Multiagent Distributed Ontology Learning

Leen-Kiat Soh
Computer Science and Engineering
University of Nebraska
115 Ferguson Hall
Lincoln, NE
(402) 472-6738

lksoh@cse.unl.edu

ABSTRACT

In this paper, we describe a framework for distributed ontology learning embedded in a multiagent environment. The objective of this framework is to improve communication and understanding among the agents while preserving agent autonomy. Each agent maintains a dictionary for its own experience and a translation table. The dictionary allows the agent to compare and discover relationships between a pair of words or concepts, while the translation table enables the agent to learn and record (a selected portion of) the vocabulary of its neighbors that is useful for the collaboration among the agents. The motivation for this distributed ontology learning is that each agent has its own experience and thus learns its own ontology depending on what it has been exposed to. As a result, different agents may use different words to represent the same experience. When two agents communicate, agent *A* may not understand what agent *B* and that hinders collaboration. However, equipped with the distributed ontology learning capabilities, agents are able to evolve independently their own ontological knowledge while maintaining translation tables through learning to help sustain the collaborative effort.

Keywords

Multiagent systems, distributed learning, ontology learning, Dempster-Shafer belief system

1. INTRODUCTION

In the real world, where human agents are autonomous, distributed, and capable of individual learning, there are different languages. To communicate or collaborate, humans speaking different languages either learn a common language or use a translator. Learning a common language that is not a person's native incurs imposes additional effort on that person and may result in disadvantages. However, speaking through a translator may be not cost-effective and may be not feasible in some applications. Similarly, in a multiagent environment, autonomous and distributed agents may encounter different events, gather different experiences, and learn different ontologies. The focus of this paper is to describe a distributed ontology learning framework in a multiagent environment.

In our framework, each agent maintains a collection of experience cases. Each experience case is a list of words with a list of classifying concepts. There are two ways that an agent can learn experience cases. First, users can teach them—by supplying a list of words and what the classifying concepts are for that list of words. Second, an agent can learn a new experience case through its interactions with its neighbors. As a result, each agent learns its own concepts based on its experiences and specialties. When a new experience case arrives, the agent needs to incorporate it into its dictionary and its translation table. This is supported by three important components: conceptual learning, translation, and interpretation, with a Dempster-Shafer belief system [1] as the underlying structure to maintain ontology consistency.

Our discussion here is related to [2]. In [2], however, the agents were not able to learn collaboratively in a multiagent system. Instead, the learning was conducted only between two agents via exchange of concepts (ontologies) where the agents were neither able to adapt to changes in concept definitions nor able to handle multiple assertions from different neighbors. Moreover, our framework addresses translation and interpretation of concepts, query processing and composition for collaboration among agents, and action planning based on traffic and agent activities, which indirectly control the learning rates of the agents.

2. METHODOLOGY

In our framework, the multiagent system is one in which agents can exchange queries and messages to learn about each other's ontology. To improve the communication and collaboration efficiency, agents determine whether some translation is worth learning, which neighbors to communicate to, how to handle and distribute queries, and how to plan for agent activities. The framework consists of two sets of components: operational and ontological. The operational components allow the agents to work together in a multiagent system. The ontological components allow the agents to communicate and understand each other.

2.1. Operational Components

There are three important operational components: query processing, action planning, and query composition. Note that in our framework, an agent sends out a query to its neighbor when it needs to find some additional experience cases for that some classifying concepts. The query consists of the concepts and may consist of also some experience cases that the agent already has. When a user submits a new experience case, the agent also treats it as a query. Thus, each agent must be able to process queries. Agents are required to compose queries as well as they also need to relay or distribute queries to other agents by modifying the queries in their own words. Finally, for the system to be effective, the query distribution and the ontology learning behavior are supported by an action planning component that makes decision based on the agent's environment such as message traffic and neighborhood profile.

2.2. Ontological Components

There are three important ontological components in our framework: conceptual learning, translation, and interpretation. We represent an experience case as a vector. Each vector consists of the classifying concept and then a list of words describing that concept. A concept may have many different experience cases. Different concepts may be used to classify the same list of words, resulting in different experience cases. A word may appear in different experience cases for different concepts as well. These experience cases can be further linked under a concept hierarchy. Moreover, for each concept, the agent also learns the description vector, combining all relevant experience cases together. This allows the system to incrementally learn and evolve existing ontologies.

2.2.1. Conceptual Learning

This component has three structures: description vectors, concept hierarchies, and semantic rules. A descriptor vector is concept-specific. A concept hierarchy links several concepts together. A semantic rule distinguishes a concept from all other concepts.

