is also not present in both

the documents. In this case, we see that the relevance is determined by the presence of the term `using` or `based` instead of `application`, because both `using` or `based` are terms similar to `application` and both the documents retrieved for the query are relevant.

Problem

The above scenarios demonstrate that the relevance for a query and the aspect is determined not only by the presence of the aspect and query terms but also by terms similar to the aspect as well as terms determined by the relationship between the query and the aspect. Determining the set of relevant terms is challenging, and it would be easier if we had a knowledge base containing information like $\langle \text{road_vehicle_detection}, \text{application}, \text{computer_vision} \rangle$, which would help us determine terms dependent on the query and the aspect. Apart from that, we have to develop techniques to determine terms similar to the aspect as well. So, it is important to model the relevance of the terms dependent on the query and the aspect using a domain-specific knowledge base.

3.1.1 Problem Definition

In this section, we will formally define some key terms related to the problem.

Definition 7. Query. *Users express their information need as strings of words called queries. In the context of this problem, a query q is an entity in the domain of Computer Science.*

Definition 8. Aspect. *An aspect a is a keyword that describes a sub-topic of the query q for which the user wishes to retrieve documents. Providing an aspect along with the query specifies the set of relevant documents. For instance, as shown in Table (3.1), for the query `genetic algorithm`, a paper titled “Multi-Objective Genetic Algorithm for Robust Clustering with Unknown Number of Clusters” is relevant for the **application** aspect. An aspect can be represented by a relationship or entity in a technical knowledge base. Note that a document can be relevant for multiple aspects of the same query.*

3.2 Approach and Contributions

In this chapter, we develop a new retrieval model to return relevant documents for a query and an aspect. We need a new model because simply expanding the query with the aspect term and using standard retrieval techniques such as query likelihood will assign a high relevance score to documents that contain the query and the aspect term both, which may not be the best document to recommend. Using techniques like pseudo-relevance feedback [112] will provide a set of terms for query expansion to retrieve better results, but the terms are not generated by considering the relationship between the query and the aspect. While the use of aspects in the retrieval of documents has been widely used in faceted search [82, 34] and diversification of search results [165, 12], they address a fundamentally different problem than ours. We take a query and an aspect as input and retrieve documents relevant for both, while these techniques take only the query as an input to their system. After that, they identify important aspects for the query so that the underlying problem of ambiguous queries or diversification of results is solved. On the other hand, we address a novel problem that takes a query and an aspect as input and retrieves relevant documents for the same.

We use the idea of language models [153] for the same. It assumes that the document and the query are samples of their underlying probability distributions. If we have an estimate of these distributions, then we can rank the documents based on the similarity of their language models with the language models estimated for the document and the query. In our case, we have two inputs to the system, a query, and an aspect. So, we have to estimate the language model for both. The language model for the aspect models the set of terms relevant for the aspect, such as `using` or `based`. The language model for the query models the terms relevant for the query and aspect using TeKnowbase. A language model for the query and the aspect is estimated by a mixture of both distributions.

Figure 3.5 describes the components of our system. There are 3 main components. After the user enters a query and an aspect, the query suggestion component suggests alternative queries for the query and the aspect to the user. The user can choose one of the

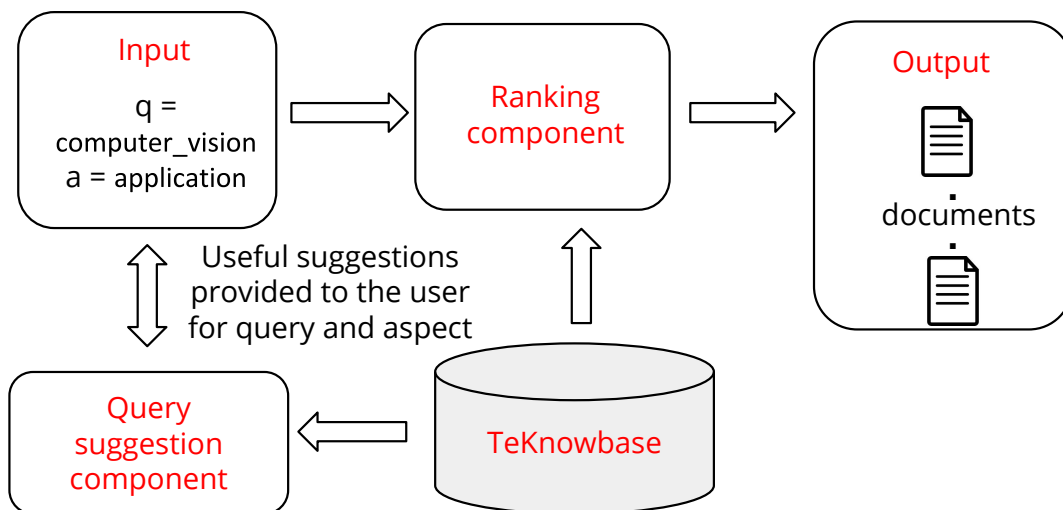


Figure 3.5: Architecture of ASK. The input consists of the query (q) and an aspect (a). The query suggestion component suggests alternative queries for the query and the aspect to retrieve documents. The ranking component returns a ranked list of documents for the query and the aspect. Both of these components use a domain-specific knowledge base in the background

suggested queries to retrieve a relevant document. The second component is the ranking component which returns a ranked list of documents relevant for the query and the aspect. The suggestions and the ranking of documents is done using an aspect-based retrieval model (described in Section 3.4.2)

Contributions. We summarize our main contributions below:

1. We introduce the novel task of aspect-based retrieval of research papers for the query.
2. We develop ASK, a novel language model-based technique to solve the task of

aspect-based retrieval using TeKnowbase.

3. Devised rules to determine the relevance of a document for a query and an aspect.

Organization. The rest of the chapter is organized as follows—Section 3.3 describes the related work. Section 3.4 describes our retrieval model that makes use of TeKnowbase to estimate the language model for the query and the aspect. Section 3.5 describes the experiments, evaluation scheme, results, and discussion. Finally, we conclude in Section 3.6.

3.3 Related Work

In this section, we describe related work in the area of aspect-based document retrieval and query suggestion.

3.3.1 Aspect-based Document Retrieval

Aspect-based retrieval or faceted retrieval allows users to retrieve documents along different dimensions. These dimensions can be as simple and known beforehand (such as time or venue) or could be complicated relationships between the query and the document (that have to be generated based on the query). For instance, while searching for news articles, one can refine the search either by time or venue. Aspect-based search has been explored for both regular and academic searches. A summary of related contributions is presented below:

3.3.1.1 Regular Search

In the context of regular search, several systems providing faceted search have been built for many years. In [82], the author proposed design interfaces for hierarchical faceted

search. [187] addressing the vocabulary problem for faceted search. [103] developed a personalized faceted retrieval system. In all of these systems, the facets are predefined properties directly mapped to the document. In our case, we do not know the mapping between the facet value and the document. The retrieval model that we propose returns a ranked list of documents given a query and facet/aspect as input.

In many cases, the standard facets/ aspects are not enough to provide a great user experience [202]. In such scenarios, they have to be automatically generated. [202, 76, 94] proposed methods to *generate* aspects for queries. While it primarily deals with determining how good the generated aspects are with respect to correctness and novelty, we are interested in retrieving relevant documents given a query and an aspect. We can use these techniques to generate aspects for our setting and then use our model to retrieve documents relevant to the generated aspects.

Aspects have been represented by subsets of query words. To illustrate, the query `Microsoft Office 2007 reviews` has two aspects, `Microsoft Office 2007` and `reviews`. Aspects have been used for diversifying search results [12], [165], [48], [47]. They have been used to handle results when query can have multiple interpretations, such as the query `java` which has two interpretations—`programming_language` and `island`. For ambiguous queries, [12] identified categories for the query and the document according to a taxonomy. However, all queries in our case consist of concepts, and there is hardly any need for disambiguation. We are more interested in exploring the *relationship* of the query words with the words in the document. [165] used query reformulations suggested by existing search engines as sub-queries to retrieve diverse results. [48] used query subtopics as aspects. [47] proposed an algorithm to automatically determine topic terms to diversify search results.

3.3.1.2 Academic Search

In the context of academic search, research paper recommendation has been thoroughly studied and a number of systems have been developed for the same. These include recom-

mendation tools that retrieve papers based on the user’s interest [180, 113] or similar to a given set of papers [155]. [70] built *Scienstein* that allows users to search for papers using authors or reference lists apart from the usual keyword-query search. These tools make use of the underlying structure of the citation graph as well as machine learning techniques on the document text. [52] made an effort to provide faceted retrieval for research papers in computer science. The facets were—publication years, authors, or conferences. These facets are different from aspects in our scenario. [34] proposed techniques to further categorize the relationships between the query paper and the recommended paper. These relationships are expressed in the form of facets like `background`, `alternative_approaches`, `methods` and `comparisons`. [42] used similar facets to summarize scientific papers. These facets are extracted by identifying the context of the text surrounding the citation. Although these facets are similar to aspects in our problem, they describe the relationship between two papers, unlike our case where the relationship has to be identified between a keyword query and the paper.

3.3.2 Query Suggestion

The purpose of query suggestion is to recommend a list of related queries to the entered query usually done using query logs or search results. These fall into two major categories—1) query auto-completion, where the suggested queries start with the prefix entered by the user, 2) query recommendation, where the suggestion need not necessarily start with the entered prefix, but is related to the entered query. A summary of these techniques in both regular and academic search is presented below:

3.3.2.1 Query Suggestion using Query Logs

Most of the large-scale search engines rely on query logs to generate query suggestions. In [18, 93], authors cluster query logs to find the most similar suggestion for the query. In [21], authors extract frequently occurring phrases from query logs and suggest them

to the user. [69] proposed a retrieval model to generate suggestions for task-based search by using sub-tasks for the entered query as suggestions. The probabilities are estimated using data from query logs and suggestions generated by popular search engines. [151] proposed a neural model to generate suggestions for queries that do not occur in the query log. [53] developed a model to suggest queries using query logs, suggestions offered by popular search engines and community-created knowledge bases. Apart from only using query logs, session information has been used by [31], [177], [68] to generate suggestions. Modifications done to the original query by the user to obtain relevant information in a single session have been utilised by [118] and [97] to generate suggestions. In [44], authors identify pages that satisfy user's need after multiple reformulations and use them to generate query suggestions. Query flow graphs and query-URL bipartite graphs (generated from query logs) have also been used to generate suggestions in [27], [132] and [128]. The main data source in all of the above techniques is obtained from query logs, which makes them unsuitable to be applied for our task.

3.3.2.2 Query Suggestion without Query Logs

In [23], [22], authors develop an instant search system that computes and presents search results online and refreshes the results with each letter typed or modified by the user. [25] developed a probabilistic retrieval model to suggest queries in the absence of query logs using key phrases extracted from relevant documents. All of these techniques provide query auto-completion, where the suggestions start with the prefix already entered by the user. To use these techniques, we have to reformulate our query by adding the aspect term to it. Doing this drastically reduces the number of candidate suggestions starting with the reformulated query, so these techniques cannot be applied and we need a new technique to retrieve relevant query suggestions for a query and aspect.

3.3.2.3 Query Suggestion in Academic Search

Not a lot of work has been done to generate query suggestions for academic search. Efforts to suggest queries for professional search have been made in [101] where the authors suggest a list of ranked boolean queries for a topic query entered by the user. While the approach is interesting, suggesting boolean queries is not a good idea as users prefer keyword query suggestions over boolean suggestions in academic search [37]. In [195], authors develop a suggestion mechanism for academic queries to establish the importance of query suggestions in academic search. They use query logs as the data source and simulate a user-interaction model to generate suggestions. Again, the absence of query logs makes it unsuitable to be adapted in our setting.

To the best of our knowledge, aspect-based retrieval for academic search is a novel problem. **ASK** solves this task by estimating a language model for a given query and aspect using TeKnowbase. The relationships in TeKnowbase are used to represent aspects and an inferencing technique using meta-paths has been proposed to address the sparsity of relations in TeKnowbase.

3.4 Aspect based Retrieval

We introduce the novel problem of aspect-based search and develop ASK (Aspect-based academic Search using domain-specific KBs) to solve it. ASK takes a query and an aspect as input and assists the users by providing useful query suggestions for the query and the aspect. It then retrieves a ranked list of relevant documents for the query and the aspect. ASK uses TeKnowbase as the source of domain knowledge. We now give an overview of the main components of ASK as follows:

1. **Input.** ASK takes a query and an aspect as input. Both these concepts have been defined in Section 3.1.1.

2. **Domain-Specific Knowledge Base.** ASK uses TeKnowbase [191, 192] as the domain-specific knowledge base to assist in suggestion of queries and retrieval of documents relevant to the query and the aspect provided as input. TeKnowbase consists of entities such as `nearest_neighbor_search`, `robotics`, or `boosting`, and taxonomic as well as other domain-specific relationships like `application`, `implementation`, or `algorithm`. Examples of triples in TeKnowbase are: $\langle \text{robotics}, \text{application}, \text{nearest_neighbor_search} \rangle$ and $\langle \text{tiny_encryption_algorithm}, \text{algorithm}, \text{block_cipher} \rangle$. The triple $\langle \text{robotics}, \text{application}, \text{nearest_neighbor_search} \rangle$ conveys information that `robotics` is an application of `nearest_neighbor_search`. This also means that the terms appearing in entities connected via a relationship type in TeKnowbase have a higher probability of appearing in documents relevant for the aspect described by that relationship. We use this idea to determine the relevance of the document for the query and the aspect.
3. **Query Suggestion.** To assist a user in retrieving relevant documents, ASK suggests alternate queries for the query and the aspect. The query suggestion component has two parts 1) candidate suggestion generation and 2) ranking of suggestions. The procedure of generating and ranking query suggestions is described in Section 3.4.4. These are ranked using a language model [153] estimated for the query and the aspect using TeKnowbase which is described in detail in Section 3.4.2.
4. **Ranking.** Given a query, the documents are ranked using retrieval models [171]. In this work, our idea is to estimate a language model [153] for each aspect and rank the documents in the corpus using the language model estimated for the query and the aspect. Section 3.4.2 describes our language model for the query and the aspect. The procedure of ranking the documents using this model is described in Section 3.4.5.
5. **Output.** The system returns a ranked list of relevant documents for the query and

LM for “Movies”		LM for “Fruits”	
Term	Probability	Term	Probability
movie	0.3	fruit	0.33
cinema	0.21	apple	0.18
director	0.23	plant	0.2
script	0.25	nutrition	0.3
apple	0.002	building	0.0015
seeds	0.005	movie	0.0008
...

Table 3.2: Example of Language Models for two documents about “Movies” and “Fruits”

the aspect.

3.4.1 Language Models

Language models are used to model unstructured documents [153]. They assign some probabilities to a set of terms or words. Each document d (which is a collection of terms) is assumed to be a sample from such a language model M_d . To illustrate, a document about “Movies” will have a high probability for terms such as `movie`, `cinema` or `director` as compared to another term such as `apple`. On the other hand, a document about “Fruits” will have a high probability for terms such as `apple` or `plant`. Table 3.2 shows language models for two hypothetical documents about Movies and Fruits.

The true language model associated with a document cannot be estimated since the document is only a sample from it. So, we can use the Maximum Likelihood Estimation (MLE) approach to estimate the language model such that the seen document’s likelihood is maximized. The probability of each term in the document can be estimated using the term frequencies in the document using the following equation:

$$P(w|M_d) = \frac{tf(w, d)}{length(d)} \quad (3.1)$$

where $tf(w, d)$ is the frequency of term w in document d and $length(d)$ is the total number of terms present in document d . However, this probability distribution assigns zero probability to terms not present in the document. This is not accurate because the actual LM for the document can have a non-zero probability assigned to terms not present in the document. So, smoothing techniques such as Dirichlet Smoothing [217] (Equation 3.2) are used to smoothen this distribution.

$$P(w|M_d) = \frac{tf(w, d) + \mu P(w|C)}{length(d) + \mu} \quad (3.2)$$

where $P(w|C)$ is the probability of term w in the entire collection and μ is a Dirichlet smoothing parameter. It is generally set to 1500. The key idea of Dirichlet smoothing is to increase the document's length by μ number of terms generated according to the background probability distribution.

Query Likelihood. Using language models, Ponte and Croft introduced query likelihood techniques to model the relevance of a document d for a query q [153]. Query likelihood estimates a language model for each document in the collection and ranks them by the likelihood of seeing the query terms as a random sample given that document model. The probability $P(d|q)$ of a document being relevant to the query q can be written as follows using the Bayes rule:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \quad (3.3)$$

$P(q)$ is the same for all documents, so this quantity can be ignored. $P(d)$ is the prior probability which can be assumed to be the same for all the documents, independent of the query. This leads us to the following equation:

$$P(d|q) \propto P(q|d) \quad (3.4)$$

The probability that d is relevant for q is proportional to the probability of observing the query as a random sample from the document's language model. The document's language model is estimated as follows:

$$M_d(w) = \frac{tf(w, d) + \mu P(w|C)}{|d| + \mu} \quad (3.5)$$

where $tf(w, d)$ is the frequency of a term in d . This distribution is further smoothed using Dirichlet Smoothing. Because it estimates from document length, estimates from short documents and long documents are comparable.

Given a document model M_d , $P(q|d)$ is estimated as follows:

$$P(q|d) = \prod_{w \in terms(q)} M_d(w) \quad (3.6)$$

where $terms(q)$ is the set of terms present in q . The order of terms in the query becomes irrelevant and it is treated as a bag of words/terms, and the probability of observing a term is independent of observing another term. The documents are then ranked in decreasing order of $P(q|d)$.

3.4.2 Aspect-based Retrieval Model

We use language models to solve the problem of aspect-based retrieval of documents as well as query suggestions. As already described in Section 3.4.1, using query likelihood techniques where the query has been augmented with the aspect term will assign a high relevance score to documents containing the query and the aspect term, which may not lead to the best documents at the top positions. Pseudo-relevance techniques model the probability of terms from the list of top-k documents retrieved for the query, assuming that these top-k documents are highly relevant. The terms that are assigned the highest probability are chosen to be expanded to the query. These techniques do not consider the relationship between the query and the aspect to model the probability of the set of

terms, due to which the best documents might not be returned. We need to model the probability of terms considering the relationship between the query and the aspect also.

To address these issues, we propose the approach for re-ranking the top-k documents relevant for the query (retrieved using the query likelihood method). We model the relevance of these top-k documents using an aspect-dependent probability (independent of the query) and a query and aspect-dependent probability. These two components are described as follows:

1. Aspect Dependent Prior Probability. It is important to model the relevance of terms for the aspect. As already pointed out in Section 3.1, there are certain terms apart from the term describing the aspect that are likely to appear in a document relevant for the aspect, independent of the query. In the case of *application* aspect, presence of terms like *usage* or *using* indicates that the document is relevant for the *application* aspect. The probability of a term w appearing in d given an aspect a , independent of the query, is called the prior. It is denoted by $P(w|a)$.

2. Query and Aspect Dependent Probability. Apart from modeling the relevance of the terms for the aspect, there are terms determined by the query and the aspect together. In the example of *genetic_algorithm* and *application* aspect, introduced in Section 1, *clustering* is a term highly relevant for *genetic_algorithm* and *application* aspect. This is because clustering is known to be an application of the genetic algorithm. So, we model this as the probability of a term appearing in d given a query q and aspect a which is denoted by $P(w|q, a)$.

3. Mixture of the Two Probability Distributions. The final relevance of a term is modeled by a mixture of the two probabilities. Equation (3.7) describes our probability distribution. It is denoted as MM .

$$MM(w) = \lambda P(w|a) + (1 - \lambda)P(w|q, a) \quad (3.7)$$

Our mixture probability distribution is used to score the query suggestions as well as documents relevant for the query and the aspect, described in Section 3.4.4 and 3.4.5, respectively. We describe the estimation of the components of our mixture model in Section 3.4.3.

3.4.3 Estimation of Probabilities

In this subsection, we describe the techniques that we used to estimate the mixture model described by Equation 3.7. We made use of the Open Research Corpus and algorithms already implemented on Galago to estimate these probabilities. We later conducted our experiments on the same dataset (described in Section 3.5). We now describe the estimation of the query dependent and the query independent component as follows.

3.4.3.1 Estimation of Query-Independent Component

$P(w|a)$ models the relevance of the term w for an aspect a as described in Section 1.. This probability can be estimated from a set of relevant documents for this aspect. To acquire a set of relevant documents for the aspect, we first conducted a user study with Computer Science researchers (not directly related to the project) and were able to collect 500 relevant documents for each aspect. However, the estimation from a small set of documents of size 500 would be inaccurate. So, we used heuristics to collect more documents relevant to the aspect. These two techniques are described in detail as follows:

- 1. User Study to Collect Relevant Documents for the Aspect.** We describe the steps of the user study experiment as follows:

1. We first chose 10 queries (entities from the domain of Computer Science). The queries were chosen based on the users' familiarity with them.
2. We also chose 3 aspects, *application*, *algorithm*, and *implementation*. The annotation task involved determining the aspect label for a query and a document out of these 3 labels. Note that a document can have more than one aspect label or none of these. To determine these labels, we referred to the rules described in Step 4.
3. The next step involved determining the candidate set of documents to be evaluated by the user for each query. We obtained this set by choosing the top-10 documents retrieved for the query augmented with terms for aspect i.e. "query + aspect". We then showed the users the query and a candidate document and asked them to determine the relevance of the document for the query and all the aspects.
4. Here we describe the rules and the questions that were asked to the users to determine the 3 different aspect labels for a document and the query.
 - (a) **Application.** If the queried entity is used as a solution in the *problem* addressed by the paper, then the paper is relevant for the application aspect for the query, and a score of 2 is assigned. If not, a score of 0 is assigned. For instance, for the query `autoencoder`, the paper "Exploring autoencoders for unsupervised feature selection" addresses the *problem* of feature selection *using* autoencoders. So, autoencoders solve the problem of feature selection and this paper is relevant for the application aspect for `autoencoder`. On the other hand, the paper "Parallelizing the Sparse Autoencoder" addresses the challenges faced in the fast training of autoencoders. So, `autoencoder` is not used to solve the problem addressed by the paper and it gets a score of 0. There can be cases when a query belongs to the problem and solution both, such as, in the paper "Deep auto-encoder neural networks in reinforcement learning", a new framework is proposed to jointly train deep autoencoders with RL algorithms. In such cases, a score of 1 is assigned. Table 3.3 summarizes these rules. The documents that were assigned a score of 1 or 2 were considered relevant for the **application** aspect.

Score	Criteria	Example
2	i) Queried topic is used to solve the problem proposed in the paper, iii) queried topic is not a part of the problem	Exploring autoencoders for unsupervised feature selection
1	i) Queried topic is used to solve the problem, ii) Queried topic is a part of the problem	Deep auto-encoder neural networks in reinforcement learning
0	i) Queried topic does not solve the problem	Parallelizing the Sparse Autoencoder

Table 3.3: Set of rules to determine the scores for papers relevant for the query autoencoder and application aspect

- (b) **Algorithm.** The first step in identifying if a paper is relevant for the algorithm aspect is to identify whether it is/is not relevant for the application aspect for the query. This is required because an algorithm proposed to solve a different problem using the queried entity should be given less score than an algorithm that addresses a problem for the query. Given a paper “New hybrid evolutionary algorithm for solving the bounded diameter minimum spanning tree problem”, it proposes a new algorithm for solving a variant of the minimum spanning tree problem. So, this should be given a higher score than a paper “Image registration with MST algorithm”, which proposes an algorithm for image registration using a minimum spanning tree. So, if the problem addressed by the paper is related to the query and an algorithm is proposed as the solution, then a score of 3 is assigned. If the query is used in the solution and an algorithm is proposed in the paper, then a score of 2 is assigned. If an algorithm is used/improved but not proposed in the paper, then a score of 1 is assigned. In other cases, 0 is assigned. Table 3.4 summaries these rules. The documents that were assigned a score of 1, 2 or 3 were considered relevant for the **algorithm** aspect.
- (c) **Implementation.** We first determine if the paper is relevant for the applica-

Score	Criteria	Example
3	i) Queried topic does not solve the problem proposed in the paper, ii) an algorithm is the novel contribution of the paper	New hybrid evolutionary algorithm for solving the bounded diameter minimum spanning tree problem
2	i) Queried topic is used to solve the problem, ii) an algorithm is the novel contribution of the paper	Image registration with MST algorithm
1	i) some algorithm is used/improved but not proposed in the paper	A weighted coding in a genetic algorithm for the degree constrained minimum spanning tree problem
0	None of the above	The minimum spanning tree: An unbiased method for brain network analysis

Table 3.4: Set of rules to determine the scores for papers relevant for the query `minimum spanning tree` and `algorithm` aspect

tion aspect of the query. If not, and a solution is proposed that deals with one of the following—1) improving the run-time/space complexity or 2) building a software/prototype implementation/hardware implementation, then a score of 2 is assigned. If the paper is relevant for the application aspect and the solution deals with one of the above-mentioned cases, then a score of 1 is assigned. In other cases, 0 is assigned. To illustrate, “Genetic Ant Algorithm for Continuous Function Optimization and Its MATLAB Implementation” gets a score of 2 for `genetic_algorithm`, because the paper is not relevant for the application aspect for genetic algorithm and it also describes a MATLAB implementation *of* genetic algorithm. The paper “A Novel Approach towards FPGA Implementation of a Multiple Parameter Optimization Technique using Genetic Algorithm” is scored 1 because it is also relevant for the application aspect. The genetic algorithm solves the problem of implementing multiple parameter optimization techniques on FPGA, and because it deals with implementations on hardware, it is relevant for the implementation aspect. Table 3.5 summarizes these rules. The documents that were assigned a score of 1 or 2 were considered relevant for the **implementation** aspect.

2. Targeted Search for an Aspect with Filtering. Obtaining annotations for documents is a difficult task and only using 500 annotated documents for each aspect may lead to the estimation of an inaccurate model. In order to increase the size of our annotations, we used heuristics to collect more documents given an aspect. We formulated a query containing only the aspect as a keyword and retrieved documents for it using the query likelihood model. For instance, for the *application* aspect, we collected all the documents returned for the query “application”. However, on manual inspection, we found that all of these documents were not papers relevant for the application aspect but only contained the term “application” in the title or abstract. So, we retained only those documents which contained the phrase “application of” in the title on the intuition that such documents describe some application. Similarly, we used phrases like “algorithm for” and “implementation of” to filter the set of documents relevant for the algorithm and

Score	Criteria	Example
2	i) Queried topic is not used to solve the problem, ii) proposed technique deals with one of the following—1) improving the run-time/space complexity or 2) building a software/prototype implementation/hardware implementation	Genetic Ant Algorithm for Continuous Function Optimization and Its MATLAB Implementation
1	i) Queried topic is used to solve a different problem, ii) proposed technique deals with one of the following—1) improving the run-time/space complexity or 2) building a software/prototype implementation/hardware implementation	“A Novel Approach towards FPGA Implementation of a Multiple Parameter Optimization Technique using Genetic Algorithm”
0	None of the above	A genetic algorithm tutorial

Table 3.5: Set of rules to determine the scores for papers relevant for the query `genetic algorithm` and `implementation` aspect

the implementation aspect, respectively. While this filtering technique returns the set of documents relevant for the aspect with good accuracy, using this technique with query and aspect will still not consider the relationship between them, and the results may not be relevant.

Having a set of ground-truth documents D for each a , we estimated $P(w|a)$ as follows:

$$P(w|a) = \frac{1}{|D|} \sum_{d \in D} \frac{tf(w, d)}{\sum_{w' \in d} tf(w', d)} \quad (3.8)$$

where $tf(w, d)$ is the frequency of the term in document d . The probability of a term w is proportional to its frequency of occurrence in the document set D .

3.4.3.2 Estimation of Query-Dependent Component

To determine the relevance of a term for a query and the aspect, we used relations in TeKnowbase. The relationships in TeKnowbase can represent the aspect that our system takes as input. Note that any relation in TeKnowbase can be used as an aspect in our setting. Let R_a denote the relation in TeKnowbase representing the aspect a . As already described in Section 2, the terms in entities connected to R_a from q have a higher probability of appearing in documents relevant for the aspect a , such as, given the triple $\langle \text{clustering}, \text{application}, \text{genetic_algorithm} \rangle$, the presence of `clustering` in a document indicates high relevance for the query `genetic_algorithm` and *application* aspect. However, TeKnowbase is sparse, so, we need some mechanism to predict entities participating in relation R_a with q to improve the estimation of terms. To do so, we propose a link prediction technique based on two well-known algorithms.

1. **Path Ranking Algorithm (PRA)** [111] estimates the probability of reaching an entity e from a query q in the knowledge base by doing random walks over paths connecting q and e . It uses this probability to compute a final score for e being related to q via relation R_a .

2. **MetaPath2Vec** [55] is an embedding based technique, which computes vector representations for entities given a path such that entities that are likely to be connected by that path are given vector representations closer to each other and those unlikely to be connected are given vector representations farther away from each other. It also considers those entities that are not connected via any path to the query, unlike PRA.

The reason for using these two algorithms is to include both direct and indirect inferencing of relations. Direct inferencing is achieved using PRA since it only considers those entities as candidates which are connected via some path to the query. The indirect inferencing is achieved using MetaPath2Vec since all entities, irrespective of whether they are connected via any path to the query are considered as candidates. A linear combination of the scores returned by these two algorithms is then used to finally score the entities as potential candidates in participating in relation R_a with q . Both these algorithms use the idea of meta-paths [182]. A meta-path is formally defined as follows:

Definition 9. Meta-path. A meta-path \mathcal{P} [111] between nodes v_1 and v_{l+1} in a knowledge base is a path defined as a sequence of edges or walks denoting the sequence $v_1 \xrightarrow{R_1} v_2 \xrightarrow{R_2} \dots \xrightarrow{R_l} v_{l+1}$. So, $\mathcal{P} = R_1 \circ R_2 \circ \dots \circ R_l$ denotes a composite relation between v_1 and v_{l+1} .

Figure 3.6 (a) shows a meta-path $\mathcal{P} = \text{application} \circ \text{type} \circ \text{type_inverse}$ between nodes `robotics` and `incremental_heuristic_search`. Both the entities also participate in the triple $\langle \text{robotics}, \text{application}, \text{incremental_heuristic_search} \rangle$. This implies that the meta-path described by $\text{application} \circ \text{type} \circ \text{type_inverse}$ is a potential representative of `application` relationship in TeKnowbase. In Figure 3.6 (b), `clustering` and `genetic_algorithm` are not connected via any relation. However, the same meta-path $\text{application} \circ \text{type} \circ \text{type_inverse}$ exists between `clustering` and `genetic_algorithm`. So, the triple $\langle \text{clustering}, \text{application}, \text{genetic_algorithm} \rangle$ can be inferred using this meta-path.

