Planning for Reasoning with Multiple Common Sense Knowledge Bases

YEN-LING KUO and JANE YUNG-JEN HSU, National Taiwan University

Intelligent user interfaces require common sense knowledge to bridge the gap between the functionality of applications and the user's goals. While current reasoning methods have been used to provide contextual information for interface agents, the quality of their reasoning results is limited by the coverage of their underlying knowledge bases. This article presents *reasoning composition*, a planning-based approach to integrating reasoning methods from multiple common sense knowledge bases to answer queries. The reasoning results of one reasoning method are passed to other reasoning methods to form a reasoning chain to the target context of a query. By leveraging different weak reasoning methods, we are able to find answers to queries that cannot be directly answered by querying a single common sense knowledge base. By conducting experiments on ConceptNet and WordNet, we compare the reasoning results of reasoning composition, directly querying merged knowledge bases, and spreading activation. The results show an 11.03% improvement in coverage over directly querying merged knowledge bases and a 49.7% improvement in accuracy over spreading activation. Two case studies are presented, showing how reasoning composition can improve performance of retrieval in a video editing system and a dialogue assistant.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Theory and methods*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents*

General Terms: Design, Algorithms, Experimentation

Additional Key Words and Phrases: Common sense, intelligent user interface, interface agent, contextual reasoning, commonsense reasoning

ACM Reference Format:
Kuo, Y.-L. and Hsu, J. Y.-J. 2012. Planning for reasoning with multiple common sense knowledge bases. ACM Trans. Interact. Intell. Syst. 2, 3, Article 17 (September 2012), 24 pages.
DOI = 10.1145/2362394.2362399 http://doi.acm.org/10.1145/2362394.2362399

1. INTRODUCTION

Common sense knowledge is an essential element in building intelligent systems. It enables computers to infer new facts or to perform actions, using common sense about the world. Applications can then interact with users more intelligently. It also helps mitigate the software brittleness bottleneck by falling back on common sense knowledge when purely domain-specific knowledge fails.

Due to the increasing complexity of users' tasks, one-to-one correspondence between user interface controls and system functionality may not be sufficient to accomplish a user's goals. Common sense knowledge has been proposed to bridge the gap between

The reviewing of this article was managed by the special issue associate editors, Henry Lieberman and Catherine Havasi.
This work was supported in part by National Science Council and National Taiwan University under Grants NSC 99-2221-E-002-139-MY3, 10R80919-6, and 10R70500.
Author's address: Y.-L. Kuo (corresponding author); email: a33kuo@gmail.com.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permission may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701, USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2012 ACM 2160-6455/2012/09-ART17 $15.00
DOI 10.1145/2362394.2362399 http://doi.acm.org/10.1145/2362394.2362399

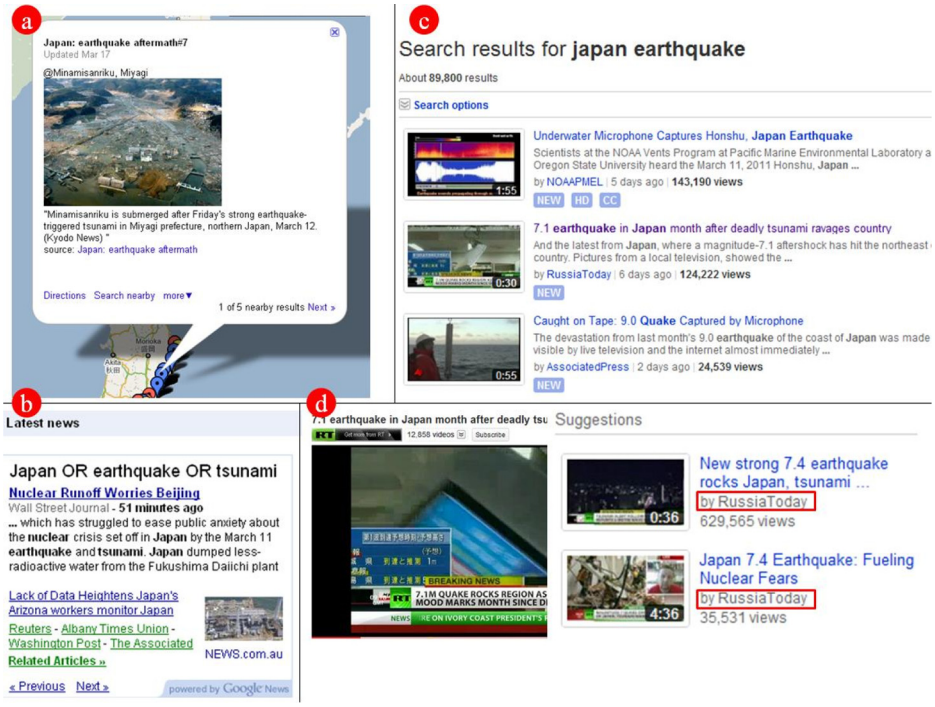


Fig. 1. Traditional media browsing: (a) map view, (b) chronological view, (c) keyword search, (d) video suggestion by producer.

users' goals and the functionalities of applications [Lieberman 2009]. An interface agent makes use of a broad range of common sense knowledge, such as event-subevent structure, temporal relationships, and specific facts to provide assistance to users [Lieberman et al. 2004].

Imagine we are browsing media items on the Web to gather information about the earthquake that struck Japan on March 11, 2011. It is not easy to achieve a comprehensive understanding of this event using conventional media browsing interfaces. Most of the interfaces provide only a single view to list media items. Figures 1(a) and 1(b) sort news according to timeline or location. We need to browse through every item in the collection to find the desired items. The search or recommendation functionalities are only text search and recommendation by authors (see Figures 1(c) and 1(d)). When we searched for video of "Japan tsunami", there were no videos mentioning the victims in the earthquake. These problems limit a user's experience in media browsing. An interface agent with common sense knowledge would act as an assistant for users. It would organize the media items using the context of an event, rather than simply as a context-independent list.

In order to make an intelligent choice of media appropriate to the user's context, it is necessary for the interface agent to have knowledge about the subject matter. This kind of contextual information can be used not only in media browsing but also in context-aware and storytelling interfaces. Conventionally, directly querying common sense *knowledge bases* (KBs) and spreading activation over common sense semantic networks are the most common approaches to get contextual information. Although many common sense KBs have convenient APIs, the sparsity of KBs and possible inaccuracy of reasoning results are problems that may be worse with common sense than

dealing with databases that store more factual information. The results returned by a direct query have high accuracy but low coverage. The results returned by spreading activation, on the other hand, have high coverage but low accuracy. If the coverage/accuracy is low, the interface agent may fail to help users, and user satisfaction might even be lower than if the agent were not present.

This research investigates the problem "How do we improve the coverage of common sense knowledge agents using multiple knowledge sources?" Instead of exploiting heuristics to improve the quality of contextual reasoning results, this article builds a reasoning layer on top of multiple common sense KBs and presents *reasoning composition*, a planning-based approach to combining different reasoning methods, to find an inference chain to the target concepts. We construct a profile of each reasoning method and synthesize them using a planning procedure. With reasoning composition, we are able to reason with multiple common sense KBs at the same time if any particular KB fails to answer the query.

This article starts by reviewing the problem of providing contextual information for interface agents and proposes an approach to answer queries of a particular type of contextual information. We then present our model and profile for the composition algorithm. Evaluation was designed to test the improvement in coverage and accuracy of reasoning results using our reasoning approach. Finally, we present case studies incorporating reasoning composition into video editing and dialogue assistance interfaces to see how the differences between reasoning approaches reflect on the interfaces. The article concludes our discussion with a review of other commonsense reasoning methods for intelligent systems.

2. CONTEXTUAL COMMONSENSE REASONING FOR INTERFACE AGENT

In this section, we review the contextual commonsense reasoning problem in building an interface agent. With a more robust contextual reasoning technique, interface agents would have better goal-achieving abilities.

2.1. What Does Common Sense Offer for Interface Agents?

Interface agents are intelligent systems, which can give users effective assistance in accomplishing specific tasks. Instead of asking users to initiate every action required in the system, the interface agent may observe a user's actions and manipulate the objects displayed in the interface [Lieberman 1997]. In order to fulfill this goal, the agent should be equipped with large amounts of knowledge about the end users or about specific domains [Maes and Kozierok 1993]. However, the ability of an agent is limited by the knowledge it possesses and by the actions it has seen the user perform.

Common sense knowledge has been proposed to mitigate the problem of software brittleness [Lenat et al. 1986]. With common sense knowledge of different domains, such as people's goals and actions, the interface agent can model a broader range of users and context. For example, Roadie [Lieberman and Espinosa 2007] uses the temporal relationship between concepts to provide plausible actions for the planner to plan the possible sequence of actions when using a consumer electronics product. Storied Navigation [Shen et al. 2009] is another interface that uses the context of concepts to represent a sentence and match the annotated videos to compose video stories.

2.2. Resources for Contextual Common Sense Reasoning

Many resources have been developed to provide different kinds of contextual information of concepts. It is straightforward to equip a variety of applications with common sense by querying these resources using APIs. For example, one may ask if a specific assertion is present in the corpus, possibly accompanied by a confidence score.

