Mining How-to Task Knowledge
From Online Communities

Masterarbeit im Fach Informatik
Master’s Thesis in Computer Science
von / by
Cuong Xuan Chu

angefertigt unter der Leitung / supervised by
Prof. Dr. Gerhard Weikum

betreut von / advised by
Dr. Niket Tandon

begutachtet von / reviewers
Prof. Dr. Gerhard Weikum
Dr. Jilles Vreeken

November 2016
Acknowledgements

To do the research and write this thesis, I received assistance from a great number of people. Firstly, I would like to express my sincere gratitude to my supervisor, Prof. Dr. Gerhard Weikum who is also my mentor from the very beginning when I started my Master course at Saarland University. Prof. Dr. Gerhard Weikum have instructed me a lot about specialized knowledge and give me a lot of valuable feedback when I have been doing this research and studying at Saarland University.

I also would like to express my sincere gratitude to my advisor, Dr. Niket Tandon, who directly worked with me. From him, I have gained a lot of valuable experience on the research and advices for my future career.

I also would like to thank my colleagues, who helped me a lot on studying and living at Saarland.

Finally, I would like to thank my families, who have always encouraged and supported me throughout the period of my Master course.

Saarbrücken, November 2016,

Cuong Xuan Chu
Hilfsmittelerklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Non-plagiarism Statement

I hereby confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, 20 November 2016,

Cuong Xuan Chu

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, den 20 November 2016,

Cuong Xuan Chu
Abstract

Nowadays, knowledge graphs have become a fundamental asset for search engines which need background commonsense knowledge for natural interactions. A fair amount of user queries seek information on problem-solving tasks such as painting a wall or repairing a bicycle. While projects like ConceptNet and Webchild have successfully compiled large amounts of knowledge on properties of objects in our daily life, there is still a big gap regarding knowledge on everyday activities, especially problem-solving tasks (how-to knowledge). Recent efforts to automatically compile commonsense have one or more the following weaknesses: (i) they ignore activity commonsense, (ii) they operate at a small scale, (iii) their outputs are not semantically organized, (iv) they are domain-specific (e.g. cooking scripts or movie scripts). All of them lack how-to knowledge.

The goal of this work is to overcome these limitations and compile a large-scale, semantically organized, domain-independent formal knowledge base on tasks and task-solving steps, by tapping the contents of online communities such as WikiHow.

We employ Open-IE techniques to extract noisy candidates for tasks, steps and the required tools and other items. For cleaning and properly organizing this data, we devise embedding-based clustering techniques. The resulting knowledge base, HowToKB, includes a hierarchical taxonomy of disambiguated tasks, temporal orders of sub-tasks, and attributes for involved items.

A comprehensive evaluation of HowToKB shows high accuracy. As an extrinsic use case, we evaluate automatically searching related YouTube videos for HowToKB tasks.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>vii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation and Problem</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Approach and Contribution</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Thesis Outline</td>
<td>5</td>
</tr>
<tr>
<td><strong>2 Background and Related Work</strong></td>
<td>7</td>
</tr>
<tr>
<td>2.1 Knowledge and Knowledge Bases</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1 Encyclopedic Knowledge</td>
<td>7</td>
</tr>
<tr>
<td>2.1.2 Commonsense Knowledge</td>
<td>8</td>
</tr>
<tr>
<td>2.1.3 Activity Commonsense Knowledge</td>
<td>8</td>
</tr>
<tr>
<td>2.2 KB Construction</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 Commonsense KB Construction</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1.1 Manual</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1.2 Semi-automated</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1.3 Automated</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2 Activity Commonsense KB Construction</td>
<td>13</td>
</tr>
<tr>
<td>2.3 KB Organization</td>
<td>15</td>
</tr>
<tr>
<td>2.3.1 Synonym</td>
<td>15</td>
</tr>
<tr>
<td>2.3.2 Hypernym</td>
<td>17</td>
</tr>
<tr>
<td><strong>3 HowToKB System</strong></td>
<td>19</td>
</tr>
<tr>
<td>3.1 HowToKB System Overview</td>
<td>19</td>
</tr>
<tr>
<td>3.2 KB Construction</td>
<td>20</td>
</tr>
<tr>
<td>3.2.1 Concepts and Notation</td>
<td>20</td>
</tr>
<tr>
<td>3.2.2 Information Extraction Method</td>
<td>23</td>
</tr>
<tr>
<td>3.3 KB Organization</td>
<td>25</td>
</tr>
<tr>
<td>3.3.1 Similarity Measures</td>
<td>27</td>
</tr>
<tr>
<td>3.3.2 Combining Features</td>
<td>28</td>
</tr>
<tr>
<td>3.3.3 Clustering Algorithm</td>
<td>28</td>
</tr>
<tr>
<td><strong>4 Experiments</strong></td>
<td>33</td>
</tr>
<tr>
<td>4.1 Experimental Setups</td>
<td>34</td>
</tr>
</tbody>
</table>

ix
Chapter 1

Introduction

1.1 Motivation and Problem

Motivation: In the last decade, many knowledge graphs have been built and become key assets of search engines. The most popular type of them is encyclopedic. These knowledge graphs have knowledge about entities and their relationship, which can be used to answer entity-centric queries, and as semantic features for recommendations in search engines.

However, there is also a large fraction of user queries are not about entities, but rather seek information on problem-solving tasks, so-called how-to queries. Examples are: How to decorate your house for Halloween? How do I repair the flat tire on my bicycle? How can I avoid high altitude sickness on my upcoming trip to the Himalayas? To answer these queries, knowledge graphs need to have commonsense knowledge about human activities, especially tasks and their problem-solving steps. Unfortunately, recent commonsense knowledge graphs totally lack the task-solving knowledge, some of them even ignore knowledge about human activities. This means, all of them do not support this class of queries at all.

Current search engine systems like Google or Bing usually return ten blue links with descending order. To get the answer for these queries, users have to go into each website and read all content from them. However a knowledge base (KB) with well-organized knowledge on tasks and their steps could boost the search result quality and user interaction.
Recently, online communities have compiled extensive information about such how-to tasks, such as WikiHow\(^1\), Snapguide\(^2\), eHow\(^3\), Howcast\(^4\) and others. WikiHow is probably the largest of these communities, which contains a wealth of instructive contents including textual and visual on a wide variety of tasks, breaking them down into sub-tasks with detailed descriptions and pictorial illustrations. This is a great source for humans; however, it is merely in textual form and far from rigorously represented machine knowledge that can directly support how-to queries.

**Problem Statement:** The goal of this work is to fill this void by automatically distilling and semantically organizing task knowledge from how-to communities like WikiHow. For example, painting a wall is a task, along with attributes like participating agents - a human, participating objects (prop) - wall, paint, roller, typical location and time. This knowledge can be constructed in a frame-style representation (task frame), as illustrated in Figure 1.1.

![Figure 1.1: Example of a task frame](image)

Furthermore, the tasks must be semantically organized in taxonomy with relations between task frames, which includes temporal and hierarchical. This means, given a task, we want to know which tasks can happen before and after this task (temporal) or what the parent task is and what the sub-tasks of this task (hierarchical) are. For example,

\(^{1}\)http://www.wikihow.com/
\(^{2}\)https://snapguide.com/
\(^{3}\)http://www.ehow.com/
\(^{4}\)http://www.howcast.com/
from figure [1], cover floor happens before paint a wall and redo floor happens after paint a wall and so on. Semantic organization can make the knowledge base more consistent and cleaner by grouping similar tasks (here, color a ceiling is similar to paint a wall) and inducing the sense of tasks when they are ambiguous. For example, the WikiHow phrase use keyboard leads to one frame about tasks that involve a computer keyboard and another frame about playing music on a keyboard.

1.2 Approach and Contribution

**Approach:** We have developed methods for extracting and cleaning task knowledge from WikiHow. The resulting knowledge base, called HowToKB, contains over half of million tasks and sub-tasks organized into a clean taxonomy. Each task or sub-task is represented by a frame with attributes for parent task, preceding sub-task, following sub-task, required tools or other items, and linkage to visual illustrations. As the text in the raw input is often noisy and ambiguous, we have devised techniques for canonicalizing tasks and items: grouping all information for the same meaning into one frame, and distinguishing different meanings by different frames.

Prior work along these lines is scarce. Encyclopedic KBs like YAGO (Suchanek et al., 2007), Freebase (Bollacker et al., 2008), as mentioned above, are extracted from large datasets like Wikipedia which contains huge amount of events, but do not have sufficient coverage of problem-solving tasks, hence only represent the knowledge about facts between entities. Recently, people are trying to build commonsense knowledge bases which can know all general knowledge that most people possess, for example, Cyc (Lenat 1995), ConceptNet (Liu and Singh, 2004) and Webchild (Tandon et al., 2014). However, all of these KBs also do not contain knowledge about problem-solving tasks.

The most related projects are the work on procedural knowledge by Yang and Nyberg (2015) and the work on Knowlywood (Tandon et al., 2015). Yang and Nyberg (2015) also harnesses WikiHow, but mostly treats it as a text corpus for query expansion. There is no attempt to canonicalize sub-tasks (i.e., identify the same-meaning sub-tasks across different tasks) and to infer structured attributes for tasks. Tandon et al. (2015) constructs an activity knowledge bases from narrative texts like movie scripts. These general activities are not related to how-to tasks since they do not have hierarchical relationship between activities, and the methodology in this work is very different from the current method. Tandon et al. (2015) uses computationally expensive semantic-parsing techniques, whereas HowToKB is largely built by light-weight clustering. Our methodology is a pipeline of methods which include two main phases:
First, we apply Open-IE techniques to WikiHow articles, in order to extract noisy and ambiguous candidates for task and sub-tasks, along with additional information like location, time and participants. The pipeline system from Tandon et al. (2015) can be used to extract these information but it depends on WordNet (Miller, 1995) and VerbNet (Schuler, 2005) dictionaries for sense disambiguation while we do not want to restrict ourselves to WordNet or VerbNet senses. Other systems like SRL systems Christensen et al. (2010) and NER systems (Nadeau and Sekine, 2007) require large amount of training data that we do not have. Meanwhile, Open IE systems can extract these information without depending on any existing resources as well as requiring any training data.

Subsequently, we use judiciously devised clustering techniques to clean and organize these candidates, and to infer attribute values. To canonicalize tasks and sub-tasks, we harness word embeddings to distinguish different meanings of the same phrase (e.g., “use keyboard”). For this phase, using word sense disambiguation techniques (Navigli, 2009) is another option, but once again, these methods also depend on existing resources such as WordNet. Meanwhile, our clustering techniques do not depend on any existing resources, have small run-time and still successfully inducing the sense of task frames.

The quality of the constructed HowToKB knowledge base has been evaluated by crowdsourcing. These experiments evaluate all steps in our system such as extracting candidates for task and sub-tasks, efficiency and effectiveness of our clustering techniques, and the quality of our final knowledge base. The result shows that for all task attributes, we achieve precision above 80 percent, and well above 90 percent for some attributes. As an extrinsic use case, we developed a technique for searching relevant YouTube videos to tasks in HowToKB. By using all information from task frames, we can improve the result on searching related videos.

**Contribution:** The salient contributions of this research are:

- The first method that automatically constructs a large knowledge base on how-to tasks, by tapping into WikiHow data and combining Open-IE techniques for candidate generation with clustering techniques for candidate canonicalization.