We now describe the key concepts related to the Path Ranking Algorithm from [111] and

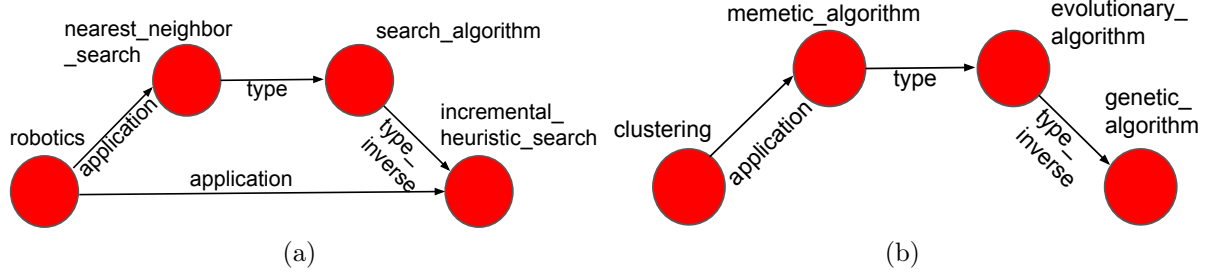


Figure 3.6: Examples of meta-paths for application relation. (a) Meta-path between `robotics` and `incremental_heuristic_search`. The meta-path $\mathcal{P} = \text{application} \circ \text{type} \circ \text{type_inverse}$ exists between them and they are also connected by application relation. (b) Meta-path between `clustering` and `genetic_algorithm`. `clustering` and `genetic_algorithm` are not related by application relation but still it can be inferred because of the existence of the same meta-path $\mathcal{P} = \text{application} \circ \text{type} \circ \text{type_inverse}$ between them

MetaPath2Vec algorithm from [55].

Path Ranking Algorithm (PRA). Given a meta-path $\mathcal{P} = R_1 \circ R_2 \circ \dots \circ R_l$, the domain of \mathcal{P} is defined as follows:

$$\text{domain}(\mathcal{P}) = \{e : R_1(e, e') = 1\} \quad (3.9)$$

Similarly, range of \mathcal{P} is defined as:

$$\text{range}(\mathcal{P}) = \{e' : R_l(e, e') = 1\} \quad (3.10)$$

where $R(e', e) = 1$ if there exists an edge with type R that connects e to e' in the knowledge base and is 0 otherwise. $R_1 R_2 \dots R_l$ are relationships labels found in a knowledge base.

Given a source node e_i and a meta-path \mathcal{P} , a path-constrained random walk defines a

Algorithm 1: Algorithm to generate the set of meta-paths for a relation R_a

Input: R_a : the relation representing the aspect a , TKB
Output: MP , $count$

```

1 // Lines 1-13: Extract the set of meta-paths and count their frequency for a
  given relation  $R_a$  from  $TKB$ 
2  $MP = \emptyset$ 
3 foreach  $(e_1, e_2)$  s.t.  $R(e_1, e_2) = 1$  do
4   foreach  $i \in 2$  to  $l$  do
5     foreach  $\mathcal{P} = R_1 \circ R_2 \circ \dots \circ R_i$  s.t.  $R_1(e_1, a_2) \wedge R_2(a_2, a_3) \wedge \dots \wedge R_i(a_i, e_2)$ 
6       do
7         if  $\mathcal{P} \in MP$  then
8            $count(\mathcal{P}) = count(\mathcal{P}) + 1$ 
9         else
10           $count(\mathcal{P}) = 1$ 
11           $MP = MP \cup \{\mathcal{P}\}$ 
12        end
13   end
14 return  $MP$ ,  $count$ 

```

probability distribution as follows. If \mathcal{P} is an empty path, then

$$h_{e_i, \mathcal{P}}(e_j) = \begin{cases} 1, & \text{if } e_i = e_j \\ 0 & \text{otherwise} \end{cases} \quad (3.11)$$

For a non-empty path, it is defined recursively as follows:

$$h_{e_i, \mathcal{P}}(e_j) = \sum_{e' \in range(\mathcal{P}')} h_{e_i, \mathcal{P}'}(e') \cdot P(e_j | e'; R_l) \quad (3.12)$$

where $P(e_j | e'; R_l) = \frac{R_l(e', e_j)}{R_l(e', \cdot)}$ is the probability of reaching node e_j from node e' with a one step random walk with edge type R_l .

Since PRA is an algorithm for inferring the presence of a given relation R_a between pairs of entities in a knowledge base using meta-paths, it first has to identify the set of useful meta-paths for the relation R_a . Algorithm 1 shows the steps to extract and score meta-paths useful for a relation R_a . It takes the length l of the path as a parameter, and extracts meta-paths of length at most l . The main step in the algorithm involves iterating over all pairs (e_i, e_j) of entities such that $R_a(e_i, e_j) = 1$, and extracting meta-paths of length up to l between them.

Given a set of paths MP extracted for a relation R_a , they are treated as features of a linear model and the strength of the existence of a relationship between e_i and e_j is computed as follows:

$$score_{e_i}(e_j) = \theta_1 h_{e_i, \mathcal{P}_1}(e_j) + \theta_2 h_{e_i, \mathcal{P}_2}(e_j) + \dots + \theta_n h_{e_i, \mathcal{P}_3}(e_j) \quad (3.13)$$

where the θ_i 's are the weights that are learned over all such relation paths.

MetaPath2Vec. MetaPath2Vec [55] is an embedding-based technique for link prediction in heterogeneous graphs. It introduces the heterogeneous skip-gram model, which is modeled after the skip-gram model [135] introduced by Mikolov for the Word2Vec algorithm. The heterogeneous skip-gram model is used to model the neighborhood of a node in a heterogeneous graph using meta-path-based random walks. Given a meta-path \mathcal{P} , the algorithm assigns closer vector representation to entities highly likely to be connected by \mathcal{P} and farther representations to pairs to entities that are unlikely to be connected by \mathcal{P} .

We selected the top-k meta-paths based on their frequency of occurrences as input to metapath2vec and obtained vector representations V_e for every entity e . This is described in lines 1–2 of Algorithm 2.

Algorithm 3 describes the steps to estimate probability of an entity e being related to query entity e_q via the relation R_a . It takes as input MP , the set of meta-paths extracted using Algorithm 1, *count*, the data structure storing the frequencies of the meta-paths

in MP , V , the vector representations obtained for entities by training MetaPath2Vec algorithm using Algorithm 2, and E , the set of entities in TKB . It returns a set of probabilities $P(E|q)$, which is the probability of each entity e being related to e_q via relation R_a . The various components of the algorithm are described as follows:

Algorithm 2: Algorithm to generate vector representations V

Input: MP : the set of meta-paths extracted for relation R_a , $count$, TKB , k

Output: V

- 1 top_k = Top k meta-paths in MP scored according to $count$
 - 2 V = Embedding representations for entities obtained by training MetaPath2Vec algorithm on TKB with top_k paths
 - 3 **return** V
-

1. Direct Inference using Meta-Paths. We follow a more straightforward approach to score the paths. In Algorithm 1, we also maintain a count of the number of times a meta-path is extracted for R_a . If a meta-path is observed more times for a relation R_a , it is considered more useful. So, instead of using a training set of positive and negative relation instances, we directly use the frequency to score the paths. Given a set of meta-paths MP extracted for the relation R_a , PRA assigns scores to entity pairs by using a linear combination of probabilities of reaching the target entity from the source entity via those meta-paths. This score is a measure of the existence of the relation R_a between the entity pair. So, we computed the score between pairs of entities using this approach. Having a set of meta-paths $MP = \mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$, we computed the score for each node reachable from source e_i as follows:

$$score_{e_i}(e_j) = \alpha_1 h_{e_i, \mathcal{P}_1}(e_j) + \alpha_2 h_{e_i, \mathcal{P}_2}(e_j) + \dots + \alpha_n h_{e_i, \mathcal{P}_n}(e_j) \quad (3.14)$$

where each α_i is set to $count(\mathcal{P}_i)$.

Since we have to estimate a probability distribution, we convert $score_{e_i}(e_j)$ to a probability

Algorithm 3: Algorithm to estimate $P_a(E|q)$

Input: E : set of entities in TKB , V : vector representations for entities after training MetaPath2Vec algorithm, MP , $count$

Output: $P_a(E|q)$

```

1 // Compute  $score_{e_q}(e)$  from  $e_q$  to all the entities  $e$  in  $E$ 
2 foreach  $e_1 \in E$  do
3   |   foreach  $\mathcal{P} \in MP$  do
4     |   |    $score_{e_q}(e_2) = score_{e_q}(e_2) + count(\mathcal{P}) * h_{e_q, \mathcal{P}}(e_2)$ 
5     |   |   end
6   |   end
7   |   foreach  $e_1 \in E$  do
8     |   |   // Compute probability using PRA algorithm
9     |   |    $DI_{e_q}(e_1) = \frac{\exp(score_{e_q}(e_1))}{\sum_{e_k \in |E|} \exp(score_{e_q}(e_k))}$ 
10    |   |   // Compute probability using embedding representations obtained using
11    |   |   |   MetaPath2Vec
12    |   |    $h'_{e_q}(e_1) = \frac{\exp(\cosine_{sim}(V(e_q), V(e_1)))}{\sum_{e_k \in |E|} \exp(\cosine_{sim}(V(e_q), V(e_1)))}$ 
13    |   |   // Compute mixture of the two probabilities
14    |   |    $P_a(e_1|e_q) = \beta * DI_{e_q}(e_1) + (1 - \beta) * h'_{e_q}(e_1)$ 
15  |   |   end
16  |   end
17  end
18  return  $P_a(E|q)$ 

```

distribution using softmax as follows:

$$DI_{e_i}(e_j) = \frac{\exp(score_{e_i}(e_j))}{\sum_{e_k \in |E|} \exp(score_{e_i}(e_k))} \quad (3.15)$$

where E is the set of entities in TeKnowbase. The steps for direct inference of relationship are described in lines 2–9 of Algorithm 3.

2. Indirect Inference using Meta-paths. We used the softmax function to convert cosine similarities between entities into a probability distribution as described in Equation

(3.16). This is described in line 11 of Algorithm 3.

$$h'_{e_i}(e_j) = \frac{\exp(\text{cosine_sim}(V(e_i), V(e_j)))}{\sum_{e_k \in |E|} \exp(\text{cosine_sim}(V(e_i), V(e_k)))} \quad (3.16)$$

3. Combining Direct and Indirect Inferencing. The final probability distribution is a mixture of the $DI_{e_i}(e_j)$ and $h'_{e_i}(e_j)$. β is used to mix the two distributions. This is described by Line 13 of Algorithm 3.

$$P_a(e_i|e_j) = \beta * DI_{e_i}(e_j) + (1 - \beta) * h'_{e_i}(e_j) \quad (3.17)$$

4. Final Estimation of Mixture Distribution. Since the documents and query suggestion are represented as bag of words (described in details in Section 3.5.2 and 3.4.5), we defined the distribution over terms instead of entities. We used the following equation:

$$P(w|q, a) = \sum_e P_a(e|e_q), \text{ s.t. } w \in \text{terms}(e) \quad (3.18)$$

where $\text{terms}(e)$ is the set of words present in the entity e . e_q is the entity that q represents. Equation (3.8) estimates the prior probability of the term given only the aspect. Equation (3.18) estimates the probability of a term given the query and the aspect both. The overall probability is a mixture given by $P(w|a)$ and $P(w|q, a)$. λ is used to mix the two distributions. The final distribution, as described at the beginning of the section is given below:

$$MM(w) = \lambda P(w|a) + (1 - \lambda) P(w|q, a) \quad (3.19)$$

The query suggestions and the documents are ranked in increasing order of divergence of the distribution described by Equation (3.19) with their language models, described in detail in Section 3.4.4 and Section 3.4.5, respectively.

3.4.4 Generating Query Suggestions

Query suggestions are either generated from query logs [18, 93] or from the set of relevant documents [25, 53, 69]. In the absence of logs for our system, we used the relevant documents to generate the set of candidate suggestions. The following steps describe the procedure to generate and rank the suggestions given a query and an aspect.

1. Extracting Key Phrases. We extracted key phrases/keywords from the top-1000 documents (retrieved using the query-likelihood model) relevant to the query using the popular keyword extracting algorithm RAKE⁴. RAKE extracts important key phrases from text and also assigns confidence values to them.

2. Determining Candidate Suggestions. However, on manual inspection, we found that not all key phrases were good candidate suggestions. We retrieved, for instance, key phrases such as `k choose or describe generalizes` for the query `latent_dirichlet_allocation`, which do not describe any information about the query. We applied the score assigned by RAKE as a filter and only retained those key phrases whose score was ≥ 5 . Even after applying this filter, we found key-phrases that were too long and assigned high scores by RAKE, such as `multiresolution airport detection via hierarchical reinforcement learning saliency model traditional airport detection methods usually utilize geometric characteristics` from the top-1000 documents for the query `latent_dirichlet_allocation`. This should not be considered as a candidate suggestion. We removed such suggestions by only retaining those key phrases that had at most 5 terms/words. The key phrases retained after applying these two filters form our candidate suggestions.

3. Ranking Suggestions. We estimated language models for each suggestion. Given a suggestion s , a document model M_s is estimated for it. It is described by the following

⁴<https://pypi.org/project/rake-nltk/>

equation. Dirichlet smoothing is used to smoothen M_s [217].

$$M_s(w) = \frac{tf(w, s) + \mu P(w|C)}{length(s) + \mu} \quad (3.20)$$

where $tf(w, s)$ is the frequency of w in s , and $P(w|C)$ is the probability of w appearing in the entire collection. $P(w|C)$ assigns a probability to words not seen in the suggestion to make it accurate. This probability is estimated as being proportional to the general frequency of the word in the entire collection. The relevance of a suggestion for a query and the aspect is modeled by the risk associated with using MM to approximate M_s . This is expressed by KL-divergence between MM and M_s . The suggestions are returned in an order of increasing KL-divergence.

$$KL(MM||M_s) = \sum_w MM(w) \log \frac{MM(w)}{M_s(w)} \quad (3.21)$$

3.4.5 Ranking of Documents

We represent a document as bag of words and estimated a language model M_d for it. Dirichlet smoothing is used to smoothen M_d . It is denoted by the following equation:

$$M_d(w) = \frac{tf(w, d) + \mu P(w|C)}{length(d) + \mu} \quad (3.22)$$

where $tf(w, d)$ is the frequency of w in d , and $P(w|C)$ is the probability of w appearing in the entire collection. We use a risk-minimization framework to rank the documents. The risk associated with using MM to approximate M_d is expressed by KL-divergence between MM and M_d . The documents are returned in an order of increasing KL-divergence.

$$KL(MM||M_d) = \sum_w MM(w) \log \frac{MM(w)}{M_d(w)} \quad (3.23)$$

3.5 Experiments

We evaluated the quality of the generated query suggestions as well as the retrieved documents using our mixture model probability distribution. Section 3.5.1 describes how our technique performs better than the state-of-the-art in the task of document retrieval. Section 3.5.2 demonstrates that the query suggestions generated using our mixture model are better than those generated by state-of-the-art techniques. The source code for the techniques used is available at:

<https://bitbucket.org/prajnaupadhyay/ask/src/master/>

3.5.1 Experiments for Document Retrieval

In this section, we describe the experiments performed to compare the quality of top-5 documents retrieved for benchmark queries and aspects using our technique as well as the baselines.

3.5.1.1 Setup

Dataset. We used the Open Research Corpus dataset⁵ for our experiments. It consists of 39 million published research papers in the domain of computer science, neuroscience, and biomedical. We used the title and abstract fields of the data since the full text was not available due to copyright issues. We used Galago⁶, which is an API for experimenting with text search to index the Open Research Corpus dataset. Some of the baseline models (such as the query likelihood model and the pseudo relevance feedback techniques) are already implemented in Galago.

⁵<https://allenai.org/data/data-all.html>

⁶<https://www.lemurproject.org/galago.php>

Aspects. We experimented with 3 different aspects, namely, *application*, *algorithm* and *implementation*. Note that we can take in any other aspect—one that exists as a relation in TeKnowbase, such as *technique* or *software*, or any other aspect as described in [76] or [202] even if it does not exist as a relation in TeKnowbase. We set λ and β to 0.5 in Equations (3.7) and (3.17) respectively. We restricted ourselves to meta-paths of size at most 3, which was also the size of paths used in the Path Ranking Algorithm [111]. We set $k=5$ for choosing the top-k meta-paths for generating embeddings using MetaPath2Vec (described in Section 2.).

Benchmarks. Our benchmark queries were taken from the set of 100 queries released by [208]. We specifically chose those queries that existed as entities in TeKnowbase to evaluate the performance of aspect-based retrieval using KB. Out of these 100 queries, 43 existed as whole entities in our knowledge base. Note that the relevance can still be calculated for entities not in TeKnowbase as well. For those entities, the second component of Equation 3.19 will be zero, and the relevance will be determined by the aspect-based distribution alone. Our queries are shown in Table 3.6.

Sr no.	Query
1	artificial intelligence
2	augmented reality
3	autoencoder
4	big data
5	category theory
6	clojure
7	cnn
8	computer vision
9	cryptography
10	data mining
11	data science
12	deep learning

13	differential evolution
14	dirichlet process
15	duality
16	genetic algorithm
17	graph drawing
18	graph theory
19	hashing
20	information geometry
21	information retrieval
22	information theory
23	knowledge graph
24	machine learning
25	memory hierarchy
26	mobile payment
27	natural language
28	neural network
29	ontology
30	personality trait
31	prolog
32	question answering
33	recommender system
34	reinforcement learning
35	sap
36	semantic web
37	sentiment analysis
38	smart thermostat
39	social media
40	speech recognition
41	supervised learning

42	variable neighborhood search
43	word embedding

Table 3.6: Benchmark queries

Baselines. As pointed out in Section 1, we can explicitly add the keyword representing the aspect to the query and use a standard retrieval model to generate results. We can further use the relevance model to obtain feedback or use diversification techniques to improve the results. Retrieving papers relevant for a query and an aspect can also be treated as a text classification problem, where a classifier can be trained for each aspect to assign relevant and non-relevant labels to the documents. Owing to the success of neural networks in text classification [99], [96], [219], [212], we used state-of-the-art neural network classifiers to classify the documents into one or more of these aspects given a query (baseline number 6). Following are our baselines.

1. **Query likelihood model with query only (QL+query).** Query likelihood [153] estimates a language model for each document in the collection and ranks them by the likelihood of seeing the query terms as a random sample given that document model. Equation 3.24 describes the equation compute this probability.

$$P(d|q) \propto P(q|d) = \prod_{w \in \text{terms}(q)} \frac{tf(w) + \mu P(w|C)}{|d| + \mu} \quad (3.24)$$

where $tf(w)$ is the frequency of the term w in the document d . Dirichlet smoothing is used to smoothen this probability. μ is the parameter for Dirichlet smoothing and is set to 1500. $P(w|C)$ is the probability of observing the term in the entire collection and is proportional to the frequency of the term in the corpus.

2. **Query likelihood model with query + aspect name (QL + query + aspect).** We used the same model as above, only the query was changed. We added

the terms `application`, `algorithm` or `implementation` to the query based on the aspect required and retrieved the results. Table 3.7 gives examples of how we modified the query for this baseline.

3. **Query expansion with pseudo relevance feedback on QL + query + aspect (QL + query + aspect + QE).** We performed pseudo-relevance feedback over the results obtained by QL + query + aspect to improve the results. The query was expanded by 100 terms selected using the pseudo-relevance technique. Top 1000 documents were used for feedback and the weight of the original query was set to 0.75. Given a query with k keywords q_1, q_2, \dots, q_k , the pseudo-relevance model is described by the following equation:

$$P(w|q_1, q_2, \dots, q_k) = \sum_{M \in \mathcal{M}} P(M)P(w|M) \prod_{i=1}^k P(q_i|M)$$

where \mathcal{M} is the set of feedback document models. \mathcal{M} consists of top-1000 documents retrieved by searching for query+aspect.

4. **Query expansion using TeKnowbase relationships and entities. (QL + query + aspect + TKB).** This baseline uses TeKnowbase to expand the query using relationships and entities. Given a query q and aspect described by keyword a , we formulated a new query as $q' = q \ a \ e_{-1} \ e_{-2} \ \dots \ e_{-n}$, where each e_{-i} , $1 \leq i \leq n$ is an entity connected to the entity represented by q and relation described by keyword a in TeKnowbase. Table 3.7 lists the expanded query used for this baseline for query `variable_neighborhood_search` and `application` aspect. The results are then retrieved using query likelihood model.
5. **Search result diversification using xQUAD and TeKnowbase. (xQUAD + TKB)** Diversification of search results is necessary when the query submitted to the retrieval system is ambiguous or has multiple aspects to be explored. xQUAD is a diversification framework that uses sub-queries to retrieve diverse results. These sub-queries cater to different aspects of the original query and help to return documents that address diverse aspects. We are specifically interested in using the

xQUAD diversification technique with the sub-queries generated from TeKnowbase. Given a query q and aspect a , the set of sub-queries was represented by $\{q \ a \ e, \forall \langle q \ a \ e \rangle \in TKB\}$. For instance, for a query `variable_neighborhood_search` and aspect *application*, the sub-queries that we used are described in Table 3.7.

The xQUAD diversification framework is described as follows. Given a query q , an initial ranking of documents R , the number of documents to be retrieved τ and a parameter λ , xQUAD re-ranks the set of documents in R according to a probabilistic framework described in Algorithm 4.

Algorithm 4: xQUAD probabilistic framework

```

1  $S = \phi$ ;
2 while  $|S| < \tau$  do
3    $d^* = \operatorname{argmax}_{d \in R \setminus S} (1 - \lambda)P(d|q) + \lambda P(d, \bar{S}|q)$ ;
4    $R = R \setminus \{d^*\}$ 
5    $S = S \cup \{d^*\}$ 
6 end
7 return  $S$ 

```

Given an initial ranking of documents R , a new ranking is iteratively computed by choosing the document that maximizes the value returned by the following equation:

$$(1 - \lambda)P(d|q) + \lambda P(d, \bar{S}|q) \quad (3.25)$$

There are two components in this equation that model the relevance and the diversity of the documents. Their mixture is determined by λ . The relevance of the document is determined by $P(d|q)$. This value can be estimated using standard language modeling techniques. The diversity component is described by $P(d, \bar{S}|q)$. This models the probability of observing this document given that it is not already present in the set S of documents already retrieved. To derive this component, the set of sub-queries of the original query q is considered. These sub-queries model the different aspects of q and help in retrieving documents that address these aspects and are thus diverse from one another. Given a set of sub-queries $Q = \{q_1, q_2, \dots\}$

generated for q , the diversity component can be further extended as follows:

$$P(d, \bar{S}|q) = \sum_{q_i \in Q} P(q_i|q)P(d, \bar{S}|q_i) \quad (3.26)$$

where $P(q_i|q)$ is the probability of relative importance of q_i to q . Assuming that retrieving a document d is independent of the documents present in S , Equation 3.27 can be written as:

$$P(d, \bar{S}|q_i) = P(d|q_i)P(\bar{S}, q_i) \quad (3.27)$$

Given $S = \{d_1, d_2, \dots, d_n\}$, $P(\bar{S}|q_i)$ can be further written as:

$$P(\bar{S}|q_i) = P(\overline{d_1, d_2, \dots, d_n}|q_i) = \prod_{d_j \in S} (1 - P(d_j|q_i))$$

The final relevance equation can be written as:

$$(1 - \lambda)P(d|q) + \lambda \left(\sum_{q_i \in Q} P(q_i|q)P(d|q_i) \prod_{d_j \in S} (1 - P(d_j|q_i)) \right) \quad (3.28)$$

6. **Neural Networks for classification (HANN).** We used hierarchical attention neural networks [212] (HANN) for classifying the paper/abstract into one/more/none of the 3 categories: `algorithm`, `application` and `implementation`. The training data consisted of 1500 annotated documents collected using the scheme as described in Section 3.5.1.2. The training was done for 200 epochs after which the accuracy started converging. Top 1000 candidate documents retrieved according to query likelihood model (used in Equation (3.24)) for each query were re-ranked according to the following equation:

$$P(d|q, a) = HANN(d, a) * P(d|q)$$

where $HANN(d, a)$ is the probability of the document being classified into aspect a returned by the neural network classifier. $P(d|q)$ is estimated using Equation

(3.24).

7. **Mixture Model (MM)**. This is our retrieval model indicated by **MM** and described by the Equation (3.7). Top 1000 documents retrieved according to the query-likelihood model for the benchmark queries were re-ranked. The documents were returned in increasing order of KL-divergence as described in Equation (3.23).

3.5.1.2 Evaluation Methodology

Evaluation Scheme. In the absence of an extensive ground-truth dataset, we conducted a crowd-sourced user-evaluation exercise to measure the performance of our model. We built a web-based interface to facilitate the evaluations. We recruited students and researchers of Computer Science (not directly related to the project) to conduct the evaluations. A total of 110 users participated in this experiment. They were asked to evaluate the top-5 documents returned by our method as well as competing techniques for pairs of a given query and an aspect. Determining if a paper is relevant for a query and an aspect is not straightforward. So, instead of asking the evaluators if a paper is relevant for an aspect, we developed multiple rules to determine whether a paper is relevant for the query and the aspect and asked the evaluator to answer yes or no. These rules were already described previously in Section 3.4.3.

Evaluation Metrics. Each benchmark query and abstract pair was graded by at least 2 evaluators. We converted the response from the experts into a graded relevance scale using the rules described in Table 3.3, 3.4 and Table 3.5. We took the average of the relevance values marked by the two users for each query and document pair. We computed the **Precision**, which is the fraction of relevant documents retrieved to the total number of retrieved documents, and **Discounted Cumulative Gain (DCG)** to measure the performance of our model as well as the baselines. DCG and precision are computed as follows:

Baseline	Modified query	Retrieval model
QL + query + aspect	variable neighborhood search application	Query likelihood
QL + query + aspect + QE	variable neighborhood search application	Query likelihood with pseudo-relevance feedback
QL + query + aspect + TKB	variable neighborhood search application knapsack graph problems vehicle routing problems arc routing	Query likelihood
xQUAD + TKB	variable neighborhood search, sub-queries used for diversification were: <ol style="list-style-type: none"> 1. variable neighborhood search application knapsack 2. variable neighborhood search application graph problems 3. variable neighborhood search application vehicle routing problems 4. variable neighborhood search application arc routing 	xQUAD

Table 3.7: Given the query variable neighborhood search and aspect *application*, different queries were constructed to retrieve results for different baselines. This table lists these queries for 4 baselines.

$$DCG_p = \sum_{i=1}^p \frac{rel_i}{\log_2(i+1)} \quad (3.29)$$

$$Precision_p = \frac{1}{p} \sum_{i=1}^p rel_i \quad (3.30)$$

where rel_i is the relevance score for the document retrieved at the i th position and p is the total number of documents retrieved.

3.5.1.3 Results and Discussions

Table 3.8 shows the results for *algorithm*, *application* and *implementation* aspects. We observed that for all the 3 aspects, our model outperforms the rest of the baselines in terms of precision and DCG.

The DCG@5, precision@5, and precision@1 values for our technique for *algorithm* aspect are 6.27, 0.7, and 0.75, respectively, and are the best values amongst all aspects and all baselines. This means that at least 3 out of 5 documents retrieved by our model for algorithm aspect were marked as relevant, and for 75% of the queries, our model retrieved a relevant document at the 1st position. **xQUAD + TKB** comes second after our model, with a DCG@5 of 5.16, precision@5 of 0.58, and precision@1 of 0.62. **QL + query + aspect + QE** performs better than **QL + query + aspect** by using pseudo-relevance feedback techniques to perform query expansion but performs poorer than our model. It obtained a DCG@5 of 5.12, precision@5 of 0.58, and precision@1 of 0.61 as compared to **QL + query + aspect**, which obtained a DCG@5 of 5.03, precision@5 of 0.56, and precision@1 of 0.59. Using relations in TeKnowbase to expand the query in **QL + query + aspect + TKB** and generating sub-queries in **xQUAD + TKB** improved results over **QL + query + aspect + QE**, but still their performances were worse than our model. The results of the neural network classifier were evaluated to be very poor.

For *application* aspect, our model performed the best amongst all the baselines. We obtained a DCG@5 of 2.64, precision@5 of 0.45 and precision@1 of 0.47, which implies that the document retrieved at the 1st place for almost 50% of the queries were judged as relevant. **xQUAD + TKB** comes second and obtained a DCG@5 of 2.56, precision@5 of 0.43, and precision@1 of 0.43. **QL + query + aspect + TKB** obtained a DCG@5 of 2.56, precision@5 of 0.43, and precision@1 of 0.41. The performance of **QL + query + aspect + QE** is identical, except its DCG@5 is 2.5, which is lower than DCG@5 for **QL + query + aspect + TKB**. **QL + query + aspect + TKB** and **QL + query + aspect + QE** both performed better than **QL + query + aspect** for *application* aspect. The values obtained for **QL + query + aspect** for DCG@5 was 2.38, precision@5 was 0.41, and precision@1 was 0.35. The results retrieved by HANN were poor.

The DCG@5 and precision@5 for *implementation* aspect obtained by our technique were 2.33 and 0.44, which are the best values amongst all the baselines. **QL + query + aspect + QE** comes second, with DCG@5 value of 2.29 and precision@5 value of 0.37. **xQUAD + TKB** and **QL + query + aspect + TKB** obtain the same performance and come 3rd. They obtain a DCG@5 value of 2.0 and precision@5 value of 0.32. Both of them performed better than the simple addition of aspect term to the query i.e. **QL + query + aspect**, which obtained a DCG@5 of 1.92 and precision@5 of 0.30. The performance of HANN is poor for the implementation aspect as well, with 0.79 obtained for DCG@5 and 0.16 for precision@5.