An interface agent uses these resources to propose possible actions so that users can select from these actions. Most of these resources represent common sense sentences as a semantic network. ConceptNet [Havasi et al. 2007] encodes about 20 relationships for applications to retrieve specific facts or conceptual relationships. WordNet [Miller 1995] groups words of the same meaning into synsets and organizes them in a hierarchical structure, which is used for generalizing concepts or finding synonyms, antonyms, etc. LifeNet [Singh and Williams 2003] and EventNet [Espinosa and Lieberman 2005] utilize the event-subevent structure of the Open Mind Common Sense[1] corpus to produce a network for reasoning about temporal relationships. A goal network [Smith and Lieberman 2010] is also used for facilitating goal recognition and planning.

2.3. Problem Definition

The two approaches for reasoning are directly querying the concepts, computing similarities of concepts [Speer et al. 2008], or applying spreading activation in the network. The spreading activation algorithm traverses links from the selected concepts and includes every assertion of the specified relation types into the reasoning results. Although it is convenient for interface agents to use APIs to access the contextual reasoning results, there are still two major problems in the reasoning quality.

(1) *Coverage*. Coverage is like "recall" except that it is about inference rather than just retrieval. Compared with the amount of common sense knowledge humans have, the coverage of current common sense KBs is relatively sparse. A single KB may not contain sufficient knowledge of the domain an interface agent requires.
(2) *Accuracy*. Accuracy is like "precision" in information retrieval. Since the spreading activation algorithm returns every concept that appears in the inference chains, it carries the danger of bringing irrelevant concepts to applications. For example, we can find "tour" when we request the related concepts of "president" just because "the White House is used for tours." If the interface agent has lots of noisy data, it is very likely to give useless information to users and lower the predictability of the interface.

Both problems diminish the interface agent's ability to provide assistance to users. Previous work tries to consult other knowledge sources when the coverage of the desired domain is low. Most interface agents also use heuristics to get rid of noise or manage the inference chain to guarantee its quality. For example, Roadie [Lieberman and Espinosa 2007] only considers the actions that can be used by the planner. This approach helps constrain the actions generated by EventNet. Storied Navigation [Shen et al. 2009] uses WordNet for concept expansion and the context provided by ConceptNet as a filter to remove the irrelevant concepts. However, it is hard to apply these heuristics to other application domain or common sense KBs. Coverage and accuracy remain unsolved problems for interface agents that would like to incorporate common sense.

2.4. Ideas for Solving This Problem

Instead of developing heuristics to improve the quality for different application domains, we think that it is important to provide contextual reasoning results of higher accuracy and coverage for interface agents. In order to improve the coverage, multiple KBs should be used to surpass the limitation of the concepts and reasoning power in a single KB. Previous research also shown that the results of reasoning in different

[1]http://openmind.media.mit.edu/

Table I. Reasoning Methods

Name	KB	Relations used in this reasoning method
Geralizer	WordNet	Hypernym
Geralizer	ConceptNet	IsA
Similar-concept finder	ConceptNet	All relation
Action finder	ConceptNet	CapableOf, UsedFor, and Causes
Actor finder	ConceptNet	CapableOf and Desires
Location finder	ConceptNet	AtLocation and LocatedNear
Object finder	ConceptNet	HasA, MadeOf, UsedFor, and AtLocation
Goal finder	ConceptNet	Desires, CauseDesires

partitions of a KB can be combined to perform reasoning [Amir and McIlraith 2005]. However, assertions in common sense semantic networks are not suitable for logical reasoning because the data may be noisy. We need to have other approaches designed for reasoning in semantic networks.

Since most inference chains in contextual reasoning can be decomposed into multiple steps, our idea for improving coverage of reasoning results is to use different weak reasoning methods in different steps to produce reasoning results. The synthesis of multiple KBs is then achieved by combining those reasoning steps. We can use the additional concepts/relations in other KBs to enhance the coverage of reasoning results. We can also select reasoning methods in our inference chain that we have reason to believe lead to answers rather than include every activated concept.

3. REASONING COMPOSITION

In this section, we provide a model for composing reasoning methods. Selecting a series of weak reasoning methods as steps in the inference chain to answer queries can be viewed as a planning problem. With the definition of state transition in composition, we can use a planning algorithm to select reasoning methods to form the inference chain. The proposed descriptions for reasoning methods and a planning procedure are used for answering queries.

3.1. Reasoning Methods for Contextual Information

The context of a concept can be categorized into different categories or dimensions [Lenat 1998]. Every time an application sends a request for the contextual information of a concept, it asks for a particular category of context. In order to get the context of a category, reasoning methods utilize different relation types in a common sense semantic network to retrieve the desired context of an input concept. For example, a reasoning method uses the "CauseDesire" relation in ConceptNet if it wants to find the goal of users when it detects that the user is doing a particular action. These simple reasoning methods form the basis for longer inference chains.

We summarize the most used contextual information finding methods in Table I. They are *generalizer, similar-concept finder, action finder, actor finder, location finder, and object finder*. Generalizer uses the class hierarchy in a semantic network to generalize a concept to its parent class. For instance, it should return "disaster" for "earthquake." Similar-concept finder gets the most similar concepts for an input concept. In most cases, similarity is defined by the shared attributes/properties of concepts. Therefore, a similar-concept finder should identify that "tsunami" and "flood" are similar concepts because they are disasters and bring lots of water. Other reasoning methods use relations as a filter to retrieve the neighboring subevents/actors/locations/objects

of an input concept. We require all of these methods to be efficiently computable in large semantic networks so that we won't suffer from the increase of complexity.

3.2. Reasoning Composition as a Planning Problem

With multiple available reasoning methods, we can then use other reasoning methods if we cannot find the output when using a single reasoning method. For example, we do not have the location of a golden retriever in ConceptNet, but we do have "Golden retriever is a dog" and "You can find a dog in a park" in ConceptNet; we can then use the generalizer as an intermediate step to get "dog" and use "You can find a dog in a park" to retrieve "park" as a related location of golden retriever.

The problem now is selecting a sequence of reasoning methods to get the target context. Selection of reasoning methods corresponds to the selection of actions in a planning problem. Therefore, it is straightforward to model it as a planning problem. In the following text, we start with two common sense KBs, ConceptNet and WordNet, and the query "find related locations of golden retriever" to explain how reasoning composition works as a planning problem. The concepts in the common sense KBs and the available reasoning methods form the planning domain. Its states and operators are defined as follows.

(1) *State*. A state in this representation contains all the concepts in the available common sense KBs. In our example, they are all the concepts in ConceptNet and WordNet. Each concept is attached with one of the three labels: "unactivated", "new", and "activated." Unactivated concepts are concepts that haven't been the input/output for any reasoning method. The concepts labeled "new" are concepts that are output by a reasoning method but haven't been the input for any reasoning method. If a concept is involved in input of one reasoning method and output of another reasoning method, we label it as "activated."
(2) *Initial state*. In the initial state, only the input concept in the query is labeled "new"; all the other concepts are labeled as "unactivated." The plan is empty. From the sample query, we know that only the input query concept "golden retriever" is labeled as "new."
(3) *Goal state*. The goal state is the state in which at least one of the concepts with "new" labels falls in the category of context specified in an application's query. In our example, the concepts that are locations such as "park", "store", etc, can be labeled as "new" in the goal state. However, there are over a million states satisfying this requirement. We need to choose the state that is most related to the input query as our goal. The evaluation of relatedness could be the distance to the initial state since "semantic drift" occurs in traversing a semantic network.
(4) *Operator*. An operator is a reasoning method such as the "Location finder" or "Generalizer" listed in Table I. It has input and output concepts. The precondition is that the input concepts of a reasoning method must be labeled "new." The state changes after executing a reasoning method. The effect of executing a reasoning method is to turn the labels of input concepts from "new" to "activated" and turning the labels of output concepts from "unactivated" to "new." For example, when we apply "Generalizer" on initial state, it turns the label of "golden retriever" from new to activated and turns the label of "dog" and "animal" from unactivated to new.

The final output of reasoning composition is the sequence of instantiated reasoning methods and concepts that are labeled "new" and match the specified output category in the goal state. These output concepts are the answer to the input query. In our example, "park" and "home" are plausible answers to the input query; the corresponding plan may be "we use action finder to find the related actions of golden retriever, and

then use location finder to get the locations in which these actions take place." The output sequence is used to help us explain our answers to applications.

3.3. Profile of a Reasoning Method

In order to generate a composition plan over these weak reasoning methods automatically, we need to have a description of a reasoning method. The *profile* of a reasoning method defines the precondition and effect of an operator so that we can find a path through the state space. However, since there are over 300,000 concepts in the combination of ConceptNet and WordNet, it is not feasible to describe a reasoning method with all the available concepts. We solve this problem by assigning concept types to the ground concepts. Instead of describing the precondition and effects of a reasoning method by concepts, we describe them by concept types. The concept types categorize the plausible state transitions. Only concepts of the defined concept types can be instantiated as the input of a reasoning method. The following paragraphs give our definition of profiles.