- The publicly available knowledge base HowToKB, which contains over a half of million tasks in the form of semantic frames and all data which is extracted from WikiHow. HowToKB is publicly available at [http://tinyurl.com/HowToKB-WWW](http://tinyurl.com/HowToKB-WWW).

- Extensive experiments showing the quality of HowToKB and an improvement on Youtube search for how-to tasks. By using crowdsourcing, a task ground truth was built and can be used for future researches.
The task frames of HowToKB are valuable for use-cases such as problem-solving search like video search (e.g. Youtube), provide background knowledge for human-computer interaction and can aid tasks like visual understanding and generating descriptions for visual contents. Figure 1.2 shows some examples on applications of HowToKB. The developed methodology is general and can be carried over other online communities.

1.3 Thesis Outline

The outline of the thesis is as follows:

- In the current chapter, we have introduced the problem, established the research issues, and outlined the contributions.

- Chapter 2 contains background and reviews general related works on knowledge, knowledge bases, its acquisition and organization.

- Chapter 3 describes our system for building HowToKB in details, which includes the architecture of our system, KB construction and KB organization.

- Chapter 4 discusses the resulting knowledge base, HowToKB, and its use-case.

- Chapter 5 presents the conclusion of the thesis and suggests new research directions for future works.
Chapter 2

Background and Related Work

This section provides an overview of several types of knowledge such as encyclopedic knowledge, commonsense knowledge, activity commonsense knowledge that also includes how-to task knowledge and some knowledge bases (KBs) which present these types of knowledge. We also discuss the state-of-the-art methods to automatically construct KBs, especially activity commonsense KBs.

2.1 Knowledge and Knowledge Bases

2.1.1 Encyclopedic Knowledge

Encyclopedic knowledge is the most popular knowledge which represents facts about instances of classes like person, location, organization, product, etc. These facts are focused on individual entities and describe either (i) encyclopedic knowledge of entities, for instance, (Max Planck isA physicist), (ii) relations between entities, (Max Planck isFatherOf Erwin Planck) and (iii) events involving entities, (Max Planck bornIn Kiel).

There are many large-scale encyclopedic knowledge bases like Freebase, DBpedia or Yago (Bollacker et al., 2008; Auer et al., 2007; Suchanek et al., 2007); they have become major assets for enriching the Web to answer semantically rich queries. These KBs both use Wikipedia as the main source to automatically extract entities and relations between them. Until now, there are billions of facts about millions of entities that were extracted with high accuracy. They are heavily used at large companies such as Google, Facebook, Microsoft, and others.
2.1.2 Commonsense Knowledge

In artificial intelligence research, commonsense knowledge is the collection of facts and information that an ordinary person (children, etc.) is expected to know. Rather than facts about instances of classes, commonsense knowledge embodies facts about classes and concepts. Similar to encyclopedic knowledge, these facts describe either (i) properties of concepts, \((water \text{ hasProperty} \text{ liquid})\), (ii) relations between concepts, \((monitor \text{ partOf} \text{ computer})\), and (iii) interactions between concepts, \((singer \text{ sings} \text{ song})\). Not like encyclopedic knowledge which can be obtained explicitly from the Web text such as Wikipedia or WordNet, commonsense knowledge is presented across multimodal documents, including text, image, videos. That implies why commonsense knowledge is used in many AI tasks, for instance, text mining, object recognition, machine translation, etc.

Some commonsense KBs like Cyc, ConceptNet, Webchild [Lenat [1995] Liu and Singh [2004] Tandon et al. [2014]] have been built and developed towards storing all these commonsense information. Most of them were extracted from Web content, either manually or automatically, with hundred thousands of concepts and millions of facts.

2.1.3 Activity Commonsense Knowledge

Activity commonsense knowledge is a part of commonsense knowledge. As discussed in section 1, activity commonsense knowledge describes all information related to everyday human activities (have a romantic dinner on the beach, shopping, etc.). Activity commonsense knowledge can be considered as interaction between concepts, however, the first concept is usually belong to person classes like men or women. Problem-solving task (how-to knowledge) is also one kind of activity commonsense, which usually involves a specific object and has several sub-tasks (e.g. make a cake, repair a bicycle).

Although many commonsense KBs have been built, there is still a big gap regarding knowledge on everyday activities. Some KBs like Cyc, SUMO [Niles and Pease 2001], ConceptNet contain some activity knowledge but not much. Recently, there is a KB that was built to focus on extracting human activities, which is Knowlywood [Tandon et al. 2015]. Until now, the Knowlywood pipeline produced around one million unique activity instances, grouped into about 500 thousands of activity synsets. However, the general activities from Knowlywood are still not related to how-to tasks.

[https://en.wikipedia.org/wiki/Commonsense_knowledge_(artificial_intelligence)]
2.2 KB Construction

A typical knowledge based system has two types of sub-systems: a knowledge base and an inference engine. The knowledge base represents the knowledge about everything in the world and the inference engine can reason about those knowledge and use rules and other form of logic to deduce new facts or highlight inconsistencies (Hayes-Roth et al., 1983). KB construction is one main step of building a knowledge based system, which involves knowledge acquisition and completion. In this section, we only focus on discussing some popular methods which are used for constructing commonsense KBs.

2.2.1 Commonsense KB Construction

Generally, methods of commonsense KB construction can be classified into three types: manual, semi-automated and automated. In manual approaches, commonsense knowledge is extracted manually by experts or even non-experts using knowledge authoring tools. The semi-automated approaches try to acquire additional knowledge via machine learning approaches based on manually-extracted knowledge before. Meanwhile, in automated approaches, knowledge is automatically extracted from unstructured text via machine learning and natural language processing techniques.

2.2.1.1 Manual

Most of the earliest knowledge-based systems are expert systems, e.g. manually-constructed systems. The advantages of these systems are having high accuracy and easily maintaining. However, building these systems takes a lot of costs and the KBs have low coverage.

Cyc: is a project started in 1984 by Douglas Lenat and is developed by the Cycorp company. The long-standing goal of this project is assembling a comprehensive ontology and knowledge base of everyday commonsense knowledge, to help AI applications perform human-like reasoning. Cyc includes hundreds of thousands of terms and millions of assertions by codifying them using ontology language (CycL). Since each assertion should be considered true only in certain contexts, Cyc puts each of its assertions into one or more explicit contexts. Because of much time consuming, there is a method which was proposed to automatically acquire more knowledge into Cyc and we will talk later.

WordNet: is a lexical database for English (Miller, 1995). Because text understanding task requires information about words and their meanings, WordNet groups English words into sets of synonyms called synsets. It is carefully handcrafted, containing more than 155,000 words, organized in over 117,000 synsets. In WordNet, there are some
semantic relations between words and between word senses. Beside of synonymy, the
basic relation, WordNet also includes hypo/hypernymy which organizes the meanings
of nouns into a hierarchical structure, mero/holonymy which distinguishes component
parts, substantive parts and member parts, and entailment relations between verbs are
also coded in WordNet. WordNet has high accuracy but low coverage of concepts, number
of relations and number of assertions.

**VerbNet:** is a verb lexicon compatible with WordNet but explicitly stated syntactic
and semantic information, using Levin verb classes to systematically construct lexical entries (Schuler, 2005). For each verb class, VerbNet describes relevant thematic roles, semantic restrictions on the arguments and syntactic frames. They are hierarchical domain-independent and mapped to other lexical resources such as WordNet.

### 2.2.1.2 Semi-automated

Semi-automated approaches collect human knowledge faster, easier and more efficient
than manual approaches, but still try to ensure that these knowledge are accurate by
using manual intervention.

**Web-based Cyc:** is a method of using a combination of Cyc and the Web, accessed via
Google, to assist in entering knowledge into Cyc (Matuszek et al., 2005). This method
automatically selects interesting, productive queries generated by CycL, translates them
into one or more English query strings and puts them into search engine systems. From
the search results, the relevant information will be extracted and parsed. If they have
not been in Cyc yet, they will be reviewed by an ontologist or a human volunteer for
accuracy, using a tool specific to that task (Witbrock et al., 2005), and asserted into
the knowledge base if they are correct. By using shallow natural language parsing, this
method allows the retrieval, verification, and review of unconstrained facts at a higher
rate than that achieved by human knowledge representation experts working unassisted.

**ConceptNet:** is a large-scale commonsense knowledge base collected from OMCS database
and other resources like Wikipedia. With OMCS database, at first, a set of extraction
rules were used to map OMCS English sentences which are semi-structured into Con-
ceptNet binary-relation assertions. As a result, nodes in ConceptNet are either verbs,
noun phrases, prepositional phrases or adjectival phrases. These nodes are then nor-
malised by using stemming and spellchecker. In the final step, the assertions will be
“relaxed” by merging or generalisation to make the knowledge base more consistent and
having higher coverage.

In ConceptNet 3 (Havasi et al., 2007), the new architecture was significantly reorganized
to be easily updated, populated from different data sources and searching in complex
queries. In the last version, ConceptNet 5 (Speer and Havasi 2013), the knowledge is incorporated from other crowd-sourced resources like Wikipedia with their own communities and editing processes. By that, ConceptNet 5 can expand its domain, and apply to many different text-understanding applications. Until April 2012, ConceptNet 5 contains about 8.7 million assertions connecting 3.9 million concepts which are represented in different language such as English, Chinese, Portuguese, etc.

While WordNet is optimized for lexical categorization and word-similarity determination and Cyc is optimized for formalized logical inference, ConceptNet is optimized for making practical context-based inferences over real-world texts.

2.2.1.3 Automated

Instead of using hand-labeled data or hand-crafted patterns to enable relation-specific extraction, automated approaches use machine learning and natural language processing tools to extract information from data sources automatically. The extracted information from these methods is usually represented as two types: schema-free and schema-based. The concepts and relations in schema-free KBs do not follow any ontology, hence are ambiguous and difficult for reasoning. In contrast, schema-based KBs have concepts mapped to ontology and relations less ambiguous, so it is easier to reason on schema-based KBs.

Schema-free

The knowledge from schema-free KBs is usually extracted by using Open Information Extraction (Open IE) tools such as Text Runner (Banko et al. 2007), Reverb (Etzioni et al. 2011), Ollie, Open IE 4.2 (Mausam et al. 2012), ClausIE (Del Corro and Gemulla 2013). These KBs have high recall but low precision.

In general, Open IE systems extract a large number of relational tuples (Arg1, Pred, Arg2) without any relation-specific training data required. Text Runner is the first Open IE system which was designed to extract a large dataset from Web. First, given a sentence, Text Runner uses a Naive Bayes model with unlexicalized POS and NP-chunk features, trained by using examples heuristically generated from the Penn Treebank, to parse the sentence and label them as trustworthy or untrustworthy extractions. Besides using Naive Bayes, there were several subsequent works which used a linear-chain CRF (Banko et al. 2008) or Markov Logic Network (Zhu et al. 2009) to improve extractions. Another Open IE system, WOE (Wu and Weld 2010), uses Wikipedia as a source of training data for extractors, which results in more improvements over Text Runner. In this system, they use dependency parse features for training, that leads to significantly increase in precision and recall over shallow linguistic features, but much more cost in
efficiency. The limitations of these systems are incoherent extractions and uninformative extractions. That means the relations are sometime not meaningful and neither are extractions.