Table 3.9 shows examples for 3 queries along 3 specified aspects. By modeling the aspect and query dependent probability explicitly, we were able to retrieve better results as compared to changing the query by adding aspect terms and performing pseudo-relevance feedback over the results as described in Section 1 and 3.4. The second paper retrieved for *genetic_algorithm* by **xQUAD + TKB** for *application* aspect does not describe any application of *genetic_algorithm* but contained a few terms like “a wide application prospect” in the abstract due to which it was retrieved in the top positions. Both the papers retrieved by our method address the application aspect for *genetic_algorithm* even if “application” is not mentioned in the title. Using sub-queries with **xQUAD + TKB**

Table 3.8: Results for algorithm, application and implementation aspect

Approach	Algorithm			Application			Implementation		
	DCG @5	p@5	p@1	DCG @5	p@5	p@1	DCG @5	p@5	p@1
MM	6.27	0.70	0.75	2.64	0.45	0.47	2.33	0.44	0.40
QL+query	2.69	0.3	0.33	1.42	0.25	0.22	1.05	0.16	0.23
QL+query+aspect	5.03	0.56	0.59	2.38	0.41	0.35	1.92	0.30	<i>0.43</i>
QL+query+aspect+QE	5.12	<i>0.58</i>	0.61	2.5	<i>0.43</i>	0.41	<i>2.29</i>	<i>0.37</i>	0.49
QL+query+aspect+TKB	<i>5.16</i>	<i>0.58</i>	0.61	<i>2.56</i>	<i>0.43</i>	0.41	2.0	0.32	0.42
xQUAD + TKB	<i>5.16</i>	<i>0.58</i>	<i>0.62</i>	<i>2.56</i>	<i>0.43</i>	<i>0.43</i>	2.0	0.32	0.42
HANN	0.77	0.1	0	0.44	0.08	0.02	0.79	0.16	0.01

for `genetic_algorithm` did not improve the retrieval performance over **QL + query + aspect** because `genetic_algorithm` is not connected to any entity via `application` relation, so no sub-queries were formed for expansion. Adding relevance feedback terms also did not work because the list of pseudo-relevant documents did not contain relevant documents in the first place due to plain keyword-based retrieval. Similarly, for *algorithm* aspect, the top 2 papers retrieved for `computer_vision` are not about algorithms but overview papers and were retrieved in the top positions because of the occurrence of `algorithm` in their abstracts. The papers retrieved by our model both propose algorithms for computer vision-related problems. For *implementation* aspect, the first paper retrieved by our model describes methods to parallelize autoencoder and the second paper discusses the implementation of autoencoder on hardware, so both are relevant. The second paper retrieved by xQUAD + TKB mentions the term `implement` multiple times but is not relevant. So, it is clear that our technique models the semantics associated with the query and the aspect better than any other keyword-based technique.

Table 3.9: Top-2 papers retrieved for 3 queries along 3 aspects for our method and a competing baseline

Approach	genetic algorithm	computer vision	autoencoder
MM	<i>Application</i> Genetic Ant Algorithm for Continuous Function Optimization and Its MATLAB Implementation	<i>Algorithm</i> Communication in a hybrid multi-layer MIMD system for computer vision	<i>Implementation</i> Parallelizing the sparse autoencoder
	Solve Zero-One Knapsack Problem by Greedy Genetic Algorithm	Job-shop scheduling applied to computer vision	Memristor crossbar based unsupervised training
xQUAD + TKB	Recent Developments in Evolutionary and Genetic Algorithms: Theory and Applications	Computer Evolution Promise	Vision: Performance of the Fixed-point Autoencoder
	Comparison and Analysis of Different Mutation Strategies to improve the Performance of Genetic Algorithm	Raydiance: A Tangible Interface for Teaching Computer Vision	Autoencoder using kernel method

3.5.2 Experiments for Query Suggestion

Apart from ranking documents, we used our mixture model to generate suggestions for a query and an aspect. We first obtained the list of candidate suggestions from relevant documents, as described in Section 3.4.4 and then ranked them in increasing order of divergence with the estimated mixture model. The divergence is measured according to Equation 3.20. Top 10 suggestions were used for evaluation.

3.5.2.1 Setup

We used the same dataset, aspects, and benchmark queries used for the document ranking experiment (described in Section 3.5.1).

Baselines. We experimented with the following baselines:

1. **Similarity based phrase search (SimSearch).** This approach searches for candidate suggestions to find those that contain the user-submitted query. As stated already, we can change the query by adding the keyword describing the aspect to retrieve results. So, for each aspect, we formulated such queries and retrieved suggestions that contained the query and the aspect keyword. However, this retrieved very few suggestions. So, we only retrieved suggestions that contained the aspect term in the suggestions and ranked them in decreasing order of score assigned by RAKE.
2. **Mixture Model.** This is our technique that ranks suggestions in increasing order of divergence with the estimated mixture model.

3.5.2.2 Evaluation Methodology

Evaluation Scheme. Due to the lack of a standard dataset for evaluation, we used humans for evaluating the generated query suggestions. To evaluate the suggestion for the query and the aspect, we first retrieved the top-10 documents by expanding the query with the generated suggestion. The evaluation consists of two parts. 1) First, only the suggestion is shown to the evaluator who is asked the following questions i) does the suggestion contain terms apart from the query, aspect, or common modifiers like various, common, different, etc. If yes, then it is considered interesting, otherwise, it is not interesting. For e.g. suggestions such as “common artificial intelligence applications” or “various artificial intelligence applications” are not interesting because apart from the

query and the aspect terms, it only contains modifiers like common or various. Such suggestions are simple re-phrasings of the original query. 2) The next step is to show a suggestion and the top-10 documents retrieved to the evaluator and ask if any one of them is relevant for the query and the aspect. The relevance of a document for the query and the aspect is determined in a similar fashion as described in Section 4.1.2. If any one of the top-10 documents is found to be relevant for the query and the aspect, the second question is answered as yes. We mark the suggestion as relevant for the query and the aspect if the answer to questions 1) and 2) is yes. For the query `artificial_intelligence` and aspect *application*, the suggestion *artificial intelligence-based financial application* is relevant because 1) it consists of terms apart from the query and the aspect which are not common modifiers and 2) it helps retrieve a document “Innovative methods for improving portfolio management based on artificial intelligence instruments”, which is relevant for the application aspect, at the second position, when expanded with the query. To be relevant, the suggestion need not describe an actual application but should contain terms that can retrieve a relevant document in the top positions. To illustrate, for the query `autoencoder` and aspect *application*, the suggestion *application layer ddos attack* is relevant, even though it does not describe an application of autoencoder because autoencoder has been used to detect distributed denial of service attacks. The paper “Detection of Application Layer DDoS attack by feature learning using Stacked AutoEncoder” is retrieved at the 1st position when this suggestion is expanded with the query.

Metrics. We used **Precision** to evaluate the quality of query suggestions.

3.5.2.3 Results and Discussion

The results of the evaluation of query suggestions are shown in Table 3.10. We further filtered these results (for both the techniques) by only retaining those suggestions that mention at least one entity from TeKnowbase. These techniques are referred to as SimSearch_E and MM_E. Our mixture model retrieves better suggestions than SimSearch

for all three aspects. The best precision is observed for *algorithm* aspect, which obtained a precision@10 of 0.68, precision@5 of 0.66, and precision@1 of 0.65, which is higher than SimSearch, which obtained a precision@10 of 0.52, precision@5 of 0.5, and precision@1 of 0.35. This means that more than 6 out of 10 suggestions retrieved for algorithm aspect on an average by our technique were relevant, and for two-thirds of the baseline queries, we retrieved a relevant suggestion at the top position. On the other hand, SimSearch could retrieve an average of 5 relevant suggestions out of 10 queries, and could only retrieve a relevant suggestion at 1st position for 35% of the queries. For the application aspect, our technique retrieved relevant suggestions for more than 50% of the queries at 1st position, while it was only 32% for SimSearch. Our precision values were also better than SimSearch at 5 and 10. For the implementation aspect, we retrieved a relevant suggestion for almost 60% of the queries at the top position by using our technique, while this percentage was 52% for SimSearch. Additionally, we were able to retrieve more than 6 out of 10 relevant suggestions on an average for the implementation aspect using our technique while SimSearch could retrieve more than 5 out of 10. It is clear that our technique retrieves better quality suggestions for each aspect than those retrieved using SimSearch.

Approach	Algorithm			Application			Implementation		
	p@10	p@5	p@1	p@10	p@5	p@1	p@10	p@5	p@1
MM	0.68	0.66	0.65	<i>0.445</i>	<i>0.453</i>	0.56	<i>0.55</i>	<i>0.55</i>	<i>0.52</i>
SimSearch	0.52	0.50	0.35	0.37	0.26	0.32	0.42	0.44	0.39
MM _E	0.68	0.66	0.65	0.54	0.52	<i>0.54</i>	0.653	0.62	0.59
SimSearch _E	0.52	0.50	0.35	0.32	0.28	0.3	0.45	0.55	0.48

Table 3.10: Quality of query suggestions generated for algorithm, application and implementation aspect. MM is our technique while MM_E is the technique that retains only those suggestions containing at least an entity from TeKnowbase. SimSearch is the baseline, SimSearch_E is the technique that retains only those suggestions containing at least an entity from TeKnowbase.

Table 3.11, 3.12 and 3.13 show top-5 suggestions generated for *algorithm*, *application*, and

implementation aspect respectively. It is clear that our technique is successful in ranking relevant suggestions at higher positions. For instance, for `computer_vision` and *algorithm* aspect, we are able to rank important algorithms related to computer vision, like *scale edge detection algorithm based* higher as compared to suggestions like *computer vision algorithm* or *many computer vision algorithm* retrieved by simple keyword search. For *application* aspect for query `genetic algorithm`, the baseline retrieved suggestions that are not relevant although they contain *application*. To give an idea, *genetic algorithm special-purpose application domain toolboxes* retrieves paper that describes a manual for a software that helps new users understand the genetic algorithm, which is not relevant for *application* aspect. On the other hand, the suggestion *genetic algorithm based data clustering* retrieved by our technique is relevant even if the application word is not mentioned in it. For *implementation* aspect, our technique ranks relevant suggestions like *model predictive control implementation* in the second position and higher than baseline, which retrieves *paper presents two different implementations*, which is not relevant. Also, our technique returns suggestions like *based image retrieval CBIR system*, which does not contain the word *implementation*, but is still relevant, because autoencoders have been used to speed up the retrieval of images in content-based image retrieval.

Table 3.14 reports the timings required for generating the suggestions as well as ranking the documents. The experiments were conducted on a 32-core Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz with 128GB of RAM. The computation of KL-divergence has been parallelized for reducing the time overhead. The primary goal of this work has been to improve the quality of results retrieved for the query and the aspect and the time overhead can be further improved in the future.

3.6 Conclusion

In this paper, we developed **ASK** (**A**spect-based academic **S**earch using domain-specific **K**B), which introduced and solved the task of aspect-based academic search. Given a

Table 3.11: Comparison of the suggestions retrieved by our model and the baseline for query *computer vision* and *algorithm* aspect

MM	SimSearch
iterative optimization algorithm based	computer vision algorithms
scale edge detection algorithm based	computer vision algorithm
simple correlation based algorithm	many computer vision algorithms
based algorithm	three papers present optimization algorithms
approximate algorithm based	image processing algorithms

Table 3.12: Comparison of the suggestions retrieved by our model and the baseline for query *genetic algorithm* and *application* aspect

MM	SimSearch
multi objective genetic algorithm application	genetic algorithm intrusion detection system ppt application
genetic algorithm application	genetic algorithm user runs custom applications using
adaptive genetic algorithm application	genetic algorithm special purpose application domain toolboxes
genetic algorithm greedy algorithm application	genetic algorithm real world embedded industrial application
genetic algorithm based data clustering	genetic algorithm produce net application java technology

Table 3.13: Comparison of the suggestions retrieved by our model and the baseline for query autoencoder and *implementation* aspect

MM	SimSearch
examined memristor crossbar based implementation	examined memristor crossbar based implementation
model predictive control implementation	paper presents two different implementations
system performance evaluation based idf based information retrieval system	embarrassingly simple implementation
based image retrieval cbir system	model predictive control implementation could achieve accuracy

Times(s)/Aspects	Algorithm	Application	Implementation
Time for documents retrieval	144	127	82
Time for query suggestion	38.41	28.35	11.82

Table 3.14: Average time taken for document retrieval and query suggestion for the 3 aspects in seconds

query and an aspect, **ASK** returns a ranked list of documents as well as query suggestions that address that aspect for the query. The key idea used was to estimate a language model for the query and the aspect both using a technical knowledge base. We used the relationships in the knowledge base to represent aspects and used meta-paths to infer such relationships over entities not connected via this relation. The final distribution was estimated by taking a mixture of the query-independent and the aspect-query dependent distribution. We tested our model for 43 queries and 3 aspects with satisfactory results. The results retrieved by our mixture model were evaluated to be better than a number of state-of-the-art keyword-based, relevance feedback, diversification, and neural models.

In the future, we can make the query suggestions more diverse to each other by adding a diversity component while ranking, similar to the approach described in Algorithm 7 in Chapter 4. Further, we can focus on reducing the time-overhead of the aspect-based search and query suggestions.

The aspect-based search of articles can be applied to other technical domains as well, such as Biology/Bio-informatics. *The Gene Ontology Resource* stores information about different genes and proteins, their functions and the biological processes they are involved with. One such example of a triple in the GO resource is $\langle \text{TOP2A}, \text{function}, \text{DNA topoisomerase/gyrase; participates in chromosome condensation} \rangle$, which states that the TOP2A gene is involved in chromosome condensation. A user, who is interested to retrieve scientific articles that describe the functions involved with a gene can specify `function` as an aspect and `TOP2A` as the query. Additionally, gene functions can also be inferred using our meta-path based or any other existing techniques [65] to further improve the language model for the query and the aspect.

Chapter 4

PreFace: Faceted Retrieval of Prerequisites

4.1 Motivation and Problem

A student who wishes to study a new topic will face difficulty understanding certain concepts, for which she does not have the required *prerequisite* knowledge. A prerequisite [119] for a concept a is another concept b that can be recommended to be studied before a for a better understanding of a . For instance, the prerequisites of `convolutional_neural_network` include `artificial_neural_network` and `backpropagation`, and Figure 4.1 shows a graph of prerequisites for convolutional neural networks. The nodes in the graph represent the prerequisites and the edges indicate the prerequisite relationship.

Moreover, a topic in Computer Science usually has multiple aspects of understanding. To illustrate, for the query `convolutional_neural_network`, the set consisting of concepts like `face_perception`, `data_classification` and `security_applications` can be recommended as prerequisites for the *application* aspect, because all of them describe ap-

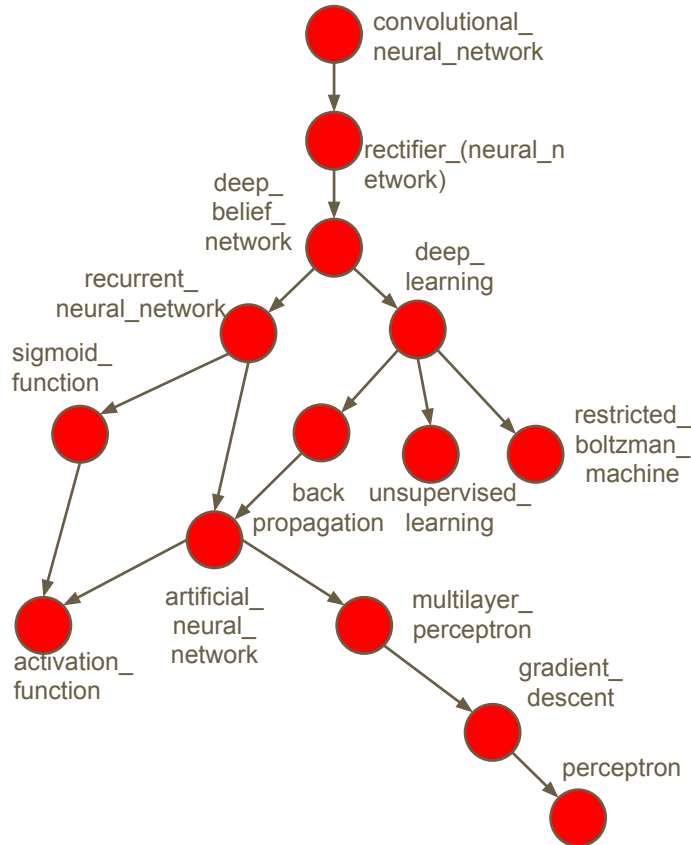


Figure 4.1: Graph of prerequisites for `convolutional_neural_network`

plications of `convolutional_neural_network`. Another set of concepts, such as `matlab` and `python` can be recommended for the `software` aspect, since they help implement `convolutional_neural_network`. It would be helpful to have a retrieval system that, takes a topic as a query and returns a prerequisite graph, as shown in Figure 4.1 and prerequisites suggested in groups, called facets, relevant for multiple aspects of the query, as shown in Table 4.1.

We now discuss in detail the issues that a computer science enthusiast faces when she comes across a new topic. She will start by querying for that topic on the web. However, there is no guarantee that the documents returned will help her understand the topic.

Convolutional neural network

From Wikipedia, the free encyclopedia

In [deep learning](#), a **convolutional neural network** (**CNN**, or **ConvNet**) is a class of [deep neural networks](#), most commonly applied to analyzing visual imagery.^[1] They are also known as **shift invariant** or **space invariant artificial neural networks** (**SIANN**), based on their shared-weights architecture and [translation invariance](#) characteristics.^{[2][3]} They have applications in [image and video recognition](#), [recommender systems](#),^[4] [image classification](#), [medical image analysis](#), [natural language processing](#),^[5] and [financial time series](#).^[6]

CNNs are [regularized](#) versions of [multilayer perceptrons](#). Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to [overfitting](#) data. Typical ways of

Figure 4.2: Snapshot of the first paragraph of Wikipedia page for `convolutional_neural_network` as on 23 July 2020

The following examples will illustrate the scenario.

Scenario 1: Figure 4.2 shows a snapshot of Wikipedia page for `convolutional_neural_network`. This page references a number of other Wikipedia pages, such as `backpropagation`, `deep_learning` and if the student is not aware of these topics, then she has to refer to the Wikipedia pages of these concepts as well. This leads to a chain of searches, and the student will end up spending a significant amount of time trying to understand the concept [189]. It is desirable that there exist techniques to automatically generate a prerequisite graph for the query, as shown in Figure 4.1.

Scenario 2: To solve the issue described above, a number of techniques to determine prerequisites for a concept have been proposed over the years. Most of these techniques address this problem by constructing prerequisite functions that take in a pair of con-

Notable libraries [\[edit \]](#)

- **Caffe**: A library for convolutional neural networks. Created by the Berkeley Vision and Learning Center (BVLC). It supports both CPU and GPU. Developed in **C++**, and has **Python** and **MATLAB** wrappers.
- **Deeplearning4j**: Deep learning in **Java** and **Scala** on multi-GPU-enabled **Spark**. A general-purpose deep learning library for the JVM production stack running on a C++ scientific computing engine. Allows the creation of custom layers. Integrates with Hadoop and Kafka.
- **Dlib**: A toolkit for making real world machine learning and data analysis applications in C++.
- **Microsoft Cognitive Toolkit**: A deep learning toolkit written by Microsoft with several unique features enhancing scalability over multiple nodes. It supports full-fledged interfaces for training in C++ and Python and with additional support for model inference in **C#** and Java.
- **TensorFlow**: **Apache 2.0**-licensed Theano-like library with support for CPU, GPU, Google's proprietary **tensor processing unit (TPU)**,^[126] and mobile devices.
- **Theano**: The reference deep-learning library for Python with an API largely compatible with the popular **NumPy** library. Allows user to write symbolic mathematical expressions, then automatically generates their derivatives, saving the user from having to code gradients or backpropagation. These symbolic expressions are automatically compiled to **CUDA** code for a fast, **on-the-GPU** implementation.

Figure 4.3: Snapshot of the libraries mentioned in the Wikipedia page for `convolutional_neural_network` as on 23 July 2020

cepts and determine whether one is a prerequisite of the other [119], [116], [149] or by constructing prerequisite graphs [198], [211] and [167]. These functions generally use the concepts referred to in its description found on the web or textbooks. However, it's not necessary these concepts will lead to extraction of good quality of prerequisites. For e.g. in Figure 4.2, concepts like `image_recognition` or `video_recognition` are also mentioned in the Wikipedia page, along with other concepts such as `computer_vision` in subsequent paragraphs. These are *applications* of `convolutional_neural_networks`, and should be recommended to the student only if she wishes to study the *applications* of `convolutional_neural_networks`. Similarly, Figure 4.3 lists some libraries, such as `keras` or `tensorflow`, that are implemented in `python` or `c`. Because all of these concepts

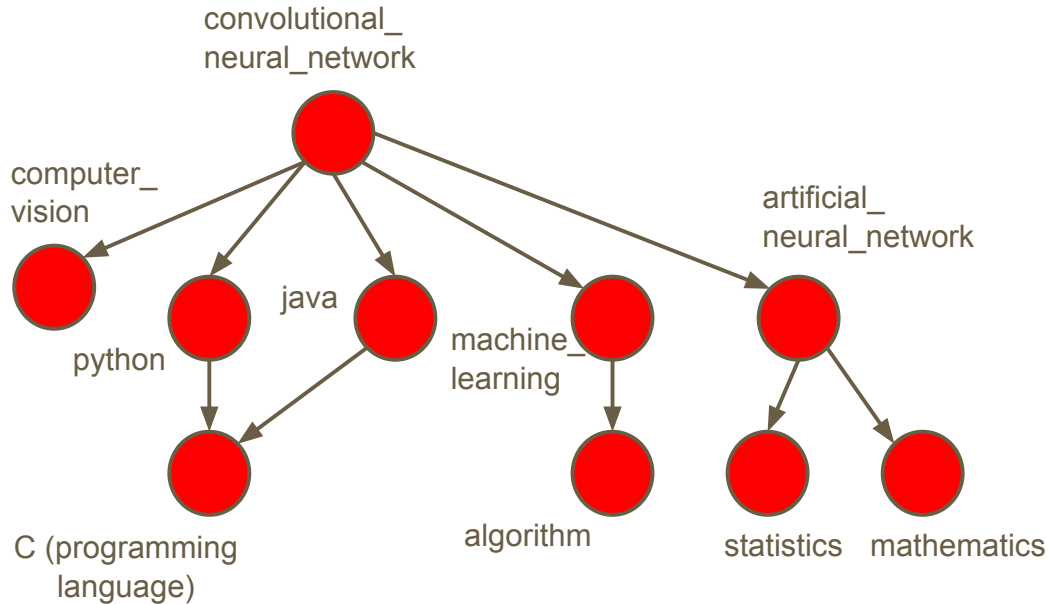


Figure 4.4: Prerequisites returned for `convolutional_neural_network` by *RefD*

are mentioned in the Wikipedia article for `convolutional_neural_network`, existing techniques, such as *RefD* [119], return them as prerequisites, as shown in Figure 4.4. It returns `python`, `C (programming language)` along with `artificial_neural_network` in the same prerequisite graph. While some of the concepts such as `artificial_neural_network` shown in Figure 4.4 should be recommended in the graph, concepts such as `python` or `C_(programming_language)` should be suggested to the user only if she wishes to implement convolutional neural networks. As a result, existing techniques suffer from *poor precision and recall* in retrieving prerequisites.

4.1.1 Problem Definition

In this section, we formally describe the terminologies that will be used throughout this chapter. Some of these terms have already been defined in the previous chapters. We

Table 4.1: Example of 4 faceted prerequisites retrieved by *PreFace* for `convolutional_neural_network`

Facets	Aspects
gradient_descent, optimization, backpropagation,	algorithm
security_application, face_perception	application
activation_function, transfer_function, sigmoid,	function
matlab, python	software

include them here again for the sake of completeness.

Definition 10. Concept. A concept C_i is any technical topic that can be studied and understood. In our case, it is an entity from the domain of Computer Science.

Definition 11. Concept Space. C is the set of all concepts called the concept space. $|C| = m$.

Definition 12. Query. A query $q \in C$ is a concept from the concept space about which the user would like to study.

Definition 13. Prerequisite. A prerequisite of a concept q is another concept p that can be recommended to be studied before q . In other words, having a knowledge of p improves the understanding of q . This is denoted by $q \rightarrow p$. A concept q cannot be a prerequisite of itself, so $q \not\rightarrow q$. Given the concept `convolutional_neural_network`, some prerequisites that can be suggested are `artificial_neural_network`, `rectifier` or `matlab`.

Definition 14. Necessary Prerequisites. A necessary prerequisite of a concept q is a prerequisite b which **has to** be studied before q . This is denoted by $q \implies b$. An absence of such a relationship is denoted by $q \not\implies b$. As explained before, `artificial_neural_network` and `rectifier` are necessary prerequisites for `convolutional_neural_network`.

Some properties of a necessary prerequisite are:

1. **Irreflexive:** A concept q cannot be a necessary prerequisite of itself, i.e. $q \not\Rightarrow q$.
2. **Asymmetric:** If $q \Rightarrow b$, then $b \not\Rightarrow q$.
3. $\forall b$ s.t. $q \Rightarrow b$, $q \rightarrow b$ holds.

Definition 15. Prerequisite Function. Let $P_f : C \times C \rightarrow \{T, F\}$ be any prerequisite function. For $C_i, C_j \in C$, $P_f(C_i, C_j)$ returns T (true) if C_i is a prerequisite of C_j , otherwise, F (false).

Definition 16. Prerequisite Graph. $P_G = (V, E)$ is the prerequisite graph, where each $C_i \in C$ is a node in V and there exists an edge from node C_i to C_j if C_j is a necessary prerequisite of the concept C_i .

To acquire an overall understanding, one has to understand different *aspects* of the queried topic. A knowledge of a software such as `matlab` can help the user implement `convolutional_neural_network`. So, `matlab` can be suggested as a prerequisite for the `software` aspect of `convolutional_neural_network`. If the aspect is `application`, then a concept such as `face_perception` can be recommended since it is known to be an application of `convolutional_neural_network`. If the aspect is `algorithm`, then prerequisites such as `optimization` or `gradient_descent` can be recommended. So, we formally describe an aspect as follows:

Definition 17. Aspect. An aspect a is a keyword that describes some subtopic of q . $q \rightarrow (p, a)$ denotes that the prerequisite p of q can be recommended to understand aspect a of query q . In the context of this chapter, an aspect can be an entity or a relation in a technical knowledge base.

Aspects can be used to define *query facets*. According to [56], “a query facet is a set of items which describe and summarize one important aspect of a query. Here a facet item is typically a word or a phrase. A query may have multiple facets that summarize the information about the query from different perspectives.”. We extend this idea of query facets to prerequisites of a query and define a **facet** as follows.

Definition 18. Facet. A facet refers to a set F of prerequisites of q that can be recommended for an aspect a of a query. This means for all $p \in F$, $q \rightarrow (p, a)$. To illustrate, the set of concepts $\{\text{matlab}, \text{octave}\}$ is a facet which can be suggested for software aspect of `convolutional_neural_network`. The set of concepts $\{\text{gradient_descent}, \text{optimization}\}$ can be suggested for algorithm aspect of `artificial_neural_network`.

4.2 Approach and Contributions

We propose PreFace, an approach to automatically generate a prerequisite graph and identify facets of prerequisites for a query using TeKnowbase. As already shown in Figure 4.4, existing techniques such as RefD [119] return prerequisites that may not be necessary. PreFace retains necessary prerequisites in the prerequisite graph by using the notion of similarity to the queried concept. We use graph embedding representations for entities trained on TeKnowbase to determine the similarity of concepts.

Additionally, PreFace also returns *facets* of prerequisites for different aspects of interest of the query. Existing techniques can be used to solve the two sub-problems separately, namely facet extraction [170], [63], [94] and prerequisite determination [119], [38] i.e. we can first extract facets for a query and then only retain concepts that are prerequisites to the query for each facet. However, this approach does not guarantee good results. The existing query-based facet extraction systems from knowledge bases [94, 63] use paths to group entities in a knowledge graph into facets. This may lead to facets consisting of entities not similar to each other with respect to the query. Figure 4.5 shows the path which connects `medical_imaging` and `fast_fourier_transform` to `convolutional_neural_network`. [94] constructs facets by grouping nodes connected via the same sequence of nodes and edges from `convolutional_neural_network` together. In this case, the sequence $\langle \text{convolutional_neural_network}, \text{implementation_inverse}, \text{gpgpu}, \text{application_inverse} \rangle$ is the sequence of path that places `medical_imaging` and `fast_fourier_transform` in the same facet. Although both are applications of `gpgpu`, both of

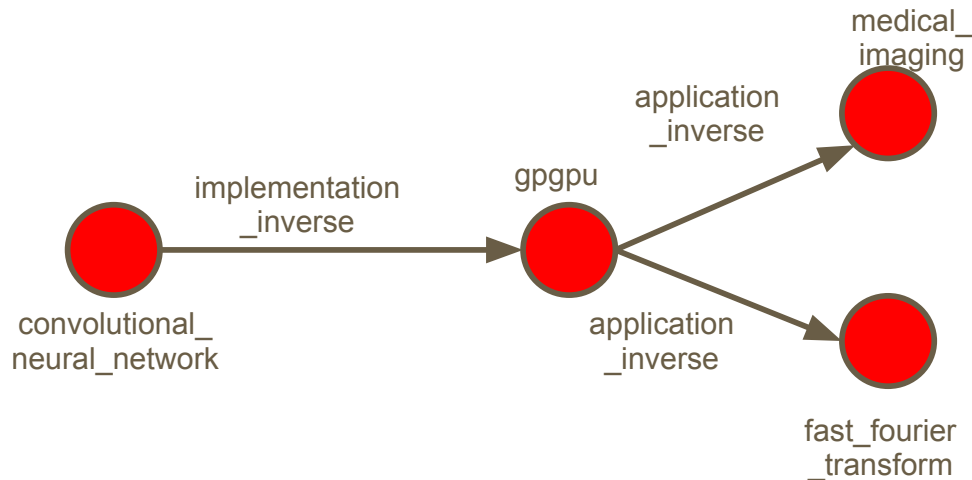


Figure 4.5: Path connecting `convolutional_neural_network` to `medical_imaging` and `fast_fourier_transform`

them are not applications of `convolutional_neural_network`. So, the sequence of path $\langle \text{convolutional_neural_network}, \text{implementation_inverse}, \text{gpgpu}, \text{application_inverse} \rangle$ might not always lead to concepts that are applications of `convolutional_neural_network`, but there is no way to specify this information in these techniques. As a result, it returns facets that are not semantically similar to each other with respect to the query. We address these issues in a two-step process—we first construct highly similar facets by hierarchically clustering [50] key-phrases relevant to the query. We represent these key-phrases as bag-of-words and entities using TeKnowbase which clusters highly similar key-phrases into the same facet. Next, we estimate language models [153] for the facets as well as the query. We make use of the aspect-based relevance model described in Chapter 3 and an existing prerequisite function to estimate the aspect based relevance of the query so that facets for different aspects are returned. Finally, these facets are ranked in increasing order of their similarity of their language models with that of the query.