First, when submitted, each concept is supported by *n* experience cases, in which each case has a list of words. Then, this component examines these experience cases to update the description vector for that concept. A description vector consists of a list of word-frequency pairs. For each word found in all the experience cases describing the same concept, the agent finds its frequency. In this manner, the agent learns the different significance of the words describing a concept.

Second, each agent uses a set of concept hierarchies to organize the concepts. If a concept name is a word in the description vector of another concept, then we say that the concept is part of that another concept.

Third, for the purpose of query matching and ontological learning, the conceptual learning includes deriving semantic rules that help discriminate one concept from another. To construct rules, we feed the vector field into an inductive learner. The learner parses the vectors into a decision tree that deterministically allocates each example into a semantically unique branch. Branches will then be traversed—attribute values extracted and comparatives introduced—to arrive at rules. An example rule is:

```
If(university > 0.20) and
 (nebraska > 0.35) and (lincoln > 0.5)
and(omaha < 0.1) and then
 NU.
```

The above rule says that if frequency of the word "`university`" appears in the description vector is greater than 20%, and the frequency of the word "`nebraska`" is more greater than 35%, and the frequency of the word "`lincoln`" is greater than 50 percent, and the frequency of the word "`omaha`" is less than 10%, then the concept is "`NU`." This rule thus is used in query processing: if a query asks for experience cases under the concept "`NU`," the agent knows how to evaluate the relevance of the description vectors to the query.

So, conceptual learning gives us a description vector for each concept. Each description vector specifies a list of words (with their frequencies) for that particular concept. The collection of the description vectors is the agent's dictionary. Further, the agent builds a group of concept hierarchies to link the concepts together. This allows matching to find part-of concepts. Moreover, the agent derives semantic rules to distinguish each concept. With these, the agent is able to tell whether a description vector has anything to do with a particular concept. This is critical to our ontological learning. We assume that the concept names may differ among agents or users while the words describing these concepts are of the same vocabulary, of common examples.

2.2.2. Translation

An agent sends out query when it needs to obtain some experience cases or a description vector for a particular concept. When the queried agent processes the query, it first matches the concept to its translation table. This is translation.

If a credible translation is found, then the queried agent simply sends back its experience cases or a description vector (depending on what the querying agent has asked for) associated with the concept translated.

Each agent maintains a comprehensive translation table. Each table lists the concepts that the agent knows and maps them to the corresponding concepts of the neighboring agents. Only translations that are credible will be recorded in the table.

In the beginning, each agent has an empty repository of translation tables. At birth, an agent learns from the users' submission (or queries). Then the agent learns about the relations it has with its neighbors through two functional occasions. First, when it queries another agent for certain information (experience cases or description vectors). Second, when it receives a query from another agent. When an agent queries another agent for certain information and if the queried agent responds positively with its own semantics, the querying agent will duly interpret it and update it in its translation table. When an agent receives a query from another agent, if it does not have a readily available and up-to-date translation, then the agent interprets the semantics that accompany the query. At the end of the interpretation, if the agent is able to recognize the semantics, it then reflects the learned mappings in the translation tables.

2.2.3. Interpretation

When a query fails to be matched through during translation, it is forwarded to the interpretation module. There are two steps here: clarification and matching.

Clarification. At this step, the queried agent first requests from the querying agent the semantic rule that characterizes the concept in question. The querying agent may not have a semantic rule for that concept and thus may be unable to satisfy the request. Then the queried agent may ask for a description vector supporting the concept in question. Once again, the querying agent may or may not have the vector for the request. If the queried agent does not get further clarification on the concept, the interpretation step stops. However, if the querying agent is able to provide a semantic rule, then the process moves to the matching step. If the querying agent is able to provide only a description vector, then the queried agent has to perform conceptual learning to understand the concept before moving to the next step. In the end, if a clarification is achieved, the queried agent will have a semantic rule characterizing the concept in question. Then, the interpretation module is ready to perform matching.

Note that a querying agent may not have a semantic rule for the concept in question when it simply relays that query from another agent. A querying agent may only have a description vector if the concept has not been incorporated into the ontology of the agent.

Matching. When matching the semantics of the query with the semantics of a resident rule, if the two sets are synonymic, then a mass of 1.0 will be added to the interpretation scores. When we say 'synonymic', we refer to the inclusiveness and exclusiveness when comparatives are involved. For example, if the querying semantic has a semantic component "`university > 0.20`" and the resident semantic has "`university > 0.15`", then the resident semantic component is said to be inclusive of the querying semantic component. A similar observation can be said about the less-than comparative. Hence, two semantics are synonymic when the resident components include the querying components. In the sense of rule-based systems, if a semantic is matched, then the resident rule is fired by asserting the concept name entailed by the resident rule with a mass of 1.0. The translation between the concepts that describe the synonymic semantics will then be recorded in the translation tables.