Reverb is considered as the second generation of Open IE systems, which implements a general model of verb-based relation phrases. Reverb uses two constraints to overcome the limitations of previous works. The first is syntactic constraint. This constraint requires the relations to match some specific POS tag patterns such as verb, verb phrase or a phrase which starts with a verb and ends with a preposition. Because the first constraint eliminates incoherent extractions and tries to reduce the number of uninformative extractions, it sometimes makes the relations too specific. The second constraint, lexical constraint, tries to separate these phrases based on an intuition that is a valid relation phrase should take many distinct arguments in a large corpus. Reverb has two weaknesses which are (i) the relations are mediated by verbs and (ii) the context is ignored, thus extracting tuples are not asserted as factual.

Ollie is the next generation of Open IE systems which tries to overcome the weaknesses of Reverb. From a set of high precision tuples extracted by Reverb, Ollie retrieves all sentences in a Web corpus that contains all content words in the tuples. By using Malt Dependency Parser (Nivre et al., 2006) on these sentences, they want to remove noise and make sure all sentences represent the original information of tuples. Finally, open pattern templates which include syntactic and semantic pattern are used to map the dependency path to open extractions. This mapping step also includes analyzing context on extractions. Open IE 4 is the successor to ReVerb and Ollie, which improves extractions by using a semantic role labeling module (SRLIE) (Christensen et al., 2010) and Relnoun. SRLIE is designed for Web-scale information extraction, which is based on PropBank (UUIC-SRL) (Kingsbury and Palmer, 2002). From the dependency graph of the sentence, a SRL system produces SRL frames that are then sent to SRLIE to get n-ary extractions. Not like a typical tuple, an n-ary extraction can include more than one second argument (Arg1, Pred, Arg2, Arg2,...) and other information like time or location. Meanwhile, Relnoun is an extractor which focuses on extracting noun-mediated relation, for instance, (United States president Barack Obama).

While Reverb and Ollie are usually not good at extracting simple clauses from long sentences, ClausIE exploits linguistic knowledge about the grammar of the English language to first find all possible clauses in a sentence. Based on these shorter clauses, ClausIE can produce high-precision extractions. This method is dramatically slower due to the dependency parser but has higher quality and higher coverage.
Schema-based

Schema-based KBs are built based on a set of good patterns, then have high precision but low coverage. Webchild is a commonsense KB which was built by using this method. From a small number of seeds obtained from WordNet, Webchild gathers assertions from the Web by using pattern matching on a huge N-gram corpus of Google. The goal of WebChild is to provide refined hasProperty relationships between nouns and adjectives like shape, color, and taste, into specific and more informative relations. In constructing KB, Webchild uses the label propagation techniques to disambiguate noun-adjective sense pairs.

NELL (Carlson et al., 2010) is another schema-based KB which acquires two types of knowledge: (i) knowledge about specified semantic categories, such as cities, companies and (ii) knowledge about semantic relations such as hasOfficesIn (organization, location). The initial KB takes an ontology about categories and relations with a set of seed examples as inputs. Using a couple semi-supervised model, NELL extracts all candidate facts from the Web. These candidate facts are then verified and promoted to the belief-facts which are put into initial ontology. Believes in the KB are considered as training instances and NELL keeps learning the model day by day since believes keep updating into the ontology. The weaknesses of NELL are having low coverage and very expensive computation.

2.2.2 Activity Commonsense KB Construction

As discussed above, human activity is a type of commonsense knowledge, which can be seen as interactions between concepts. However, there are very few researches working on mining activity knowledge.

Many previous researches focus on event temporal ordering tasks. For example, in Regneri et al. (2010), the authors propose an unsupervised learning method to order events from scripts. Firstly, by using crowdsourcing via Amazon Mechanical Turk, they collect natural language descriptions of script-specific event sequences, then use a multiple sequence alignment algorithm to build a graph which represents the script’s temporal structure. Another research also introduces an unsupervised algorithm which uses coreferring arguments in chains of verbs to learn script-like information about the world, including both event structures and the roles of their participants, but without predefined frames, roles, or tagged corpora (Chambers and Jurafsky, 2009). Additionally, Modi and Titov (2014) use neural networks to learn embeddings for events, then adapt these representations for temporal ordering tasks.
On the other hand, recent commonsense KBs either ignore activity commonsense, operate at a small-scale or are domain-specific. Cyc and SUMO contain some activity knowledge but the amount of activity knowledge is very small. For instance, SUMO develops a standard upper ontology that defines high-level concepts such as Object, Process, etc. For building ontology for Process, SUMO attempts to use over 3000 English verbs which are grouped into 48 near-synonyms classes \cite{Levin1993} in which they already eliminated the verb classes that do not seem to refer to genuine processes, then uses first-order logic to describe interactions (process) between concepts. ConceptNet also includes some activity commonsense which are extracted semi-automatically. ConceptNet considers activities as concepts and between these concepts, there are temporal relations such as hasSubevent, hasFirstSubevent, hasLastSubevent or causal relations such as Causes, MotivatedByGoal, hasPrerequisite.

Knowlywood is the most recent research working on mining activity knowledge. Knowlywood defines an activity frame which includes an activity (verb + object) and other information like location, time and participating agent. Working on narrative texts, the authors develop a pipeline to systematically compile semantically refined activity frames automatically. Firstly, they use information extraction techniques (ClausIE) on data sources, map the output to WordNet and VerbNet and then use integer linear programming (ILP) to disambiguate and construct candidate activity frames. After that, they use Probabilistic Soft Logic (PSL) for graph inference and use WordNet for taxonomy construction. In the final graph, activity frames are considered as notes, the edges are the relations between them, which include temporal (previous/next), hypernymy and synonymy. Knowlywood aims at general activities such as romantic dinners, wedding speeches, etc., rather than how-to knowledge for task solving. Moreover, the input sources for Knowlywood – narratives texts like movie scripts – require a very specific methodology for knowledge extraction. Knowlywood results in a heavy bias towards activities that are salient in movies, such as dramatic farewells or killings.

Another research related to our work is on process knowledge by \cite{YangNyberg2015}. That work also tapped into the contents of the WikiHow community. However, the goal was – different from ours – to support keyword search for how-to contents by means of query expansion in an IR-style manner. They make no attempt to canonicalize tasks and semantically organize their attributes into frames.

Overall, our research is the first research working on automatically constructing a large-scale knowledge base on how-to tasks.
Table 2.1: Overview of works on mining activity commonsense knowledge

<table>
<thead>
<tr>
<th>Encyclopedic</th>
<th>Commonsense Knowledge</th>
<th>Activity Commonsense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yago, Freebase, etc.:</td>
<td>Cyc</td>
<td>Knowlywood:</td>
</tr>
<tr>
<td>- Factual (e.g. bornOn)</td>
<td>Manual</td>
<td>- Activity extraction</td>
</tr>
<tr>
<td>- Entity oriented (e.g. Person)</td>
<td>Limited size</td>
<td>- Semantically organized</td>
</tr>
<tr>
<td>- Event but no common activities</td>
<td>Do not focus on activities</td>
<td>- Domain specific (movie scripts)</td>
</tr>
<tr>
<td></td>
<td>ConceptNet</td>
<td>- Do not have problem-solving tasks</td>
</tr>
<tr>
<td>- Crowdsourced</td>
<td>- No semantic activity or task frame</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Webchild</td>
<td></td>
</tr>
<tr>
<td>- Do not focus on activities</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.3 KB Organization

KB organization or KB canonicalization is a crucial phase of building KBs. Because the knowledge extracted from texts usually contains noise, the goal of KB organization is to remove redundancy and make the KB more consistent. Generally, the techniques used in this step are not much different between encyclopedic KBs and commonsense KBs. By using ruled-based or machine learning-based approaches, KB organization groups relations in knowledge graph into two types of classes: synonym or hypernym.

2.3.1 Synonym

The problem of clustering synonymous relations has been widely studied. To solve this problem, it is required to disambiguate senses of entities or concepts and relations between them. Many schema driven KBs such as YAGO, Freebase, DBpedia avoid this problem by assigning unique ids for each component of every triple (subjects, predicates, objects). By that, these information are canonical, however, the coverage of these KBs is much lower than KBs which are built by using “open IE” techniques.

With the KBs using open IE techniques, one approach to disambiguate entities, concepts or relations is linking them to an existing dictionary, such as Wikipedia, Freebase or WordNet. In general, entity linking can be considered as ranking problem where each mention generates a candidate list of entities, then the best entity is selected by using a machine learned model. For instance, Hachey et al. (2013) use a Support Vector Machine (SVM) ranking model with several local features such as cosine similarity between the query context and the text of the candidate entity page, combination of candidate categories—Wikipedia classifications. Meanwhile, Ratinov et al. (2011) optimize the local
model by adding an accurate disambiguation context (Wikipedia title) for each mention. However, there is one problem with these methods, which is many pages have new entities that are not in a KB (out-of-KB).

On the other hand, there are also many approaches which do not use entity linking for disambiguation. PATTY (Nakashole et al., 2012) uses syntactic ontological-lexical patterns and sequence mining algorithms to gather a general class of relation phrases. By using argument overlap statistics and phrase syntax similarity, PATTY organizes these relation phrases into synsets and infers lexical type signatures. Actually, their goal is to construct a taxonomy of relation phrases, and if two phrases are both semantically more general than other, they are synonymous. NELL also organizes extracted relations into groups of synonyms, however, NELL uses a set of binary L2-regularized logistic regression models, one per category, to classify relations based on various morphological features (words, capitalization, affixes, part-of-speech, etc.). That means the number of clusters is limited and pre-defined.

Universal Schema (Riedel et al., 2013) uses probabilistic models of matrix factorization and collaborative filtering to find similarity scores between relational phrases. This matrix combines information of structured (schema driven) and unstructured (open IE) data, with the rows coming from running cross-document entity resolution on pre-existing structured databases and the columns coming from the union of surface forms and DB relations. Through several models, this method can learn vector representations for phrases, which allows to encode asymmetry implicature between them.

RESOLVER (Yates and Etzioni, 2009), on the other hand, uses HAC to cluster Open IE relations in TextRunner data. RESOLVER uses a generative, probabilistic model for finding the similarity between strings. There are two sub-models in RESOLVER. The first model is a simple model for predicting whether pair of strings is synonymous based on string similarity (i.e., edit distance). The second model is Extracted Shared Property (ESP) for predicting whether pair of strings co-refers based on their distributional similarity. RESOLVER combines these models to compute an overall prediction of synonymy decisions and uses this metric inside HAC. Another work using HAC for clustering Open IE relations is (Galarraga et al., 2014). In this work, they provide some similarity functions such as attribute overlap, string similarity, IDF token overlap, etc. as features of HAC to cluster noun phrases. With verb phrases (relations), their method requires that subjects and objects of the Open IE triples are already clustered by using their noun phrase clustering algorithm or mapping them to Freebase. Then verb phrases can be clustered by using rule mining which is based on AMIE algorithm (Galarraga et al., 2013) that can learn Horn rules or using HAC, similar to noun phrases.
Unlike above methods, the work in Grycner et al. (2014) presents a large-scale statistical relations method for clustering relational phrases using Probabilistic Soft Logic (PSL). PSL is a declarative language for specifying templates for probabilistic graphical models. These models, which are known as hinge-loss Markov random fields (Bach et al., 2013), describe the probability densities of continuous random variables in the range $[0,1]$. Inference in these models is a convex optimization task, which can be solved efficiently. In this work, PSL models develop three sets of rules including (1) standard similarity functions for argument similarity and textual similarity, (2) transitivity rule which ensures that similarity relation between relational phrases is transitive and (3) negative prior on the inference predicate similar. Each PSL model runs in two steps which are learning weights of rules, performed on a separate training data set, and inference. This method can be scalable to be applied to 200,000 relational phrases.