In short, PreFace takes a query in the domain of Computer Science and returns the

following:

1. A prerequisite graph P_G of necessary prerequisites, which have to be necessarily studied before the query. For e.g, Figure 4.1 shows a prerequisite graph returned for `convolutional_neural_network`. It consists of nodes such as `artificial_neural_networks` and `backpropagation`, which are necessary prerequisites for `convolutional_neural_network`.
2. A set of faceted prerequisites for them. For example, Table 4.1 shows prerequisites for 4 interesting facets—i) algorithm, ii) application, iii) function, and iii) software

Contributions. The salient contributions of this work are:

1. Introduced the idea of necessary prerequisites and the novel problem of faceted retrieval of prerequisites
2. Proposed techniques to construct a prerequisite graph consisting of necessary prerequisites.
3. Development of a language model-based framework to retrieve interesting facets for a query of interest using TeKnowbase.
4. User evaluations to compare our approach with existing techniques for prerequisite determination and facet extraction.

Organization. This rest of this chapter is organized as follows. Section 4.4 describes the components of PreFace. Section 4.5 describes the experimental setup. In Section 4.3 we describe the related work in this field.

4.3 Related Work

While ours is the first attempt at developing a retrieval model for faceted prerequisite extraction, both of these techniques i.e. facet extraction and prerequisite determination have been independently explored. Below we review both approaches.

4.3.1 Facet Extraction

Extracting facets (or aspects) has been studied over a long time, using knowledge graphs and/or search results. Below we list related work in each of these areas.

4.3.1.1 Facet Extraction for Regular Search

Facets are either predefined categories on the corpus or are built dynamically based on the query.

Static Facet Categories. Among existing works that use static facet categories are Ontogator [129] and mSpace [170] that use RDF graphs to annotate images to facilitate faceted browsing. [148] developed BrowseRDF, which helps the user browse an RDF graph by providing constraints to be applied on graph properties. They also proposed metrics to measure the quality of facets and rank them. [77] helps a user answer complex queries using a faceted search on Wikipedia. The properties are extracted from Wikipedia info-boxes and displayed to the user for further refining the results. gFacet[84] and Visi-Nav [79] are tools that provide visualization of the web of data supported with faceted filtering techniques using RDF graphs and properties.

Dynamic Generation of Facets. Among the systems that generate facets dynamically are those that build SPARQL queries on the fly to be executed on the respective

SPARQL endpoints. [64] and [63] have used these approaches to build facets for a query. The authors proposed QDMiner [56] to retrieve facets from search results by extracting frequently occurring lists in relevant documents. These lists were extracted from structured data in the documents, like HTML lists or tables. [94] proposed QDMKB that improved the results generated by [56] using FreeBase. Another extension to QDMiner was done by [102] where they improved the quality of facets generated by using a probabilistic graphical model. Both QDMiner and QDMKB assume that the corpus is rich in meta-data, which is not always true. The techniques that use knowledge bases to generate facets assume that they are sufficient, which may not be the case for domain-specific graphs.

Constructing Faceted Hierarchies Constructing a hierarchy or taxonomy of items from a document collection has been a popular area of research. [45] proposed a system to construct facet hierarchies for a text corpus and then assigned the documents to each of these facets. [199, 178, 131]. This is different from our facets which are co-ordinate terms and not hierarchies.

4.3.1.2 Faceted Academic Search

In the context of academic search, [70] built *Scienstein* that allows users to search for papers using authors or reference lists apart from the usual keyword-query search. These tools make use of the underlying structure of the citation graph as well as machine learning techniques on the document text. [52] made an effort to provide faceted retrieval for research papers in computer science. The facets were—publication years, authors, or conferences. These facets are different from aspects in our scenario. [34] proposed techniques to further categorize the relationships between the query paper and the recommended paper. These relationships are expressed in the form of facets like `background`, `alternative_approaches`, `methods` and `comparisons`. [42] used similar facets to summarize scientific papers. These facets are extracted by identifying the context of the text

surrounding the citation. The facets used by these papers are different from the facets that we are interested in. Our facets are facets for prerequisites of the query, which is a concept, while the facets described in the related work are relationships between the research papers.

4.3.2 Prerequisite Determination

Prerequisite Functions. Among the techniques that take a pair of concepts and return whether one is a prerequisite of the other, [184] used crowd-sourcing to create a gold standard dataset that was used to train a classifier using features from Wikipedia. This classifier takes two concepts (names of Wikipedia pages) as input and tells whether one is a prerequisite of the other. In [119], the authors proposed *RefD* using frame semantics to compute prerequisites between concepts using Wikipedia. *RefD* represents each concept as a frame of related concepts and measures the reference distance between two concepts based on this representation. Wikipedia has also been used as source of features to build a supervised classifier in [167]. The authors make use of Wikipedia clickstream data to build the classifier. In [121], the authors used *RefD* to infer concept prerequisite relationships from course prerequisite pairs. They later proposed active learning techniques to reduce the amount of training data in [120]. [74] have proposed information theoretic measures to determine dependencies between concepts in a scientific corpus. In [149], authors have trained classifier using lecture transcripts from MOOCs to improve the prediction of prerequisite pairs. [211] have proposed an optimization framework to model prerequisite links among concepts as latent links which can be used to infer prerequisite links between course pairs across universities. Supervised learning model has been used in [126] to determine prerequisite relations by extracting high quality phrases from educational data.

Organization of Topics. A number of techniques have been proposed to present information in an organized manner. These include generating hierarchies over document

collections [105] or ordering documents in a sequence. [172] proposed metro-maps to show the developments between research papers. [201] proposed methods to generate reading orders for a concept of interest in the domain of physics [73] and [91].

Although a number of techniques exist that solve the two sub-problems that we are concerned with, to the best of our knowledge, there exists no other system that solves both of these problems. Please note that we do not compete with any prerequisite function or are proposing a new technique to determine prerequisites. Our approach takes *any* prerequisite function and automatically identifies facets using the function and TeKnow-base.

4.4 PreFace

We propose PreFace, which is a system to extract prerequisites as well as facets for a queried topic of interest. Figure 4.6 shows the components of PreFace. It takes a query and returns i) a prerequisite graph, where the nodes are the necessary prerequisites, and the edges indicate the prerequisite relationship. ii) a ranked list of faceted prerequisites for the query. The two components of PreFace are shown in the figure.

4.4.1 Prerequisite Graph Generation

In this section, we describe the technique to generate the prerequisite graph for a query. We use the idea of frame semantics to represent a concept and describe it in the next paragraph.

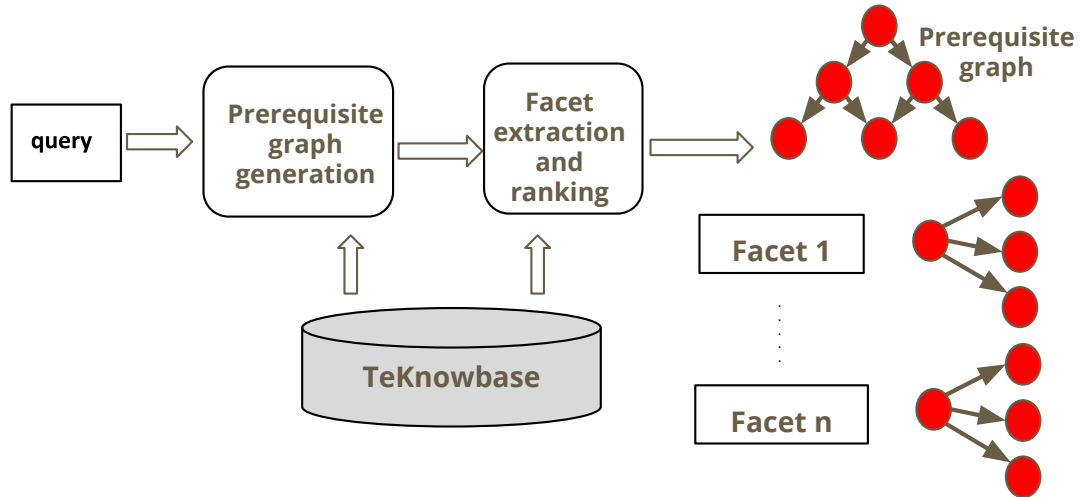


Figure 4.6: Architecture of PreFace

4.4.1.1 Frame Semantics

Frame semantics [35] is a theory in linguistics developed by Charles J. Fillmore. The key idea proposed by Fillmore is that a concept is defined by a set of related concepts called *frame*. For instance, a concept such as `knuckle` has to be understood along with the knowledge of `finger`, `hand` or `arm`. So, these terms constitute the frame for `knuckle`. A frame is defined formally as follows.

Definition 19. Frame. *Given a concept a , a frame f_r for a is a set $f_r = \{a_1, a_2, \dots, a_{|f_r|}\}$, where each a_i , $1 \leq i \leq |f_r|$ is a concept related to a . To illustrate, a frame for `convolutional_neural_network` is shown in Figure 4.7. The set of concepts that are referred to in articles or tutorials describing a can be used as a frame for a . A popular technique to determine the frame for a is to use the concepts mentioned in the Wikipedia page of the article describing a .*

The frame plays a crucial role in determining the prerequisites of a concept. Using the set of concepts mentioned in the Wikipedia page for `convolutional_neural_network` will lead

```
{deep_belief_network, artificial_neural_network,  
rectifier_(neural_networks), multilayer_perceptron,  
backpropagation, deep_learning, neocognitron,  
convolutional_neural_network, sigmoid_function,  
recurrent_neural_network, long_short-term_memory,  
time_delay_neural_network}
```

Figure 4.7: A frame for convolutional_neural_network

to the presence of concepts such as `keras` or `torch` in its frame. This results in retrieving prerequisites such as `python` or `C`, which are relevant for the *software* aspect. Instead, if we can determine a better frame that does not contain any of these concepts, then we can improve the precision of necessary prerequisites.

We use the notion of similarity between the concepts to determine a better frame for a concept. The key idea used is to remove concepts highly dissimilar to the queried concept from the frame. To define the similarity between the concepts, we make use of their neighborhoods in TeKnowbase. As an example, the neighborhood of `convolutional_neural_network` consists of `neural_network`, `deep_learning` or `machine_learning`, which is also the neighborhood of `time_delay_network`. So, both these concepts are similar to each other. On the other hand, concepts like `python` or `C` have different neighborhoods, so their similarity to `convolutional_neural_network` is low. Graph embedding-based techniques, like Node2Vec, [75] are popular ways to assign vector representations to nodes in a graph in such a way that the nodes that have similar neighborhoods are assigned vector representations closer to each other. We use the embeddings generated for entities in TeKnowbase using Node2Vec as vector representations for the entities, using which we develop techniques to prune the frame. Our technique is described in Section 4.4.1.2

4.4.1.2 Our Approach

We propose an algorithm to determine the correct frame for a query q to be used with *any* prerequisite function. The idea of frames has been used previously in prerequisite functions, like RefD [119]. Algorithm 5 describes our algorithm. After the frame has been determined, we generate the prerequisite graph for the query using Algorithm 6. Following steps describe these two algorithms in detail:

- 1. Determining Frames.** As stated in the examples in Section 4.1, using the set of concepts mentioned in a tutorial (such as Wikipedia page) as the frame for the query returns both necessary and soft prerequisites. We have to remove such concepts from the

Algorithm 5: Generate frames

```

Input:  $Candidates(q), q, TKB$ 
Output:  $Frame$ : New frame computed for all candidates
1  $Candidates(q) = Candidates(q) \cup q$ 
2 foreach  $e \in Candidates(q)$  do
3    $S = \text{set of concepts in the original frame for } e$ 
4   // Construct new frame for a concept  $e$ 
5   Let  $D$  be a  $|S| \times |S|$  array
6   foreach  $(e_i, e_j) \in S \times S$  do
7      $D_{i,j} = \text{cosine\_distance}(V(e_i), V(e_j))$ 
8     //  $V_i$  and  $V_j$  are the vector representations for entities  $e_i$  and  $e_j$ 
      obtained by training Node2Vec algorithm on  $TKB$ 
9   end
10  Cluster  $S$  using agglomerative clustering with  $D$ 
11  Use CH-index to choose the best cluster set and set the frame  $Frame(e) = \text{the cluster to}$ 
      which  $e$  belongs
12 end
13 return  $Frame$ 
14

```

given frame. Using a similarity threshold to remove concepts from the frame, or fixing the size of the frame to keep the k most similar entities needs an additional parameter—size or similarity threshold as input. So, we propose the idea of partitioning the set of concepts in the frame using agglomerative clustering [50]. The steps for partitioning and choosing the appropriate frame are described in Lines 2–13 in Algorithm 6. It takes the query q , a function $Candidates$ that returns a set of candidate prerequisites for q , and TKB , which is TeKnowbase. The set of candidate concepts are usually considered to be all concepts reachable up to 2-hops from the Wikipedia article of the concept. We describe the steps of the algorithm in detail as follows:

1. Let S be the set of concepts in the original frame of q , e_i be each concept in S and $V(e_i)$ be the vector representation of each e_i in S .
2. We compute the pairwise distances between concepts in S using cosine distance between their vector representations. (Lines 5–9)

Algorithm 6: Generate prerequisite graph P_G for q

Input: $Frame, TKB, P_F, q, Candidates(q)$
Output: P_G

```

1 // Determine prerequisites for query  $q$  using new frames
2  $P_C = \emptyset$ 
3 foreach  $c \in Candidates(q)$  do
4   | if  $P_F(c, q, Frame(c), Frame(q)) == True$  then
5   |   |  $P_C = P_C \cup c$ 
6   |   | continue
7   | end
8  $P'_C = P_C$ 
9 // Improve recall of prerequisites for  $q$ 
10 foreach sibling  $s_i$  of  $q$  in  $TKB$  do
11 | determine the cosine similarity of  $s_i$  with  $q$  using their embedding representations
12 end
13  $L$  = list of siblings of  $q$  in  $TKB$  ranked in decreasing order of cosine similarity
14 foreach  $e_i$  in  $L$  do
15 |  $Candidates(e_i)$  = Set of candidate prerequisites for  $e_i$ 
16 |  $P_D = \emptyset$ 
17 | foreach  $c \in Candidates(e_i)$  do
18 |   | if  $P_F(c, e_i, Frame(c), Frame(e_i)) == True$  then
19 |   |   |  $P_D = P_D \cup c$ 
20 |   |   | continue
21 |   | end
22 |   | if  $((P'_C \cup P_D) - P_C) < \alpha$  then
23 |   |   |  $P'_C = P'_C \cup P_D$ 
24 |   |   | break
25 | end
26 // Construct prerequisite graph  $P_G$  for  $q$ 
27 Construct a graph  $P_G$  where nodes belong to  $P'_C$ 
28 foreach pair of nodes  $(a, b)$  in  $P_G$  do
29 | Add edge from  $a$  to  $b$  if  $P_F(a, b, Frame(b)) == True$ 
30 end
31 From  $P_G$ , delete edges  $(a, b)$  such that  $b$  is a descendant in the knowledge base's taxonomy of  $a$ .
32 Detect cycles in  $P_G$  and remove the edge that has the lowest score returned according to
   | prerequisite function  $P_F$ 
33 return  $P_G$ 

```

3. Next, we cluster the set S using agglomerative clustering. Agglomerative clustering [50] is a bottom-up clustering strategy that initially assumes each item in the set as a single cluster. It then merges the most similar pairs of clusters together at each iteration. To measure the inter-cluster distance, a single linkage clustering strategy is used. The single linkage clustering strategy uses the distance of the most similar items between two clusters as the distance between those two clusters. At each iteration, two pairs of most similar clusters are merged until we are left with the entire set S as a single cluster. (Line 10)
4. The optimum cluster is chosen using CH-index by optimizing the inter-cluster and intra-cluster distances. The clustering that minimizes the intra-cluster distance and maximizes the inter-cluster distance is chosen as the optimum clustering. (Line 11)
5. After we have chosen the optimum clustering, we choose the cluster the queried concept belongs to as the frame. The concepts that belong to the same cluster as the query are highly similar to the query, and the concepts that are assigned to different clusters are the ones different from the query. Figure 4.7 shows the cluster that was returned as the frame for `convolutional_neural_network` using our algorithm. The original frame i.e. the set of concepts mentioned in the Wikipedia page for `convolutional_neural_network` consisted of concepts such as `python` or `C`, which now do not appear in the cluster that `convolutional_neural_network` belongs to. (Line 11)

We determine the frame for every candidate prerequisite of q using the above technique.

2. Determining Prerequisites. We then use the prerequisite function $RefD$ to compute q 's prerequisites using the frames that we constructed for the concepts in the previous step. $RefD$ or the reference distance between two concepts A and B is defined as:

$$RefD(A, B) = \frac{\sum_{i=1}^k r(c_i, B) \cdot w(c_i, A)}{\sum_{i=1}^k w(c_i, A)} - \frac{\sum_{i=1}^k r(c_i, A) \cdot w(c_i, B)}{\sum_{i=1}^k w(c_i, B)} \quad (4.1)$$

where C is the concept space, each concept c_i in C has a weight $w(c_i, A)$ associated with A , and the references are encoded by setting $r(c_i, B) = 1$, if B is referenced by c_i in either Wikipedia pages, books or cited in papers. $RefD$ uses a threshold $thresh$ to determine prerequisite relationship between A and B . Using this formula, a concept A is a prerequisite of concept B in the following way:

$$RefD(A, B) \in \begin{cases} (thresh, 1], & \text{if B is a prerequisite of A} \\ [-thresh, thresh], & \text{if no prerequisite relation exists} \\ [-1, -thresh), & \text{if A is a prerequisite of B} \end{cases} \quad (4.2)$$

Using $RefD$ and the new frames, we determined the prerequisite relationship between the query and all the candidate prerequisites. Our set of prerequisites are denoted by P_C . (Lines 2–7 of Algorithm 6).

3. Augmenting Prerequisites from a Similar Concept. We propose the idea of improving the recall of retrieved prerequisites P_C by identifying similar concepts and adding their prerequisites to the set of prerequisites obtained for q . We make use of TeKnowbase taxonomy for the same. TeKnowbase taxonomy helps us identify other concepts which share the same parent as that of q . These concepts are called the *siblings* of the query. TeKnowbase identifies the most similar sibling for q using cosine similarity of the vector representations of the concepts trained using Node2Vec [75]. Some examples of similar concepts that we obtained are listed in Table 4.2. For concepts which do not have any siblings, we consider the space of all concepts ranked according to their cosine

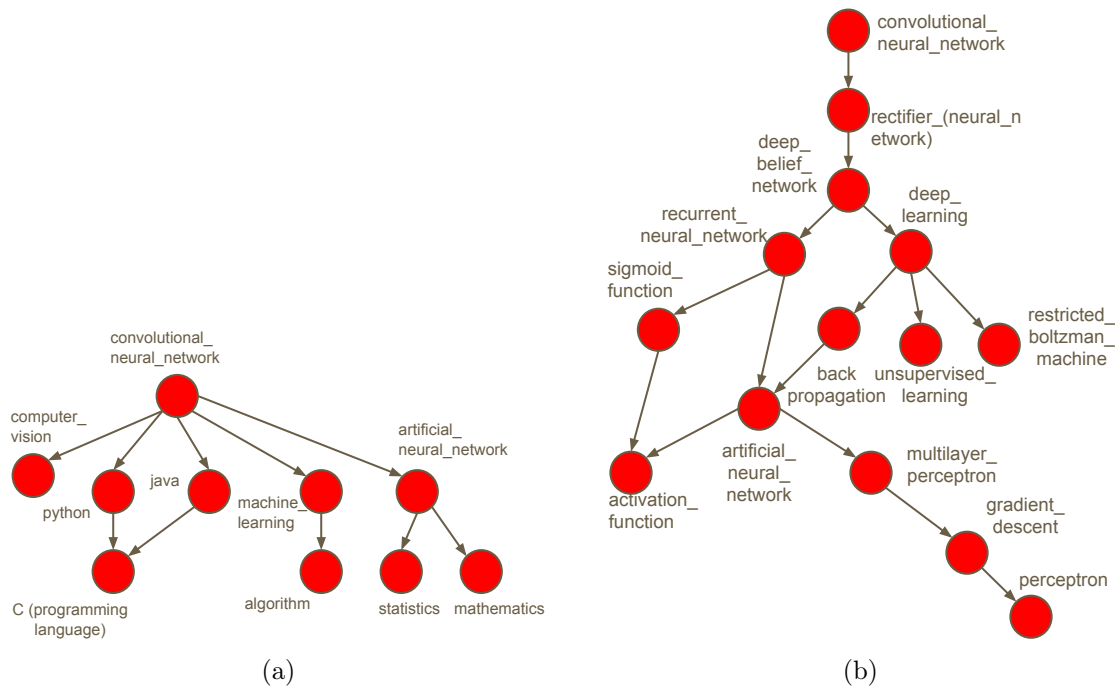


Figure 4.8: Prerequisite graph obtained for `convolutional_neural_network` using different frame representations shown in Table 4.7. (a) Prerequisite graph obtained using the original frame (b) Prerequisite graph obtained using the frame generated using our technique

similarity value to q . We follow the following procedure—we maintain a list of similar concepts ranked according to their cosine similarity with q . We compared the list of prerequisites retrieved for q with that of the most similar concept. Let P_D be the set of prerequisites retrieved for the similar concept. We set a parameter α which is the minimum number by which we want to increase the prerequisites retrieved for q using similar concepts. If $|P_D - P_C| < \alpha$, then we move to the next similar concept and add its prerequisites. We repeat this procedure until the number of new prerequisites added to P_C is greater than α . Continuing with the example, `time_delay_neural_network` was retrieved as the most similar concept to `convolutional_neural_network`. The prerequisites retrieved for `time_delay_neural_network` consisted of `feedforward_neural_network` and

Concept	Similar Concept
<code>convolutional_neural_network</code>	<code>time_delay_neural_network</code>
<code>bloom_filter</code>	<code>quotient_filter</code>
<code>longitudinal_redundancy_check</code>	<code>cyclic_redundancy_check</code>
<code>canonical_lr_parser</code>	<code>simple_precedence_parser</code>
<code>space_hierarchy_theorem</code>	<code>time_hierarchy_theorem</code>

Table 4.2: Similar concepts

`decision_boundary`, which were not retrieved for `convolutional_neural_network`. Adding these prerequisites improves the recall of prerequisite retrieval for the query (Lines 10–25 of Algorithm 6). We experimentally demonstrate this in Section 4.5.

4. Constructing the Prerequisite Graph P_G , Pruning and Generating Order.

For every pair of concepts in P_C , we further determined if one is a prerequisite of the other using the prerequisite function. We also removed those pairs of prerequisites (a, b) where a was a descendant in TeKnowbase taxonomy of B . For instance, `binary_search_tree` cannot be a prerequisite of `binary_tree`. We remove cycles by removing the edge that has the lowest score returned by the prerequisite function (Lines 27–33). A topological sort of the graph returns a reading order for q . Figure 4.8 (a) shows the prerequisite graph constructed for `convolutional_neural_network` using RefD. RefD considers all the concepts mentioned in the Wikipedia page of `convolutional_neural_network` in its frame. This includes `keras` or `torch`. As a result, concepts such as `python`, `C` or `java` were returned as the prerequisites. The prerequisite graph constructed using the frame generated by our technique is shown in Figure 4.8 (b). It does not consist of prerequisites such as `python` or `C`, but concepts like `backpropagation` or `perceptron` are present, which are necessary prerequisites.

4.4.2 Generating and Ranking Facets for Prerequisites

The previous section described techniques to construct a prerequisite graph of necessary prerequisites. In this section, we describe techniques to extract facets of prerequisites for the queried topic. We formulate this as a retrieval problem where the system first extracts candidate facets and then ranks them based on their relevance to the query.

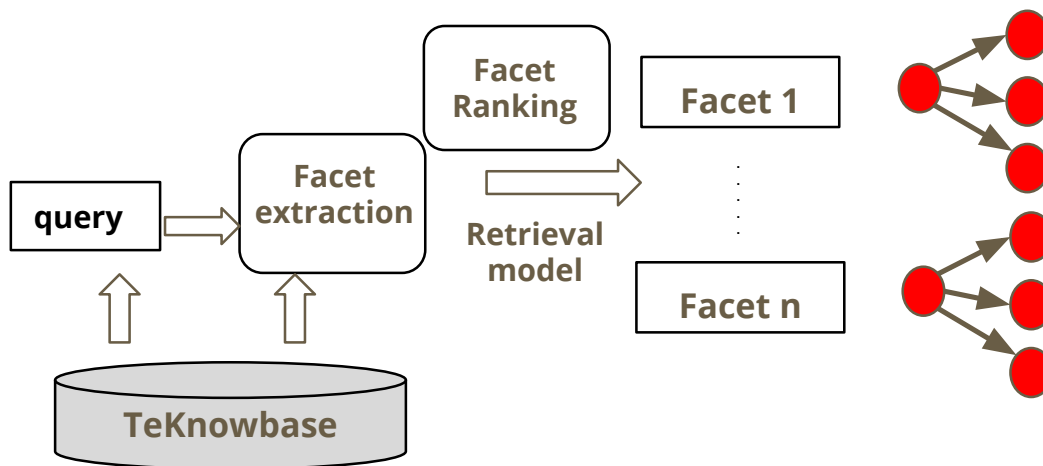


Figure 4.9: Components of the Facet Extraction and Ranking System

4.4.2.1 Components of Facet Extraction and Ranking System

Figure 4.9 shows the components of the facet extraction system. An overview of these components is as follows:

1. **Facet extraction.** As already described in Section 4.1, the quality of facets retrieved by existing techniques is not the best. So, we need a different technique to extract these facets. The facet extraction component extracts candidate facets for the query entered by the user. This component is described in detail in Section 5.1.3.1.

2. **Retrieval model.** This component uses language models to model the query as well as the facets. We propose the idea of estimating language models for the query as well as the facets and ranking the facets based on their similarity to the language model of the query. The language model for the query is estimated using a prerequisite function, aspect-based retrieval model described in Chapter 3, and TeKnowbase. This component is described in detail in Section 4.4.2.3.
3. **Ranking of facets.** This component ranks the set of candidate facets based on their relevance with the query. It uses KL-divergence measure to rank the facets. This component is described in detail in Section 4.4.2.4.
4. **TeKnowbase.** TeKnowbase is the backbone of our facet extraction system. It helps in the extraction as well as the modeling the relevance of the facets.

We now describe the 3 main components of our system elaborately in the next subsection.

4.4.2.2 Facet Extraction

A facet is a group of prerequisites that describe some aspect of the query. Existing techniques retrieved facets of poor quality because the concepts in each facet were not highly similar to each other. To address this issue, we used TeKnowbase to ensure that highly similar concepts belonged to a facet. We used the relevant documents returned in the top positions for the query to generate candidate facets. The key idea used is to extract frequently occurring key-phrases from these documents and cluster them to generate facets. Below we describe in detail these two steps:

1. **Extracting Key-Phrases from Documents.** To extract key phrases relevant to the query, we followed a similar approach that was used to extract candidate query suggestions for a query (described in Chapter 3, Section 3.4.4). We indexed the Open

Research Corpus dataset¹ on Galago² and retrieved top-1000 documents for a query. We then used Rake³, a tool to extract and score essential phrases from a document. To further clean the list of extracted phrases, we performed two data cleaning operations as follows—i) we retained only those phrases that had a length of at most 5, and ii) phrases with a score (returned by Rake) less than five were removed from the set of candidate phrases. The main reason for using phrases instead of entities as facets is because phrases capture the context better than entities. Consider the two phrases—“applications like robot navigation” and “robots using new camera technologies” retrieved for query `computer vision`. The phrases mention entities from TeKnowbase like `robot navigation`, `robots` and `camera`, which are highly relevant for the *application* aspect for `computer vision` by our aspect-based retrieval model (already described in Chapter 3 and later described in Section). Additionally, both these phrases consist of terms such as `applications` and `using` which are not entities in TeKnowbase, but are also highly relevant terms for the *application* aspect for `computer vision` by our aspect-based retrieval model. So, using words along with entities helps us model the relevance of key phrases.

2. Clustering Key-Phrases into Facets. The next step is to cluster these phrases and obtain a list of candidate facets. To cluster these phrases into semantically related groups, we used a hierarchical clustering algorithm with complete linkage to extract our facets. Please note that any clustering algorithm can be used to generate clusters of facets. Before clustering, we have to represent the phrases as feature vectors. So, we used a bag of entities representation for the key-phrases along with a bag of words. The phrases were tagged with entities from TeKnowbase. Additionally, this set was expanded by adding entities situated at a 1-hop distance from already tagged entities in TeKnowbase. This was done to capture better context. Consider the phrase `using backpropagation` retrieved for `artificial neural networks`. `backpropagation` will be tagged in this phrase. The triple $\langle \text{backpropagation, type, algorithm} \rangle$ exists in TeKnowbase, so `algorithm` exists in the

¹<https://allenai.org/data/s2orc>

²<https://www.lemurproject.org/galago.php>

³<https://pypi.org/project/rake-nltk/>

1-hop neighborhood of `backpropagation`, and will be added to its feature set. So, it will now have high similarity to other phrases containing `algorithm`, and will appear in the same cluster as them. The distance between the phrases was measured using cosine distance. We adopted the agglomerative strategy for hierarchical clustering which is a bottom-up approach. Each data point starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. While merging two clusters, The Complete-Link Clustering technique [50] considers the similarity of two clusters as the similarity of their most dissimilar members. The set of candidate facets are the clusters returned by the hierarchical clustering algorithm.

4.4.2.3 Retrieval Model

The next step is to estimate language models for the candidate facets extracted from the previous step. We use TeKnowbase and the Open Research Corpus [15] to estimate a language model using the relevance equation (Equation 4.3), and then rank the candidate facets using a probabilistic framework described in Algorithm 7. Algorithm 7 takes as input the query q , τ , the number of facets to be returned, λ , the mixing parameter to combine the two components of relevance equation, and F , the set of candidate facets. It greedily selects the best facet at each iteration that balances the trade-off between its relevance to the query and dissimilarity to the facets already retrieved. We discuss our framework in detail in the following section.