3.3.1. Property. In order to specify the precondition and effect of reasoning methods, we consider the following attributes as the properties of an instance of a reasoning method. These properties help us define the operators for the planner to use.

(1) *Input concept*. An input concept of a reasoning method is specified by an interface agent or the output of other reasoning methods (i.e., the concepts with "new" as their labels.) For example, "golden retriever" is an input concept for generalizer and its output "dog" can be the input concept for the action finder to get related actions.
(2) *Input concept type*. We assign concept types to the ground concepts to prevent listing all the possible concepts in the profile. Intuitively, every concept must be assigned to a concept type when we use it in natural language. The input concept type helps us constrain the state transitions. With the concept types, our composition procedure only instantiates reasoning methods that have the same input concept type as the concept type of a concept labeled "new." For example, if we have a "new" concept "home" of concept type "location", we don't need to instantiate the operator "location finder" because there is no location information for two locations. These concept types prune search space so that we can find the plausible composition chains efficiently and reduce noise by not considering the concepts in incompatible concept types.
(3) *Output concept type*. The output concept type is also attached to every concept output by the reasoning methods. With the concept type for an output concept that is labeled "new", we can check if it is in the concept types requested by the interface agent. If it is not our targeted output, we continue the reasoning chain and instantiate other reasoning methods with new output concepts.
(4) *Access KB*. In most cases, a reasoning method accesses only one common sense KB. Different common sense KBs may provide the same reasoning function for the applications. For instance, both WordNet and ConceptNet can generalize a concept using their own class hierarchy. We can only distinguish between the two reasoning methods by describing the KBs they access.

Using these profiles, we define the precondition of a reasoning method as a state that contains a concept that matches the input concept type of a reasoning method. The effect of a reasoning method is moving to a new state by adding a concept of the specified output concept type to the current state.

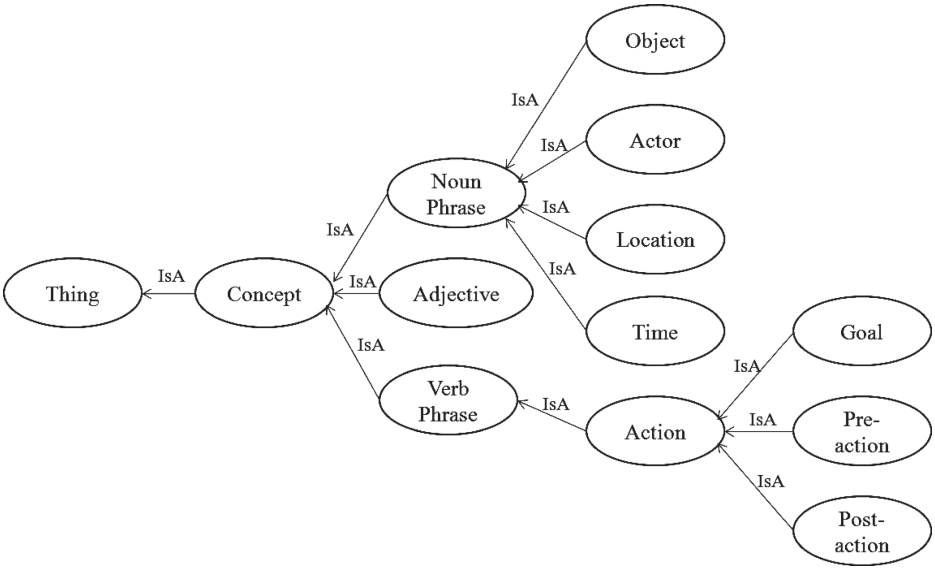


Fig. 2. The ontology of concept types.

Table II. Input and Output Concept Types of Reasoning Methods

Name	Input concept type	Output concept type	Example input/output
Geralizer	Noun phrase	Noun phrase	golden retriever/animal
Similar-concept finder	Any concept	Any concept	dog/cat
Action finder	Object, actor, and location	Verb phrase	dog/run
Actor finder	Action	Actor	teach/teacher
Location finder	Object, actor, and action	Location	dog/home
Object finder	Object, actor, location, and action	Object	play/frisbee
Goal finder	Adjective, action, object	Goal	hungry/eat

3.3.2. Concept Type. The input/output concept types used in the profile of reasoning methods follow the ontology defined in Figure 2. The goal of this ontology is to keep reasoning methods connected to natural language, the representation interface agents use to interact with users. Therefore, concepts are associated with three major types: noun phrase, verb phrase, and adjective, which correspond to grammatical concepts we use in our daily conversation. An interface agent can use natural language tools, for example, the NLTK toolkit[2] or the Stanford dependency parser[3], and its application domain to determine the concept type of an input query concept. For instance, if the parser annotates "park" as a noun and WordNet hierarchy classifies it as "location", then we know location is the concept type of park. The reasoning method generates the output concepts along with their types so that the output concepts can be used to match with the input of other reasoning methods. The type of the output concept is uniquely determined by the reasoning method that generates it.

Table II summarizes the input and output types of each reasoning method. This kind of abstraction simplifies the state transition with natural language concepts. By

[2]http://www.nltk.org/
[3]http://nlp.stanford.edu/software/lex-parser.shtml

matching with concept types rather than natural-language text, we turn concepts into a more computationally useful and efficient form for the planner to use.

3.3.3. Input Query. Once the state space and reasoning methods are defined, the interface agent can directly use an input query to find the contextual information of a concept. An input query should contain two parts. The first part of the input query is the text of the input concept, which is represented in natural language. The second part specifies the input concept type and output concept type of the query. The input/output concept type is denoted by types introduced in the previous paragraph. For example, we use "golden retriever:actor;location" as our input query if we would like to find the related locations of the actor "golden retriever."

An input query identifies the initial state and the condition of the goal state. In the initial state, only the input concept is labeled as "new". In the goal state, at least one concept that is labeled "new" matches the output concept type specified in the input query. Our composition algorithm handles the input query, composes the plan, and executes reasoning methods to get the contextual information of the input concept.

3.4. Composition Procedure

With the KBs and profiles of reasoning methods, we can run our composition algorithm over the available reasoning methods with the given input query. However, we cannot use classical planning algorithms in our problem since all classical planning algorithms, for example, STRIPS [Fikes and Nilsson 1971] and GraphPlan [Blum and Furst 1997], are offline algorithms. Our reasoning methods dynamically label concept as "new" according to the input concept. We cannot generate a composition plan in the beginning if we do not know the output concepts for every input concept in advance. There are planners designed for handling contingency [Pryor and Collins 1996]. However, their implementations are too slow to be of practical use in handling the requests from user agents. In order to deal with the output concepts given in the runtime of reasoning methods, we apply forward-search on our state space to generate the composition plan.

3.4.1. Composition by Forward-Search. We chose a state-space representation for the planning graph at the beginning of this section. Each node of the planning graph represents a state. Therefore, given an initial state, we can compose a plan by choosing an operator whose precondition is true in the current state until the current state satisfies the goal. In our problem, we select the reasoning methods whose input concept types match the types of the concepts that are labeled "new" until the type of the new concepts matches the target type defined in the input query.

In the query for the contextual information of a concept, we would like to get concepts that are most relevant to the input concept. The longer a reasoning chain is, the more likely it will suffer from semantic drift. For example, we may use a long reasoning chain such as "golden retriever is a kind of pet, people may have pets, people can see a movie, and a movie is in a movie theater" to get the result "golden retriever is related to movie theater." However, a movie theater is far from our initial domain, for instance, pets. To alleviate this problem, we require our planning algorithm to have a *finite horizon*. We do not keep activating reasoning methods whenever we find a match of concept type. Instead, we stop to explore a new state if the state has low relevance to the input query. The horizon of this composition problem is determined by the relevance of a state to the input concept. In this article, we use $\frac{1}{\|I\| \times d}$ as the evaluation function of relevance for a state, where $\|I\|$ is number of "new" and "activated" concepts in the state, and d is the depth of the state to the initial state. The $\|I\|$ here indicates the number of intermediate concepts involved in this branch of state transition. The

longer the reasoning chain is or the more the intermediate concepts involved in the reasoning chain, the more likely it would include irrelevant concepts in the results. For example, in our previous long reasoning chain, the state that results from activating the action finder for people has a large $\|I\|$ since there are lots of actions people are involved in. The results would be less reliable if we consider all the people-related actions. Therefore, the relevance evaluation can also be viewed as a pruning procedure to help us remove the irrelevant states.

In our problem, the contextual information is provided by different reasoning methods in different KBs. Therefore, there may be polysemy and redundancy in different KBs and inconsistency in the crowd-sourced KBs. For example, the concept "apple" is related to a fruit in WordNet but related to a company in Wikipedia; the concept "vegetable" is related to both healthy and unhealthy in ConceptNet because the common sense knowledge contributed by the general public contains the statement that "fruit is a kind of food" and "food can be healthy"/"food can be unhealthy." For different KBs containing polysemy, we can summarize the knowledge in a KB into a KB profile to match the KBs that contain the desired knowledge domain [Kuo and Hsu 2011]. For concepts that have conflicts, we still maintain all the concepts in the output of the reasoning method but associate them with a confidence score that indicates the reliability of the concept. In this article, we normalize the frequency of an assertion and multiply it by the relevance of the state as in our confidence scores. Applications can then use these scores to decide which concept to use. In most cases, an application would choose the concepts that agree with most people as its default.