2.3.2 Hypernym

The hypernymy relation can be considered as subsumption information between relational phrases. For instance, the relation got married to has the hypernym is in a relationship with.

Actually, in PATTY, the authors also introduce a subsumption hierarchy, but the hypernymy links are very few and sparsity. RELLY (Grycner et al., 2015) uses PSL to construct a hypernymy graph from PATTY and other KBs like YAGO, WordNet. Similar to Grycner et al. (2014), the PSL model extracts a set of rules which combine statistic features, semantic information and structural constraints. The statistical information includes argument overlap and alignments to WordNet. The semantic information is type information and the structural constraints describe the transitivity and acyclicity. RELLY has four stages: data pre-processing which assigns confidence scores of 0 or 1 for the binary-valued semantic predicates in the PSL model, rule weight learning which uses an EM algorithm on a small hand-crafted data set, inference which predicates hyponym, and perform additional cleaning (or thresholding).

Another research working on mining hypernymy relations is Berant et al. (2011). In this paper, the authors use Integer Linear Programming (ILP) to learn a globally-optimal set of entailment rules for typed predicates. Using only ILP for finding the best of entailment rules under a transitivity constraints has two problems: ambiguity and scalability. To overcome these problems, the authors, firstly, train an entailment classifier to define whether one triple entails another. They use WordNet to generate positive and negative examples, extract features based on distributional similarity scores and build the model. After that, they extend the ILP model by modifying the objective function and adding
some constraints. Finally, for scalability, they decompose the graph into small graphs and use incremental ILP on each small graph.

Others

There are also some researches which address the task of constructing taxonomy of relations which includes both synonymy and hypernymy relations. HARPY (Grycner and Weikum 2014), an extension of PATTY, wants to align phrase synsets from the PATTY taxonomy with verb synsets in WordNet. By that, they can enhance many PATTY phrases with the clean hypernyms of WordNet, augment the subsumption hierarchy, and extend WordNet verb senses with the lexical type signatures derived from PATTY. For this purpose, they build a directed candidate alignment graph. The vertices of this graph include synsets of relational phrases in PATTY, verb senses from WordNet and features of either phrases or verbs. Meanwhile, the edges describe relations between phrases, verbs, and features, which include hypernymy, similarity and vertex-feature edges. Every edge is weighted by counting frequency of features and/or similarity scores, or is simple set to 1 for binary cases (e.g., hypernymy edges). After building this graph, the authors use a random-walk algorithm to find “strong path” between relational-phrase vertices and verb-sense vertices. This method generates 600K hypernymy links, but with low precision.

Knowlywood, on the other hand, leverages the high-quality relations in WordNet to organize activity frames. They construct a graph between activity frames by using a computationally expensive technique and relying on sense disambiguation via WordNet. The weights of edges in this graph are derived from relatedness strength. The relation between two activity frames will be defined as synonym or hypernym by comparing the semantic distance score, which is WordNet path similarity (Tandon et al. 2015) to a threshold which is determined manually.

Overall, KB organization is usually used on KBs which extracted knowledge based on “open IE” techniques, does not depend on whether they are encyclopedic or commonsense KBs. However, the previous works either (1) only organize the relations (or triples) of KBs or (2) depend on existing resources, such as Wikipedia, Freebase or WordNet. In our work, we leverage all information of how-to task frames which include main task themselves (surface forms, like in Knowlywood), contextual information like location, time, participants and the inter-attribute of the task frame such as parent, previous/next or sub-task to organize them without using any other existing resource. Our method also has much more light-weight than the one Knowlywood uses.
Chapter 3

HowToKB System

3.1 HowToKB System Overview

The HowToKB system is a pipeline of methods and tools for automatically building the how-to task KB. Figure 3.1 illustrates the architecture of HowToKB. There are two main components in our system, which are KB construction and KB organization.

- **KB Construction**: In this phase, we first crawl all data, both textual and visual content, from total of 168697 articles from WikiHow. We then use Open IE 4.2
to extract tasks, participants, location and time from the data. To remove noise from extraction results, we normalize tasks into two forms: weak form (stopword removing from original extractions) and strong form (head verb + head noun, both are frequent in WordNet) and only keep tasks if they have both two forms. By leveraging the well-structured data of WikiHow, we construct the relations between task frames, which include temporal relation (previous/next tasks) and hierarchical relation (parent/sub-tasks).

- **KB Organization:** We propose a new clustering algorithm that includes both bottom-up and top-down hierarchical clustering to cluster task frames. We use all information from task frames (surface form of the task, additional information and context) as features and learn weights of features by using a logistic regression model. To the end, the similarity score between two task frames is used for clustering.

### 3.2 KB Construction

#### 3.2.1 Concepts and Notation

In natural language, how-to tasks are typically referred to by verbal phrases (e.g., “repair” or “repair a tire”). To a first degree, WordNet provides a repository of such phrases. However, WordNet mostly contains single verbs and misses out on many compound phrases. Also, noun phrases may express tasks as well (e.g., “tire repair”). Therefore, we consider extended phrases, not necessarily present in WordNet, as cues for expressing tasks.

**Definition 3.1** (Normalized extended phrase). A phrase (noun phrase or verb phrase) whose head-word is present in WordNet, is called an extended phrase (e.g., “tire repair” has head-word “repair”). These phrases can be normalized either strongly or weakly. Strong normalization reduces the phrase to its head word, while weak normalization only performs stemming and removes leading articles.

**Definition 3.2** (Task phrase). A task phrase consists of \((v, prep, o)\) where \(v\) is a verb or a normalized extended verb phrase and \(o\) is a noun or a normalized extended noun phrase and \( prep\) is a preposition linked with \(v\) (optional). For example, paint on wall.

Task phrases are the anchors for compiling knowledge on how-to tasks into semantic frames.
Definition 3.3 (Task frame). A task frame is a task phrase enhanced with

- attributes: location, time, participating agent, participating object, category;
- hierarchical relations: parent task and sub-tasks;
- temporal relations: previous task and next task.

Finally, images or videos showing the tasks are also stored in a frame. Each attribute can have zero, one or multiple entries with confidence scores $\in [0, 1]$.

Table 3.1 shows an example for a task frame.

<table>
<thead>
<tr>
<th>attribute</th>
<th>notation</th>
<th>value</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>loc(a)</td>
<td>house</td>
<td>noun-phrase</td>
</tr>
<tr>
<td>time</td>
<td>time(a)</td>
<td>weekend</td>
<td>WN-time</td>
</tr>
<tr>
<td>parti. agent</td>
<td>parta(a)</td>
<td>student</td>
<td>WN-living</td>
</tr>
<tr>
<td>parti. object</td>
<td>parta(a)</td>
<td>brush</td>
<td>WN-non-living</td>
</tr>
<tr>
<td>category</td>
<td>cat(a)</td>
<td>house</td>
<td>WH-categories</td>
</tr>
<tr>
<td>parent-task</td>
<td>parent(a)</td>
<td>decorate house</td>
<td>task</td>
</tr>
<tr>
<td>sub-task</td>
<td>sub(a)</td>
<td>clean wall</td>
<td>task</td>
</tr>
<tr>
<td>prev-task</td>
<td>prev(a)</td>
<td>buy paint</td>
<td>task</td>
</tr>
<tr>
<td>next-task</td>
<td>next(a)</td>
<td>dry the wall</td>
<td>task</td>
</tr>
</tbody>
</table>

Table 3.1: Example frame on task $a =$ paint a wall

The WikiHow community organizes its contents with one article dedicated to one task. The articles are semi-structured, containing: (i) textual and visual content, (ii) article level statistics like number of views and quality ratings and (iii) a category selected from a hierarchy of more than 3000 categories. A WikiHow article describes various how-to methods to solve a task. Each method mentions the required tools and presents a step-by-step procedure. Each step contains a heading, denoting the step’s main action, followed by a textual description and representative images. Figure 3.2 shows an example of articles from WikiHow.

To distill knowledge from a WikiHow article into machine-readable form, we hypothesize on the structure of articles. We assume four dimensions of structure:

- Hierarchical structure: The article describes a task that can be split into several solution methods. Every method has a heading that indicates the main action for the solution method, which can be seen as a sub-task of the article-level task. Similarly, a step-level headings can be interpreted as sub-tasks of the per-method sub-tasks.
Temporal structure: Method-level and step-level actions have temporal ordering. Suppose a method consists of a sequence of three step-level actions, $a_1$, $a_2$, and $a_3$. We interpret this as $a_1$ preceding $a_2$ and $a_2$ preceding $a_3$.

Categorical structure: Tasks at all levels can be assigned to the same category as the category for the entire article.

Frame attribute structure: Every method contains a list of required tools and other objects which become the participants of a task. Steps are depicted by images or videos; these visual contents become the visual representative of a step-level action. Users rate an article with a score between 1 and 5 (5 being best). Users land on an article either through search (of any text present in the article) or by traversing the category taxonomy. Thus, these ratings and views are carried over to all sub-tasks in the article.

We obtain sub-tasks at all levels from the step-by-step headings in an article – figure 3.3. We do not consider the – more verbose and hence noisier – descriptions to extract sub-tasks. Consider an article-level description that we annotate for illustration: 

$<\text{CAUSE}>$ Is your clothes iron experiencing a midlife crisis? $</\text{CAUSE}>$ ... $<\text{METADATA}>$ Fear not, citizen! Cleaning your iron can be quick... $</\text{METADATA}>$. In the absence of annotated examples, it is very difficult to tell apart useful parts of this description (cause) and the non-useful part (metadata can be tips, warnings or simply notes). Similarly, consider a step-level description: $<\text{SUBACTIVITY}>$ Your damp cloth can be soaked
Either way, make sure the iron is dry before using. While there is useful information presented in this description, it is very costly to annotate articles this way. Processing full descriptions in an automated manner is left for future work.

### 3.2.2 Information Extraction Method

Our IE goal is to extract tasks, participants, location and time from a heading. Consider the example heading: Use distilled water and vinegar in the reservoir in summers. Here, the tasks are: use distilled water and use vinegar. The location and time for both tasks are the reservoir and summers, respectively. The method to perform these extractions must meet the following requirements:

1. **Sentence sub-structure.** As the headings are not long, we must capture all the information present. In this example, we need to break the sentence into two clauses.
2. **Semantic role labels.** We have five role labels: task, participating object, participating agent, location and time.
3. **Open domain.** Task knowledge is evolving and open domain; so the IE method must not be restricted to a specific domain.
Design choices. We considered a number of design choices for the IE method, including the following.

- Knowlywood semantic parsing pipeline (Tandon et al., 2015): Carrying over our prior work to the new setting of distilling task knowledge may satisfy requirements (1.) and (2.), but would critically rely on VerbNet (Schuler, 2005) and WordNet dictionaries for additional sense disambiguation. For constructing the HowToKB, we do not want to restrict ourselves to VerbNet or WordNet senses.