1. Modeling Facet Relevance. The overall relevance of a facet is determined by a query-dependent and independent component. A desirable facet should contain necessary as well as soft prerequisites for the query, which is modeled by the query-dependent component. At the same time, a facet should consist of a large number of highly similar concepts, which is determined by the query-independent component. The overall relevance is determined by combining these two components. Both of them are described as follows:

Algorithm 7: Probabilistic framework for PreFace (λ, τ, F, q)

```

1  $S = \emptyset$ 
2  $f^* = \operatorname{argmax}_{f \in F} Q(f) * \frac{1}{D_{KL}(PreFace||M_f)}$ 
3  $S = S \cup \{f^*\}$ 
4 while  $|S| < \tau - 1$  do
5    $f^* = \operatorname{argmax}_{f \in F \setminus S} Q(f) * \frac{D_{KL}(M_f||M_S)}{D_{KL}(PreFace||M_f)}$ 
6    $F = F \setminus \{f^*\}$ 
7    $S = S \cup \{f^*\}$ 
8 end
9 return  $S$ 

```

1. **Query dependant facet relevance.** The query dependent facet relevance is denoted by $PreFace(w|q)$. This equation is represented as:

$$PreFace(w|q) = \lambda Preq(w|q) + (1 - \lambda) \sum_i P(q_i|q)P(w|q, q_i), \quad (4.3)$$

where $\sum_i P(q_i|q) = 1$. The query dependent facet relevance is modeled by two components. The first component is determined by the prerequisites of q , estimated using prerequisite functions. It is denoted by $Preq(w|q)$, which is called the prerequisite probability. This models the probability of a term w appearing in a prerequisite of q . The second component models the relevance for different aspects of the query. It is denoted by $\sum_i P(q_i|q)P(w|q, q_i)$, where each aspect for q is denoted by q_i . These two components are described as follows:

- (a) **Prerequisite probability.** $Preq(w|q)$ models the probability of a word w appearing in a prerequisite of q . Any existing prerequisite function can be used to estimate this probability.
- (b) **Query and aspect probability.** We also have to identify the aspects of the query, and then model the relevance of a term for each aspect. This is modeled by the second component of our relevance equation. Given an aspect

q_i , the relevance of a term given a query q and aspect q_i is modeled using ASK, an aspect-based retrieval model, proposed in Chapter 3 [190]. The query and aspect component is denoted by $P(w|q, q_i)$ and is modeled as follows:

$$P(w|q, q_i) = \gamma P_{ind}(w|q_i) + (1 - \gamma) P_{dep}(w|q, q_i) \quad (4.4)$$

where $P(w|q_i)$ is the component determined by the aspect alone and $P(w|q, q_i)$ is the query dependent component, determined by both the query and the aspect. The final relevance is given by a mixture of the two components. γ is used to mix these two components to determine the final probability distribution. This model has been discussed in detail in Chapter 3.

2. **Query independent facet relevance.** The query independent facet relevance models the relevance of the facet, independent of the query. This is generally determined by the strength of similarity between key phrases in a facet. At the same time, a facet that contains a large number of similar key phrases is more important than one that contains a lesser number of key phrases provided the strength of similarity between the two is the same. So, we use the size of the facet as a measure of its quality, after a similarity threshold is applied to it. The query independent facet relevance is modeled by $Q(f)$. This component is used while ranking the facets.

2. Modeling Facet Diversity. The user should be recommended facets that are diverse to one another. This means that if a facet describing the algorithms for understanding `convolutional_neural_network` appears in the top-k retrieved facets, it is desirable that they are not suggested again. This is done by returning facets whose language model diverges the least from the query and the most from the language model of already retrieved facets. This is achieved by the following equation:

$$\frac{D_{KL}(M_f||M_S)}{D_{KL}(PreFace||M_f)} \quad (4.5)$$

where M_f is the language model of a facet f , M_S is the language model representation of the set of facets already retrieved and PreFace is the language model for the query. $D_{KL}(M_f||M_S)$ is the KL divergence between the language model for the facet and the language model of already retrieved facets. $D_{KL}(PreFace||M_f)$ is the KL divergence between the language model for the query and the language model of the facet. They are calculated according to the following equation:

$$D_{KL}(M_f||M_S) = \sum_w M_f(w) \log \frac{M_f(w)}{M_S(w)} \quad (4.6)$$

$$D_{KL}(PreFace||M_f) = \sum_w PreFace(w) \log \frac{PreFace(w)}{M_f(w)} \quad (4.7)$$

The language model of a facet f and set S is described by the following equations. These models are smoothed using additive smoothing techniques.

$$M_f(w) = \frac{tf(w, f) + 1}{length(f) + |V|} \quad (4.8)$$

$$M_S(w) = \frac{\sum_{f_i \in S} tf(w, f_i) + 1}{\sum_{f_i \in S} length(f_i) + |V|} \quad (4.9)$$

In other words, the greater the divergence between the language model of the facet and language model of S , the better is the facet and the lower the divergence between the query language model and the facet language model, the better the facet.

3. Estimation of Components. In this section, we describe the techniques to estimate the probabilities that are used in our probabilistic framework.

1. **Estimating $Preq(w|q)$.** This component models the probability of words likely to appear in the prerequisites of the query. To estimate this component, we can use any standard prerequisite function from the literature. For our work, we used the already introduced prerequisite function, $RefD$ (or reference distance) [119], because it is a simple, unsupervised metric and gives good results. $RefD$ takes two concepts as input and returns a score for the prerequisite relationship between them and the concepts that return a score greater than $thresh$ are considered as prerequisites. Given such a prerequisite function that returns a score between pairs of concepts, we estimated $Preq(w|q)$ as follows:

$$Preq(w|q) = \sum_i \beta_i Preq_s(w|s_i) \quad (4.10)$$

where s_i is the entity in that shares the same parent in TeKnowbase taxonomy. $Preq_s(w|s_i)$ is estimated as follows:

$$Preq_s(w|s_i) = \frac{\sum_{e_k \in ent(w)} RefD(e_k, s_i)}{\sum_{e_k \in C} RefD(e_k, s_i)}, RefD(e_k, s_i) > thresh \quad (4.11)$$

where C is the concept space and $ent(w)$ is the set of entities that contain the term w in TeKnowbase. The intuition behind using a mixture of $Preq_s(w|s_i)$, where s_i is an entity that shares the same parent as q is that siblings in TeKnowbase taxonomy have similar prerequisites, and we can make our distribution more accurate by including concepts that were not otherwise returned as prerequisites for the query.

2. **Estimating query and aspect probability.** The second component in Equation 4.3 models the query and aspect probability. To estimate this component, we have to identify aspects for the query. These aspects should be both highly relevant to the query have high coverage. We estimated this component using TeKnowbase. The following steps describe the procedure to estimate the second component using TeKnowbase.

- (a) **Acquiring and scoring the set of entities.** We used the set of key-phrases

retrieved using the procedure described in Section 1. to extract the set of relevant entities E . We tagged entities from TeKnowbase in those key-phrases and scored each entity $e \in E$ using the following formula:

$$score(e) = co_occ(e, q)RefD(e, q), RefD(e, q) > thresh \quad (4.12)$$

where $co_occ(e, q)$ counts the number of times e and q have appeared together in a document across all the relevant documents retrieved for q . $RefD(e, q)$ returns the score depicting how much is e a prerequisite of q .

- (b) **Using TeKnowbase entities as aspects.** Having a set of entities E , we have to partition it into groups such that each group is highly relevant to the query as well as is semantically similar to each other. We can then choose a representative from each group as an aspect for q . To do this, we used the links in TeKnowbase, assuming that entities connected to each other in the knowledge base are semantically similar to each other. We achieved this by creating an induced sub-graph G on TeKnowbase using the set of entities in E and then applied the star clustering algorithm [17] on G to cluster entities into groups. The Star Clustering algorithm works on a graph and takes a similarity threshold σ as input. It retains edges that have strength greater than σ and then clusters the nodes of the graph into groups. It is a greedy algorithm that chooses the node with the highest degree first and assigns all its adjacent nodes to its cluster. It repeats the procedure with other nodes that have not yet been assigned to any cluster. Star clustering algorithm is fast with a run time complexity of $O(V + E)$ and does not need the number of clusters as input. In TeKnowbase, all links are assumed to have a strength of 1, so σ was set to 1. After retrieving the set of clusters, we scored each cluster using the following equation:

$$score(C) = \sum_{c_i \in C} score(c_i) \quad (4.13)$$

where $score(c_i)$ is the score of each entity calculated using Equation 4.12. We retained the top-10 clusters and chose the highest scored entity in each cluster

as an aspect. The clusters which contain more entities appearing frequently with the query will get a higher score. The highest scored entity in that cluster can be used as a representative.

- (c) **Using TeKnowbase relationships as aspects.** Apart from using entities in TeKnowbase as aspects, we expanded the set by adding relationships from TeKnowbase as aspects. We used the aspects that were used in [190], namely *algorithm*, *application* and *implementation*, and also additionally used two relationship types—*type* and *technique*, as the aspects.
- (d) **Estimating query independent aspect probability.** We estimated the query-independent aspect probability by explicitly querying the aspect alone and retrieving the top documents. To further clean the list, we retained only those documents that mention the aspect term in the title or abstract. Having this set of documents D , we estimated the query independent aspect probability as follows:

$$P(w|q_i) = \frac{1}{|D|} \sum_{d \in D} \frac{tf(w, d)}{\sum_{w' \in d} tf(w', d)} \quad (4.14)$$

- (e) **Estimating query dependent aspect probability.** We estimated this component from both the search results and TeKnowbase. To estimate this component from search results, we explicitly queried for q and q_i and retrieved top ranked documents. To further make our estimation accurate, we only retained those documents that contained both the query and aspect terms in either the title or the abstract. We then estimated $P(w|q, q_i)$ as follows:

$$P_{docs}(w|q, q_i) = \frac{1}{|D|} \sum_{d \in D} \frac{tf(w, d)}{\sum_{w' \in d} tf(w', d)} \quad (4.15)$$

where D is the set of documents with each document containing the query and the aspect term both. To improve the accuracy of our estimation, we made use of TeKnowbase. TeKnowbase consists of entities connected to q via relation described by the aspect q_i . So, the words appearing in entities connected to the q via q_i should be given a higher probability. However, TeKnowbase is

sparse, and only using those links to estimate this probability will not result in an accurate estimation. So, we used the approach adopted in [190] and Chapter 3 for improving the estimation using meta-paths in TeKnowbase. The key idea used is to go through the pairs of entities related via a given relationship type described by the aspect q_i and extract the set MP of meta-paths between them. We then scored every meta-path in MP according to their frequency. Having a set of meta-paths $MP = P_1, P_2, \dots, P_n$, we computed the score for each node reachable from source e_i using the Path Constrained Random Walk Algorithm [111] as follows:

$$score_{e_i}(e_j) = \alpha_1 h_{e_i, P_1}(e_j) + \alpha_2 h_{e_i, P_2}(e_j) + \dots + \alpha_n h_{e_i, P_n}(e_j) \quad (4.16)$$

where α_i 's are weights for each of these paths. We set $\alpha_i = count(P_i)$ in Equation 4.16. Since we have to estimate a probability distribution, we convert $score_{e_i}(e_j)$ to a probability distribution using softmax and denote it as $DI_{e_i}(e_j)$.

$$DI_{e_i}(e_j) = \frac{\exp(score_{e_i}(e_j))}{\sum_{e_k \in |E|} \exp(score_{e_i}(e_k))} \quad (4.17)$$

where E is the set of entities in TeKnowbase. This is converted into $P_{KB}(w|q, q_i)$ as follows:

$$P_{KB}(w|q, q_i) = \sum_e DI_e(q) \quad (4.18)$$

where $w \in terms(e)$. We then compute the query and aspect probability by mixing the probability distributions given by Equation 4.15 and Equation 4.18. The final probability for query and aspect, both, is given as follows:

$$P(w|q, q_i) = \alpha P_{docs}(w|q, q_i) + (1 - \alpha) P_{KB}(w|q, q_i) \quad (4.19)$$

- (f) **Estimating the probability of aspect given a query.** This models how important is the aspect q_i to q . All the aspects can be given equal probability

or the probability can be proportional to $score(q_i)$, described by Equation 4.12. Given a set of aspects, the second component of the final relevance equation is a linear combination of the query and the aspect probabilities, described as follows:

$$\sum_i P(q_i|q)P(w|q, q_i) \quad (4.20)$$

The final relevance equation is a mixture of Equation 4.20 and Equation 4.10 and the facets are ranked using Equation 4.21.

4.4.2.4 Ranking of Facets

The facets are re-ranked at each iteration, until the desirable number (τ) of facets are retrieved. This is described by the following equation:

$$f^* = \operatorname{argmax}_{f \in F \setminus S} Q(f) * \frac{D_{KL}(M_f || M_S)}{D_{KL}(PreFace || M_f)} \quad (4.21)$$

where $Q(f)$ is the quality of a facet, described in the previous section. The facet retrieved at this position is removed from the pool of candidate facets and added to S .

4.4.2.5 Item Ranking

After the facets have been ranked, we have to rank the entities in the facet to be shown as prerequisites to the user. To do so, we first identified a representative element re for each facet. We tagged entities in the facet and chose the most frequently occurring entity as the representative. After choosing the representative, we computed the score for each entity e tagged in the facet as follows: $freq(e)sim(e, re)$, where $freq(re)$ is the number of times e appears in that facet, and $sim(e, re)$ is the normalized cosine similarity between the vector representations of entities e and re . These vector representations were obtained by training the Node2Vec algorithm on TeKnowbase. The entities in each facet are then

ranked in decreasing order of this score.

4.5 Experiments

In this section, we discuss our experiments to evaluate the quality of the prerequisite graph as well as faceted prerequisites. The source code for our technique is available at:

<https://bitbucket.org/prajnaupadhyay/preface>

4.5.1 Experiments for Necessary Prerequisites

In this section, we describe the experiments to compare the quality of prerequisites retrieved by our technique (henceforth referred to as OWN_KB) as well as the baselines (listed in Section 4.5.1.2).

4.5.1.1 Setup

We used TeKnowbase and a Wikipedia dump of 01 December 2015. The prerequisite function *RefD* proposed in [119] was used along with our technique used to generate the prerequisite graph described in Section 4.4.1. We note that *any* prerequisite function can be used here, and our knowledge base-based technique will remain the same.

4.5.1.2 Baselines

Apart from comparing our results with the results generated by an existing prerequisite function, we also compared our results with other techniques that can help us obtain a

better frame for the query. We used different representations for the entities in TeKnowbase obtained from different sources, such as using Word2Vec on free text. These vector representations can then be provided as input to our algorithm to cluster the existing frame. This would compare how well existing representations, such as vectors obtained using Word2Vec compare with knowledge graph embedding representations, in generating a better frame for the query.

RefD: This is the prerequisite function described in [119]. It used the entire Wikipedia page as a frame.

OWN_TEXT: We first trained phrase vectors for entities in TeKnowbase using Word2Vec. We used the text present in the Wikipedia page of each entity as input to the Word2Vec algorithm. The embeddings generated are used in our technique as described in Section 4.4.1.

OWN_COMBINED_{CONCAT}: This used the concatenation of the embeddings generated from TeKnowbase and Word2Vec described in Section 4.4.1.

OWN_COMBINED_{SUM}: The representation here was obtained by taking a sum of the vector representations generated from TeKnowbase and Word2Vec.

OWN_KB: This is our technique, outlined in Section 4.4.1, that uses the pruned frame generated using embeddings from the knowledge base and adds the prerequisites of a similar entity.

4.5.1.3 Benchmarks

We selected 35 concepts as our benchmark queries which cover different sub-topics of computer science like data structures, algorithms, systems, and machine learning. They are listed in Table 4.3.

Table 4.3: Benchmark queries

Sr no.	Query
1	adaboost
2	adaptive_huffman_coding
3	associative_array
4	avl_tree
5	b-tree
6	beam_search
7	bidirectional_search
8	binary_heap
9	binomial_heap
10	bloom_filter
11	brownboost
12	cache_invalidation
13	canonical_lr_parser
14	circular_buffer
15	clock_synchronization
16	convolutional_neural_network
17	d-ary_heap
18	database_index
19	finalizer
20	fractal_tree_index
21	logitboost
22	longitudinal_redundancy_check
23	luhn_mod_n_algorithm
24	message_passing
25	multi_level_feedback_queue
26	quotient_filter
27	radix_tree
28	reference_counting
29	serializability
30	space_hierarchy_theorem
31	time_hierarchy_theorem
32	van_emde_boas_tree
33	vector_clock
34	verhoeff_algorithm
35	wavl_tree

4.5.1.4 Gold Standard

The gold-standard list of prerequisites for the 35 concepts was generated by 2 experts. We divided the concepts between them. At first, we made a list of prerequisites returned by all the techniques (our technique as well as the baselines) for each concept. We showed this list to them and asked them to choose the necessary prerequisites. Later, we asked them to add prerequisites that were not returned by any of the techniques. We ensured that the concepts added by the experts had a Wikipedia page by manually looking for them on Wikipedia because we are using Wikipedia for our experiments. For instance, concepts like `hash_function` and `hashing` which were included as prerequisites for `bloom_filter` were treated as one because there is no separate Wikipedia page for `hashing` (they represent the same concept). Also, the experts included `false_positive` and `false_negative` as prerequisites to `bloom_filter`. On looking for a Wikipedia page for these concepts, we found that there was one combined page—`false_positives_and_false_negatives` describing both these concepts and no separate Wikipedia pages for both. So, we used `false_positives_and_false_negatives` in such cases. Finally, the experts exchanged their list of prerequisites to proofread the list of prerequisites written by the other. Differences were resolved by discussion. The average number of prerequisites was 11 per concept with the minimum and maximum being 8 and 15 respectively.

4.5.1.5 Evaluation Methodology and Metrics

Precision, Recall and F1-Scores. We evaluated the quality of prerequisites first. For each benchmark query, we computed the precision, recall, and F1 scores using the gold standard. We also compared the precision and recall values after using different frames and adding prerequisites from a similar concept.

Quality of Prerequisite Graph. In addition to measuring the accuracy of the nodes in the prerequisite graph, it is also important that we measure the quality of the reading

order that the prerequisite graph generates. To do this, we have to evaluate the edges and edge sequences in the graph. Since we do not have the gold standard values for every pair of concepts in the reading order, we relied on the expert's judgment for the same. We asked two experts to rate each edge (a, b) in P_G on a scale of 1 to 4. That is,

Option 1. When b has to be necessarily studied before a

Option 2. When b is a field of study i.e. a is a concept belonging to the broad area of b

Option 3. When a is a necessary prerequisite of b , i.e. inversely related

Option 4. When b is relevant for implementation or some other aspect of a and should not be suggested as necessary prerequisite for a

We computed the *accuracy* of edges as follows:

$$Acc_1 = \frac{\sum_{e_i \in E_1} I(e_i)}{|E_1|}, \text{ where } I(e_i) = 1, \text{ if the edge } e_i \quad (4.22)$$

is either marked as Option 1, else $I(e_i) = 0$

where E_1 is the set of all edges in the prerequisite graph.

Only evaluating the edges in P_G is not enough. A graph P_G with 5 edges all marked either as Option 1 except one will have 0.8 as the value for Acc_1 . The value of Acc_2 will be 0 if all the edge-sequences of length 2 involve the edge not marked as 1. So, it is important that we also evaluate the edge sequences of length 2. The accuracy of edge-sequences of length 2 is computed as follows:

$$Acc_2 = \frac{\sum_{e_i \in E_2} I(e_i)}{|E_2|}, \text{ where } I(e_i) = 1, \text{ if both the edges in edge sequence } e_i \quad (4.23)$$

are marked as Option 1 or Option 2, else $I(e_i) = 0$

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

Representation Scheme	Precision	Recall	F1-Score
RefD	0.25	0.36	0.25
OWN_KB	0.77	0.31	0.41
OWN_TEXT	0.25	0.12	0.13
OWN_COMBINED _{CONCAT}	0.48	0.24	0.30
OWN_COMBINED _{SUM}	0.39	0.22	0.26

Table 4.4: Precision, Recall and F1 scores obtained after using different frames for different baselines. Pruning the frame improves the precision of necessary prerequisites

Representation Scheme	Precision	Recall	F1-Score
RefD	0.25	0.36	0.25
OWN_KB	0.74	0.60	0.62
OWN_TEXT	0.21	0.15	0.14
OWN_COMBINED _{CONCAT}	0.42	0.37	0.35
OWN_COMBINED _{SUM}	0.35	0.34	0.30

Table 4.5: Precision, Recall and F1 scores obtained after identifying the most similar sibling using OWN_KB, OWN_TEXT, OWN_COMBINED_{CONCAT} and OWN_COMBINED_{SUM} and adding its prerequisites. The prerequisites of siblings from knowledge graph taxonomy are roughly the same and improve recall by a significant margin

4.5.1.6 Results

Precision, Recall and F1 Scores. Table 4.4 compares the precision, recall and F1 scores of different baselines after improving the frames. OWN_KB obtains the best precision of 77%. RefD, OWN_COMBINED_{CONCAT}, OWN_COMBINED_{SUM} and OWN_TEXT obtain 25%, 48%, 39% and 25% precision respectively. Clearly, this is due to using better frames generated by our technique on TeKnowbase. The recall of OWN_KB is 31% and lower than that of RefD but it still performs better in terms of F1-score while the recall obtained by OWN_TEXT, OWN_COMBINED_{CONCAT} and OWN_COMBINED_{SUM} are even worse.

Our technique to identify the sibling/most similar concept and add its prerequisites improved the recall of prerequisite retrieval. We used this idea with different baselines as well—OWN_KB, OWN_COMBINED_{CONCAT}, OWN_TEXT and OWN_COMBINED_{SUM}. Table 4.5 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 77% but the recall was low—only 31%. The recall improved to 60%, almost by a factor of 2 at the cost of bringing the precision down by only 4% and improving the F1-score from 0.41 to 0.62. Hence, it performs the best. However, the improvement is not very significant for OWN_TEXT, OWN_COMBINED_{CONCAT} or OWN_COMBINED_{SUM}. The reason is poor quality of frames retrieved by them that returned non-relevant prerequisites. The recall of OWN_COMBINED_{CONCAT} improved from 24% to 37% at the cost of bringing precision down from 48% to 42% and the F1-score increased from 0.30 to 0.35. The performance of OWN_COMBINED_{SUM} is worse than OWN_COMBINED_{CONCAT} for precision but better in terms of recall. The F1-score retrieved by OWN_COMBINED_{CONCAT} is better than OWN_COMBINED_{SUM}. OWN_TEXT performs the worst. Its recall increased by a small margin—from 12% to 15% at the cost of 4% reduction in precision—25% to 21%, with a meagre increase in F1-score. The reason for it performing poorly is using concepts that occur near to each other in text in the frame, as justified using example of `convolutional_neural_network` in Section 4.4.1. The poor performance of OWN_TEXT led to the sub-par performance of OWN_COMBINED_{CONCAT} and OWN_COMBINED_{SUM} since they are constructed from OWN_TEXT embeddings. Overall, OWN_KB performs better than each of the other techniques due to the better quality of frames.

User Evaluation. Table 4.6 lists the accuracies— Acc_1 and Acc_2 obtained from user evaluation. The accuracies were calculated as described in Section 4.5.1.5. Again, the prerequisite graph generated using OWN_KB obtains the highest accuracy of 84% for edges and 68% for edge sequences of length 2. OWN_COMBINED_{CONCAT} obtains an accuracy of 62% for the edges which is not bad but performs poorly for edge sequences of length 2 (44%). OWN_COMBINED_{SUM} performs similarly and obtains 61% accuracy for edges and 39% for edge sequences of length 2. However, they perform better than

Representation Scheme	Acc_1	Acc_2
RefD	0.39	0.17
OWN_KB	0.84	0.68
OWN_TEXT	0.45	0.25
OWN_COMBINED _{CONCAT}	0.62	0.44
OWN_COMBINED _{SUM}	0.61	0.39

Table 4.6: Results from user evaluation

RefD. This is evident because we earlier found that OWN_COMBINED_{CONCAT} and OWN_COMBINED_{SUM} obtained better precision than RefD. The prerequisite graph generated using OWN_TEXT performs the worst with 45% and 25% accuracies for edges and edge sequences of length 2. Overall, OWN_KB is the winner.

Discussion. Figure 4.10 (a) shows the prerequisite graph generated for `convolutional_neural_network` using the baseline OWN_TEXT. It uses vector representations for entities in TeKnowbase obtained using Word2Vec algorithm. It obtained the vector representations for the entities using the text of the Wikipedia pages for training. Table 4.7 shows the frame that we obtained for `convolutional_neural_network` by clustering the original frame using OWN_KB as well as OWN_TEXT. The frame that was generated for `convolutional_neural_network` using OWN_TEXT contained a number of concepts which should not be present in the frame of `convolutional_neural_network`. However, the frame generated by OWN_KB returned relevant concepts in the frame. The most similar entity for `convolutional_neural_network` identified using text embeddings was `gesture_recognition`. `gesture_recognition` is a concept relevant for the *application* aspect for `convolutional_neural_network`, but it was returned as the most similar concept because they appear together in text multiple times. On the other hand, the most similar concept returned by OWN_KB was `time_delay_network`. The prerequisite graph for `time_delay_network` returned concepts such as `feedforward_neural_network` and `decision_boundary` which were not returned in the prerequisite graph for `convolutional_neural_network`. This led to improvement in recall of retrieved prerequisites.

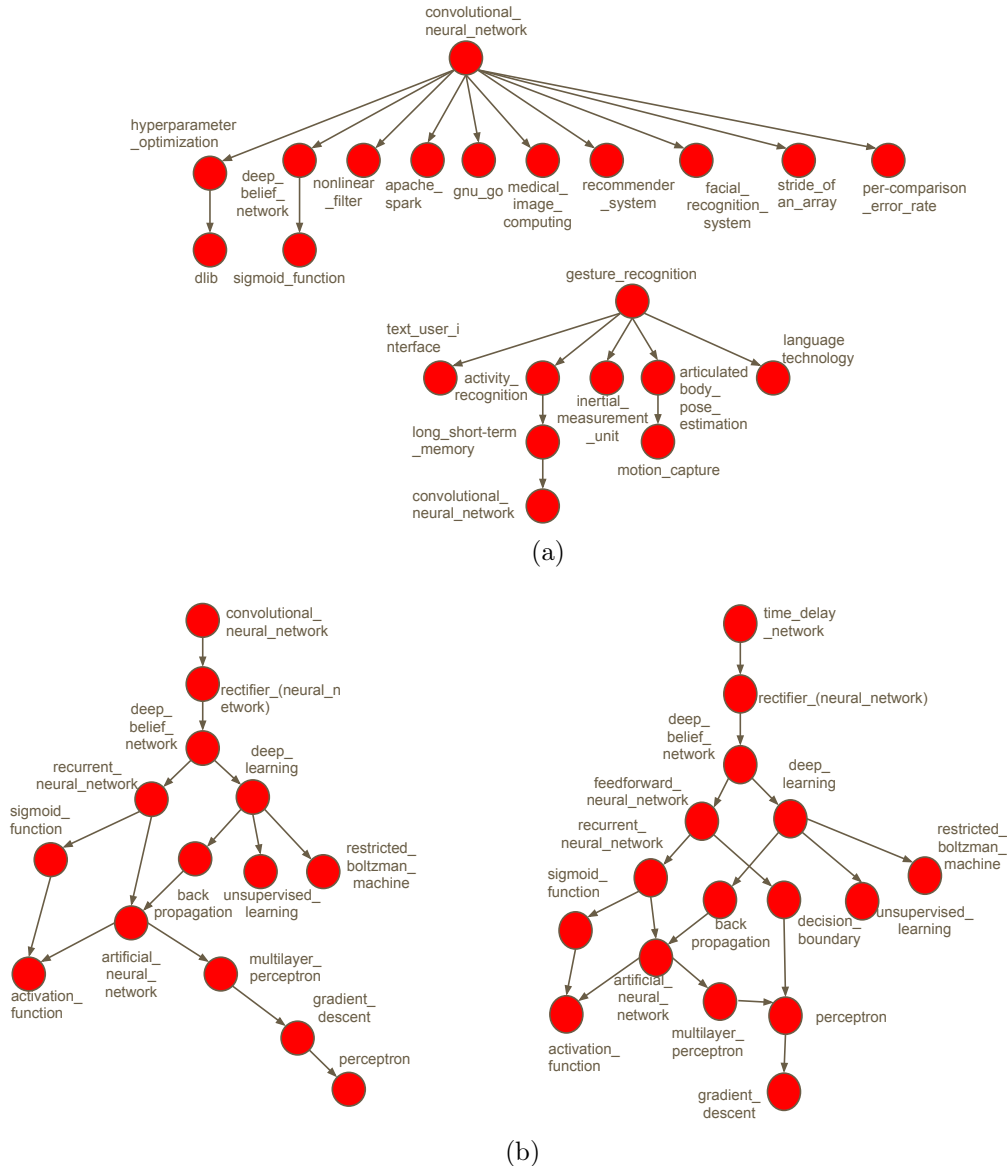


Figure 4.10: Prerequisite graph obtained for `OWN_TEXT` and `OWN_KB` (a) prerequisite graph obtained for `convolutional_neural_network` using `OWN_TEXT`. The most similar concept returned using our techniques and word embeddings for `convolutional_neural_network` was `gesture_recognition` and its the prerequisite graph is also shown in the figure. (b) Prerequisite graph obtained for `convolutional_neural_network` using our technique and embeddings generated from `TeKnowbase`. The figure also shows the prerequisite graph for `time_delay_neural_network`, which was identified as the most similar concept to `convolutional_neural_network`.

Baseline	Frame generated for convolutional_neural_network
OWN_KB	deep_belief_network, artificial_neural_network, rectifier_(neural_networks), multilayer_perceptron, backpropagation, deep_learning, convolutional_neural_network, sigmoid_function, neocognitron, recurrent_neural_network, long_short-term_memory, time_delay_neural_network
OWN_TEXT	c_sharp, apache_spark, per-comparison_error_rate, deep_belief_network, hyperparameter_optimization, long_short-term_memory, reti, facial_recognition_system, neocognitron, c, stride_of_an_array, nonlinear_filter, recommender_system, medical_image_computing, dlib, q-learning, gnu_go

Table 4.7: The frames generated for convolutional_neural_networks by our technique OWN_KB and the baseline OWN_TEXT

Hence, we saw that using embeddings from our technical knowledge base resulted in both improved precision and recall. The improvement in precision was due to better frames. By choosing the entities with similar neighborhoods in TeKnowbase to be added to the frame, we retrieved the necessary prerequisites. The recall improved because we were able to identify the most similar concept to the query and use its prerequisites. We also saw that knowledge graph embeddings performed better than text or text and KB concatenated embeddings in obtaining better precision and recall.

4.5.2 Experiments for Faceted Prerequisites

4.5.2.1 Setup

We experimented with the Wikipedia and the Open Research Corpus datasets. We indexed the Open Research Corpus⁴ on Galago⁵ and used it to retrieve top-1000 results for each query. We evaluated the top-5 facets for each query and in each facet, the top 3 items for each query.

4.5.2.2 Benchmark Queries

We chose 30 queries from the set of topics released by [116] (listed in Table 4.8), which is a set of concepts annotated with their prerequisites. We used these annotations to measure the precision of prerequisites retrieved by our technique as well as the baselines. We additionally conducted user studies to evaluate the precision of concepts that were retrieved but not already in that dataset. We restricted ourselves to queries that have a Wikipedia page because the prerequisite function *RefD* uses Wikipedia to compute reference distance between concepts.