Our agent is also equipped to handle relevant matching since a synonymic matching is rare. Suppose we have a querying semantic "`(college > 0.13 nebraska > 0.1237 cornhusker > 0.10)`" and the resident semantic "`(cornhusker > 0.2000 university > 0.1200 nebraska > 0.1138)`". Now, the first semantic component is not matched. The second semantic component is matched. The third semantic component is partially matched: the semantic token is matched but the comparative requirement is not fulfilled. Suppose the number of resident semantic components is N_{sc}, the mass of the assertion of the above partially-matched rule is

$$Mass = \frac{\sum_{i=1}^{N_{sc}} matched(resident\ con\ comp_i, query\ con)}{N_{sc}}$$

If a resident semantic component is matched, then the function *matched* returns 1.0. If the semantic token of a resident semantic component is not found in the querying semantic, then the function *matched* returns 0.0. If the semantic token matches but its semantic frequency is excluded (from the range), then *matched* returns

$$1.0 - \left| resident\ con\ \ freq - query\ con\ \ freq \right|$$

3. DEMPSTER-SHAFER BELIEF SYSTEM

An agent performs two types of learning. It learns incrementally, refining its concepts whenever there is a new submission. It also learns collaboratively, refining its translation table whenever there is a query that prompts the agent to ask for help from its neighbors. The underlying problem is how to combine the various assertions made by the relevant matching that we discussed earlier in a consistent manner. Towards this end, we incorporate the Dempster-Shafer theory [1] for building a belief system that receives evidence and maintains global beliefs in its assertions consistently.

Suppose that all the concept names that an agent understands, stored in its ontologies, are of the frame of discernment or universe *U*. A proposition in favor of a concept name, Γ, is thus an assertion as previously described. Thus, the set of all propositions is $\mathrm{P}(U)$, the power set of *U*. Let $m : \mathrm{P}(U) \to [0,1]$ be a function—a basic probability assignment—satisfying conditions for a certainly false proposition, $m(\varnothing) = 0$, and for a certainly true proposition, $\sum_{\Gamma \subseteq U} m(\Gamma) = 1$. The belief function, $Bel : \mathrm{P}(U) \to [0,1]$, is defined in terms of the basic probability assignment *m*: $Bel(\Gamma) = \sum_{\alpha \subseteq \Gamma} m(\alpha)$. This tells us the degree of belief associated with the proposition Γ as the probability mass associated with Γ and its subsets. The plausibility of a proposition is further defined as $Pls(\Gamma) = 1 - Bel(\neg\Gamma)$. Hence a proposition is always bound by $[Bel, Pls]$ in terms of the confidence in its perceived truthfulness. To combine various pieces of evidence for building up beliefs in favor of various propositions, the Dempster's rule of combination is used. Suppose we are given two assignments (two pieces of evidence), m_1 and m_2, and we want to combine them into a single piece of evidence. Hence, we compute

$$[m_1 \oplus m_2](\Gamma) = \frac{\sum_{\alpha \cap \beta = \Gamma} m_1(\alpha) m_2(\beta)}{1 - \sum_{\alpha \cap \beta = \varnothing} m_1(\alpha) m_2(\beta)},$$

where $\Gamma \neq \varnothing$, and $[m_1 \oplus m_2](\varnothing) = 0$.

For example, suppose after matching the semantics to our rule base, we arrive at two assertions: `NU` with mass 0.7 and `Monet` with mass 0.2. Hence, m_1 corresponds to our belief:

`{NU}` 0.7
Θ 0.3

and m_2 corresponds to our belief:

`{Monet}` 0.2
Θ 0.8

Then we can compute their combination m_3 using the rule of combination above (presented as a table below):

	{NU}	**0.7**	Θ	**0.5**
{Monet} 0.2	{ }	0.14	{Monet}	0.10
Θ **0.8**	{NU}	0.56	Θ	0.40

3.1. Concept Disambiguation

Until now, after the rule-based, semantics-driven assertions and the evidential combination, we arrive at a set of evidential intervals for the propositions or concept names. For our disambigua-

tion process, we follow the two axioms of evidential interval analysis:

Axiom 1 *The higher the belief and the plausibility values, the more credible the proposition is.*

Axiom 2 *The closer the belief value is to the plausibility value, the more credible the proposition is.*