- SRL systems: Some SRL systems satisfy requirement (1.). Most SRL systems are trained on location, time, and, participants, but not for open-domain asks or activities. Thus, using an existing SRL system for acquiring task knowledge would need a large amount of labeled training data that we do not have.

- NER systems (Nadeau and Sekine, 2007): Merely running named entity recognition (NER) does not satisfy (1.) and does not provide the desired kinds of labels for (2.). Similar to SRL systems, these methods require large amounts of training data. Lexicon based methods for NER require lexicons, and being in an open domain, the lexicons are not fixed. For instance, a table can be a location or a participating object depending on the context, and the lexicon does not help in distinguishing these cases.

- Open IE systems: ReVerb only satisfies (3.), while Ollie satisfies (1.) and (3.). OpenIE 4 satisfies all of (1.) and (3.) and partially (2.), as it lacks understanding tasks, participating objects, and, participating agents.

Therefore, we choose OpenIE as our primary method for task IE, specifically using the Open IE 4 software.

We expand the inputs – WikiHow headings – before passing them to Open IE 4. Headings typically begin with a verb, while Open IE 4 expects sentences or phrases that begin with a subject. To fix this, we prefix each heading with the subject You. For instance, an extraction of an heading from WikiHow “paint a bedroom wall on weekend” would be “(You; paint; a bedroom wall; T:on weekend)”.

Postprocessing OpenIE output. Open IE 4 yields subject-predicate-object triples, where each of the three components is a textual surface phrase. We postprocess this output first by imposing two kinds of normalization. For weak normalization, we remove stop-words from the task phrase. For strong normalization, we identify the head verb and head noun and combine them into a task name. For instance, the initially extracted phrase paint a bedroom wall has the weak normalization paint bedroom wall and the strong normalization paint wall.
A second stage in post-processing the Open IE output is to impose semantic type restrictions on the subject and object of triples. Initially, the arguments of a triple are surface forms (noun phrases or verb phrases), without any restriction. To ensure that a triple really refers to a how-to task, we require the subject to be a task name (if already in our dictionary) or a noun phrase or a temporal expression. For high recall, we only impose these constraints on the head words of the phrases while allowing an open set of phrases. As dictionary we use frequent words in WordNet. WordNet also allows us to populate a list of time expressions, by taking all synset words on time domain (WN-time).

The predicates in an Open IE triple are also surface forms and often verb phrases. We map such predicates to the task frame attributes by constructing a role label as the concatenation of the predicate and object phrases. We filter the usually noisy role labels for location and time, using the object constraints defined above.

### 3.3 KB Organization

Semantically organizing the extracted task phrases and their candidate frames is crucial for building a clean and consistent KB. The KB construction, as described in the previous section, has redundancy from the WikiHow contents. There are two main motivations of KB organization after construction:

- **Removing redundancy.** For instance, considering two frames for tasks paint a wall and color a ceiling, they are highly related to each other and heavily overlap in their sub-tasks and frame attributes. Grouping such nearly synonymous task frames together can make the KB more precise and consistent.

- **Disambiguation of task names.** For instance, the task use a keyboard can be interpreted as use a music keyboard or use a computer keyboard. It is important to separate these two senses, and have one frame for each of the two meanings.

To cope with these issues, we cast semantic organization into a clustering problem. The method to perform this clustering problem must meet the following requirements:

1. **Working on non-numeric data.** Our targets are task frames whose most of attributes are categorical. To leverage these attributes for clustering, the method needs to be able to work on them.

2. **The number of cluster is not pre-defined.** It is apparent since we do not know how many clusters task frames are belong to.
3. **Efficiency on large datasets.** Efficiency is always one of the main issues in clustering problem, especially when working on large datasets. Since our clustering problem also has to work on over one million of task frames, efficiency should be taken into consideration.

**Design choices.** We considered several design choices for the clustering methods, including the following:

- **Representative-based clustering** such as Naive Bayes or K-means. The Naive algorithm one by one generates all possible clusterings, which is infeasible on large datasets. Meanwhile with K-means, it is mandatory to compute a median, which requires numerical vector representation of each data point. Moreover, they both require the number of cluster pre-defined, hence do not satisfy requirements (1) and (2).

- **Probabilistic clustering** such as expectation-maximization (EM). These methods also do not satisfy requirements (1) and (2). EM is a mixture model which needs to know the number of clusters, also have to compute the log-likelihood of the data so it could only operate on numerical data.

- **Density-based clustering** such as Grid-based or DBSCAN ([Ester et al., 1996](#)). These methods satisfy requirements (1) and (2) since they only require a similarity measure between two data points (like DBSCAN) and we don’t have to specify the number of clusters. However, these methods also require to specify many parameters such as global minimal density, parameter which controls the minimum size of a cluster and parameter which controls the required density. Moreover, in grid-based methods, the number of grid cells increases exponentially with dimensionality, hence it is difficult to satisfy requirement (3).

- **Hierarchical clustering.** These methods consist of two main approaches: agglomerative bottom-up and divisive top-down. They can satisfy requirements (1) and (2) but it is also not easy to satisfy (3).

Therefore, we choose hierarchical clustering methods to solve our clustering problem but need to improve their efficiency. Through the learning step which will be discussed later, we devised a hierarchical clustering algorithm, with a computationally inexpensive bottom-up phase followed by a top-down algorithm. The experimental result shows that our proposed clustering algorithm is not only effective but also efficient.
3.3.1 Similarity Measures

Clustering the task frames requires computing pairwise similarity measures. We define similarity measures per frame attribute and then combine them linearly. The coefficients for this combination are learned from a small number of labeled training samples.

**Categorical Similarity**: The WikiHow category taxonomy is a tree, containing over 3000 categories with a maximum depth of 8 levels. We use the WU-Palmer measure (Wu and Palmer, 1994) to compute similarity between the categories of two task frames:

\[
f_{\text{cat}}(c_1, c_2) = \frac{2 \cdot \text{depth}(\text{lca}(c_1, c_2)) + 1}{\text{depth}(c_1) + \text{depth}(c_2) + 1}
\]  

(3.1)

**Lexical Similarity**: This function computes the lexical similarity between the task names as:

\[
sim(\text{surface}(a1), \text{surface}(a2)) = f_{w2v}(\text{verb}(a1), \text{verb}(a2)) \cdot f_{w2v}(\text{noun}(a1), \text{noun}(a2))
\]  

(3.2)

Here, \(\text{sim}\) is a simple edit distance between two words (the head verbs or the head nouns in the two task names), and \(f_{w2v}\) is an embedding-based similarity. Specifically, we are using Word2Vec (Mikolov et al., 2013). To this end, we train a distributional representation over POS-tagged WikiHow data. The similarity between two POS-tagged words is the cosine similarity between their embeddings:

\[
f_{w2v}(w_1, w_2) = \cosine(E_{w_1}, E_{w_2})
\]  

(3.3)

where \(E_w\) is the embedding of word \(w\).

**Vector similarity**: Location, time, participating agent and participating object are all vectors of strings. To compute the similarity of a pair (e.g., location vectors of two task frames), we use the weighted Jaccard coefficient, with weights coming from equation 3.3 as:

\[
f_{\text{list}}(L_1, L_2) = \frac{\sum_{w_i \in L_1} \sum_{w_j \in L_2} f_{w2v}(w_i, w_j)}{|L_1||L_2|}
\]  

(3.4)

Analogously, for attributes that are lists of tasks, like prev-task, next-task, sub-task and parent-task, we plug the task name similarity from Equation 3.2 into Equation 3.4.
3.3.2 Combining Features

The similarity between two task frames can now be computed as follows:

\[ sim(a_1, a_2) = \frac{1}{1 + \exp(-f_{sim}(a_1, a_2))} \]  

(3.5)

Here, \( f_{sim}(a_1, a_2) \) is a linear combination of all features:

\[ f_{sim}(a_1, a_2) = w_0 + \sum_{i=1}^{N} w_if_i(a_1, a_2) \] 

(3.6)

where \( f_i(a_1, a_2) \) is the similarity measure for the \( i^{th} \) attribute of the task frames. We learn the parameters \( w_i, i \in [0, N] \), by training a logistic regression model.

For this purpose, we manually create a training set of 5500 data points (5000 training points, 500 test points) for the logistic regression model, using two labels: near-synonymous frames or dissimilar frames. To learn robust weights for our features, we need to make sure our training data contains hard cases. For instance, two task frames brew beverage in Beer and Cider category and brew beverage in Coffee category have different sub-task sets, hence are dissimilar to each other. Similarly, make an Italian dish and make a German dish are totally different in the way how to do it, even though they have same strong-norm form make dish. However, in general cases, tasks having nearly similar sub-task and frame attribute sets should be similar to each other — broil steak and grill steak.

For efficient estimation of the — very sparse — pairwise similarity matrix between all task frames, we introduce pruning of dissimilar pairs. The empirical observation on which this heuristics is founded, is:

\textit{Two task frames are dissimilar with an empirical confidence of 99.9\% if a combination of their categorical and lexical similarity is less than a threshold.}

Based on this observation, we first compute the overall similarity of two task frames by using only two of the basic similarity measures, a fairly light-weight computation. Only for the pairs that survive this filter step (i.e., could potentially be near-synonymous), we need to perform the more expensive full computation using all similarity measures.

3.3.3 Clustering Algorithm

As we do not know the appropriate number of clusters, we use hierarchical clustering until reaching a stopping criterion. As mentioned above, there are two approaches to hierarchical clustering: agglomerative bottom-up and divisive top-down. Both have high
$O(N^3)$ computational complexity (typically $O(N^3)$) and are too slow for our purpose. Therefore, we devise a two-phase method, which first performs a crude bottom-up algorithm to construct super-clusters and then splits these in a top-down manner as needed.

**Bottom-up phase**

The bottom-up algorithm starts with lexical grouping based on equation 3.2. This quickly groups all task frames which have the same strongly normalized name into the same group that we refer as a lexical cluster (see Algorithm 1). For instance, two task frames `paint the living room wall` and `paint the bedroom wall` have the same normalization `paint wall` and are grouped together. For each lexical cluster, we use the strong normalization as a key for subsequent steps.

Next, the bottom-up algorithm merges two lexical clusters based on the similarity of their keys using Equation 3.2. This results in distributional clusters. We want to ensure that if two task frames can be clustered, they will always be in the same distributional cluster. Therefore, we use our empirical observation to ensure that if two keys are dissimilar they are not grouped but if they are not dissimilar they may or may not be similar (see Algorithm 2). Note that two tasks whose strong-norm is same but are very different senses would be clustered together at this stage e.g. use music keyboard and use computer keyboard.

Our bottom-up algorithm has a small run-time and significantly pre-clusters.