Algorithm 1 shows how the forward-search composition algorithm works. Taking our query for the location of golden retriever as an example, Figure 3 is the resulting state transition after applying the forward-search composition. In Figure 3, the "unactivated" concepts are omitted due to lack of space. (The number of unactivated concepts is equal to the number of concepts in all KBs minus the activated and new concepts.) For query "golden retriever:actor;location", the input query concept q is golden retriever, input type t_i is actor, and output type t_o is location. We use 0.002 as our relevance threshold. In the initial state, only golden retriever is marked as "new". Since the output list L is empty at the beginning, we can activate the reasoning methods whose precondition matches the initial state. We choose "ConceptNet location finder", "ConceptNet action finder", and "WordNet generalizer" for execution. After execution, we cannot find any output by location finder. The first branch turns golden retriever to "activated" and stops at the first step. Even if we fail to retrieve answers from one reasoning method, we still have the results from other reasoning methods. In this example, action finder labels "chase frisbee" because it finds "golden retriever is capable of chasing frisbee" in ConceptNet; "generalizer" labels "dog" and "retriever" because it finds "golden retriever is a kind of dog/retriever" in WordNet. Since the derived states do not contain the concepts that match the desired output type "location", we evaluate their relevancy to the initial state to decide if we can continue the reasoning chain. The second branch gets 0.5 and the third branch gets 0.33 as their relevance score. Because these confidence scores are larger than the threshold, we continue to select reasoning methods for execution. (Otherwise, we will break the loop and return an empty list.) The activated concept "golden retriever" is labeled as "activated" to prevent a loop in our composition algorithm. In the third iteration, both branches activated location finder and find the output "park" and "home". Our algorithm adds them to the output concept list and returns the list as the related locations for "golden retriever."

3.4.2. Rule-Based Implementation. In our forward-search algorithm, we would like to get all the concepts in the nearest goal states. So, we choose to concurrently execute all the matched reasoning methods. In the actual implementation, it is hard to maintain

ALGORITHM 1: Forward-Search Composition (q)

Input: A set of KBs $\mathcal{K}$, an input query concept q with type t_i, and a output concept type t_o, and a relevancy threshold θ
Output: A list of related concepts L with confidence scores.
$L \longleftarrow \emptyset$; /* Output concept list. */
$d = 0$; /* Length of state transition. */
Label the input query concept q as "new" in the state space;
repeat
  S = Execute the reasoning methods $\mathcal{R}$ whose precondition matches the current states to get the derived states;
  $d = d + 1$;
  for *each state s in S* **do**
    C = Get all the concepts that are labeled "new" in s;
    for *each concept c in C* **do**
      if *type of c matches* t_o **then**
        Add c with confidence score into L;
      end
    end
    I = Get all the concepts that are labeled "new" and "activated" in s;
    if $\frac{1}{\|I\| \times d} > \theta$ **then**
      continue;
    else
      break;
    end
  end
until $L \neq \emptyset$;
Sort concepts in L by confidence;
return L

the status of every concept because there are over 300,000 concepts in one state and 6 branches of a state at a time. Our program needs to keep track of over 2 million concepts in the worst case.

From the example in Figure 3, we know that most of the concepts are "unactivated" in reasoning composition. We can simply keep track of the "new" and "activated" concepts and use them to decide which reasoning method should be activated. In our implementation, we use the ontology of profiles to maintain the instantiated reasoning methods and input/output concept, and a rule reasoner to infer possible state transitions for the current state whenever the new output concepts are generated by a reasoning method.

Ontology of Profiles. The instances of activated input/output concepts and reasoning methods are stored in the ontology of profiles. This ontology is different from the common sense KBs, which are used for representing the structure of common sense semantic networks. It represents the current state of composition. The instances of reasoning methods tell us which reasoning method is chosen and its associated input/output. We use this ontology as another layer to manipulate the selection of reasoning methods. Starting with an instance of an initial concept and the classes of reasoning methods, we can infer possible reasoning methods of the input concepts and assert output concepts into this ontology to continue the inference chain.

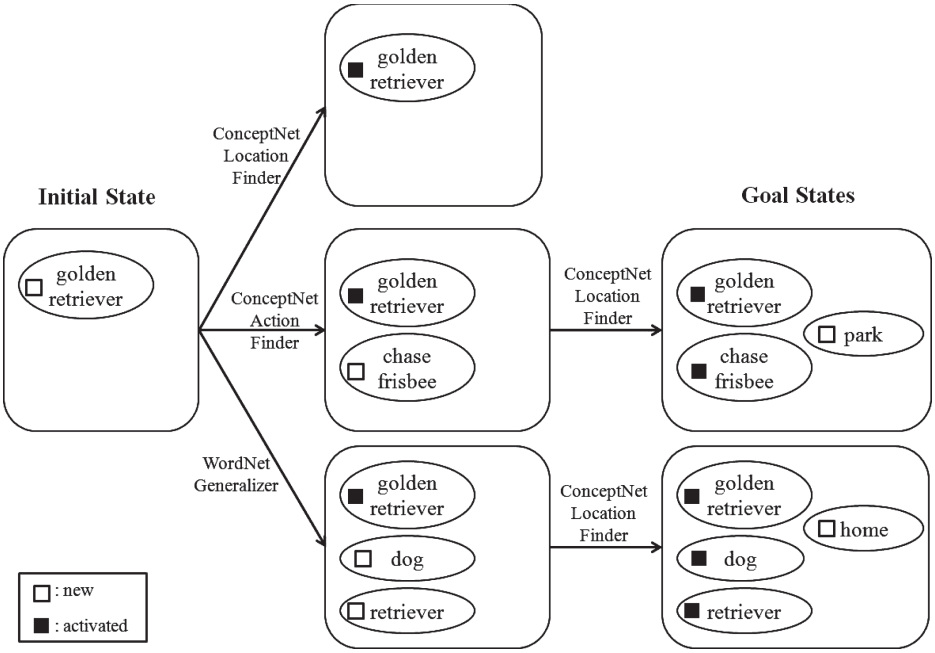


Fig. 3. Example of state transition in reasoning composition.

Reasoning Rules. Using a rule reasoner, a reasoning method is executed only if the input concept and its profile satisfy the rules. This requirement simulates previous forward-search algorithm to get reasoning results. Two rules are selected for basic composition of reasoning methods. They are listed as follows.

(1) If there is a new input query and its input concept type matches the input concept type of a reasoning method, infer a new instance of the reasoning method and use the new input concept as its input. For example, the input concept of our query is "golden retriever" and it is an "actor". It searches for all the reasoning methods that take "actor" as input. In this case, it activates "action finder" because action finder returns related actions for actors, objects, and actions.
(2) If the concept type of a current output concept matches the input concept type of a reasoning method, infer a new instance of the reasoning method and use the output concept as its input. As in our golden retriever example, the output related action of golden retriever is "chase frisbee". The rule reasoner then activates "location finder" because the location finder takes actors, objects, and actions as its input.

Rule-Based Composition Algorithm. Our procedure of rule-based composition is illustrated in Algorithm 2. In each iteration, our system asserts the current output concepts of reasoning methods into the ontology of reasoning profiles to make state transition. A rule reasoner is applied on the ontology to infer instances of reasoning methods using the composition rule defined in this section. The inferred reasoning methods are then executed to add new output concepts into the ontology. This procedure continues until the type of output concepts matches the targeted output type, which is our goal in contextual common sense reasoning, or the expected relevance of the inference results is low. The expected relevance can simply be estimated by an evaluation function using the number of concepts involved in the composition I and the length of the reasoning chain d as in Algorithm 1.

ALGORITHM 2: Rule-Based Composition (q)

Input: An ontology P that contains the profiles of reasoning methods, a input query concept q with type t_i, and a output concept type t_o, and a relevancy threshold θ.
Output: A list of related concepts L.
$L \longleftarrow \emptyset$; /* Output concept list */
$I \longleftarrow \emptyset$; /* Intermediate concepts in reasoning chain. */
$d = 0$; /* Length of reasoning chain. */
Assert input concept q with type t_i into P;
$\mathcal{R}$ = Apply rule reasoner on P to infer possible reasoning methods;
repeat
  C = Execute the inferred reasoning methods $\mathcal{R}$ to get reasoning results;
  for *each concept c in C* **do**
    if *type of c matches* t_o **then**
      Add c into L;
    else
      Assert c into P; Add c into I;
    end
  end
  $d = d + 1$;
  if $\frac{1}{\|I\| \times d} > \theta$ **then**
    $\mathcal{R}$ = Apply rule reasoner on P to infer possible reasoning methods;
  else
    break;
  end
until $L \neq \emptyset$;
Sort concepts in L by confidence score;
return L

3.4.3. Discussion. Our composition process facilitates the interoperation of common sense KBs because our algorithms automatically use other reasoning methods to compose reasoning chains. Since we use reasoning methods as building blocks and execute them independently, we can easily add more reasoning methods to the ontology of profiles as one of the available reasoning methods for composition. This composition algorithm can also be incorporated into a multiagent common sense integration system [Kuo and Hsu 2011] to support the cooperation of distributed common sense KBs.