4.5.2.3 Baselines

As already stated in Section 4.1, we can solve the two sub-problems separately using existing state-of-the-art techniques, as in the baselines mentioned below:

1. **QDMKB + RefD.** QDMKB [94] is a state-of-the-art facet retrieval technique for extracting facets from knowledge bases and search results. It improves upon the results of its predecessor, QDMiner [56], and other state-of-the-art techniques QF-I and QF-J

⁴<http://s2-public-api-prod.us-west-2.elasticbeanstalk.com/corpus/>

⁵<https://sourceforge.net/p/lemur/wiki/Galago/>

Sr no.	Query
1	artificial neural network
2	backpropagation
3	collaborative filtering
4	computer vision
5	conditional random field
6	context -sensitive grammar
7	cross entropy
8	dimensionality reduction
9	generative adversarial networks
10	genetic algorithm
11	gradient descent
12	hidden markov model
13	latent dirichlet allocation
14	linear regression
15	logistic regression
16	optical character recognition
17	pagerank
18	probabilistic context-free grammar
19	question answering
20	recursive neural network
21	regular expression
22	reinforcement learning
23	sentiment analysis
24	shallow parsing
25	singular value decomposition
26	spectral clustering
27	speech recognition
28	statistical machine translation
29	word-sense disambiguation
30	word2vec

Table 4.8: Benchmark queries

[102]. It uses a knowledge base for returning facets. QDMKB groups together entities into facets on the basis of their being connected via a sequence of nodes and edges. *RefD* is a state-of-the-art prerequisite function already described in Section 4.4.1. We implemented QDMKB and extracted facets for a query on TeKnowbase. Then, for each facet, we retained only those entities that were returned as prerequisites for the query according to RefD.

2. RefD + TKB. We can also retrieve facets for prerequisites of a query by clustering them into groups. We used TeKnowbase links to cluster the candidate prerequisites into meaningful groups. We obtained the candidates for the query from its top-ranked search results by tagging entities in the phrases extracted from those results. We then constructed an induced sub-graph from those entities on TeKnowbase and applied the star clustering algorithm [17]. This same procedure was used to automatically determine aspects by clustering the entities using TeKnowbase links using the star clustering algorithm. Each cluster was scored according to the following equation. The clusters were then ranked according to $score(C)$ (Equation 4.24).

$$score(C) = \sum_{c_i \in C} RefD(c_i, q), RefD(c_i, q) > thresh \quad (4.24)$$

3. PreFace. This is our technique that extracts facets by clustering them and then ranks by representing them as language models. They are ranked according to a probabilistic framework described by Algorithm 7.

4.5.2.4 Evaluation Scheme

There are two components to evaluate—1) quality of facets 2) quality of prerequisites. Owing to the lack of a standard dataset for evaluation, we used human evaluators (computer science researchers) to evaluate the generated facets. The top 5 facets with 3 items

each were shown to two evaluators (computer science researchers) who evaluated them for their quality.

Evaluation of Facet Quality. To evaluate the quality of facets retrieved, we conducted the following experiments. Each facet was evaluated for the quality of clustering and ranking of facets. We used semantic similarity to measure the quality of clusters retrieved by each technique and DCG (Discounted Cumulative Gain) to measure the ranking of facets. The methodology to evaluate both the qualities is described as follows:

1. **Ranking of facets.** To evaluate the ranking quality, each user was shown a representative concept from each facet and asked if that concept was relevant to the query. To further assist the user, the 3 most similar concepts to the representative concept were also shown to get an idea about the facet. Based on this, the user had to decide if this facet was relevant. Some instructions that were provided to the users are shown in Table 4.9. For *Preface*, the entity that was most frequently occurring in the facet was shown as the representative. For *RefD + TKB*, the representative was the star center of each cluster generated by the star clustering algorithm. For *QDMKB + RefD*, we chose the item in the facet that obtained the highest score for RefD. The relevance scores could be 0: not relevant at all, 1: somewhat relevant, and 2: highly relevant. We then used these scores to compute DCG values to judge the ranking quality.
2. **Clustering Quality.** We have to evaluate the quality of facets generated by our technique as well as the baselines. Our facets were generated by clustering phrases extracted from top-k documents where each phrase was represented using features from TeKnowbase. The competitors are i) a state-of-the-art facet extraction algorithm QDMKB on TeKnowbase combined with the prerequisite function RefD and ii) facets extracted by clustering entities using star clustering algorithm on TeKnowbase. To judge the clustering quality, we asked the user to score the similarity between other entities in the facet to the representative of the facet. The score

Relevance score	Criteria
0	Representative concept is a field of study or is too broad a concept to be a facet topic
1	Representative concept is ambiguous, but the facet as a set of concepts is coherent
2	Representative concept is very related to the query and to the other concepts in the facet

Table 4.9: Criteria for different relevance judgements for ranking quality

could be 2: very similar, 1: somewhat similar, and 0: not similar at all. To illustrate, `quadratic_convex_function` and `general_smooth_nonlinear_function_approximator` are similar to each other because both are functions, whereas `algorithm` and `integer` are not similar to each other. For *PreFace*, we showed the top-3 most similar elements to the representative in the cluster. For *RefD + TKB*, we showed the top-3 entities that were added to the clusters after their star centers were chosen. For *QDMKB + RefD*, the top 3 entities returned according to their prerequisite scores were shown. We computed the score for each pair and normalized the similarity score so that it lies between 0 and 1.

Evaluation of Prerequisites. Apart from evaluating the quality of facets, we also have to evaluate the prerequisites retrieved by our technique. We used the set of prerequisite concept pairs released by [116] as the ground truth. Since the definition of our prerequisites is broader than in earlier work, we performed a user study to judge the relevance of prerequisites not present in the dataset. Each prerequisite was annotated by 2 evaluators (computer science researchers) with scores of 0 or 1. A score of 1 was assigned if the prerequisite was judged to be necessary or supplementing the understanding of the query. The same set of concepts that were shown to the user for evaluating the quality of facets were shown to be evaluated as prerequisites for all the baselines.

Techniques	DCG @5	Cluster similarity
PreFace	5.80	0.95
QDMKB + RefD	4.26	0.80
RefD + TKB	5.56	0.77

(a)

Techniques	Precision
PreFace	0.76
QDMKB + RefD	0.636
RefD + TKB	0.68

(b)

Table 4.10: Tables showing results for a) DCG and cluster similarity values for facet ranking and facet quality, respectively, for all 3 techniques b) Precision of prerequisites retrieved by *PreFace* and competing techniques.

4.5.2.5 Results and Discussion

Facet Quality. Table 4.10 (a) shows the values for facet quality for all 3 techniques. Our technique outperforms the baselines in retrieving better quality of clusters and their ranking.

1. **Clustering Quality.** The average cluster similarity for our technique was 0.98, which is very high and the highest among other techniques. This was possible because of the bag-of-words and entities representation of items using TeKnowbase. QDMKB + RefD obtains a cluster similarity score of 0.8. The reason for it performing worse than *PreFace* is that not all properties return a coherent set of facets. As already pointed out in Section 5.1.3.1, the facets generated by QDMKB + TKB on TeKnowbase are poor in quality, because the sequence of edges that they use as properties may not lead to similar entities, and they also miss out on a number of prerequisites of the query. The property represented by the sequence of edges `implementation_inverse`, `gpgpu`, `application_inverse` from `convolutional_neural_network` leads to `medical_imaging` and `fast_fourier_transform`, which are returned in the same facet, when actually, they are not relevant for the same aspect for `convolutional_neural_network`. `medical_imaging` is an application of `convolutional_neural_network`, while `fast_fourier_transform` is a technique

used by `convolutional_neural_network`. This is clearly not a good clustering because the entities are not semantically similar to each other with respect to the query. *RefD + TKB* obtains a cluster similarity score of 0.77. The reason for the lower similarity value is that the semantics of relations are not considered while clustering the entities. For instance, the 3rd facet retrieved by *RefD + TKB* consists of `programming_language`, `computer_science`, `data_mining`. This set is not a highly coherent set of concepts but has appeared in a single cluster because they are connected to each other in TeKnowbase via the triple $\langle \text{programming_language}, \text{issoftware_notations_and_tools}(\text{relatedTo}), \text{computer_science} \rangle$. As a result, they were assigned to the same cluster by *RefD + TKB*.

2. **Ranking quality.** The ranking quality of the facets is measured using DCG. Our technique outperforms the baselines in the ranking of facets. *QDMKB + RefD* performs worse because it fails to retrieve at least 5 facets for all the queries. *RefD + TKB* performs better than *QDMKB + RefD* but worse than *PreFace*. The reason for this is for most of the queries, some of the facets retrieved at the top positions have a field of study as the representative element, such as suggesting a facet about `statistics` or `mathematics` is not a good idea. Overall, *PreFace* generates good quality facets as compared to its competitors.

Quality of Prerequisites. Table 4.10 (b) shows the precision of retrieved prerequisites. *PreFace* outperforms both the baselines by obtaining a precision of 0.76 across all queries. *RefD + TKB* comes second in retrieving prerequisites to our technique because it uses *RefD* to construct facets. It obtains a precision of 0.68. *QDMKB + RefD* performs the worst because it returns few prerequisites in each facet and the facet is also not relevant to the query. Table 4.11 shows the prerequisites retrieved for `convolutional_neural_network` for *PreFace*, *RefD + TKB* and *QDMKB + RefD*. *QDMKB + RefD* retrieved prerequisites like `control_system` or `integrated_circuit` because they were items in its facets but are not prerequisites. *PreFace* retrieves correct prerequisites like `image`, `algorithm` and `optimization` in facets that are coherent. This shows that our retrieval system is able to return better prerequisites than other techniques.

Table 4.11: Facets containing prerequisites retrieved for `convolutional_neural_network` by PreFace, QDMKB + RefD, and RefD+TKB for top 3 facets

PreFace	QDMKB + RefD	RefD + TKB
algorithm, genetic_algorithm, backpropagation, gradient_descent	artificial_neural_network, genetic_algorithm, support_vector_machine	algorithm, genetic_algorithm, evolutionary_algorithm, logic
function, transfer_function, activation_function, linear_function	artificial_intelligence, parsing	statistics, feature, mathematics, regression_analysis
computer_security, parallel_computer, security_systems, security_applications	machine_learning, rgb_images	programming_language, computer_science, data_mining

4.6 Conclusion

In this chapter, we have developed *PreFace*, a system to automatically generate a prerequisite graph and faceted prerequisites for a concept of interest. To the best of our knowledge, ours is the first system that solves this problem. *PreFace* extracts and ranks facets by using a language model representation for the facets and balancing the trade-off between its relevance and diversity. The probabilities are estimated using a domain-specific knowledge base in Computer Science and a large corpus of research papers. Our evaluation of the results over a benchmark set of queries shows that *PreFace* retrieves better facets and prerequisites than existing state-of-the-art techniques.

Chapter 5

PreFace++: Faceted Retrieval of Prerequisites and Technical Data

We have developed PreFace++ [194], a prototype and an extension of PreFace [193]. Like PreFace, it takes a technical entity as a query and generates a prerequisite graph for the query, consisting of its necessary prerequisites. It also identifies interesting facets for the query and recommends prerequisites for them. The additional functionality included in PreFace++ is the retrieval of research papers and technical posts for the different facets identified by PreFace. It makes use of TeKnowbase, the Open Research Corpus¹ and the StackOverflow dataset² for the same.

5.1 System Architecture

The architecture of PreFace++ is shown in Figure 5.1. It consists of 4 main parts.

¹<https://allenai.org/data/s2orc>

²<https://archive.org/download/stackexchange>

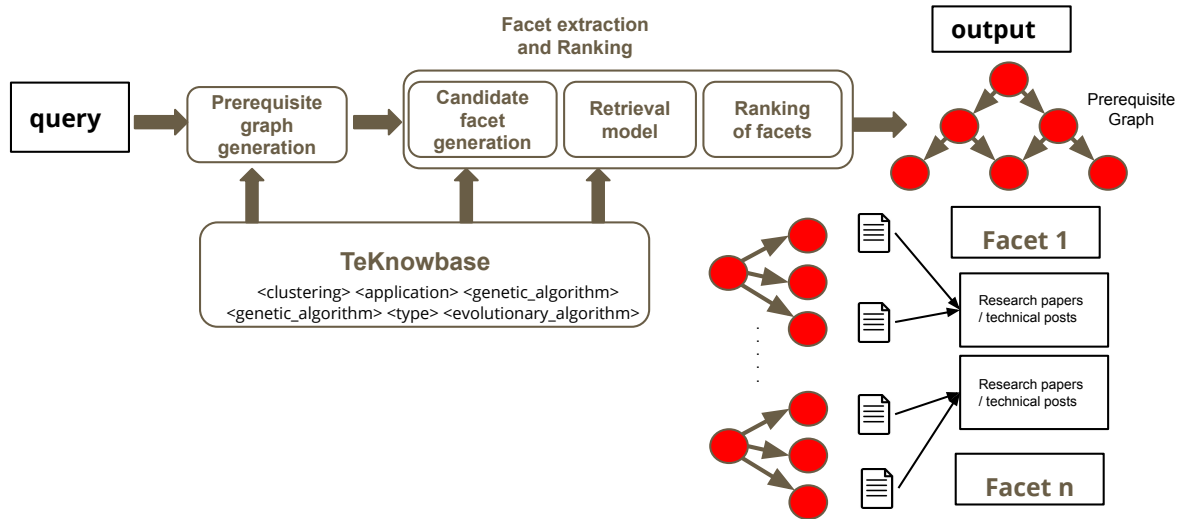


Figure 5.1: Architecture of PreFace++. Apart from returning a prerequisite graph and faceted prerequisites, it also returns research papers and technical posts from StackOverflow relevant to the query and the facets.

5.1.1 TeKnowbase.

TeKnowbase [191] forms the backbone of our system. The construction and evaluation of TeKnowbase have been described in Chapter 2.

5.1.2 Prerequisite Graph Generation

PreFace++ takes a query, which can be any technical entity in Computer Science as input and returns a prerequisite graph. The nodes in the graph are necessary prerequisites and the edges indicate a prerequisite relationship. To construct this graph, we follow the following steps:

1. We constructed a tree by traversing the concepts mentioned in the first paragraph of the Wikipedia page of the query for two hops.

2. However, not all the concepts mentioned in the first paragraphs are prerequisites. To remove such concepts and only retain necessary prerequisites, we first used *RefD* [119]. A drawback of only using *RefD* is that it returns prerequisites for different facets together along with the necessary prerequisites. So, we have to remove concepts belonging to different facets.
3. To remove concepts belonging to different facets from the prerequisite graph, we used the idea of similar neighborhoods. We have observed that prerequisites relevant for different facets of the query share different neighborhoods in TeKnowbase. We captured the idea of neighborhoods using knowledge graph embeddings generated using Node2Vec [75] on TeKnowbase. Node2Vec assigns vector representations closer to each other to concepts that share similar neighborhoods. So, concepts that are prerequisites for different facets get vector representations farther away from the query's. So, we retained only those concepts in the prerequisite graph whose similarity exceeds a given threshold from the queried concept. The cosine similarity between the vector representations of the concepts was used as a measure of the similarity between the concepts. Only those concepts that satisfy the following equations are retained:

$$\text{cosine_sim}(V(q), V(e)) > \text{thresh}, \text{RefD}(q, e) > \theta \quad (5.1)$$

where *thresh* is the similarity threshold between the concepts, θ is the threshold used by *RefD*, and $V(e)$ is the vector representation of concept e obtained by training Node2Vec algorithm on TeKnowbase. The similarity between the concepts is measured using the cosine similarity between their vector representations.

5.1.3 Facet Generation and Ranking

This component determines interesting facets and prerequisites for the query. The details of this procedure have been described in Chapter 4. To make the technique work faster, we used an alternative approach to generate candidate facets.

5.1.3.1 Candidate Facet Generation

The candidate facets were generated by clustering the keyphrases relevant for the query. These key phrases were extracted from the top-k relevant documents for the query in the Open Research Corpus dataset. Additionally, we extracted more keyphrases from posts where the query has been tagged in 14 topics from the StackOverflow dataset. We used the tool RAKE to extract these keyphrases. The procedure to generate candidate facets in Chapter 4 and [193] uses hierarchical clustering with complete linkage, which is a time-consuming operation. So, we used a faster approach for generating candidate facets from the phrases. We describe the steps of this approach as follows:

1. First, we tagged entities from TeKnowbase in keyphrases relevant for the query.
2. Then, we represented the keyphrases using a bag of words and entities and expanded this set by adding entities situated at a 1-hop distance from already tagged entities in TeKnowbase to capture better context.
3. Then, for each entity that was tagged, we chose a set of similar phrases based on the Jaccard similarity of the entity with the phrase until a given similarity threshold was crossed.
4. So, we were finally left with a set of clusters of phrases which were returned as candidate facets.

5.1.3.2 Retrieval

The facets extracted from the previous step and the query were then represented as language models (LMs) [153], which assume that the query and documents are samples of underlying probabilistic processes. To estimate the LM of the query, we used *RefD* and our earlier proposed technique on aspect based retrieval (Chapter 3, [190]) using TeKnowbase. The details about the estimation of these LMs is given in Chapter 4 and [193].

5.1.3.3 Ranking of Facets.

Then, the facets were ranked balancing the trade-off between their relevance to the query and diversity among each other. KL divergence was used as a measure of dissimilarity between the LMs of the facets and the query, described in detail in [193].

5.1.4 Retrieval of Research Papers and Technical Posts

This component returns the following:

1. **Sentences containing the Query and the Facet Terms.** To better understand the relationship between the query and the facet terms, PreFace++ returns sentences from the corpus where they occur together in a sentence.
2. **Relevant Research Papers and Technical Posts.** Not all the prerequisites identified for the facets co-occur with the query in a sentence. So, PreFace++ allows the user to explore research papers from Open Research Corpus and technical posts from StackOverflow relevant for the query and the facet. Given the terms in the query q_t and the terms in the facet f_t , we formulate a query $q_{new} = q_t f_t$ and retrieve the research papers abstracts according to the query likelihood model [153] implemented in Galago. The technical posts from StackOverflow are retrieved by querying for q_{new} in StackOverflow.

5.2 System Implementation

5.2.1 Front End

We used PHP and Javascript to develop the front end and the D3 library to render the prerequisite graph. The user interface provides auto-completion for the query as the user

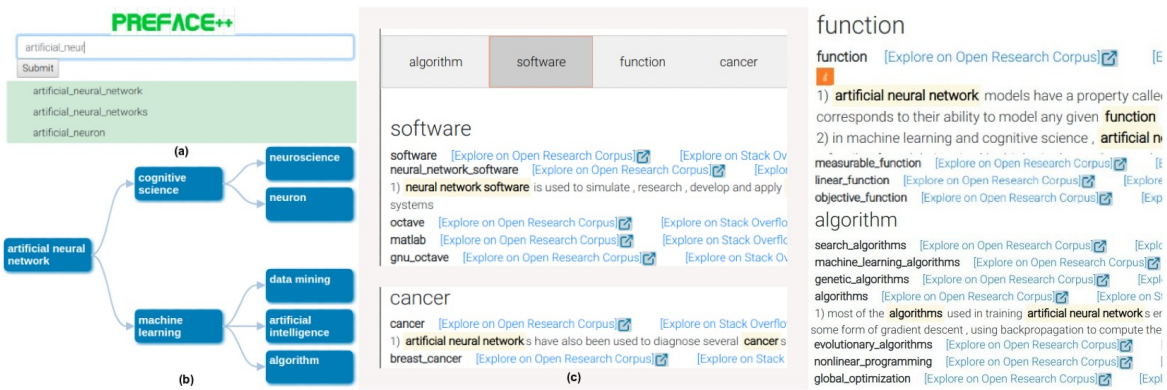


Figure 5.2: (a) Auto-completion options for the partially completed string `artificial_neur`, (b) Prerequisite graph returned for `artificial_neural_network`. The nodes in this graph are the necessary prerequisites. An edge from node `a` to `b` indicates that `b` is a prerequisite of `a`, (c) Two facets (`software` and `cancer`) with their prerequisites extracted for `artificial_neural_network`. The sentences where the facet terms and the query co-occur are also shown.

enters the terms in the search box. These are generated by querying for the entered string in the back-end database (MySQL) of entities using AJAX. Figure 5.2 shows the auto-completion options obtained for the string `artificial_neur`, the prerequisite graph, and 4 facets retrieved for `artificial_neural_network`.

5.2.2 Back End

We used Apache Tomcat and Java Servlets to set up our server. The top-1000 documents relevant for the query were retrieved from Open Research Corpus indexed on Galago. We created indexes on StackOverflow data to retrieve posts relevant to the query. Next, we extracted keyphrases from these posts and documents using a Java implementation of RAKE³. We stored pre-extracted key-phrases and LMs for a set of around 8000 entities. The key phrases and the LMs for the remaining entities were extracted on the fly and results were stored to be reused later. Then, the facets were generated using the technique described in Section 5.1.3.1, represented as language models, and smoothed using additive smoothing techniques. We parallelized the computation of KL divergence of the candidate set of facets to obtain faster results.

5.3 Conclusion

PreFace++ is a prototype of our proposed system PreFace, which assists a user in learning a new topic in the domain of Computer Science. PreFace++ takes a concept as input and returns a prerequisite graph for the concept along with interesting facets towards its understanding. It also allows the user to explore research papers and technical posts related to the query and the identified facets. In the future, we would like to extend this system to automatically generate personalized lecture notes for a query of interest. These will be generated by summarizing the prerequisite graph as well as the facets for

³<https://github.com/Linguistic/rake>

the query.

Chapter 6

Conclusion

In this thesis, we have developed a system to help a user consume a large amount of technical content on the web in a systematic way. Our system assists a user in starting to learn about a technical topic by recommending prerequisites for its basic understanding, and research papers for advanced understanding. The key component of the system is TeKnowbase, which is a knowledge base in the domain of Computer Science. The entities in TeKnowbase were extracted from Wikipedia, domain-specific websites such as Webopedia and TechTarget, and indexes of online textbooks. The relationships were determined using heuristics as well as inferencing from structured as well as unstructured sources. TeKnowbase was evaluated to contain highly accurate triples and has a higher number of domain-specific relationships in Computer Science when compared with well-known knowledge bases such as Freebase or DBPedia. TeKnowbase is freely available.

Using TeKnowbase, we developed PreFace, which generates a prerequisite graph and identifies interesting faceted prerequisites for the query. The facets are generated using TeKnowbase and a large corpus of research papers, and they are ranked using a novel retrieval model, which is estimated from TeKnowbase. The ranking of facets is achieved by balancing the trade-off between the relevance as well as the diversity of the retrieved facets. User evaluation of the pre-requisites as well as the facets show that PreFace outperforms

state-of-the-art pre-requisite retrieval and well as facet generation techniques.

The final component of our work, ASK, allows the user to specify an aspect along with the query to retrieve research papers for advanced understanding of the topic. Our retrieval model which was estimated from TeKnowbase returns better results than state-of-the-art pseudo-relevance feedback, diversification, or neural models. Apart from that, the suggestions generated by our model for the query and the aspect have been evaluated to be better than the suggestions generated using state-of-the-art techniques for the query.

6.1 Future Work Directions

We plan to explore the problems of question answering and automatic summarization in the technical domain. Knowledge graphs are an integral component of such systems. While a number of techniques have been proposed in the open domain, they will not work when used in a technical domain. This is because the open domain knowledge bases do not provide coverage in terms of entities and relations in the technical domain, so we have to use domain-specific knowledge bases like TeKnowbase. The proposed techniques when used on TeKnowbase may not always work since TeKnowbase is sparse, and these techniques are designed with the assumption that the KGs they use are not. Solving these problems in the technical domain will involve inferencing to acquire more triples from structured as well as unstructured sources.

6.1.1 Question Answering in the Technical Domain

To support interactive learning, we plan to include a question answering module that automatically provides answers to natural language questions posed by the user in the technical domain. A preliminary analysis of the questions asked about concepts from Computer Science on Google categorizes them into the following types

- **Factual Questions:** These questions either seek an entity/entities as the answer, for example, `What algorithm does Tensorflow use?`, questions that ask for definitions or reasons, such as `What is adversarial loss`, `Why is GAN hard to train`, questions that seek to know the difference between a pair of entities, for example, `What is the difference between CNN and RNN?`,
- **Yes/No Questions:** For example, `Are GANs unsupervised?`,
- **How-to Questions:** Such as, `How is GAN trained?`.

Such questions involve querying the KB as well as consulting textual sources. Answering such questions requires identifying the relationships mentioned in the question text and mapping them to relations in the knowledge base [10, 215], which may not appear in the exact same surface form. For example, instead of relation `uses` or `problems solved`, the knowledge base may consist of `application` relation. Even after identifying the surface form, the answer may not be available directly in the KB and we may need to use knowledge graph inferencing using embeddings [89]. However, it is not necessary that such embedding techniques will help us because they perform inferencing at the 1-hop level and we will need to perform inferencing over multiple hops in the KB and also involve textual sources since it is sparse.

Additionally, the system will consist of an intelligent evaluation module that will automatically generate questions to test the user's knowledge on the subject, evaluate it, and choose the next best question to ask the user depending on her answer to the question. These questions will be generated based on a learning plan for the subject, and the plans will be generated by identifying the prerequisite relationships between the concepts in the learning plan.

6.1.2 Automatic Generation of Lecture Notes

Apart from identifying the prerequisites, it would also be useful if the user is provided with descriptions of the prerequisites in the order identified by the prerequisite graph. In other words, this component will automatically generate lecture notes for the query entered by the user. This can be achieved by extracting definitions of the concepts and concatenating them in the order identified by the graph. However, this approach does not work due to the following reasons:

- The definitions for topics in Computer Science are not limited to textual definitions, but also include mathematical definitions, diagrams, examples, or code segments. For example, a description for `b-tree` should consist of a diagram, `big-o_notation` can be best described using a mathematical formula, and a `semaphore` can be understood if its pseudocode is also provided. Also, it is not necessary that all these concepts should be accompanied by all of these parts. So, automatically identifying these components for each topic is challenging. One possible way is to use Wikipedia or freely available lecture notes from sources such as Coursera and develop heuristics to extract them. We can search for section headings containing “pseudocode” or “examples” and retrieve the entire section. While efforts have been made to extract aspect-based summaries from document structure [66], all of these deal with textual components, while in our case the components are not purely textual.
- Once we have this information, we have to summarize the prerequisite graph, where the nodes are topics in Computer Science, and the edges are prerequisite relationships. It is important that the order provided by the graph be followed, as well as the relationships between the two adjacent nodes in the graph, should be described as well. Efforts have been made for the summarization of 2-length entity chains in knowledge graphs [40], but summarization of a graph containing more than 2 nodes has not been explored. One way to solve this problem is to generate a candidate list of two-length summaries for pairs of prerequisites in the graph and choose the best summary for the graph using optimization techniques.

Appendices

Appendix A

Agglomerative and Star Clustering

A.1 Agglomerative Clustering

Agglomerative clustering is a type of hierarchical clustering technique that builds a hierarchy of clusters over a set of items. Each item starts as one cluster and pairs of clusters are merged at each iteration, due to which it is also known as bottom-up clustering. Agglomerative clustering consists of the following steps:

1. Compute the distance matrix for items in the set.
2. Consider each item as a cluster.
3. Merge the two closest clusters. Update the distance matrix with the new cluster.
4. Repeat Step 3 until all items have been merged into a single cluster.

Step 3 merges two closest clusters at each iteration. Given two clusters X , Y , and the distance metric between two items, $d(x, y)$, the distance between two clusters can be calculated in multiple ways. We have used two ways to compute the distance between the clusters in this thesis.

Figure A.1: Pairwise distances between item in S

	A	B	C	D	E	F
A	-	1	5	2	3	4
B	1	-	4	2	3	5
C	5	4	-	1	3	2
D	2	2	1	-	2	4
E	3	3	3	2	-	6
F	4	5	2	4	6	-

1. Single Linkage Clustering. This clustering technique considers the distance between two clusters as the distance between their most similar items.

$$D(X, Y) = \min_{x \in X, y \in Y} d(x, y) \quad (\text{A.1})$$

2. Complete Linkage Clustering. This clustering technique considers the distance between two clusters as the distance between their most dissimilar items.

$$D(X, Y) = \max_{x \in X, y \in Y} d(x, y) \quad (\text{A.2})$$

Agglomerative clustering algorithm can be visualized with the help of *dendrograms*. A dendrogram is a tree-like structure where the nodes represent the clusters and the edges represent the merges. Each level of the tree represents one iteration of the algorithm. Figure A.2 shows a dendrogram for the hierarchical clustering of a set $S = A, B, C, D, E, F$. The pairwise distances between the items are shown in Table A.1.

A.1.1 Calinski-Harabasz Index(CH-Index)

To determine the optimum clustering, Calinski-Harabasz index (or CH-index) is used. Given N items and a clustering consisting of K clusters, the Calinski-Harabasz index at

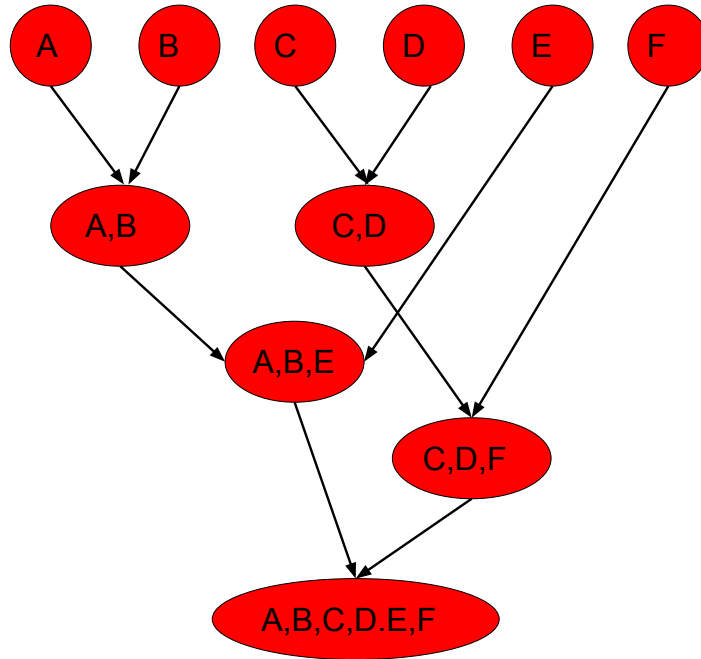


Figure A.2: Dendrogram for complete-linkage clustering algorithm applied on S

K is computed as follows:

$$CH(K) = \frac{(N - K) * inter_c(K)}{(K - 1) * intra_c(K)} \tag{A.3}$$

where $intra_c(K)$ is the intra-cluster distance of the clustering under consideration. It is the average of the within-cluster distance between the items in each cluster. This is expected to be low for a good clustering. $inter_c(K)$ is the average of the distances between the clusters. This should be high for a good clustering. The CH-index is directly proportional to the inter-cluster distance and inversely proportional to the intra-cluster distance. The clustering that obtains the highest value for CH-index is the best clustering. The inter and intra cluster distances are computed as follows:

$$intra_c(K) = \frac{1}{2} \sum_{k=1}^K \sum_{x_i \in X_k} \sum_{x_j \in X_k} d(x_i, x_j) \quad (\text{A.4})$$

$$inter_c(K) = \frac{1}{2} \sum_{k=1}^K \sum_{x_i \in X_k} \sum_{x_j \notin X_k} d(x_i, x_j) \quad (\text{A.5})$$

A.2 Star Clustering

The Star Clustering algorithm was proposed to organize the information associated with a retrieval system. It is used to cluster a set of documents by representing them as nodes of a graph. Given a set of documents D , a *similarity graph* G is constructed by considering each $d \in D$ as a node in G and using the similarity score between the documents as the strength of edges between them. Star Clustering is a well-known graph clustering algorithm. It was initially proposed to cluster a set of documents into meaningful clusters on the basis of their similarity. Given a similarity threshold σ and a graph G generated as described above, the following steps are performed by the star clustering algorithm.