**Algorithm 1: Lexical Grouping Algorithm**

*Data: All task frames extracted from WikiHow*

*Result: List of lexical clusters*

for all task frame $a_i$ do
    if $\text{strong}norm(a_i) = \text{strong}norm(a_j)$ with $i \neq j$
        then
            cluster($a_i$) = cluster($a_j$) = $C$;
            key($C$) = $\text{strong}norm(a), a \in C$;
            value($C$) = \{a | a \in C\};

29
Algorithm 2: Distributional Grouping Algorithm

**Data:** List of lexical clusters, thresholds $\delta_{dist}, \delta_{cat}$

**Result:** List of super clusters

for all lexical clusters $C_i$ do
  if $\text{sim}(\text{key}(C_i), \text{key}(C_j)) < \delta_{dist}$ and $\text{sim}(\text{cat}(a), \text{cat}(a')) < \delta_{cat}$, $a \in C_i, a' \in C_j$ then
    can not merge $C_i, C_j$ together;
  else
    merge $C_i, C_j$ into the same super cluster;

Top-down Phase

Each of the coarse-grained super-clusters is now refined by recursive splitting. Different super-clusters can be processed in parallel. This method is not only efficient but also scalable, which is essential when working on the large datasets.

We use a simple heuristics for splitting clusters. We identify the two most dissimilar frames within a cluster, use them as seeds of two sub-clusters, and then assign all other frames to the closer one of the seeds (see Algorithm 3-a). These steps are repeated for each of the clusters until a stopping criterion is satisfied. We considered the Bayesian Information Criterion (Schwarz, 1978) to decide when to stop splitting, but it turned out that simple thresholding on average intra-cluster similarity is very effective. So we chose this alternative, giving us more flexibility in producing coarser or more fine-grained clusters (depending on the threshold value).

Algorithm 3-a: Divisive-TopDown Algorithm

**Data:** A super cluster $C$, threshold $\gamma$

**Result:** List of final clusters

if $\text{avgIntraClusterSim}(C) < \gamma$ then
  $C', C'' = \text{split}(C)$;
  $\text{divisive} - \text{topdown}(C')$;
  $\text{divisive} - \text{topdown}(C'')$;
else
  return $C$ as a final cluster;
**Algorithm 3-b: Two-way Splitting Function**

**Data:** An input cluster $C$

**Result:** Two cluster children

compute pair-wise similarity of all task frames in $C$;

$a^* = \arg \min_{a_j \in C} \text{avgSim}(a_j, C \setminus \{a_j\})$;

$C' = \{a^*\}$;

$C'' = C \setminus \{a^*\}$;

**for all task frame $a_i$ in $C''$ do**

| if $\text{avgSim}(a_i, C') > \text{avgSim}(a_i, C'')$ then |
| $C' = C' \cup \{a_i\}$; |
| $C'' = C'' \setminus \{a_i\}$; |

return $C', C''$;

---

**Figure 3.4:** Example of our clustering algorithm

**Updates**

WikiHow and how-to guides in general, have been quickly gaining popularity. Our algorithm should be able to efficiently organize such new content. Hierarchical algorithms are update friendly because the new task frame must be recursively merged (considering the stopping criteria) into the more similar cluster (either left or right cluster traversing down). We show that our simple divisive clustering avoids this expensive re-computation (see Algorithm 4-a).
Algorithm 4-a: Update Algorithm

Data: List of super clusters and fine clusters, thresholds $\delta_{\text{dist}}, \delta_{\text{cat}}, \gamma$, new task frame $a$

Result: List of clusters to update with $a$

$C^* = \text{getUpdatableCluster}(\text{super clusters}, \delta_{\text{dist}}, \delta_{\text{cat}}, a)$;

if $C^* = \text{null}$ then
  return new cluster $\{a\}$;
else
  while $\text{children}(C^*) \neq \text{null}$ do
    update($\text{children}(C^*)$, $\delta_{\text{dist}}, \delta_{\text{cat}}, \gamma, a$);
    $C^* = C^* \cup \{a\}$;
  end while
  if $\text{children}(C^*) = \text{null}$ and $\text{avgIntraClusterSim}(C^*) < \gamma$ then
    split($C^*$);
  end if
end if

Algorithm 4-b: Updatable Candidate Clusters Algorithm

Data: List of clusters, thresholds $\delta_{\text{dist}}, \delta_{\text{cat}}$, new task frame $a$

Result: Cluster which can be updated with $a$

for all input clusters $C_i$ do
  if $\text{sim}(\text{key}(C_i), \text{strongnorm}(a)) < \delta_{\text{dist}}$ and $\text{sim}(\text{cat}(a), \text{cat}(a')) < \delta_{\text{cat}}, a' \in C_i$ then
    can not merge $C_i, C_j$ together;
  else
    return $C_i$;
  end if
end for

return empty cluster;

Figure 3.5: Example of the update algorithm
Chapter 4

Experiments

Table 4.1 summarizes the size of the HowToKB: the number of HowTo tasks, before and after organization, their attributes and the associated images. We estimate the precision by extensive sampling, with assessment by crowdsourcing.

<table>
<thead>
<tr>
<th></th>
<th>Ungrouped</th>
<th>Grouped</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Tasks</td>
<td>≈1.2M</td>
<td>≈0.5M</td>
<td>0.90</td>
</tr>
<tr>
<td>#Attributes per task</td>
<td>4.8</td>
<td>12.0</td>
<td>0.90</td>
</tr>
<tr>
<td>#Images per task</td>
<td>0.8</td>
<td>2.0</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 4.1: HowToKB statistics

Table 4.2 gives an anecdotal example from HowToKB.

<table>
<thead>
<tr>
<th>Task: paint a bedroom wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
</tr>
<tr>
<td>Time</td>
</tr>
<tr>
<td>Agents</td>
</tr>
<tr>
<td>Prop</td>
</tr>
<tr>
<td>Parent</td>
</tr>
<tr>
<td>Previous</td>
</tr>
<tr>
<td>Next</td>
</tr>
<tr>
<td>Sub-tasks</td>
</tr>
</tbody>
</table>

Table 4.2: Anecdotal examples from HowToKB
4.1 Experimental Setups

We conducted extensive experiments to assess the viability of our approach and the quality of the resulting HowToKB. Our experiments cover the two modules developed in this research: knowledge extraction and KB organization.

(i) KB construction: Quality of the OpenIE triples using various OpenIE methods. We evaluate Reverb, Ollie and Open IE 4.2 for triples obtained from a random subset of WikiHow sentences/headings. Quality of task frames extracted by using Open IE 4.2 – the KB before organization phase.

(ii) KB organization: Clustering algorithm accuracy and run-time efficiency.

(iii) Intrinsic evaluation on the resulting HowToKB using crowdsourcing and comparison with other existing KBs.

(iv) Extrinsic evaluation on the use-case: Searching related YouTube videos for HowToKB tasks.

4.2 KB Construction

This section reports the performances of different OpenIE systems and quality of task frames extracted by using Open IE 4.2.

4.2.1 Information Extraction Systems

To construct a task frame from WikiHow contents, we rely on an OpenIE system. Assuming that the input WikiHow is perfectly correct, we measure how much noise is introduced by the OpenIE system we chose and other systems that could have been used.

Baselines

The family of Open IE systems process sentences to extract $SPO$ triples, where $O$ is a list of objects that can additionally include context like a role label (e.g. location, time). We consider all variants in the Open IE family, i.e. ReVerb, Ollie and Open IE 4.2 and their combinations. We compare these against our wrapper on top of Open IE 4.2.
Evaluation setup

As ground truth for triples, we construct a dataset of 50 random sentences sampled from WikiHow. For each sentence, three human judges manually annotated \( SPO \) triples for each sentence. This resulted in a comprehensive set of 59 triples. For instance, a given sentence “You purchase or rent a double flaring tool kit.” has two possible \( SPO \) triples as output: \((You; purchase; a double flaring tool kit)\) and \((You; rent; a double flaring tool kit)\).

To measure the noise introduced by the various Open IE systems, we use the standard F1 measure against the dataset. If an extracted triple is absent in the ground truth, the triple is incorrect leading to a false positives.

Based on the dataset, we also use F1 score to evaluate how much noise the post-processing step can remove from the results extracted by using Open IE 4.2.

Results

Which OpenIE system is the best? We compare the F1 scores against the triple ground truth for each of the OpenIE systems and find that Open IE 4.2 has the maximum F1 of 72.07%, followed by ReVerb at 58.59%, and, Ollie at 45.38%. To confirm if these different systems are complementary, we perform a simple ensemble of these systems, which leads to a collective F1 of 73.01%. Table 4.3 shows the result of performances of different OpenIE systems and their combinations in details.

<table>
<thead>
<tr>
<th>Systems</th>
<th>L</th>
<th>P</th>
<th>R</th>
<th>L(\cap)P</th>
<th>L(\cap)R</th>
<th>P(\cap)R</th>
<th>L(\cap)P</th>
<th>L(\cap)R</th>
<th>P(\cap)R</th>
<th>L(\cap)P(\cap)R</th>
<th>L(\cap)P(\cap)R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>44.26</td>
<td>75.47</td>
<td>70.73</td>
<td>78.57</td>
<td>68.97</td>
<td>85.71</td>
<td>52.27</td>
<td>48.61</td>
<td>67.64</td>
<td>80.00</td>
<td>51.09</td>
</tr>
<tr>
<td>Recall</td>
<td>46.55</td>
<td>68.97</td>
<td>50.00</td>
<td>37.93</td>
<td>34.48</td>
<td>41.38</td>
<td>79.31</td>
<td>60.34</td>
<td>79.31</td>
<td>27.59</td>
<td>81.03</td>
</tr>
<tr>
<td>F1-score</td>
<td>45.38</td>
<td>\textbf{72.07}</td>
<td>58.59</td>
<td>51.16</td>
<td>45.98</td>
<td>55.81</td>
<td>63.01</td>
<td>53.85</td>
<td>\textbf{73.01}</td>
<td>41.03</td>
<td>62.67</td>
</tr>
</tbody>
</table>

Table 4.3: Extraction results of different combinations of three IE systems \((L: \text{Ollie}, P: \text{Open IE 4.2}, R: \text{Reverb})\)

As the gain is marginal and given that we lose the additional benefit of semantic role labeling, we rely solely on Open IE 4.2 as the system of choice.

Removing noise by post-processing. Open IE systems attach a confidence score to every \( SPO \) triple. We empirically investigate the F1 scores at thresholds between \([0,1]\) at a step size of 0.05. We settled at a confidence score of 0.45 which gives the best F1 score of 72.07% (precision of 75.47% and recall of 68.9%). We observe that the maximum precision at any threshold is 82.5% (at a lower recall level) showing that the default confidence score does not yield very high quality results. As our goal in KB construction is to minimize the noise, the confidence score alone cannot take us very far. Our Open
IE wrapper normalizes the triples (cf. Section 3.2.2) leading to a marginal increase in F1 to 72.34%. However and most importantly, the precision increases dramatically from 75.4% to 97.1% at the cost of being more selective (recall drops from 68.9% to 57.6%) – table 4.4. So our additional wrapper on top of Open IE 4.2 is decisive for building a high-quality KB.

<table>
<thead>
<tr>
<th>System</th>
<th># sentences</th>
<th># extractions</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open IE 4.2</td>
<td>50</td>
<td>62</td>
<td>75.47</td>
<td>68.97</td>
<td>72.07</td>
</tr>
<tr>
<td>Open IE 4.2 (with post-process)</td>
<td>35</td>
<td>35</td>
<td>97.14</td>
<td>57.63</td>
<td>72.34</td>
</tr>
</tbody>
</table>

Table 4.4: Removing noise from Open IE 4.2 extraction results

4.2.2 Output of KB Construction: Task Frames

Baselines

We do not have any baseline system in this case but use crowdsourcing to evaluate the quality of task frames extracted from WikiHow data by using our Open IE wrapper.