For reasoning in a logical framework, our approach corresponds to partition-based reasoning [Amir and McIlraith 2005] where reasoning procedures are done in each partition. They then pass the inference results in one partition to other partitions to perform joint reasoning. This demonstrates that the reasoning methods in our reasoning composition may be first-order theorem provers to incorporate logical reasoning. So, it is possible that we can not only retrieve contextual information in a common sense semantic network, but also answer first-order queries with our approach.

Since we model the composition of reasoning results as a planning problem, we can also consider the cost of each reasoning method in our model as the cost of actions in a classical planning problem. We can use the cost of different reasoning methods to decide which reasoning method we would like to instantiate given our available computing resources for common sense reasoning methods.

In the next section, we'll present experiments on our rule-based implementation of reasoning composition to determine the improvement of coverage of reasoning results.

Table III. Statistics of Knowledge Bases

Knowledge base	Number of concepts
ConceptNet	274,477
WordNet	128,391

4. EVALUATION

In order to evaluate the proposed reasoning composition procedure in some example common sense semantic networks, we used the ConceptNet and WordNet APIs to implement the atomic reasoning methods. We would like to compare the coverage of reasoning results between reasoning composition and other conventional knowledge-based retrieval methods.

4.1. Experimental Setup

The English version WordNet contains approximately 130,000 words. ConceptNet 4 has over one million English sentences and a comparable number of concepts to WordNet. Given that the English ConceptNet and WordNet are currently among the largest common sense semantic networks, we decided to choose them as our experimental KBs for implementing reasoning methods. Despite the large number of concepts in both knowledge bases, as shown in Table III, they are quite inconsistent and incomplete. The overlap of concepts in English ConceptNet and WordNet is only 4.79%. This number suggests that there must be room for improvement of the coverage of query results if we can integrate the two KBs for reasoning.

4.1.1. Reasoning Method. The reasoning methods form the basis for reasoning composition. In our prototype system, six reasoning methods are used for providing contextual information of an input query. They are the generalizer, the similar-concept finder, the action finder, the actor finder, the location finder, and the object finder.

Two generalizers are implemented to find the more general class or description of a concept. They use the hypernym and IsA relations to generalize a concept in WordNet and ConceptNet, respectively. A similar-concept finder outputs concepts of the same type as an input concept. As in AnalogySpace [Speer et al. 2008], we implemented a similar concept finder by representing concepts in the English ConceptNet as a high-dimensional feature-vector space. Similar concepts are then retrieved from the proximate concepts of their inputs in this high-dimensional vector space. For other reasoning methods, we utilized the relations (see Table I) mentioned in the previous section for retrieval. The reasoning methods we implemented are summarized in Table II.

4.1.2. Ontology of Profiles. Since reasoning methods are the functions of a KB, a reasoning method can then be viewed as a service provided by a common sense KB. The ontology descriptions for service profiling, for example, OWL-S [Martin et al. 2004], can also be used for specifying the reasoning methods and their input/output. Therefore, the profiles of the aforementioned reasoning methods and their activated input/output concepts are stored in a predefined ontology that follows the OWL-S specification so that we can easily apply a rule reasoner to.

4.1.3. Rule-Based Planner. In our prototype system, we used Jena[4], a Semantic Web framework for Java, to achieve the rule-based composition. The composition rules are written in Jena rules. The generic rule reasoner of Jena is used for applying our rule set to the ontology of profiles to infer new instances of reasoning methods. The inferred

[4]http://jena.sourceforge.net/

reasoning methods are then executed and produce output concepts to add to the ontology of profiles. With new output concepts, our rule reasoner can infer more reasoning methods to incrementally build up the reasoning chain. In contrast to the reasoning in large common sense KBs, it is much more efficient to do reasoning in a small ontology of reasoning profiles.

4.1.4. Experimental Dataset. In order to compare the results of reasoning composition with other methods for querying contextual information, we sampled 750 concepts from English ConceptNet for this experiment. The sampling procedure is as follows. We first sorted concepts according to their assertion counts. Concepts with assertions whose count is less than 5 or greater than 15 are then filtered. We do not use concepts whose assertion counts are too small because we may not find them as common concepts in other KBs. The concepts with larger assertion counts are also filtered because they already have enough information for inference in ConceptNet alone. After filtering, 750 concepts are randomly sampled from the remaining concepts and are annotated with input concept types. Finally, every sampled concept is combined with one of the four output types: "actor, location, action, and object" to form 3,000 queries as defined in the previous section.

4.2. Experimental Result

These queries are used as the input of the prototype system to get contextual reasoning results. The output of our proposed composition procedure is compared to the output of two contextual reasoning approaches. One directly queries the common sense KB for the targeted contextual information. Another uses spreading activation to get the related concepts. Both the coverage and accuracy of output answers are evaluated to see how the proposed reasoning composition procedure performs.

4.2.1. Comparison with Direct Query. For this comparison, we used two baselines.

(1) *Directly query English ConceptNet*. English ConceptNet alone contains contextual information of concepts through various relations. We can simply query English ConceptNet to get the neighbors of a concept as its context.
(2) *Directly query the merged KB of ConceptNet and WordNet*. One way to get contextual information from multiple KBs is to query the merged KB without reasoning. In this experiment, the concepts with the same name in different KBs are merged into one concept in the merged KB; the relations with the same meaning, for instance, IsA and hypernym, are also merged into one relation link. The combination of KBs may also improve the coverage of reasoning results.

Coverage. The first part of the experiment looks into the coverage of reasoning results. The coverage is the ability a reasoning approach has to answer the input queries. If a reasoning approach gives an answer to the input query, we marked it as a "hit." The coverage is thus defined as the hit rate to a set of input queries:

$$hit\ rate = \frac{\#\ of\ answered\ queries}{\#\ of\ input\ queries}.$$

Figure 4 compares the hit rate of the baselines and our reasoning composition approach.

ConceptNet itself answered 1,961 queries out of 3,000 queries. If we only asked the merged KB of ConceptNet and WordNet without further reasoning, it answered 2,076 queries. The hit rate grew from 65.34% to 69.20% by combining the KBs. However, the hit rate of the merged KB is still not ideal for applications to use. Our proposed reasoning composition approach answered 2,474 queries in this experiment (i.e., hit

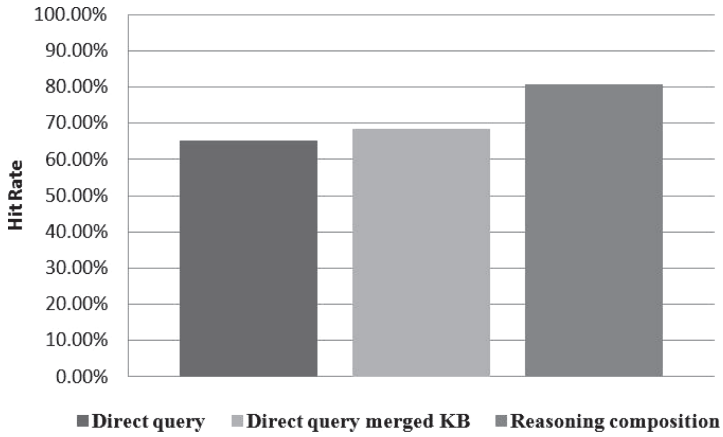


Fig. 4. Hit rate of different query approaches.

rate = 82.47%). It is shown that the reasoning composition improved the coverage of answers over 12% against the merged KB. Since the coverage represents the percentage of uses' actions that can be processed by the interface agent, we think the interface agent can handle the majority of actions with our proposed reasoning composition. The other unprocessed user actions can then be covered by designing the response to be "fail-soft", that is, to have the interface respond in a manner no worse than if there were no assistance at all from the agent.

Accuracy. In addition to the input queries answered by the merged KB, the reasoning composition approach answered 398 other queries. For the queries that were answered by the merged KB and reasoning composition, we have high confidence on their accuracy since they have direct links to the input concept in the common sense semantic networks. For other queries that are answered only by reasoning composition, they are shuffled to Amazon Mechanical Turk[5] to be voted on.

Twenty reasoning results are put in a HIT (human intelligence task) to ask workers on Mechanical Turk (Turkers) to judge if the concepts are related or not related to the input queries. In this HIT, 2 answers labeled as ground truth are mixed with other answers to ask Turkers. If a Turker rates the questions differently from the ground truth, their judgment is not included in our evaluation. Turkers whose judgment passed the ground truth test received US $0.10 for their judgment, and their rating would be considered valid in our evaluation. After this process, each reasoning result is rated as either relevant or nonrelevant by at least 3 randomly selected Turkers. It is treated as a good answer if at least half of the Turkers rated it as relevant.