1. Let $G_\sigma = (V, E_\sigma)$ where $E_\sigma = \{e \in E : w(e) \geq \sigma\}$
2. Let each vertex in G_{sigma} initially be unmarked.
3. Calculate the degree of each vertex $v \in V$.
4. Let the highest degree unmarked vertex be a star center and construct a cluster consisting of the star center and all its neighbor vertices. Mark each node in the newly constructed cluster.
5. Repeat Step 4 until all nodes are marked.
6. Represent each cluster by the document corresponding to its associated star center.

The complexity of the algorithm is $O(V + E)$, which is linear in the size of the number of nodes and edges. Star clustering does not need the number of clusters as input and automatically determines the clustering based on the similarity threshold σ .

Appendix B

QDMiner and QDMKB Approach Towards Facet Extraction

QDMiner [56] and QDMKB [94] are techniques proposed to extract facets for a query. These techniques aim to extract different sets of facets for different queries, as opposed to showing a static list of facets. QDMiner uses the search results to generate the set of facets, while QDMKB uses search results as well as knowledge bases to construct the set of facets. These two techniques are described as follows:

B.1 QDMiner

QDMiner uses the top-K documents retrieved for a query to extract query facets. It relies on listing as an important way to structure relevant or coordinate items in documents to mine query facets. The assumptions made by QDMiner are as follows:

- Important information is organized in the form of lists, which may appear in the documents i) explicitly formatted as lists, ii) in sentences separated by commas, or

iii) in tables.

- The lists that are important to the query will occur frequently in the search results returned at the top positions, while the unimportant ones appear infrequently.

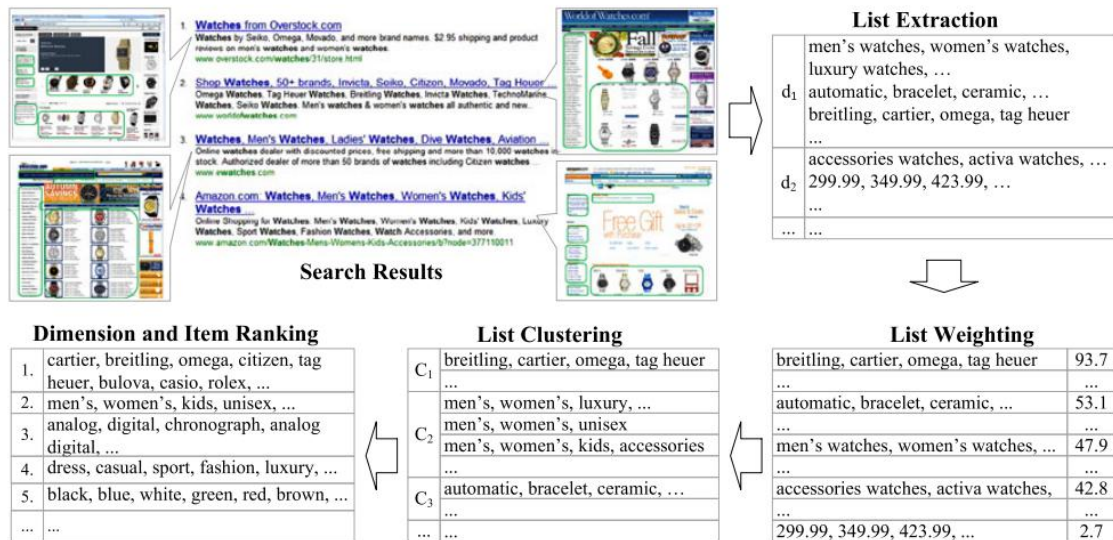


Figure B.1: Architecture of QDMiner

Figure B.1 shows the architecture of QDMiner. It takes a query as input and then retrieves the top-K documents, denoted by the set R . These documents are then fed to the system that has the following components, briefly described as follows:

- **List and Context extraction:** Lists and their context are extracted from the

HTML code or sentences appearing in each document in R . One such example of list extracted is “men’s watches, women’s watches, luxury watches, ...”.

- **List weighting:** The extracted lists are weighted based on their frequency. This is used to identify the important lists, which occur frequently in the documents.
- **List clustering:** To combine similar lists extracted from different documents, they are clustered into facets.
- **Facet weighting:** The clustered facets are re-ranked based on the frequency of the individual items in the facet.

B.1.1 List and Context Extraction

The lists are extracted from the documents based on three different types of patterns—i) free-text patterns, ii) repeat region patterns and iii) HTML tag patterns. Each list is assigned a *container node*, which is the lowest common ancestor of the items contained in that list. The previous and next sibling of the container node is extracted as well.

B.1.1.1 Free Text Patterns

It involves looking for patterns in each sentence of the document. Following patterns were looked for in the sentences:

1. **Pattern 1.** *item, item* (and | or) {other} item*, which is used to retrieve lists of watches from a sentence as follows: *We shop for gorgeous watches from Seiko, Bulova, Lucien, Piccard, Citizen, Cartier or Invicta.* This extracts a list consisting of items **{Seiko, Bulova, Lucien, Piccard, Citizen, Cartier, Invicta}**. The container node in this case is the sentence containing the extracted list.

2. Pattern 2. $\{\text{item} (:\text{-}) .+\$\text{+}\}$. It looks for information present in continuous lines that are comprised of two parts separated by a dash or a colon. An example of a sentence is as follows: *...are highly important for the following reasons:*

Consistency—every fact table is filtered consistently res...

Integration—queries are able to drill different processes ...

Reduced development time to market—the common facets are available without recreating the wheel over again. This extracts a list consisting of the following items: **{Consistency, Integration, Reduced development time to market}**. The container node in this case is the paragraph containing the items.

B.1.1.2 HTML Tag Patterns

This looks for common HTML tags such as **UL**, **OL**, **SELECT** and **TABLE** and extracts items accordingly, as shown in Figure B.2.

B.1.1.3 Repeat Region Patterns

This extracts information that is present in repeated regions in the text, extracted with the help of HTML DOM trees. In this case, a repeat region is a region that includes at least two adjacent or nonadjacent blocks, e.g. M blocks, with similar DOM and visual structures. Then, the leaf HTML nodes within each block are extracted and grouped by their tag names and display styles. Figure B.3 shows examples of a repeat region describing restaurants.

B.1.1.4 Post Processing

Each of these extracted lists is post-processed to remove useless symbol characters, such as '[' and ']', and converting uppercase letters to lowercase. Then, long items, which contain more than 20 terms are removed. Lastly, all lists that contain less than two

```

SELECT:
<select name="ProductFinder2" id="ProductFinder2" >
<option value="WatchBrands.htm" >Watch Brands</option>
<option value="Brands-Accutron.htm" >Accutron</option>
<option value="Brands-Bulova.htm" >Bulova</option>
<option value="Brands-Caravelle.htm" >Caravelle</option>
<option value="Brands-Seiko.htm" >Seiko</option></select>

UL:
<ul><li><a href="/rst.asp?q=dive">Dive</a></li>
<li><a href="/rst.asp?q=titanium">Titanium</a></li>
<li><a href="/rst.asp?q=automatic">Automatic</a></li>
<li><a href="/rst.asp?q=quartz">Quartz</a></li>
<li><a href="/rst.asp?q=gold">Gold</a></li></ul>

TABLE:
<table width="100%">
<tr><td width="10%"></td><td>White</td></tr>
<tr><td></td><td height="20">Red</td></tr>
<tr><td></td><td height="20">Black</td></tr>
<tr><td></td><td height="20">Pink</td></tr>
<tr><td height="4" colspan="2"></td></tr></table>

```

Figure B.2: Examples of HTML tags and the items they contain

unique items or more than 200 unique items are removed.

B.1.2 List Weighting

To remove lists that contain useless items, each list is scored based on the following components:

1. **Document Matching Weight.** Each list l is assigned the following score:

$$S_{DOC} = \sum_{d \in R} (s_d^m \cdot s_d^r) \quad (\text{B.1})$$



Figure B.3: Examples of repeat regions and the information contained in them

where s_d^m is the percentage of items in l present in d , and s_d^r measures the importance of d , and is set to $\frac{1}{\sqrt{\text{rank}_d}}$. This scores lists whose items appear frequently in the top-ranked documents.

2. Average Invert Document Frequency (IDF) of Items. To discount the score of lists that contain very common items that appear in almost every list, the following formula is used:

$$S_{IDF} = \frac{1}{|l|} \sum_{e \in l} idf_e \quad (\text{B.2})$$

where $idf_e = \log \frac{N - N_e + 0.5}{N_e + 0.5}$, where N_e is the total number of documents that contain item e in the corpus and N is the total number of documents.

The final importance of the list is given by the following formula:

$$S_l = S_{DOC} \cdot S_{IDF} \quad (\text{B.3})$$

The lists are sorted in descending order based on this score. The next step involves clustering these lists into facets.

B.1.3 List Clustering

Two lists consisting of a large number of overlapping items can be clustered together into a single facet. The distance between the lists has to be specified for the clustering algorithm. The distance between the lists is calculated as follows:

$$d_l(l_1, l_2) = 1 - \frac{|l_1 \cap l_2|}{\min\{|l_1|, |l_2|\}} \quad (\text{B.4})$$

Here, $|l_1 \cap l_2|$ is the number of common items between l_1 and l_2 . The distance between two clusters of lists is computed as follows:

$$d_c(c_1, c_2) = \max_{l_1 \in c_1, l_2 \in c_2} d_l(l_1, l_2) \quad (\text{B.5})$$

A modified version of the Quality Threshold (QT) Clustering algorithm [95] is used to cluster the lists. QT ensures quality by finding large clusters whose diameters do not exceed a user-defined diameter threshold. It does not need the number of clusters as input. The original algorithm assigns equal importance to all the lists, however, in this scenario, each list has a score. It is necessary that lists that have a higher score should be clustered first. So, WQT (Quality Threshold with Weighted data points) is proposed that can handle this situation. It works as follows:

1. Provide the maximum diameter and minimum weight for the clusters. Denote them by D_{max} and W_{min} , respectively.

2. The next step involves building candidate clusters. Choose the list with the highest score and build a cluster by iteratively adding other lists until the diameter of the cluster surpasses the D_{max} .
3. Add this cluster to the set of candidate clusters if the total weight of the cluster is not less than W_{min} , and remove the lists already added to this cluster from being added to subsequent clusters.
4. Repeat steps 1–3 with the reduced set of lists.

The weight of a cluster is then computed based on the number of websites from which its lists are extracted. If $Sites(c)$ is the set of websites that contain the lists in a cluster c , then $w_c = |Sites(c)|$, where w_c is an indicator of the weight of the cluster.

B.1.4 Facet Ranking

Each generated facet has to be assigned a score based on its importance as well as the importance of the lists contained in the facet. Following formula is used to score each facet:

$$S_c = \sum_{G \in \mathcal{C}} S_G = \sum_{G \in \mathcal{C}} \max_{l \in G} S_l \quad (\text{B.6})$$

where \mathcal{C} is the set of independent groups of lists G present in a facet C . S_G is the weight of a group of lists G a facet, and s_l is the weight of a list l within a group G . S_c is calculated using the techniques described as follows: The lists present in each facet are divided into multiple groups and then the weight is assigned to each group of lists. One way to divide them into groups is to check for unique websites that mention those lists. So, by assigning $\mathcal{C} = Sites(c)$, we can calculate the weight of a cluster using Equation B.6.

This technique assumes that the content of each website is different from other websites. This is rare because websites often convey the same piece of information in different

words. So, similarity or the distance between the lists have to be recomputed based on the content of the websites they are found in. The text of the websites is used to generate a context for each list. So, the text of the website is encoded into a 64-bit fingerprint, and given two lists l_1 and l_2 , the hamming distance $dist(l_1, l_2)$, between the fingerprints of their context is used to calculate the distance.

$$DUP_L(l_1, l_2) = 1 - \frac{dist(l_1, l_2)}{LS} \quad (\text{B.7})$$

where LS is the length of the fingerprint. Given this distance between the two lists, the weighted QT algorithm introduced in the previous section is used to cluster these lists so that duplicates fall into a single cluster.

B.1.5 Item Ranking

The step involves computing the importance of items in a facet. An item is more important if there are multiple highly ranked documents that mention that item. So, the weight of an item in a facet, $S_{e|c}$ as follows:

$$S_{e|c} = \sum_{s \in \text{mathcal{C}}} w(c, e, \mathcal{C}) = \sum_{G \in \mathcal{C}} \frac{1}{\sqrt{avgrank_{c,e,G}}} \quad (\text{B.8})$$

where $w(c, e, \mathcal{C})$ is the weight of the list in group of lists G , $avgrank_{c,e,G}$ is the average rank of item e within all lists extracted from group G . It is computed as follows:

$$avgrank_{c,e,G} = \frac{1}{|L(c, e, G)|} \sum_{l \in L(c, e, G)} rank_{e|l} \quad (\text{B.9})$$

where $L(c, e, G)$ is the set of all lists in c and $G \subset C$. The items within a facet are then ranked according to their weights. An item is called *qualified* if:

$$S_{e|c} > 1 \text{ and } S_{e|c} > \frac{\mathcal{C}(c)}{10} \quad (\text{B.10})$$

The items which are qualified are shown in each facet.

B.2 QDMKB

Approaches such as QDMiner use search results to extract facets. These techniques work when the search results are rich in HTML lists or tables because the initial set of lists are extracted from such elements of the HTML document. In the absence of such meta-data, knowledge bases can be used to extract query facets. QDMKB extracts facets from a large knowledge graph, such as Freebase. Figure B.4 shows the architecture of the system. It has 4 main components described briefly as follows:

- **Facet generation:** This is the first step of facet extraction that retrieves a set of relevant entities $E(q)$ given a query q . Freebase Search API is used for this purpose. After that, the direct properties, as well as the second-hop properties of entities in $E(q)$, are retrieved. All these properties are facet candidates.
- **Facet expansion:** Each facet generated by QDMiner is expanded using the properties in Freebase.
- **Facet grouping:** The generated facets are clustered into final facets so that similar facets are combined.
- **Facet weighting:** The facets are scored based on the frequency of the items appearing in the search results.

B.2.1 Facet Generation

Freebase Search API is used to retrieve relevant entities for the query string qs entered by the user. Top-5 entities are retrieved and the similarity of each of those entities and

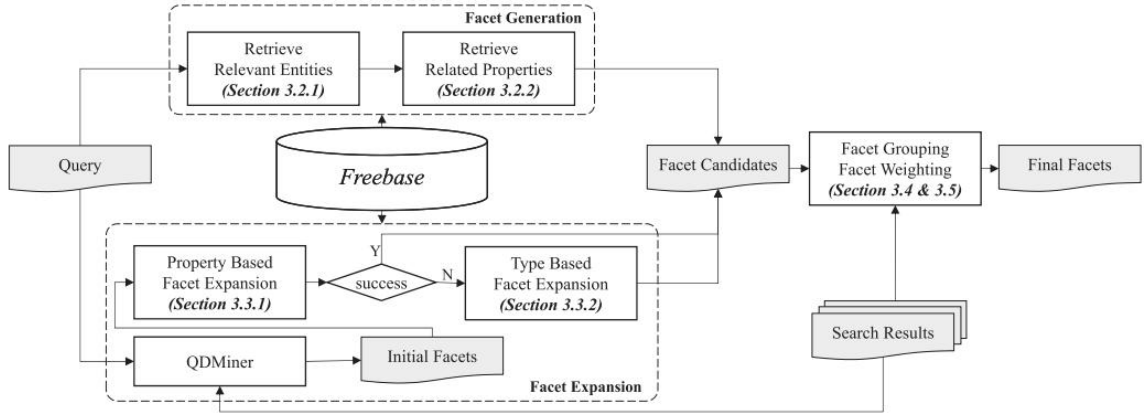


Figure B.4: QDMKB architecture

the query is computed as follows:

$$Sim(e, qs) = \max_{a \in alias(e)} \frac{|a \cap qs|}{|a \cup qs|} \quad (B.11)$$

where $alias(e)$ is an alias of the entity e taken from Freebase, $|a \cap qs|$ is the number of common terms between qs and a and $|a \cup qs|$ is the total number of unique terms in union of terms in qs and a . All entities whose similarity with the query string is less than a threshold are removed from $E(q)$.

After that, related properties of each entity e in $E(q)$ are retrieved. There are two

types of properties—i) direct properties, denoted by $P_1(e)$, and ii) properties of properties, denoted by P_2e . Direct properties consist of the target entities connected to e via some relation/property in Freebase. Direct properties also include the direct properties of e 's parent in the knowledge graph's taxonomy. For example, entity *Steven Spielberg* belongs to types *Film Producer* (*/film/producer*), *Film Director* (*/film/director*), *TV Producer* (*/tv/tv_producer*). Type *Film Director* includes properties such as *Films Directed* (*/film/director/film*) which has targets entities like *Schindler's List*, *Saving Private Ryan*, and *War Horse*. Type *TV Producer* contains properties such as *TV Programs Produced* (*/tv/tv_producer/programs_produced*) which includes target entities like *Animaniacs*, *Band of Brothers*, and *Halo*. These properties and their target entities are called direct properties which could be retrieved by Freebase Topic API. Apart from direct properties, properties of properties can be retrieved. For *Steven Spielberg*, properties of his directed films belong to its second-hop properties.

Given a direct property p entity e , let $V(e, p)$ be the set of target entities connected to e via p . $P_1(e)$ is the set of triples of form $\langle e, p, V(e, p) \rangle$. Let $e_t \in V(e, p)$ be one such target entity let p' be the direct property of e_t . Then, the second hop properties of the entities $P_2(e)$ is constructed by finding out how diverse the entities are connected to these second-hop properties. This is required because sometimes, the target entities connected to peer target entities are duplicated. Following is an example quoted from the paper “for example, *Saving Private Ryan* and *War Horse* are two target entities belonging to the property *Films Directed* of entity *Steven Spielberg*. Property *Genres* (*/film/film/genre*) of *Saving Private Ryan* has target entities including *War Film*, *Action Film*, and *Drama*, whereas the corresponding property of *War Horse* includes *War Film* and *Drama*. In this case, we tend to merge all Genres of different films to form one second-hop property, i.e. *All Genres of Directed Films of Steven Spielberg*. That is to say, if properties p' of each entity e_t in $V(e, p)$ share many common target entities, we tend to treat all these p' as a whole. From the viewpoint of source entity *Steven Spielberg*, the property *All Genres of Directed Films* makes more sense than many separate properties such as *Genres of His Saving Private Ryan* and *Genres of His War Horse*. On the contrary, if properties p' vary from each other, such as *Actors of different films directed by Steven Spielberg*, each

of them will be treated as a separate second-hop property”. The second hop properties have to be diverse enough, and the following formula computes the same:

$$Diversity(e, p, p') = \frac{|\cup_{e_t \in V(e,p)} V(e_t, p')|}{\sum_{e_t \in V(e,p)} |V(e_t, p')|} \quad (\text{B.12})$$

Separate second hop properties are constructed if the Diversity score is greater than 0.5. These second hop properties belong to $P_2(e)$. Finally, both $P_1(e)$ and $P_2(e)$ are merged with each other. If $P(e) = P_1(e) \cup P_2(e)$, then

$$P(q) = \cup_{e \in E(q)} P(e) \quad (\text{B.13})$$

The triples for which $V(e, p) \leq 1$ are removed.

B.2.2 Facet Expansion

This component expands the set of facets already generated by QDMiner using Freebase. A facet $f \in F(q)$ can be expanded using properties or types of the entities in the knowledge base.

B.2.2.1 Property-based Facet Expansion

The first step is to identify the properties that a set of facets, $F(q)$ could most likely belong to. These properties can then be used to expand this $F(q)$. The following formula ranks the properties from the knowledge base most likely to cover the items in $F(q)$:

$$score(e, f, p) = idf(p).sim(e, f, p) \quad (\text{B.14})$$

where $\text{idf}(p)$ is the inverted entity frequency for the property. $\text{idf}(p)$ is computed as follows:

$$\text{idf}(p) = \log \frac{|E|}{|V(e, p)|} \quad (\text{B.15})$$

where E is the total number of entities in Freebase. $\text{sim}(e, f, p)$ is the similarity between the facet and the property.

$$\text{sim}(e, f, p) = \frac{\sum_{I_i \in f} I(|E(I_i) \cap V(e, p)| > 0)}{|f|} \quad (\text{B.16})$$

All properties which obtain a score less than $t_{overlap}$ are removed. The remaining are considered for expansion. If no properties are left after applying the threshold or the returned set is empty, then the technique described in the next section is applied.

B.2.2.2 Facet Expansion Based on Types

The key idea used in this technique is to identify the most suitable type for the item in the facet and then apply constraints to improve the precision of items expanded using that type. At first, all the identified types of items are weighted.

1. Type Weighting. For each entity $e \in E(q)$, let $T(e)$ be the set of its types. The following equation calculates the weight of a type t for facet f :

$$\text{score}(f, t) = \text{idf}(t) \cdot \text{weight}(f, t) \quad (\text{B.17})$$

$$\text{weight}(f, t) = \frac{1}{|f|} \sum_{I_i \in f} \sum_{e \in E(I_i)} \sum_{t_e \in T(e)} \frac{I(t, t_e)}{\sqrt{R(I_i, e)} \sqrt{R(f, I_i)}} \quad (\text{B.18})$$

where $I(t, t_e) = 1$ if t equals to t_e , which means e belongs to type t . $R(I_i, e)$ is the rank of entity e within the retrieved entities for facet item I_i . $R(T_i, e)$ is the rank of facet item I_i within the initial facet. The higher a facet item and an entity are ranked, the higher the

weights of corresponding types of the entity are. $idf(t)$ is the inverted entity frequency of type t , which is $\log \frac{E}{E_t}$, where $|E_t|$ is the number of entities belonging to type t . We remove those entities whose weights falls below a threshold $t_{overlap}$, and then score the remaining ones using Equation B.18

2. Query Dependent Constraint. All the types generated in the previous technique are not correct. To improve the precision of the types, an MQL query was constructed to retrieve entities that belong to the extracted types and have common properties with the target entities. To generate the property and the target entity that will be a part of the MQL query, the following approach is deployed.

All 1st hop properties of each facet item $T_i \in f$ are retrieved from Freebase using the technique described in Section B.2.1. All the properties and their values are then transformed into a list of pairs $\langle p, e_t \rangle$, comprising of the property p and target entity e_t . RThen, only those pairs are retained where the target entity e has at least one alias that is contained by the query q . Then, each of the pair is weighted according to the following equation:

$$weight(f, p, e_t) = \frac{1}{|f|} \sum_{T_i \in f} \sum_{e \in E(I_i)} \frac{HasPair(e, p, e_t)}{\sqrt{R(I_i, e)} \sqrt{R(f, I_i)}} \quad (\text{B.19})$$

$HasPair(e, p, e_t) = 1$, if e is connected to e_t via property p . A threshold is applied to remove low weighted pairs and the highest ranked pair is chosen to apply constraints in the MQL query.

B.2.2.3 Facet Grouping

QT algorithm [107] with complete linkage is used to cluster the generated as well as facets expanded using Freebase so that facets containing similar items get clustered into single ones. The diameter of the largest cluster needs to be specified instead of the number of

clusters, which is calculated as follows:

$$Dia(C) = \max_{f_1 \in C, f_2 \in C} Dis(f_1, f_2) \quad (\text{B.20})$$

$Dis(f_1, f_2)$ is calculated as follows:

$$Dis(f_1, f_2) = \frac{|f_1 \cap f_2|}{\max |f_1|, |f_2|} \quad (\text{B.21})$$

B.2.2.4 Facet Weighting

Each facet is weighted so that it can be ranked. They are weighted in a manner similar to the one used for weighting the lists in QDMiner, as described in Section [B.1.2](#). The facets are finally ranked in decreasing order of weight and shown to the user.

Bibliography

- [1] Resource description framework (rdf): Concepts and abstract syntax.
- [2] *MeSH*, 2009 (accessed December 29, 2020). <https://www.ncbi.nlm.nih.gov/mesh/>.
- [3] *PhySH*, 2009 (accessed December 29, 2020). <https://physh.aps.org/>.
- [4] *ACM Computing Classification*, 2012 (accessed December 29, 2020). <https://dl.acm.org/ccs>.
- [5] Learning in moocs: Motivations and self-regulated learning in moocs. *The Internet and Higher Education*, 29:40–48, 2016.
- [6] Digitization of education in the 21st century, 2018.
- [7] Dramatic growth of open access september 30, 2020, September 2020.
- [8] Steven Abney. Bootstrapping. In *Proceedings of the 40th annual meeting of the association for computational linguistics*, pages 360–367, 2002.
- [9] Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. Automated template generation for question answering over knowledge graphs. *WWW*, pages 1191–1200, 2017.
- [10] Puneet Agarwal, Maya Ramanath, and Gautam Shroff. Retrieving relationships from a knowledge graph for question answering. In *Advances in Information Retrieval*, pages 35–50, Cham, 2019. Springer International Publishing.

-
- [11] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94, 2000.
- [12] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. *WSDM*, pages 5–14, 2009.
- [13] Alan Akbik and Alexander Löser. Kraken: N-ary facts in open information extraction. In James Fan, Raphael Hoffman, Aditya Kalyanpur, Sebastian Riedel, Fabian M. Suchanek, and Partha Pratim Talukdar, editors, *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction, AKBC-WEKEX@NAACL-HLT 2012, Montréal, Canada, June 7-8, 2012*, pages 52–56. Association for Computational Linguistics, 2012.
- [14] Enrique Alfonseca and Suresh Manandhar. An unsupervised method for general named entity recognition and automated concept discovery. In *In: Proceedings of the 1st International Conference on General WordNet*, pages 34–43, 2002.
- [15] Waleed Ammar, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, Ahmed Elgohary, Sergey Feldman, Vu Ha, Rodney Kinney, Sebastian Kohlmeier, Kyle Lo, Tyler Murray, Hsu-Han Ooi, Matthew E. Peters, Joanna Power, Sam Skjonsberg, Lucy Lu Wang, Chris Wilhelm, Zheng Yuan, Madeleine van Zuylen, and Oren Etzioni. Construction of the literature graph in semantic scholar. *NAACL*, 2018.
- [16] M Ashburner, C A Ball, J A Blake, D Botstein, H Butler, J M Cherry, A P Davis, K Dolinski, S S Dwight, J T Eppig, M A Harris, D P Hill, L Issel-Tarver, A Kasarskis, S Lewis, J C Matese, J E Richardson, M Ringwald, G M Rubin, and G Sherlock. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000.
- [17] Aslam, Javed A. and Pelekhov Ekaterina and Rus, Daniela. The Star Clustering Algorithm for Static and Dynamic Information Organization. *Journal of Graph Algorithms and Applications*, 8(1):95–129, 2004.

- [18] Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In Wolfgang Lindner, Marco Mesiti, Can Türker, Yannis Tzitzikas, and Athena I. Vakali, editors, *Current Trends in Database Technology - EDBT 2004 Workshops*, pages 588–596, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- [19] Saeid Balaneshin Kordan and Alexander Kotov. Sequential query expansion using concept graph. *CIKM*, pages 155–164, 2016.
- [20] Saeid Balaneshinkordan and Alexander Kotov. An empirical comparison of term association and knowledge graphs for query expansion. *ECIR*, pages 761–767, 2016.
- [21] M. Barouni-Ebarhimi and A. A. Ghorbani. A novel approach for frequent phrase mining in web search engine query streams. In *Fifth Annual Conference on Communication Networks and Services Research (CNSR '07)*, pages 125–132, May 2007.
- [22] Hannah Bast and Ingmar Weber. The completesearch engine: Interactive, efficient, and towards ir db integration. *CIDR 2007 : 3rd Biennial Conference on Innovative Data Systems Research, University of Wisconsin / Computer Science Department, 88-95 (2007)*, 01 2007.
- [23] Holger Bast and Ingmar Weber. Type less, find more: Fast autocompletion search with a succinct index. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 364–371, New York, NY, USA, 2006. Association for Computing Machinery.
- [24] Michael Bendersky, Donald Metzler, and W. Bruce Croft. Effective query formulation with multiple information sources. *WSDM*, pages 443–452, 2012.
- [25] Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. Query suggestions in the absence of query logs. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 795–804, New York, NY, USA, 2011. Association for Computing Machinery.