Evaluation Setup

As ground truth for task frames, we pick a random sample of 150 task frames and have them judged by three crowdsourcing workers on CrowdFlower\(^1\). For each task frame, several attributes can hold.

**Question Type I**: Given a task along with its attributes, we provide the context of this task and ask turkers for *one of the most likely* attributes. One automatically constructed sample question posed to turkers is *In some context like “decorate your house”, one of the most likely previous tasks when we paint a wall could be pick a color. Is it commonly true?* In this case, the task name is *paint a wall* which is in the context “decorate your house” and has a previous task *pick a color*. We obtain these contexts through the source of extraction (i.e. the title of the WikiHow page from which this task was extracted).

**Question Type II**: Given a task along with its attribute, we ask turkers for *the most likely* attribute and do not provide the context. From above example, our question now is *The most likely previous task when we paint a wall could be pick a color. Is it commonly true?* We also make an ensemble of these questions such as asking for *one of the most likely* without providing the context or asking for *the most likely* along with

\(^1\)http://crowdflower.com
the context. The goal of this experiment is to show us the effect of the way we setup for crowdsourcing.

**Answers:** The answer for these questions is *commonly true* that means our information extracted by using Open IE 4.2 is correct, otherwise *not commonly true*. We aggregate the scores from the three judges and retain those frame attribute values where at least two judges agree. CrowdFlower automatically weeds out unreliable judges, based on a set of gold-standard questions.

**Results**

On the task of evaluating quality of task frames, we realized that the answers of turkers depended on the question we provided. By asking for *one of the most likely* (for example, “one of the most likely location when we paint a wall”) and providing the context (for example, “in some contexts like ...”), the answers of turkers are different and become more reliable than asking for *most likely* and without providing context. It is apparent since all of these information are commonsense and our task phrases have the limited contexts.

To see the difference, we randomly pick 100 questions whose answers are *not commonly true*. Currently, these questions ask for *most likely* and do not provide the context. We change these questions by asking for *one of the most likely* and providing the context. In the result, 20 answers of these questions become *commonly true*. For example, the task *use transition sound* becomes *meaningful* when we provide the context “add a slide transition in powerpoint”. Table 4.5 shows the evaluation results between different crowdsourcing setups – different questions for turkers – in details.

<table>
<thead>
<tr>
<th>“most likely”</th>
<th>“one of the most likely”</th>
<th>“most likely”</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>without context</strong></td>
<td><strong>without context</strong></td>
<td><strong>with context</strong></td>
</tr>
<tr>
<td>randomly 50 questions – answer: not commonly true</td>
<td>6 answers become: commonly true</td>
<td>8 answers become: commonly true</td>
</tr>
<tr>
<td>- clean room has sub-task take all dirty dish</td>
<td>- Context: “organize your room in a fun and stylish way”, clean room has next task go shopping for food and decorations</td>
<td></td>
</tr>
<tr>
<td>randomly 100 questions – answer: not commonly true</td>
<td>20 answers become true</td>
<td></td>
</tr>
<tr>
<td>- Context “activate all cores in your pc”, activate all core is meaningful</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Context “dye cotton yarn”, mix dye has previous task soak yarn</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.5:** Result of different crowdsourcing setups

It turns out by providing more precise questions – asking for “one of the most likely” along with the context, the accuracy of task frames steadily increases. Table 4.6 shows some examples from the result of this task.
The Open IE 4.2 wrapper leads to 1.29 million task frames, derived from 1.94 million high quality triples from 1.96 million sentences from 168,697 WikiHow articles. Table 4.7 reports the high quality of the extracted task frames measured against the ground truth. This demonstrates that our KB construction stage is robust, eliminating most input noise and keeping errors very low.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Value</th>
<th>Answer</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>Sing lullabies to babies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td><strong>sing lullaby</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meaningful</td>
<td>True</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Prop</td>
<td>lullaby</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Sub-task</td>
<td>hold baby</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Sub-task</td>
<td>create rhythm</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Sub-task</td>
<td>keep environment</td>
<td>True</td>
<td>0.67</td>
</tr>
<tr>
<td>Sub-task</td>
<td>focalise on melody</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Sub-task</td>
<td>try to sing or hum soft slow song</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Context</td>
<td>Have a good day in club penguin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td><strong>tattle to friend</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meaningful</td>
<td>True</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Living being</td>
<td>friend</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>Next task</td>
<td>earn some coin</td>
<td>False</td>
<td>0.65</td>
</tr>
<tr>
<td>Next task</td>
<td>check newspaper</td>
<td>False</td>
<td>0.65</td>
</tr>
</tbody>
</table>

Table 4.6: Evaluation on task frame extracted by using Open IE 4.2

Table 4.7: Quality of task frames extracted by our Open IE 4.2 wrapper

<table>
<thead>
<tr>
<th>Task attribute</th>
<th>Total</th>
<th># Correct</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task phrase meaningfulness</td>
<td>150</td>
<td>149</td>
<td>0.99</td>
</tr>
<tr>
<td>Location</td>
<td>3</td>
<td>3</td>
<td>1.0</td>
</tr>
<tr>
<td>Participating Object (prop)</td>
<td>188</td>
<td>185</td>
<td>0.98</td>
</tr>
<tr>
<td>Participating Living Being</td>
<td>17</td>
<td>15</td>
<td>0.88</td>
</tr>
<tr>
<td>Parent task</td>
<td>106</td>
<td>106</td>
<td>1.0</td>
</tr>
<tr>
<td>Previous task</td>
<td>106</td>
<td>101</td>
<td>0.95</td>
</tr>
<tr>
<td>Next task</td>
<td>103</td>
<td>100</td>
<td>0.97</td>
</tr>
<tr>
<td>Sub task</td>
<td>108</td>
<td>107</td>
<td>0.99</td>
</tr>
</tbody>
</table>

4.3 KB Organization

The basis of KB organization is clustering which in turn relies on the quality of the similarity function. As discussed in Section 3.3.2, our similarity function achieves very high accuracy. Using all the features, it reaches a precision of 97.6% on the test set.

In the experiment described in this section, we assess the quality of the clusters obtained through our method and through the baselines.
Baselines

In practice, divisive clustering methods are often faster than agglomerative bottom-up methods. As we devised a hybrid bottom-up/top-down clustering method, a natural baseline thus is a traditional top-down divisive algorithm.

Evaluation setup

We are interested in measuring the quality of the clusters, having already evaluated the quality of pair-wise similarities. To construct ground truth, we randomly select 100 task frame pairs from a random set of clusters, where each pair comes from the same cluster. Additionally, we select 100 task frame pairs where the two frames in each pair come from different clusters. For each of these 200 pairs, CrowdFlower workers assess if a pair is similar enough to be clustered together. With this ground truth, we can compare the precision of different clusterings, i.e., the ratio of the number of correctly clustered pairs against the total number of pairs. We also measure the run-time of the two algorithms to compare their performance.

Results

Our clustering algorithm outperforms top-down clustering both in terms of speed and precision. Figure 4.1 shows that our clustering is much more efficient when increasing the size of the data set. Our gains in speed are due to the pre-clustering in the bottom-up phase, leading to much smaller start clusters for the top-down phase. Our bottom-up clustering step achieves a precision of 91.2%, a major gain over the baseline which has 71.6% precision.

To see the effectiveness of the KB organization, figure 4.2 and 4.3 show us the results for searching task use keyboard before and after organizing the KB. Although the result before KB organization includes many information as we want, these information are ambiguous between use music keyboard (red frame) and use computer keyboard. Meanwhile, after KB organization, the system returns us three different task frames, separated depending on their senses which include use music keyboard, use computer keyboard and somehow use mobile phone keyboard. From these results, it can be said that by leveraging all information from task frames and using simple clustering techniques, we can successfully induce the sense of task frames without depending on any existing resources like WordNet.
Figure 4.1: Running time of top-down vs. our clustering

![Graph showing running time comparison between top-down and our clustering methods. The x-axis represents the number of task frames, and the y-axis represents time (in sec). The graph displays a clear distinction between the two methods, with our clustering showing a marked improvement in efficiency.]

Figure 4.2: Result for searching task use keyboard before clustering

<table>
<thead>
<tr>
<th>ACTIVITY</th>
<th>use keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAGE</td>
<td>![Images of keyboard use scenarios]</td>
</tr>
<tr>
<td>CATEGORY</td>
<td>Phone, Technical Reading, Windows, Android, Computer Emotions, Symbols and ASCII Art, Mac, Input Devices, iPad, Music, Listening and Appreciation, Microsoft Word,</td>
</tr>
<tr>
<td>PARO</td>
<td>Keyboard</td>
</tr>
<tr>
<td>PARENT</td>
<td>select add new keyboard, use space key, go to loc email keyboard setting, use horizontal line tool</td>
</tr>
<tr>
<td>SUBACTIVITY</td>
<td>select add new keyboard, use space key, go to loc email keyboard setting, use horizontal line tool, integrate in other song, use the keyboard to insert degree symbol, enable keyboard</td>
</tr>
<tr>
<td>PREV</td>
<td>select add new keyboard, use space key, go to loc email keyboard setting, use horizontal line tool, integrate in other song, use the keyboard to insert degree symbol, enable keyboard</td>
</tr>
<tr>
<td>NEXT</td>
<td>select add new keyboard, use space key, go to loc email keyboard setting, use horizontal line tool, integrate in other song, use the keyboard to insert degree symbol, enable keyboard</td>
</tr>
<tr>
<td>URL</td>
<td>select add new keyboard, use space key, go to loc email keyboard setting, use horizontal line tool, integrate in other song, use the keyboard to insert degree symbol, enable keyboard</td>
</tr>
<tr>
<td>RATING</td>
<td>3.726</td>
</tr>
<tr>
<td>VIEW</td>
<td>2319566</td>
</tr>
</tbody>
</table>
4.4 Resulting HowToKB

Baselines

There is no direct competitor that provides semantically organized task frames. The best approximations, to some extent, are Knowlywood (Tandon et al. 2015) and ConceptNet (Speer and Havasi 2012). Actually, they focus on different kinds of commonsense knowledge, but they do contain some task-oriented knowledge as well. Knowlywood does not have sub-activity relations, while ConceptNet neither has a clear notion of tasks nor canonicalized frames. Therefore, similar to Tandon et al. (2015), we align HowToKB with ConceptNet-5 (CN) such that the left argument of the triple must be similar to a task name (verb [preposition] object), and the right argument can be a task or a noun concept. If necessary, we invert the relations in CN to swap the arguments and align CN relations to task frame attributes (see Table 4.8).

Evaluation setup

Given a task with its context, we ask CrowdFlower workers to fill its attributes: location, time, participants, parent, previous, next and sub-task. From the results of the experiment on sub-section 4.2.2, we realized that it is important to inform the judges the context on every question. An example is In some context such as “decorate the house”, the most likely location when we paint a wall would be? The answer for
this question is filled by the workers, for instance, *house*. For this task, we selected 50 meaningful task frames from the frames used in the other evaluations.