There are 331 out of the 398 answers rated as good answers. Although our algorithm does not guarantee high accuracy, we still have reasonable accuracy (83.17%) in our experiment. If we remove the bad answers from the reasoning results of reasoning composition, the hit rate is still maintained at 80.23%. Compared with direct query approaches, our approach improves the coverage of reasoning results over 12%, with the trade-off of some drop in accuracy. Our result shows that the coverage of answers still improves a lot after removing bad answers. Therefore, it is feasible to answer queries by composition of reasoning methods even if some errors may be produced in integrating reasoning results from noisy common sense KBs like ConceptNet.

[5]https://www.mturk.com/

Error Analysis. In our current implementation, we found that bad answers are usually the answers that are too general or the wrong concept types. The first issue comes from the generalizer and similar-concept finder. For example, we got "mall", "school", and "beach" as the related location of "play soccer" because "play soccer" can be generalized to "play" and retrieve concepts that are places for playing. This problem suggests that the reasoning methods should selectively return results so that we can produce the composition plans more accurately. The second issue comes from the ambiguity of natural language concepts. Since concepts in ConceptNet are collected in natural language texts, the users often input texts in order to form complete sentences rather than fill in the blank with answers of the expected type. Therefore, the reasoners in each KB may give output that is not our expected type for the planner to construct the plan sequence. For example, we may get activities "buy product" and "spend money" as the answer to the query "related objects of smoker".

These kinds of problems show limitations in reasoning with a collaboratively built KB. Since the data is noisy and represented in a flat network, care must be taken in combining inductive reasoning results by planning.

4.2.2. Comparison with Spreading Activation. We also evaluated reasoning results against those produced by spreading activation [Collins and Loftus 1975], a common approach to getting contextual information by propagating inferences through selected relations in the semantic network. Since spreading activation cannot differentiate all the types used in the 3,000 input queries, we grouped "actor" and "object" into one type. In this part, we chose the queries that could only be answered by reasoning composition for comparison because the ones that could be answered by direct query should also be answered by reasoning composition. After grouping by types, 398 queries answered by reasoning composition were reduced to 338 queries. We then used spreading activation on English ConceptNet to get answers for these queries.

Coverage. In this comparison, spreading activation answered only 167 out of the 338 input queries. We find that the coverage is improved using a single KB with inference, but the improvement is not as much as our reasoning composition approach, which integrates multiple KBs. The main reason is that we fail to infer any knowledge if there is no such knowledge in the existing KBs.

Accuracy. As with the comparison with direct queries, the answers output by spreading activation were also put on Amazon Mechanical Turk to verify their accuracy. We then mapped the good answers to 1 point and bad answers to 0 points. The average score for spreading activation was only 0.419; the reasoning composition got 0.916 for the same 338 queries. Comparing the scores of spreading activation with our reasoning composition using a paired t-test showed that the average 0.497 point difference was statistically significant at the $p < 0.0001$ level, with a 95% confidence interval of (0.422, 0.572).

Discussion. From our observation, we find that spreading activation on ConceptNet provides reasonable coverage but with low accuracy. For example, when we query for the objects that relate to the concept "play frisbee", it returned "have fun, computer" because both playing on a computer and playing frisbee are related to "have fun". With reasoning composition, we find that "playing frisbee takes place outdoors, outdoors is a kind of outside (infer from WordNet generalizer), sport may be at outside, and sport equipment is used for sport". Therefore, we can return "sport equipment" for the query for the objects related to "play frisbee". The result shows higher precision than ConceptNet spreading activation because we have a better plan to select reasoning methods rather than spreading arbitrarily. At the same time, we

Table IV. Found Related Actors and Objects of "president"

Spreading Activation	Reasoning Composition
(measure distance, 0.2194), (president of united state, 0.2038), (event, 0.1638), (united state america, 0.1618), (government, 0.1511), (idea, 0.1511), (plant, 0.1477), (racist, 0.1307), (jail, 0.1307), (brat, 0.1307), (star movie, 0.1306), (famous person, 0.1306), (michigan, 0.1265), (american president, 0.1286), (chess piece, 0.1279), (fool, 0.1183), (gentleman, 0.1091), (idiot, 0.032)	(leader, 0.24), (politician, 0.2), (elect official, 0.12), (chief of state, 0.12), (coporate executive, 0.23), (academic administrator, 0.12), (elect leader, 0.12), (presidency, 0.12), (head of state, 0.12)

also improved the coverage of reasoning results by introducing other reliable KBs, for example, WordNet, because better concepts and relations are involved in the reasoning process. We conclude from the comparison that using reasoning composition to combine reasoning methods provides more flexibility and inferential power than the spreading activation approach.

5. CASE STUDIES

In order to provide an easy method for interface agents to obtain contextual reasoning results, we developed a REST API. An agent simply sends an input query via the API to get contextual information for the target concept. The reasoning functionality of two interface agents are replaced by reasoning composition to see if they provide improved results for some example applications, in our case, for video editing and dialogue assistance.

5.1. Study 1: Storied Navigation

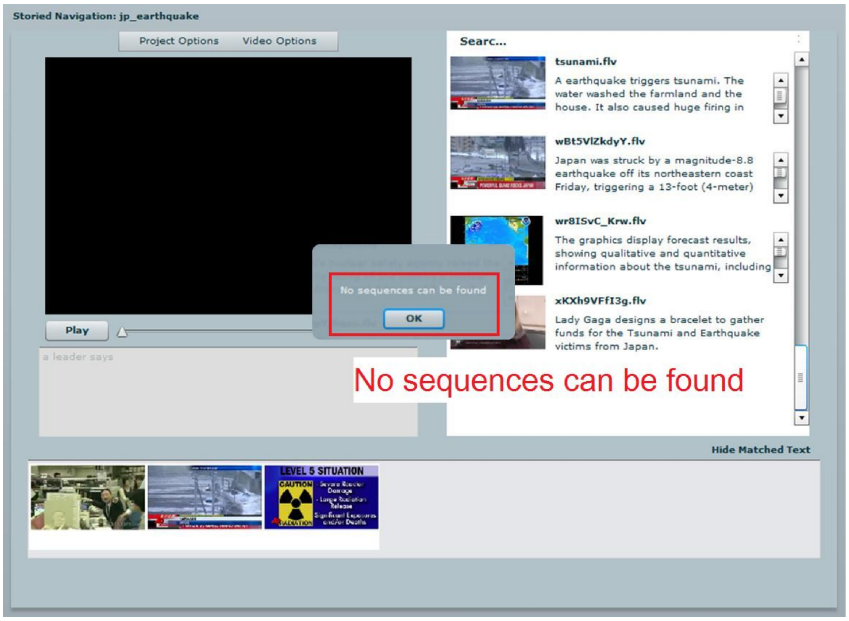
Storied Navigation [Shen et al. 2009] is a video editing system that helps editors compose a video story by selecting a sequence of videos from a set of annotated clips. After each clip is imported into the system, Storied Navigation extracts the key concepts from its annotation and represents a sentence into a concept representation of contextually related concepts. Editors can then type sentences in natural language to retrieve the video sequences that share similar concept representations.

In this study, we choose "the Japanese earthquake of March 11, 2011" as our story domain. The video corpus consisted of 23 clips with English descriptions collected from YouTube[6]. The description of clips served as the text used in the annotation phase. In our annotation phase, these videos are processed by spreading activation and by our reasoning composition API, respectively, to create a conceptual representation.

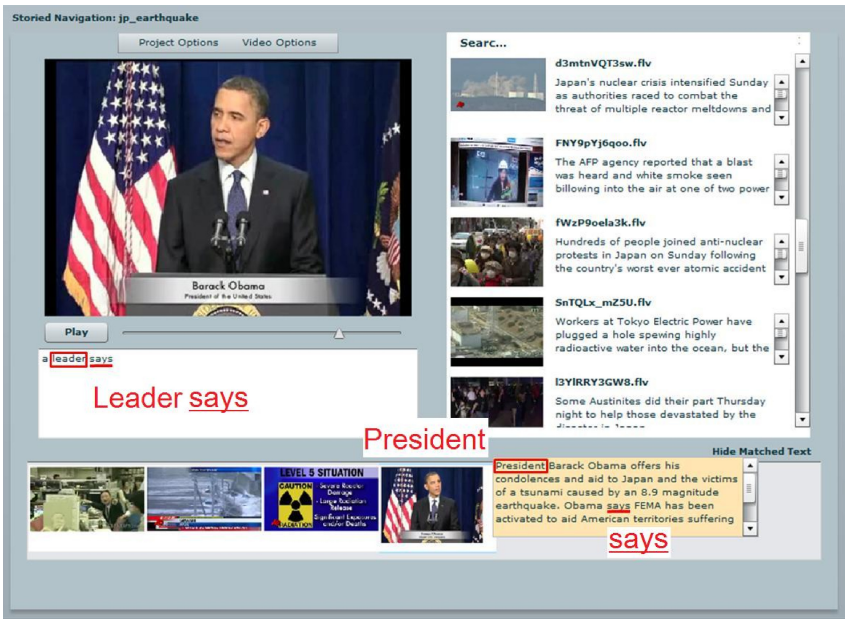
From our data, Storied Navigation used 42 input queries during the annotation phase. There are 13 out of the 42 queries answered by spreading activation on ConceptNet; 23 of them are answered by reasoning composition in ConceptNet and WordNet. Comparing the related actors and objects retrieved by the two approaches (see Table IV), we find that spreading activation returns many but diverse answers, which require filtering before the interface agent can use them. Using our reasoning composition, the interface agent is able to use the results directly because it has more relevant knowledge.