-
- [26] Olivier Bodenreider. The unified medical language system (umls): Integrating biomedical terminology, 2004.
- [27] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, and Sebastiano Vigna. Query suggestions using query-flow graphs. In *Proceedings of the 2009 Workshop on Web Search Click Data, WSCD '09*, page 56–63, New York, NY, USA, 2009. Association for Computing Machinery.
- [28] Sergey Brin. Extracting patterns and relations from the world wide web. In *International Workshop on The World Wide Web and Databases*, pages 172–183. Springer, 1998.
- [29] Razvan Bunescu and Raymond Mooney. A shortest path dependency kernel for relation extraction. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 724–731, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.
- [30] Razvan Bunescu and Marius Paşca. Using encyclopedic knowledge for named entity disambiguation. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy, April 2006. Association for Computational Linguistics.
- [31] Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Enhong Chen, and Hang Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, pages 875–883, 2008.
- [32] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an architecture for never-ending language learning. 2010.
- [33] Kaushik Chakrabarti, Surajit Chaudhuri, Tao Cheng, and Dong Xin. A framework for robust discovery of entity synonyms. In *Proceedings of the 18th ACM SIGKDD*

- international conference on Knowledge discovery and data mining*, pages 1384–1392, 2012.
- [34] Tanmoy Chakraborty, Amrith Krishna, Mayank Singh, Niloy Ganguly, Pawan Goyal, and Animesh Mukherjee. Ferosa: A faceted recommendation system for scientific articles. *PAKDD*, pages 528–541, 2016.
- [35] Charles J Fillmore. *Frame semantics*. 2006.
- [36] Jinxiu Chen, Donghong Ji, Chew Lim Tan, and Zhengyu Niu. Unsupervised feature selection for relation extraction. In *Companion Volume to the Proceedings of Conference including Posters/Demos and tutorial abstracts*, 2005.
- [37] Tsangyao Chen and Melissa Gross. Usability modeling of academic search user interface. In *Design, User Experience, and Usability: Understanding Users and Contexts*, pages 16–30, 2017.
- [38] Yang Chen, Pierre-Henri Wuillemin, and Jean-Marc Labat. Discovering Prerequisite Structure of Skills through Probabilistic Association Rules Mining. *EDM*, 2015.
- [39] Gong Cheng, Thanh Tran, and Yuzhong Qu. Relin: Relatedness and informativeness-based centrality for entity summarization. *ISWC*, pages 114–129, 2011.
- [40] Shruti Chhabra and Srikanta Bedathur. Towards generating text summaries for entity chains. In *Advances in Information Retrieval*, pages 136–147, Cham, 2014.
- [41] Paul Clough, Mark Sanderson, Jiayu Tang, Tim Gollins, and Amy Warner. Examining the limits of crowdsourcing for relevance assessment. *IEEE Internet Computing*, 17(4):32–38, 2013.
- [42] Armanand Cohan and Nazli Goharian. Scientific document summarization via citation contextualization and scientific discourse. *International Journal on Digital Libraries*, pages 287–303, 2018.

-
- [43] Mark Craven and Johan Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, page 77–86. AAAI Press, 1999.
- [44] Silviu Cucerzan and Ryen White. Query suggestion based on user landing pages. pages 875–876, 01 2007.
- [45] W. Dakka and P. G. Ipeirotis. Automatic extraction of useful facet hierarchies from text databases. *ICDE*, pages 466–475, 2008.
- [46] Jeffrey Dalton, Laura Dietz, and James Allan. Entity query feature expansion using knowledge base links. *SIGIR*, pages 365–374, 2014.
- [47] Van Dang and Bruce W. Croft. Term level search result diversification. *SIGIR*, pages 603–612, 2013.
- [48] Van Dang and W. Bruce Croft. Diversity by proportionality: An election-based approach to search result diversification. *SIGIR*, pages 65–74, 2012.
- [49] Dmitry Davidov, Ari Rappoport, and Moshe Koppel. Fully unsupervised discovery of concept-specific relationships by web mining. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 232–239, 2007.
- [50] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 01 1977.
- [51] Pedro DeRose, Warren Shen, Fei Chen, Yoonkyong Lee, Douglas Burdick, AnHai Doan, and Raghu Ramakrishnan. Dblife: A community information management platform for the database research community (demo). *CIDR*, pages 169–172, 01 2007.
- [52] Jörg Diederich, Wolf-Tilo Balke, and Uwe Thaden. Demonstrating the semantic growbag: Automatically creating topic facets for faceteddblp. *JCDL*, pages 505–505, 2007.

-
- [53] Heng Ding, Shuo Zhang, Darío Garigliotti, and Krisztian Balog. Generating high-quality query suggestion candidates for task-based search. *ECIR*, pages 625–631, 01 2018.
- [54] Xin Dong, Evgeniy Gabrilovich, Jeremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. *KDD*, pages 601–610, 2014.
- [55] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. Metapath2vec: Scalable representation learning for heterogeneous networks. *KDD*, pages 135–144, 2017.
- [56] Zhicheng Dou, Sha Hu, Yulong Luo, Ruihua Song, and Ji-Rong Wen. Finding dimensions for queries. pages 1311–1320, 2011.
- [57] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. Introducing wikidata to the linked data web. *ISWC*, pages 50–65, 2014.
- [58] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open Information Extraction from the Web. *IJCAI*, pages 2670–2676, 2007.
- [59] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th international conference on World Wide Web*, pages 100–110, 2004.
- [60] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artif. Intell.*, 165(1):91–134, June 2005.
- [61] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

-
- [62] Tobias Falke, Gabriel Stanovsky, Iryna Gurevych, and Ido Dagan. Porting an open information extraction system from English to German. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 892–898, Austin, Texas, November 2016. Association for Computational Linguistics.
- [63] Leila Feddoul, Sirko Schindler, and Frank Löffler. Automatic facet generation and selection over knowledge graphs. *SEMANTICS*, pages 310–325, 2019.
- [64] Sébastien Ferré. Expressive and scalable query-based faceted search over sparql endpoints. *ISWC*, pages 438–453, 2014.
- [65] Marc Feuermann, Pascale Gaudet, Huaiyu Mi, Suzanna E. Lewis, and Paul D. Thomas. Large-scale inference of gene function through phylogenetic annotation of Gene Ontology terms: case study of the apoptosis and autophagy cellular processes. *Database*, 2016, 12 2016. baw155.
- [66] Lea Frermann and Alexandre Klementiev. Inducing document structure for aspect-based summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6263–6273, Florence, Italy, July 2019. Association for Computational Linguistics.
- [67] Evgeniy Gabrilovich and Shaul Markovitch. Feature Generation for Text Categorization Using World Knowledge. *IJCAI*, pages 1048–1053, 2005.
- [68] Wei Gao, Cheng Niu, Jian-Yun Nie, Ming Zhou, Jian Hu, Kam-Fai Wong, and Hsiao-Wuen Hon. Cross-lingual query suggestion using query logs of different languages. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, page 463–470, New York, NY, USA, 2007. Association for Computing Machinery.
- [69] Dario Garigliotti and Krisztian Balog. Generating query suggestions to support task-based search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1153–1156, New York, NY, USA, 2017. Association for Computing Machinery.

- [70] Bela Gipp, Jöran Beel, and Christian Hentschel. Scienstein: A research paper recommender system. *ICETCCT*, pages 309–315, 2009.
- [71] Michael Glass and Alfio Gliozzo. Discovering implicit knowledge with unary relations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1585–1594, Melbourne, Australia, July 2018. Association for Computational Linguistics.
- [72] Google. Freebase data dumps. <https://developers.google.com/freebase/data>.
- [73] Jonathan Gordon, Stephen Aguilar, Emily Sheng, and Gully Burns. Structured generation of technical reading lists. *BEA@NAACL*, pages 261–270, 2017.
- [74] Jonathan Gordon, Linhong Zhu, Aram Galstyan, Prem Natarajan, and Gully Burns. Modeling concept dependencies in a scientific corpus. *ACL*, pages 866–875, 2016.
- [75] Aditya Grover and Jure Leskovec. Node2Vec: Scalable Feature Learning for Networks. *KDD*, pages 855–864, 2016.
- [76] Dhruv Gupta, Klaus Berberich, Jannik Strötgen, and Demetrios Zeinalipour-Yazti. Generating semantic aspects for queries. *JCDL*, pages 162–178, 2018.
- [77] Rasmus Hahn, Christian Bizer, Christopher Sahnwaldt, Christian Herta, Scott Robinson, Michaela Bürgele, Holger Düwiger, and Ulrich Scheel. Faceted wikipedia search. *BIS*, pages 1–11, 2010.
- [78] Xianpei Han and Le Sun. Global distant supervision for relation extraction. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [79] Andreas Harth. Visinav: A system for visual search and navigation on web data. *J. Web Sem.*, 8:348–354, 11 2010.
- [80] Takaaki Hasegawa, Satoshi Sekine, and Ralph Grishman. Discovering relations among named entities from large corpora. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 415–422, Barcelona, Spain, July 2004.

- [81] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. Dynamic factual summaries for entity cards. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 773–782, New York, NY, USA, 2017. Association for Computing Machinery.
- [82] M. A. Hearst. Design recommendations for hierarchical faceted search interfaces. *SIGIR Workshop on Faceted Search*, pages 1–5, 2006.
- [83] Marti A Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. *COLING*, pages 539–545, 1992.
- [84] Philipp Heim, Jürgen Ziegler, and Steffen Lohmann. gfacet: A browser for the web of data. 417:49–58, 2008.
- [85] Ben Hixon, Peter Clark, and Hannaneh Hajishirzi. Learning knowledge graphs for question answering through conversational dialog. *NAACL*, pages 851–861, 2015.
- [86] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. 194:28—61, January 2013.
- [87] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 541–550, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.
- [88] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs, 2020.
- [89] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. pages 105–113, 2019.

-
- [90] Hideki Isozaki and Hideto Kazawa. Efficient support vector classifiers for named entity recognition. COLING '02, page 1–7, USA, 2002. Association for Computational Linguistics.
- [91] James Gregory Jardine. *Automatically generating reading lists*. PhD thesis, University of Cambridge, UK, 2014.
- [92] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
- [93] Qixia Jiang and Maosong Sun. Fast query recommendation by search. In *AAAI*, 2011.
- [94] Zhengbao Jiang, Zhicheng Dou, and Ji-Rong Wen. Generating query facets using knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):315–329, 2017.
- [95] Xin Jin and Jiawei Han. *Quality Threshold Clustering*. 2016.
- [96] Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. *NAACL*, 2015.
- [97] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, page 387–396, New York, NY, USA, 2006. Association for Computing Machinery.
- [98] Nanda Kambhatla. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. ACLdemo '04, pages 178–181, 2004.
- [99] Yoon Kim. Convolutional neural networks for sentence classification. *EMNLP*, 2014.
- [100] Youngho Kim, Jangwon Seo, and W. Bruce Croft. Automatic boolean query suggestion for professional search. In *In SIGIR*, pages 825–834, 2011.

-
- [101] Youngho Kim, Jangwon Seo, and W. Bruce Croft. Automatic boolean query suggestion for professional search. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 825–834, New York, NY, USA, 2011. Association for Computing Machinery.
- [102] Weize Kong and James Allan. Extracting query facets from search results. *SIGIR*, pages 93–102, 2013.
- [103] Jonathan Koren, Yi Zhang, and Xue Liu. Personalized interactive faceted search. *WWW*, pages 477–486, 2008.
- [104] Alexander Kotov and ChengXiang Zhai. Tapping into knowledge base for concept feedback: Leveraging conceptnet to improve search results for difficult queries. *WSDM*, pages 403–412, 2012.
- [105] Georgia Koutrika, Lei Liu, and Steve Simske. Generating reading orders over document collections. *ICDE*, pages 507–518, 2015.
- [106] Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.
- [107] S. Yooseph L. J. Heyer, S. Kruglyak. Exploring expression data: Identification and analysis of coexpressed genes. *Genome Res*, 9(11):1106–1115, 1999.
- [108] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. *ICML*, pages 2863–2872, 2018.
- [109] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML*, 2001.
- [110] Esther Landhuis. Scientific literature: Information overload. *Nature*, 535(7612):457–458, 2016.
- [111] Ni Lao, Tom Mitchell, and William Cohen. Random walk inference and learning in a large scale knowledge base. *EMNLP*, pages 529–539, 2011.

-
- [112] Victor Lavrenko and W. Bruce Croft. Relevance based language models. *SIGIR*, pages 260–267, 2001.
- [113] Joonseok Lee, Kisung Lee, and Jennifer G Kim. Personalized academic research paper recommendation system. *CoRR*, 2013.
- [114] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.
- [115] Douglas B Lenat. CYC: a large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.
- [116] Irene Li, Alexander R Fabbri, Robert R Tung, and Dragomir R Radev. What should i learn first: Introducing lecturebank for nlp education and prerequisite chain learning. *AAAI*, pages 6674–6681, 2019.
- [117] Qi Li and Heng Ji. Incremental joint extraction of entity mentions and relations. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 402–412, Baltimore, Maryland, June 2014. Association for Computational Linguistics.
- [118] Yanan Li, Bin Wang, Sheng Xu, Peng Li, and Jintao Li. Querytrans: Finding similar queries based on query trace graph. pages 260–263, 01 2009.
- [119] Chen Liang, Zhaohui Wu, Wenyi Huang, and C Lee Giles. Measuring prerequisite relations among concepts. *EMNLP*, pages 1668–1674, 2015.
- [120] Chen Liang, Jianbo Ye, Shuting Wang, Bart Pursel, and C Lee Giles. Investigating active learning for concept prerequisite learning. *EAAI*, 2018.
- [121] Chen Liang, Jianbo Ye, Zhaohui Wu, Bart Pursel, and C Lee Giles. Recovering concept prerequisite relations from university course dependencies. *AAAI*, 2017.

-
- [122] Yankai Lin, Shiqi Shen, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. Neural relation extraction with selective attention over instances. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2124–2133, Berlin, Germany, August 2016. Association for Computational Linguistics.
- [123] Xiaohua Liu, Shaodian Zhang, Furu Wei, and Ming Zhou. Recognizing named entities in tweets. HLT '11, page 359–367, USA, 2011. Association for Computational Linguistics.
- [124] Xitong Liu and Hui Fang. Latent entity space: a novel retrieval approach for entity-bearing queries. *Information Retrieval Journal*, 18:473–503, 2015.
- [125] Colin Lockard, Xin Luna Dong, Arash Einolghozati, and Prashant Shiralkar. Ceres: Distantly supervised relation extraction from the semi-structured web. *Proc. VLDB Endow.*, 11(10):1084—1096, June 2018.
- [126] Weiming Lu, Yangfan Zhou, Jiale Yu, and Chenhao Jia. Concept extraction and prerequisite relation learning from educational data. pages 9678–9685, 2019.
- [127] Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. Neural network-based question answering over knowledge graphs on word and character level. *WWW*, pages 1211–1220, 2017.
- [128] Hao Ma, Michael R. Lyu, and Irwin King. Diversifying query suggestion results. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 1399–1404. AAAI Press, 2010.
- [129] Eetu Mäkelä, Eero Hyvönen, and Samppa Saarela. Ontogator — a semantic view-based search engine service for web applications. *ISWC*, pages 847–860, 2006.
- [130] Diego Marcheggiani and Ivan Titov. Discrete-state variational autoencoders for joint discovery and factorization of relations. *Transactions of the Association for Computational Linguistics*, 4:231–244, 2016.

-
- [131] Olena Medelyan, Steve Manion, Jeen Broekstra, Anna Divoli, Anna-Lan Huang, and Ian H. Witten. Constructing a focused taxonomy from a document collection. pages 367–381, 2013.
- [132] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM '08*, page 469–478, New York, NY, USA, 2008. Association for Computing Machinery.
- [133] Filipe Mesquita, Jordan Schmedek, and Denilson Barbosa. Effectiveness and efficiency of open relation extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 447–457, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.
- [134] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arxiv:1301.3781. 2013.
- [135] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. *NIPS*, pages 3111–3119, 2013.
- [136] George A Miller. WordNet - A Lexical Database for English. *Commun. ACM*, 38(11):39–41, 1995.
- [137] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011, Suntec, Singapore, August 2009. Association for Computational Linguistics.
- [138] Makoto Miwa and Mohit Bansal. End-to-end relation extraction using LSTMs on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1116, Berlin, Germany, August 2016. Association for Computational Linguistics.

-
- [139] Makoto Miwa and Yutaka Sasaki. Modeling joint entity and relation extraction with table representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1858–1869, Doha, Qatar, October 2014. Association for Computational Linguistics.
- [140] David Nadeau. *Semi-supervised named entity recognition: learning to recognize 100 entity types with little supervision*. PhD thesis, University of Ottawa, 2007.
- [141] Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. Scalable knowledge harvesting with high precision and high recall. *WSDM '11*, page 227–236, 2011.
- [142] Ndapandula Nakashole, Gerhard Weikum, and Fabian M Suchanek. PATTY - A Taxonomy of Relational Patterns with Semantic Types. *EMNLP-CoNLL*, pages 1135–1145, 2012.
- [143] Robert Neumayer, Krisztian Balog, and Kjetil Nørkvåg. On the modeling of entities for ad-hoc entity search in the web of data. In *Advances in Information Retrieval*, pages 133–145, 2012.
- [144] Dat Quoc Nguyen. An overview of embedding models of entities and relationships for knowledge base completion. 2017.
- [145] Thien Huu Nguyen and Ralph Grishman. Relation extraction: Perspective from convolutional neural networks. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 39–48, Denver, Colorado, June 2015. Association for Computational Linguistics.
- [146] Fedor Nikolaev, Alexander Kotov, and Nikita Zhiltsov. Parameterized fielded term dependence models for ad-hoc entity retrieval from knowledge graph. pages 435–444, 07 2016.
- [147] Feng Niu, Ce Zhang, Christopher Ré, and Jude Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *Int J Semantic Web Inf Syst*, pages 42–73, 01 2012.

-
- [148] Eyal Oren, Renaud Delbru, and Stefan Decker. Extending faceted navigation for rdf data. *ISWC*, pages 559–572, 2006.
- [149] Liangming Pan, Chengjiang Li, Juanzi Li, and Jie Tang. Prerequisite relation learning for concepts in MOOCs. *ACL*, pages 1447–1456, 2017.
- [150] Laura Pappano. The year of the mooc. *The New York Times*, 2(12):2012, 2012.
- [151] Dae Hoon Park and Rikio Chiba. A neural language model for query auto-completion. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1189–1192, New York, NY, USA, 2017. Association for Computing Machinery.
- [152] Elisabetta Poltronieri, Elena Bravo, Moreno Curti, Maurizio Ferri, and Cristina Mancini. Open access publishing trend analysis: Statistics beyond the perception. *Information Research*, 21, 06 2016.
- [153] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. *SIGIR*, pages 275–281, 1998.
- [154] Jeffrey Pound, Peter Mika, and Hugo Zaragoza. Ad-hoc object retrieval in the web of data. *WWW*, page 771–780, 2010.
- [155] Aravind Sesagiri Raamkumar, Schubert Foo, and Natalie Pang. Can I have more of these please?: Assisting researchers in finding similar research papers from a seed basket of papers. *The Electronic Library*, 2018.
- [156] Juan Ramos. Using tf-idf to determine word relevance in document queries. 1999.
- [157] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado, June 2009. Association for Computational Linguistics.
- [158] Hadas Raviv, Oren Kurland, and David Carmel. Document retrieval using entity-based language models. *SIGIR*, pages 65–74, 2016.

- [159] Sebastian Riedel, Limin Yao, and Andrew McCallum. Modeling relations and their mentions without labeled text. In José Luis Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 148–163, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [160] Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. AAI '99/IAAI '99, page 474–479, USA, 1999. American Association for Artificial Intelligence.
- [161] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.
- [162] Alan Ritter, Luke Zettlemoyer, Mausam, and Oren Etzioni. Modeling missing data in distant supervision for information extraction. *Transactions of the Association for Computational Linguistics*, 1:367–378, 2013.
- [163] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009.
- [164] Benjamin Rosenfeld and Ronen Feldman. URES : an unsupervised web relation extraction system. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 667–674, Sydney, Australia, July 2006. Association for Computational Linguistics.
- [165] Rodrygo LT Santos, Craig Macdonald, and Iadh Ounis. Exploiting query reformulations for web search result diversification. *WWW*, pages 881–890, 2010.
- [166] Sunita Sarawagi. Information extraction. *Found. Trends Databases*, 1(3):261–377, March 2008.
- [167] Mohsen Sayyadiharikandeh, Jonathan Gordon, Jose-Luis Ambite, and Kristina Lerman. Finding prerequisite relations using the wikipedia clickstream. *WWW Companion*, pages 1240–1247, 2019.

- [168] Michael Schmitz, Stephen Soderland, Robert Bart, Oren Etzioni, et al. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534, Jeju Island, Korea, July 2012. Association for Computational Linguistics.
- [169] Michael Schmitz, Stephen Soderland, Robert Bart, Oren Etzioni, et al. Open Language Learning for Information Extraction. *EMNLP-CoNLL*, pages 523–534, 2012.
- [170] MC Schraefel, Daniel A Smith, Alisdair Owens, Alistair Russell, Craig Harris, and Max Wilson. The evolving mspace platform: leveraging the semantic web on the trail of the memex. *Hypertext*, pages 174–183, 2005.
- [171] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [172] Dafna Shahaf, Carlos Guestrin, and Eric Horvitz. Metro Maps of Science. *KDD*, pages 1122–1130, 2012.
- [173] Étienne Simon, Vincent Guigue, and Benjamin Piwowarski. Unsupervised information extraction: Regularizing discriminative approaches with relation distribution losses. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1378–1387, Florence, Italy, July 2019. Association for Computational Linguistics.
- [174] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th International Conference on World Wide Web, WWW '15 Companion*, page 243–246, New York, NY, USA, 2015. Association for Computing Machinery.
- [175] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. *NIPS*, pages 926–934, 2013.

-
- [176] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211, Jeju Island, Korea, July 2012. Association for Computational Linguistics.
- [177] Yang Song and Li-wei He. Optimal rare query suggestion with implicit user feedback. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 901–910, New York, NY, USA, 2010. Association for Computing Machinery.
- [178] Emilia Stoica, Marti Hearst, and Megan Richardson. Automating creation of hierarchical faceted metadata structures. *NAACL*, pages 244–251, 2007.
- [179] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago - a core of semantic knowledge. *WWW*, pages 697–706, 2007.
- [180] Kazunari Sugiyama and Min-Yen Kan. A comprehensive evaluation of scholarly paper recommendation using potential citation papers. *Int. J. Digit. Libr.*, pages 91–109, 2015.
- [181] Le Sun and Xianpei Han. A feature-enriched tree kernel for relation extraction. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–67, Baltimore, Maryland, June 2014. Association for Computational Linguistics.
- [182] Y. Sun and J. Han. *Mining Heterogeneous Information Networks: Principles and Methodologies*. 2012.
- [183] Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 455–465, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

- [184] Partha Pratim Talukdar and William W. Cohen. Crowdsourced comprehension: Predicting prerequisite structure in wikipedia. *BEA@NAACL-HLT*, pages 307–315, 2012.
- [185] Jie Tang, Jing Zhang, Limin Yao, Juan-Zi Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *KDD*, pages 990–998, 2008.
- [186] Andreas Thalhammer and Achim Rettinger. Browsing dbpedia entities with summaries. *ISWC*, pages 511–515, 2014.
- [187] Daniel Tunkelang. Dynamic category sets: An approach for faceted search. 2006.
- [188] Peter D. Turney. Expressing implicit semantic relations without supervision. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 313–320, Sydney, Australia, July 2006. Association for Computational Linguistics.
- [189] Tyson-Bernstein, H. *A conspiracy of good intentions: America’s textbook fiasco*. Council for Basic Education, 1988.
- [190] Prajna Upadhyay, Srikanta Bedathur, Tanmoy Chakraborty, and Maya Ramanath. Aspect-based academic search using domain-specific kb. *ECIR*, pages 418–424, 2020.
- [191] Prajna Upadhyay, Ashutosh Bindal, Manjeet Kumar, and Maya Ramanath. Construction and applications of teknowbase: A knowledge base of computer science concepts. *WWW Companion*, pages 1023–1030, 2018.
- [192] Prajna Upadhyay, Tanuma Patra, Ashwini Purkar, and Maya Ramanath. Teknowbase: Towards construction of a knowledge-base of technical concepts, technical report available from <https://arxiv.org/pdf/1612.04988.pdf>, 2016.
- [193] Prajna Upadhyay and Maya Ramanath. Preface: Faceted retrieval of pre-requisites using domain-specific knowledge bases. *ISWC*, pages 601–618, 2020.

- [194] Prajna Upadhyay and Maya Ramanath. Preface++: Faceted retrieval of prerequisites and technical data. *ECIR (Demo Track)*, pages 554–558, 2021.
- [195] Suzan Verberne, Maya Sappelli, and Wessel Kraaij. Query term suggestion in academic search. In Maarten de Rijke, Tom Kenter, Arjen P. de Vries, ChengXiang Zhai, Franciska de Jong, Kira Radinsky, and Katja Hofmann, editors, *Advances in Information Retrieval*, pages 560–566, Cham, 2014. Springer International Publishing.
- [196] Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.
- [197] Xuan Wang, Yu Zhang, Qi Li, Yinyin Chen, and Jiawei Han. Open information extraction with meta-pattern discovery in biomedical literature. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '18, page 291–300, New York, NY, USA, 2018. Association for Computing Machinery.
- [198] Wang, Shuting and Liu, Lei. Prerequisite Concept Maps Extraction for Automatic Assessment. *WWW Companion*, pages 519–521, 2016.
- [199] Bifan Wei, Jun Liu, Jian Ma, Qinghua Zheng, Wei Zhang, and Boqin Feng. Dft-extractor: A system to extract domain-specific faceted taxonomies from wikipedia. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 277–280, New York, NY, USA, 2013. Association for Computing Machinery.
- [200] Aaron Steven White, Drew Reisinger, Keisuke Sakaguchi, Tim Vieira, Sheng Zhang, Rachel Rudinger, Kyle Rawlins, and Benjamin Van Durme. Universal compositional semantics on Universal Dependencies. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1713–1723, Austin, Texas, November 2016. Association for Computational Linguistics.

- [201] Gerhard Wohlgenannt, Albert Weichselbraun, Arno Scharl, and Marta Sabou. Dynamic Integration of Multiple Evidence Sources for Ontology Learning. *JIDM*, 3(3):243–254, 2012.
- [202] Fei Wu, Jayant Madhavan, and Alon Halevy. Identifying aspects for web-search queries. *JAIR*, pages 677–700, 2011.
- [203] Fei Wu and Daniel S. Weld. Open information extraction using Wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 118–127, Uppsala, Sweden, July 2010. Association for Computational Linguistics.
- [204] Sen Wu, Ce Zhang, Feiran Wang, and Christopher Ré. Incremental knowledge base construction using deepdive. *Proceedings of the VLDB Endowment*, 8:1310, 02 2015.
- [205] Chenyan Xiong and Jamie Callan. Esdrank: Connecting query and documents through external semi-structured data. *CIKM*, pages 951–960, 2015.
- [206] Chenyan Xiong and Jamie Callan. Query expansion with freebase. *ICTIR*, pages 111–120, 2015.
- [207] Chenyan Xiong, Jamie Callan, and Tie-Yan Liu. Bag-of-entities representation for ranking. *ICTIR*, pages 181–184, 2016.
- [208] Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for academic search via knowledge graph embedding. *WWW*, pages 1271–1279, 2017.
- [209] YAGO. Yago image. Image available from https://gate.d5.mpi-inf.mpg.de/webbyago3spotlx/SvgBrowser?entityIn=%3CAlbert_Einstein%3E&codeIn=eng.
- [210] Yulan Yan, Naoaki Okazaki, Yutaka Matsuo, Zhenglu Yang, and Mitsuru Ishizuka. Unsupervised relation extraction by mining Wikipedia texts using information from the web. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing*

- of the AFNLP*, pages 1021–1029, Suntec, Singapore, August 2009. Association for Computational Linguistics.
- [211] Yiming Yang, Hanxiao Liu, Jaime Carbonell, and Wanli Ma. Concept Graph Learning from Educational Data. *WSDM*, pages 159–168, 2015.
- [212] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. *NAACL*, pages 1480–1489, 2016.
- [213] Limin Yao, Aria Haghighi, Sebastian Riedel, and Andrew McCallum. Structured relation discovery using generative models. In *proceedings of the 2011 conference on empirical methods in natural language processing*, pages 1456–1466, 2011.
- [214] Alexander Yates, Michele Banko, Matthew Broadhead, Michael J Cafarella, Oren Etzioni, and Stephen Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 25–26, 2007.
- [215] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. pages 1321–1331, Beijing, China, July 2015. Association for Computational Linguistics.
- [216] Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. Relation classification via convolutional deep neural network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2335–2344, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.
- [217] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. *SIGIR*, pages 268–276, 2001.

-
- [218] Ce Zhang, Christopher Ré, Michael Cafarella, Christopher De Sa, Alex Ratner, Jaeho Shin, Feiran Wang, and Sen Wu. DeepDive - Declarative Knowledge Base Construction. *SIGMOD Record*, 45(1):60–67, 2016.
- [219] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *NIPS*, pages 649–657, 2015.
- [220] Yuhao Zhang, A. Chaganty, Ashwin Paranjape, Danqi Chen, J. Bolton, Peng Qi, and Christopher D. Manning. Stanford at tac kbp 2016: Sealing pipeline leaks and understanding chinese. *Theory and Applications of Categories*, 2016.
- [221] Yuhao Zhang, Peng Qi, and Christopher D. Manning. Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2205–2215, Brussels, Belgium, October–November 2018. Association for Computational Linguistics.
- [222] Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 35–45, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.
- [223] H. Zhao, Y. Pan, and F. Yang. Research on information extraction of technical documents and construction of domain knowledge graph. *IEEE Access*, 8:168087–168098, 2020.
- [224] Suncong Zheng, Yuexing Hao, Dongyuan Lu, Hongyun Bao, Jiaming Xu, Hongwei Hao, and Bo Xu. Joint entity and relation extraction based on a hybrid neural network. *Neurocomputing*, 257:59 – 66, 2017. Machine Learning and Signal Processing for Big Multimedia Analysis.
- [225] Nikita Zhiltsov, Alexander Kotov, and Fedor Nikolaev. Fielded sequential dependence model for ad-hoc entity retrieval in the web of data. pages 253–262, 2015.

- [226] GuoDong Zhou and Jian Su. Named entity recognition using an HMM-based chunk tagger. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 473–480, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

List of Publications

1. Prajna Upadhyay, Ashutosh Bindal, Manjeet Kumar, Maya Ramanath. Construction and Applications of TeKnowbase: A Knowledge Base of Computer Science Concepts. In *EKM workshop held with The Web Conference 2018* [Full Paper]
2. Prajna Upadhyay, Srikanta Bedathur, Tanmoy Chakraborty, Maya Ramanath. Aspect-Based Academic Search Using Domain-Specific KB. In *ECIR 2020* [Short Paper]
3. Prajna Upadhyay, Maya Ramanath. **PreFace: Faceted** Retrieval of **Prerequisites** using domain-specific Knowledge Bases. In *ISWC 2020* [Full Paper]
4. Prajna Upadhyay, Maya Ramanath. **PreFace++: Faceted** retrieval of Prerequisites and Technical data. In *ECIR 2021* [Demo Paper]
5. Prajna Upadhyay, Srikanta Bedathur, Tanmoy Chakraborty, Maya Ramanath. **ASK: Aspect-based academic Search** using domain-specific **KB**. Under review at *Journal of Scientometrics* [Full Paper]

Biography

Prajna Devi Upadhyay is a Ph.D student at Indian Institute of Technology, Delhi, India. She completed her M.Tech from National Institute of Technology, Durgapur and B.Tech from Assam University, Silchar.