Axiom 1 follows naturally from the work of the Dempster-Shafer theory. On the other hand, Axiom 2 punishes ignorance. That is, if the agent thinks that a proposition is very plausible but believes with little confidence that the proposition is true, then the agent is ignorant about the proposition. Following from the above two axioms, we devise a measure of credibility of a proposition as:

$$Credibility(\Gamma) = \frac{Pls(\Gamma) + Bel(\Gamma)}{Pls(\Gamma) - Bel(\Gamma)}.$$

During interpretation, the concept that yields the highest credibility will be the winning concept. Note that if the credibility of the winning concept is below a certain threshold, then the interpreter realizes that it does not understand or recognize the querying concept. This provision prevents low-quality recognition.

At the end of this stage, the interpreter performs one of the following: (1) If the winning concept passes the credibility test, then the agent supplies the querying agent with what it knows, i.e., the experience cases under the winning concept. The translation will also be recorded in the translation tables, or (2) If the winning concept fails the credibility test, then the agent turns to the translator module of its system.

3.2. Concept Amalgamation

This process is triggered by the combination of (1) the lack of a credible winning concept, and (2) the existence of a credible, relevant non-singleton concept structure. The objective is to promote non-singleton sets of concepts, such as {Basketball, NU}, to a recognizable concept structure. For example, suppose the set {Basketball, NU} has an evidential interval of [0.6 0.9]. Its credibility is 5.0. Suppose this passes our filter and the winning concept fails.

The amalgamation process will register the conceptual complex {Basketball, NU} to a complex-relevant table, together with the semantics that support the complex. Further, it will record the translation and its credibility to the corresponding space under the querying agent.

This provision self-motivates every agent to build and learn complex concepts, which in turns increases the complexity and level of understanding among the agents with distributed ontologies.

3.3. Belief System and Distributed Ontology Learning

Our agents conduct distributed ontology learning at different times. When an agent interacts with the user, it handles user submissions, performs inductive learning to obtain semantic rules, and builds its concept database. Since these submissions or experience cases are received sequentially, the learning is incremental. During the interaction with other agents in the environment, users can still submit both queries and new experience cases to an agent. When new experience cases are submitted, an agent revises its concept database through the belief system in the following manner. If the new experience cases are submitted with an existing concept name, then the agent essentially finds a translation between the concept name with new experience cases (evidence) and the concept name with old experience cases. The credibility of the translation is then used to re-weight mass of the existing semantic rules and the new rules derived from the new experience cases. This revision is then propagated to all related translations in the table. On the other hand, when a query is submitted and the agent fails to recognize the concept of that query, it may relay it to other agents. If another agent retrieves the relevant experience cases and returns them, then the originating agent learns that new concept using the retrieved experience cases as examples, absorbing it into its concept base via belief system in a manner similar to the aforementioned.

During interactions with other agents, an agent also learns to redirect tasks and re-formulate queries for better ontology understanding. The fundamental mechanism that enables such behavior is through the maintenance of the translation tables in the system. The presence of a unique translation table at each agent increases the autonomy of the agents in the system, allowing each to specialize for specific sets of queries and experience cases. Further, an agent is designed to relay a query that it cannot satisfy to other agents for help. An agent may not satisfy a query when it does not recognize the query concept, or does not find a relevant match, or does not retrieve experience cases with high relevance values, or does not find enough experience cases as required. As a result, the agent can use its translation table to locate useful neighbors to approach for help. If the concept query does not have a translation but the agent does have a few lowly-relevant experience cases, then it will supply those to its neighbors as examples. The key of our utilization of this type of distributed learning is that each agent has its own set of concepts to facilitate query accuracy and speed. Communications are only necessary when agents need help. Through communications, queries are relayed and concepts are shared. Thus, the agents only learn necessary translations based on their experiences, making the learning process efficient and effective.

In conclusion, the belief system allows the agents to evolve their own ontologies in the following ways: (1) infusion of new experience cases into the translation table via the concept hierarchies, (2) propagation of new credibility values to all translation entries, (3) joining terms to form complex concept names, and (4) exchange of concepts between agents that is based on the collaboration behavior between the agents (traffic congestion and agent activities).

4. REFERENCES

[1] Shafer, G. 1976. *A Mathematical Theory of Evidence*, Princeton, NJ: Princeton University Press.

[2] Williams, A. B. and Tsatsoulis, C. 1999. Diverse Web Ontologies: What Intelligent Agents Must Teach to Each Other, *Working Notes of the AAAI Spring Symposium Series on Intelligent Agents in Cyberspace*, Stanford, CA, Mar 22-24, 115-120.