The difference in reasoning results is illustrated by the following example. When an editor tries to find a public speech of a leader by typing "a leader says...", as shown in Figure 5, Storied Navigation with reasoning composition finds the video in which President Obama gave a speech to Japan because we know "president" is a kind of "leader"

[6]http://www.youtube.com



(a) Storied Navigation with spreading activation



(b) Storied Navigation with reasoning composition

Fig. 5. Storied Navigation suggests video when user types “a leader says...”.

in WordNet and “president” is “at location” of “country” in ConceptNet. Therefore, it includes “leader”, “president”, and “country” as part of the concept representation of the natural language description. We can use this context representation to match candidate videos. On the other hand, Storied Navigation with spreading activation fails

to find a relevant video because it brings many irrelevant concepts and none of them match up with the concept representation of the video descriptions.

5.2. Study 2: Dialogue Assistant

Dialogue Assistant is another interface that helps users continue their conversation with others by associating related items to a topic. It uses picture cards as communication cues. Since a conversation is similar to a storytelling process, these picture cards are grouped into the story elements actor, activity/event, location, and object, which correspond to our defined concept types. The agent generates the picture cue cards using the topic, or pictures that the user selects, to suggest plausible items for conversation. The Dialogue Assistant helps the user to brainstorm about what should be included in their conversation.

We developed two versions of Dialogue Assistant. One queries the English ConceptNet for related items. Another uses reasoning composition to get the four kinds of story elements. Figure 6 shows differences between the two approaches in handling the same action. The interface agent with reasoning composition provides more picture cues than the agent that uses a direct query alone.

The selected picture cues in a conversation session forms an association sequence to a topic. Two hypotheses are to be verified: (1) "our system helps user produce longer association sequences" and (2) "these sequences are also reasonable to other people." A user was asked to compose 10 association sequences for the topic of "earthquake" and "travel." Each sequence was required to consist of 10 picture cues because a satisfactory conversation should have a minimum length. Unfortunately, the user can only produce an association sequence of 2 or 3 picture cues using the Dialogue Assistant with direct query. The user completed 6 sequences for "earthquake" and 4 sequences for "travel" using the Dialogue Assistant with reasoning composition. After creating the association sequences, 35 online users were recruited to rate each association pair in these sequences. If the two concepts in a pair are highly associated and related to the topic, the pair got 5 points. Otherwise, it got 1 point. We use the middle score given by each user as the relevance score for a pair. Figure 7 shows the number of pairs for each middle score.

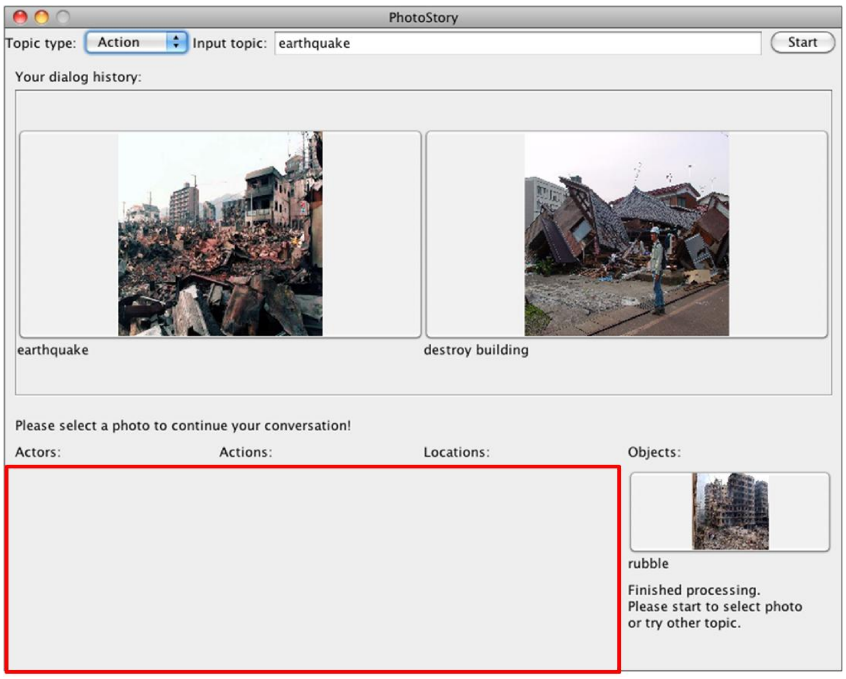
There are 55 pairs (middle score = 4 or 5) out of 90 pairs considered reasonable and related to the topic. There are 18 pairs (middle score = 2 or 1) considered to be composed of less relevant concepts and not so related to the specified topics. This showed that most of the cues suggested by Dialogue Assistant form a reasonable transition of story elements to other people.

6. RELATED WORK

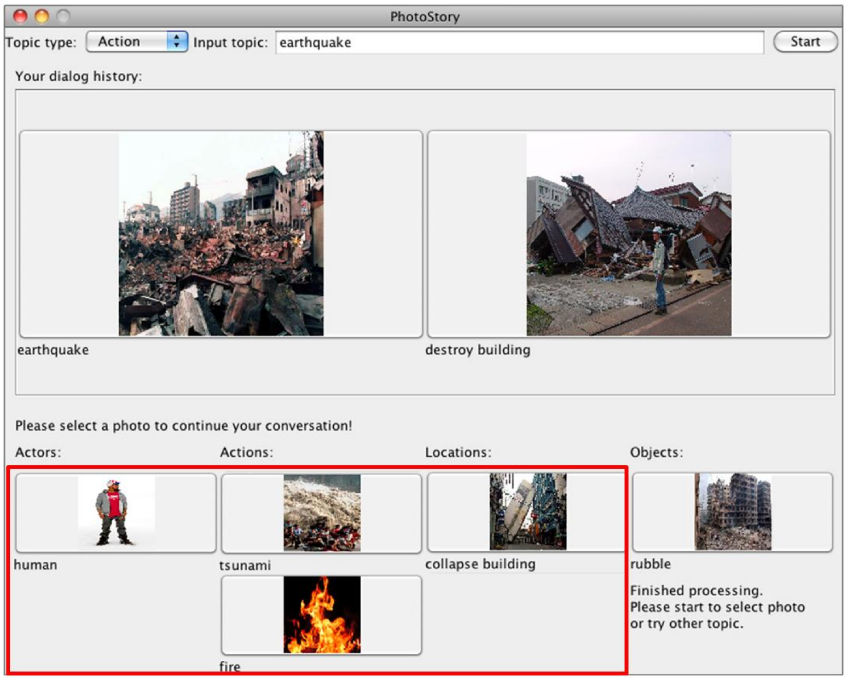
Context has long been used in common sense reasoning. In this section, we are going to discuss the relationship between retrieving contextual common sense knowledge and context. Since knowledge bases with different representations may call for different reasoning methods, our discussion of related work involves reasoning in the two most prominent representations, the semantic network and the formal logic framework.

6.1. Context in Common Sense Knowledge

Unlike classical logic reasoning, which is context-free, context has been considered as an important element in common sense reasoning. In first-order logic representations of common sense knowledge, a context generally consists of assumptions and a set of rules/assertions that are true in that context. Cyc uses what it calls a *microtheory* [Lenat 1998] to define context, according to a number of different dimensions. Microtheories are used to localize search and constrain or guide the inference in a KB.



(a) Dialogue Assistant with direct query



(b) Dialogue Assistant with reasoning composition

Fig. 6. Dialogue Assistant gives picture cues when user selects “destroy building”.

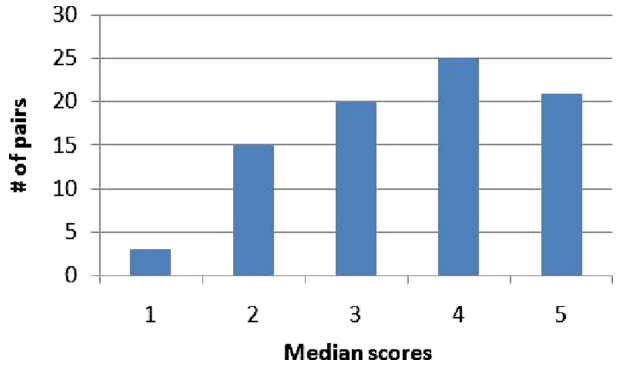


Fig. 7. Number of pairs for each middle score.

In another common sense semantic network, Contextualizer [Lieberman et al. 2004] was proposed to identify the context of each assertion. It asked users to add explicit context descriptions for each assertion. After encoding the context of a KB, applications can then retrieve the context an assertion depends on or analogous assertions across contexts. Both definitions of context use context as a filter to constrain inference. It differs from the goal of this article, which is to provide contextual information for an interface agent to use. We will need to incorporate context into conditions for reasoning method selection to produce better reasoning results.