Results

The evaluation results are shown in Table 4.9. We report the statistical significance by means Wilson score confidence intervals for $\alpha = 95\%$ (Brown et al., 2001). The results show that the clustered task frames in HowToKB have very high quality.

The activities derived from ConceptNet and Knowlywood were assessed based on the ground truth. There is very low coverage: ConceptNet has only 1 of the 50 ground-truth tasks, and Knowlywood has 4 activities that match any of the 50 tasks. These numbers confirm that the kind of how-to task knowledge that HowToKB possesses is not captured by any prior work on commonsense knowledge acquisition.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Precision</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meaningful task phrases</td>
<td>0.94±0.04</td>
<td>0.51M</td>
</tr>
<tr>
<td>Location</td>
<td>0.70±0.27</td>
<td>18K</td>
</tr>
<tr>
<td>Participating object</td>
<td>0.89±0.04</td>
<td>1.4M</td>
</tr>
<tr>
<td>Participating living being</td>
<td>0.88±0.11</td>
<td>136K</td>
</tr>
<tr>
<td>Parent task</td>
<td>0.81±0.0</td>
<td>0.92M</td>
</tr>
<tr>
<td>Previous task</td>
<td>0.83±0.05</td>
<td>0.91M</td>
</tr>
<tr>
<td>Next task</td>
<td>0.84±0.04</td>
<td>0.91M</td>
</tr>
<tr>
<td>Sub task</td>
<td>0.88±0.03</td>
<td>0.92M</td>
</tr>
<tr>
<td>Images</td>
<td>0.88±0.03</td>
<td>1.03M</td>
</tr>
</tbody>
</table>

Table 4.9: Quality and Size of HowToKB
4.5 Use case: YouTube Video Search

In order to demonstrate the usefulness of HowToKB, we studied a use case to automatically find relevant YouTube videos $D$ for a query $q$. $q$ is a task title, e.g., paint a wall. The usefulness of HowToKB would be evident if it is able to expand $q$ to $q'$ using relevant context from task frames. If $q$ is absent in HowToKB, this context comes from task frames which are semantically related to $q$.

**Baselines**

As baseline, we consider a system that does not expand $q$ and uses the terms in $q$ to match against a YouTube video’s title and description. This is a strong baseline because many instructional videos either have very informative titles that describe tasks, or contain rich descriptions.

**Evaluation Setup**

To construct the dataset ($D$), we crawl YouTube videos and gather the title, description, tags, category, and, comments. Some of this information is too general or unrelated. For example, category can be too coarse while the comments are typically opinions on the entire video sequence and may be unrelated to the query task ($q$).

As this use-case has not been previously considered, there is no ground truth that ranks $d \in D$ for $q$. Instead, we leverage WikiHow pages that can reference a YouTube instructional video. There are typically one or two linked video URLs in a WikiHow article, and these become relevant YouTube videos. The resulting dataset consists of 18,380 $q/d$ pairs, typically one relevant video per query.

This dataset can be used to measure coverage, and we explore the relevant metrics. Precision@K is equivalent to HITS@K in this setup because there is typically one relevant video per query. MRR (Mean Reciprocal Rank) is another relevant metric that measures the position of the first correct result. We consider both HITS and MRR as our evaluation metrics.

Our system expands $q$, systematically using two sets of attributes from the task frames, namely:

- Inter-task attributes ($I_{attr}$), consisting of parent task, similar task and previous or next tasks.
Results

Table 4.10 reports HITS and MRR metrics for the baseline, $I_{attr}$, $C_{attr}$, and $(I + C)_{attr}$. These are aggregated results from a total of 18,380 queries. It is clearly evident that the expansion of $q$ to $q'$ provides very rich context that leads to more than doubling the evaluation metrics. Oftentimes, YouTube videos describe the tools or objects you need for the task, or present the related tasks by providing a step-by-step procedure. HowToKB-based expansion gains advantage using these descriptions; see Table 4.11 for an example of the successful queries.

<table>
<thead>
<tr>
<th>k</th>
<th>$q$ (baseline)</th>
<th>$q + C_{attr}$</th>
<th>$q + I_{attr}$</th>
<th>$q + I_{attr} + C_{attr}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1421</td>
<td>1867</td>
<td>2199</td>
<td>2361</td>
</tr>
<tr>
<td>3</td>
<td>2411</td>
<td>3061</td>
<td>3408</td>
<td>3619</td>
</tr>
<tr>
<td>5</td>
<td>3058</td>
<td>3779</td>
<td>4077</td>
<td>4340</td>
</tr>
<tr>
<td>10</td>
<td>4259</td>
<td>5036</td>
<td>5043</td>
<td>5425</td>
</tr>
</tbody>
</table>

MRR  

0.076 0.101 0.12 0.128

Table 4.10: Results for YouTube search use case

- Contextual attributes ($C_{attr}$), consisting of location, time and participant information.

These sets provide us with more insight on the contribution of frame attributes for different categories of queries (these categories come additionally from the WikiHow task pages in the ground truth).

<table>
<thead>
<tr>
<th>$q$</th>
<th>$q + I_{attr} + C_{attr}$</th>
<th>Youtube Video</th>
</tr>
</thead>
</table>
| make caramel corn | make caramel corn ... brown sugar ... popcorn ... syrup teaspoon.. salt teaspoon ...bake soda ... vanilla ... | **Title: Gourmet Caramel Popcorn**
“Thanks Monique”
gourmet caramel popcorn ...ingredient ...popcorn tablespoon butter cup pack ...brown sugar cup ...karo syrup teaspoon ...vanilla teaspoon bake soda ... |
| dance salsa | dance salsa ... salsa master basics dance ...leader dance follower work ...cross body lead ... learn open break | **Title: Salsa Dancing for Beginners**
salsa dancing... beginner ... basics step ... salsa right turn ... ...cross body lead ... turn partner work... basic step ... |

Table 4.11: Examples of query expansions and corresponding search results
Chapter 5

Conclusion

This thesis presented HowToKB, the first comprehensive knowledge base with fine-grained attributes about how-to tasks. By tapping how-to communities such as WikiHow, combining information extraction techniques with judiciously designed form of hierarchical clustering, our methodology achieved both efficiency and effectiveness.

The first contribution of this research is the method that automatically constructs a large knowledge base on how-to tasks. This contains several novel elements:

- From the perspective of the AI community, our KB is the first large-scale KB focusing on task-solving steps, which can be used to improve search quality and experience for how-to task queries. Because mining how-to task knowledge is still a new problem, this KB has been showing a high potential to cope with other challenging research problems such as visual understanding, activity alignment or activity prediction.

- On the aspect of Data Mining, our simple method for KB organization, which combines both bottom-up and top-down hierarchical clustering techniques, has great advantages. Not only it achieves a high accuracy on inducing the senses of task frames, but it is also scalable.

- From the NLP perspective, our data sources – how-to communities such as WikiHow – not only have well-structured but also contain rich textual and visual descriptions on problem-solving tasks. Task frames extracted from these data by using information extraction techniques achieve high quality. Our research, once again, affirms the importance of quality of data sources. It can be said that rich data with a simple method can get better performance than a fancy method on poor data.
The second contribution of this research is the publicly available knowledge base HowToKB, which contains over half of a million tasks in the form of semantic frames and with linkage to visual content like images or videos and all data which is extracted from WikiHow.

The third contribution of this research is extensive experiments showing the high quality of HowToKB. The crowdsourced ground truth can be used as a resource for other researchers. Our use-case also demonstrates the usefulness of HowToKB by improving the results on searching relevant Youtube videos for how-to tasks.

Discussion

Because this is the first method working on mining how-to task knowledge, we would like to discuss both the strength and weakness of our approach.

Strength:

- Our method does not require any training data for information extraction step and any existing resources for disambiguation step. Most of previous works on KB organization depend on existing resources like WordNet or VerbNet, meanwhile our organization technique uses only WikiHow data itself (building word2vec model on WikiHow data for computing similarity and using information extracted from WikiHow data as features of clustering method).

- Our method is a combination of existing techniques, which is very flexible and can be adapted to other data sources. Open IE 4.2 can be used for any data on open-domain, while hierarchical clustering is a fundamental clustering technique which is widely used.

- Our hierarchical clustering technique allows us to easily update the KB when a new data comes.

- The frame representation is a good choice for describing how-to tasks. Each task frame can contain all information related to a how-to task, including visual information.
Weakness:

- Until now, our research only works on WikiHow data. Since each data source has a different structure, our method needs to be generalized to handle different data sources, especially for the information extraction and taxonomy construction steps.

- Our method depends on existing techniques, which directly affect the quality of our resulting KB. At the KB construction phase, we use Open IE 4.2 for extracting tasks. Although Open IE 4.2 outperforms other Open IE tools, this tool does not achieve high accuracy (72.34%). In our system, this is the step that needs to be improved.

- Our hierarchical clustering method needs to define a stopping criterion. Although hierarchical clustering methods do not require specifying the number of clusters, they do need to specify a threshold for stopping. Until now, our method assigns this threshold manually; however, it would be better if we can automatically learn this stopping criterion.

- It is not easy to evaluate the quality of our clustering method as well as our HowToKB. Most evaluation tasks need human intervention, which incurs high cost. Therefore, the number of random samples we pick from the KB is relatively small. Large-scale evaluations are desirable.

Future Work

There are many further research directions we can follow for future work.

- First, we plan to extend our current HowToKB by integrating more multimodal data (e.g., videos).

- Second, we intend to leverage even more signals present in online communities e.g. average rating, number of views, etc.

- Finally, we would like to turn this resource into an open-domain training dataset for potential applications such as activity-oriented computer vision tasks, text summarization and question answering.
List of Figures

1.1 Example of a task frame ........................................... 2
1.2 Examples on applications of HowToKB .......................... 5

3.1 HowToKB Architecture ............................................. 19
3.2 An example of articles from WikiHow ............................ 22
3.3 Example of task frame taxonomy construction ................. 23
3.4 Example of our clustering algorithm ............................ 31
3.5 Example of the update algorithm ................................. 32

4.1 Running time of top-down vs. our clustering .................. 40
4.2 Result for searching task use keyboard before clustering .... 40
4.3 Results for searching task use keyboard after clustering ...... 41
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overview of works on mining activity commonsense knowledge</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Example frame on task <em>a = paint a wall</em></td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>HowToKB statistics</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Anecdotal examples from HowToKB</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Extraction results of different combinations of three IE systems (L: Ollie, P: Open IE 4.2, R: Reverb)</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>Removing noise from Open IE 4.2 extraction results</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>Result of different crowdsourcing setups</td>
<td>37</td>
</tr>
<tr>
<td>4.6</td>
<td>Evaluation on task frame extracted by using Open IE 4.2</td>
<td>38</td>
</tr>
<tr>
<td>4.7</td>
<td>Quality of task frames extracted by our Open IE 4.2 wrapper</td>
<td>38</td>
</tr>
<tr>
<td>4.8</td>
<td>ConceptNet as KB baseline</td>
<td>42</td>
</tr>
<tr>
<td>4.9</td>
<td>Quality and Size of HowToKB</td>
<td>42</td>
</tr>
<tr>
<td>4.10</td>
<td>Results for YouTube search use case</td>
<td>44</td>
</tr>
<tr>
<td>4.11</td>
<td>Examples of query expansions and corresponding search results</td>
<td>44</td>
</tr>
</tbody>
</table>
Bibliography