6.2. Common Sense Reasoning in Semantic Networks

The ability to find similar concepts is helpful for interface agents to give suggestions by retrieving analogous items. Measures of similarity quantify how much two concepts are alike. Since most common sense semantic networks are crowd-sourced KBs, similarity measures play a role in reasoning in large and noisy semantic networks.

AnalogySpace [Speer et al. 2008] generalizes the reasoning method called *cumulative analogy* [Chklovski 2003] so that it is robust enough in a large and noisy semantic network. The assertions are divided into *concepts* and *features*, that is, descriptions of concepts such as "UsedFor eating" or "dog IsA". The knowledge in ConceptNet is represented as a sparse matrix, and its most prominent features can be identified by using singular value decomposition (SVD). Concept similarity is defined in terms of their shared features.

Blending [Havasi et al. 2009] was proposed as a technique to integrate common sense into other systems. In particular, blending of common sense knowledge with domain-specific knowledge can be done by finding an analogical closure across multiple previously separated sources of data. It applies the same reasoning procedures on multiple KBs by combining two sparse matrices linearly into a single larger matrix. Reasoning with blended knowledge bases containing overlapping information can produce inferences that would not be produced from either input alone. The blended knowledge bases are often used for identifying the prominent senses of a body of domain-specific knowledge. For instance, applications have been developed to show how the blended knowledge bases help affect sensing [Cambria et al. 2009] and word sense disambiguation [Havasi et al. 2010].

Our reasoning composition allows KBs to use their own reasoning procedures while integrating the data from multiple KBs. So, both AnalogySace and blending can also be integrated into our reasoning composition as one of our reasoning methods to provide reasoning for a specific domain. With more reasoning methods involved, we can

produce various reasoning results in common sense semantic networks for interface agents.

6.3. Commonsense Reasoning in Formal Logical Frameworks

The oldest and largest logic-based common sense knowledge base is the Cyc ontology [Lenat 1995]. The Cyc project carefully crafted knowledge into CycL, a rigorous logic-based language to ensure its correctness. Now, the OpenCyc 2.0 ontology contains hundreds of thousands of terms with millions of assertions relating the terms to each other.

Cyc has several different reasoning techniques. Many are based on a logical framework that uses deduction and theorem provers to reason about facts. Heuristics are often applied to logic-based reasoning for better efficiency. OpenCyc[7] also released its planner for reasoning out actions and events with its rules and assertions. These reasoners are exploited in Cyc's intelligent assistant [Panton et al. 2006] to recognize a user's plan. Logical reasoning is focused on providing exact answers to users and is rarely applied to the contextual reasoning required by interface agents.

7. CONCLUSION

This article investigated the problem of contextual common sense reasoning for interface agents and proposed an approach using planning to reason with multiple KBs to improve coverage of answers. Experiments have been conducted with the reasoning methods of English ConceptNet and WordNet. The results showed an 11.03% improvement in the hit rate over directly querying the merged KB and a 49.7% improvement in accuracy over conventional spreading activation. Two case studies demonstrated this proposed reasoning composition approach can be easily embedded into interface agents so that they can perform robustly.

The proposed reasoning composition approach is motivated by interface agents but is not limited to this kind of application. It is also suitable for modeling the context in most context-aware applications.

Future work may extend reasoning composition to other types of reasoning techniques. Since the reasoning methods can be viewed as multiple atomic reasoning agents for different tasks, a multi-agent system [Kuo and Hsu 2011] is a promising way to enhance the flexibility of reasoning composition.

ACKNOWLEDGMENTS

The authors would like to thank Edward Shen for providing Storied Navigation for the case study, the OMCS team at MIT for the commonsense computing tools, the anonymous reviewers for their valuable comments, and the members in NTU iAgents Lab for their constant support and many productive discussions. Special thanks go to Henry Lieberman for his insightful advice in revising this article.

REFERENCES

AMIR, E. AND MCILRAITH, S. 2005. Partition-based logical reasoning for first-order and propositional theories. *Art. Intell. 162*, 49–88.

BLUM, A. AND FURST, M. 1997. Fast planning through planning graph analysis. *Art. Intell. 90*, 281–300.

CAMBRIA, E., HUSSAIN, A., HAVASI, C., AND ECKL, C. 2009. AffectiveSpace: Blending common sense and affective knowledge to perform emotive reasoning. In *Proceedings of the CAEPIA Workshop on Opinion Mining and Sentiment Analysis*.

CHKLOVSKI, T. 2003. Learner: A system for acquiring commonsense knowledge by analogy. In *Proceedings of the 2nd International Conference on Knowledge Capture (K-CAP'03)*. ACM, New York, NY.

[7]http://www.opencyc.org/

COLLINS, A. AND LOFTUS, E. 1975. A spreading-activation theory of semantic processing. *Psychol. Rev. 82,* 6, 407–428.

ESPINOSA, J. AND LIEBERMAN, H. 2005. Eventnet: Inferring temporal relations between commonsense events. In *Proceedings of the 4th Mexican International Conference on Artificial Intelligence*. Springer, Berlin, 14–18.

FIKES, R. AND NILSSON, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Art. Intell. 2*, 189–208.

HAVASI, C., SPEER, R., AND ALONSO, J. 2007. ConceptNet 3: A flexible, multilingual semantic network for common sense knowledge. In *Recent Advances in Natural Language Processing*, Borovets, Bulgaria.

HAVASI, C., PUSTEJOVSKY, J., SPEER, R., AND LIEBERMAN, H. 2009. Digital intuition: Applying common sense using dimensionality reduction. *IEEE Intell. Syst. 24,* 4, 24–35.

HAVASI, C., SPEER, R., AND PUSTEJOVSKY, J. 2010. Coarse word-sense disambiguation using common sense. In *Proceedings of the AAAI Fall Symposium on Common Sense Knowledge (FSS'10)*. AAAI Press.

KUO, Y.-L. AND HSU, J. Y.-J. 2011. Capability modeling of knowledge-based agents for commonsense knowledge integration. In *Proceedings of the 14th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA'11)*. Springer-Verlag.

LENAT, D. 1998. The dimensions of context-space. Tech. rep., Cyc Corp.

LENAT, D. B. 1995. CYC: A large-scale investment in knowledge infrastructure. *Comm. ACM 38,* 11, 33–38.

LENAT, D. B., PRAKASH, M., AND SHEPHERD, M. 1986. CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Mag. 6,* 4, 65–85.

LIEBERMAN, H. 1997. Autonomous interface agents. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'97)*. ACM, New York, NY, 67–74.

LIEBERMAN, H. 2009. User interface goals, AI opportunities. *AI Mag. 30*, 16–23.

LIEBERMAN, H. AND ESPINOSA, J. 2007. A goal-oriented interface to consumer electronics using planning and commonsense reasoning. *Knowl. Based Syst. 20*, 592–606.

LIEBERMAN, H., LIU, H., SINGH, P., AND BARRY, B. 2004. Beating common sense into interactive applications. *AI Mag. 25*, 63–76.

MAES, P. AND KOZIEROK, R. 1993. Learning interface agents. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*. AAAI Press, 459–465.

MARTIN, D., BURSTEIN, M., HOBBS, E., LASSILA, O., MCDERMOTT, D., MCILRAITH, S., NARAYANAN, S., PARSIA, B., PAYNE, T., SIRIN, E., SRINIVASAN, N., AND SYCARA, K. 2004. OWL-S: Semantic markup for web services. Tech. rep., W3C.

MILLER, G. A. 1995. WordNet: A lexical database for English. *Comm. ACM 38,* 11, 39–41.

PANTON, K., MATUSZEK, C., LENAT, D., SCHNEIDER, D., WITBROCK, M., SIEGEL, N., AND SHEPARD, B. 2006. Common sense reasoning – from Cyc to intelligent assistant. In *Lecture Notes in Computer Science*, vol. 3864, 1–31.

PRYOR, L. AND COLLINS, G. 1996. Planning for contingencies: A decision-based approach. *J. Art. Intell. Res. 4*, 287–339.

SHEN, E. Y.-T., LIEBERMAN, H., AND DAVENPORT, G. 2009. What's next?: Emergent storytelling from video collection. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems (CHI'09)*. ACM, New York, NY, 809–818.

SINGH, P. AND WILLIAMS, W. 2003. LifeNet: A propositional model of ordinary human activity. In *Proceedings of the Workshop on Distributed and Collaborative Knowledge Capture (DC-KCAP) at K-CAP*. ACM, New York, NY.

SMITH, D. A. AND LIEBERMAN, H. 2010. The why UI: Using goal networks to improve user interfaces. In *Proceedings of the 15th International Conference on Intelligent User Interfaces (IUI'10)*. ACM, New York, NY, 377–380.

SPEER, R., HAVASI, C., AND LIEBERMAN, H. 2008. AnalogySpace: Reducing the dimensionality of common sense knowledge. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI'08)*. AAAI Press.

Received May 2011; revised November 2011; accepted April 2012