Exploiting Domain Knowledge in Making Delegation Decisions *

Chukwuemeka David Emele[1], Timothy J. Norman[1],
Murat Şensoy[1], and Simon Parsons[2]

[1] University of Aberdeen, Aberdeen, AB24 3UE, UK
[2] Brooklyn College, City University of New York, 11210 NY, USA
{c.emele,t.j.norman,m.sensoy}@abdn.ac.uk
parsons@sci.brooklyn.cuny.edu

Abstract. In multi-agent systems, agents often depend on others to act on their behalf. However, delegation decisions are complicated in norm-governed environments, where agents' activities are regulated by policies. Especially when such policies are not public, learning these policies become critical to estimate the outcome of delegation decisions. In this paper, we propose the use of domain knowledge in aiding the learning of policies. Our approach combines ontological reasoning, machine learning and argumentation in a novel way for identifying, learning, and modeling policies. Using our approach, software agents can autonomously reason about the policies that others are operating with, and make informed decisions about to whom to delegate a task. In a set of experiments, we demonstrate the utility of this novel combination of techniques through empirical evaluation. Our evaluation shows that more accurate models of others' policies can be developed more rapidly using various forms of domain knowledge.

1 Introduction

In many settings, agents (whether human or artificial) engage in problem solving activities, which often require them to share resources, act on each others' behalf, communicate and coordinate individual acts, and so on. Such problem-solving activities may fail to achieve desired goals if the plan is not properly resourced and tasks delegated to appropriately competent agents. Irrespective of the field of endeavour, the overall success of problem-solving activities depends on a number of factors; one of which is the selection of appropriate candidates to delegate tasks to (or share resources with). However, successful delegation decisions depend on various factors. In norm-governed environments, one of the factors for making successful delegation decisions is the accuracy of the prediction about the policy restrictions that others operate with.

* This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

In multi-agent environments, agents may operate under policies, and some policies may prohibit an agent from providing a resource to another under certain circumstances. Such policies might regulate what resources may be released to a partner from some other organisation, under what conditions they may be used, and what information regarding their use is necessary to make a decision. In addition, policies may govern actions that can be performed either to pursue individual goals or on behalf of another. In Emele *et al.* [1], we show that intelligent agents can determine what policies others are operating within by mining the data gathered from past encounters with that agent (or similar agents) as they collaborate to solve problems. This prior research uses a novel combination of argumentation-derived evidence (ADE) and machine learning in building stable models of others' policies. Here we explore the question that given agents may have access to some background (or ontological) domain knowledge, how can we exploit such knowledge to improve models of others' policies? To do this, we propose the use of ontological reasoning, argumentation and machine learning to aid in making effective predictions about who to approach if some other collaborator is to be delegated to perform a task on the behalf of another.

The rest of this paper is organised as follows. Section 2 discusses delegation in norm-gorverned environments. Section 3 presents our approach for learning policies. Section 4 reports the results of our evaluations. Section 5 summarises our findings, discusses related work and outlines future directions.

2 Delegating in Norm-governed environments

Task delegation, in general, is concerned with identifying a suitable candidate (or candidates) to transfer authority to act on one's behalf [4]. In other words, we are concerned with finding candidate agents that will achieve a given goal on our behalf. Implicitly, this model is based on experience gathered from past encounters. In a norm-governed multi-agent system where each agent is regulated by a set of rules, referred to as *policies* (or norms), the policies could determine what actions an agent is (or is not) allowed to perform. Delegation, in this case, is not just a matter of finding agents that possess the appropriate expertise (or resource); if an agent has the required expertise but is operating under a policy that prohibits the performance of that action, it may not take on the delegated task. Nevertheless, delegation in such settings entails finding agents who possess the required expertise (or resources), and whose policies permit it to perform the required action (or provide the required resource). If an agent is allowed to perform an action (according to its policy) then we assume it will be willing to perform it when requested, provided it has the necessary resources and/or expertise, and that doing so does not yield a negative utility. In our framework, an agent that has been assigned a task is solely responsible for that task. However, an agent can delegate an aspect of a task to another. For example, agent x is responsible for performing task T_x but could delegate the provision of some resource R_r required to fulfill T_x to another agent y_1. Provided agent y_1 has resource R_r, and does not have any policy that forbids the provision of R_r then we assume y_1 will make R_r available to x.

From the above example, we see that agent x needs to find effective ways of delegating the provision of resource R_r. In order to delegate a task successfully, we need

to find out the agent whose policy constraints will most likely, according to a chosen metric, permit it to execute the delegated task. In our framework, whenever there is a task to be delegated, policy predictions are generated alongside the confidence of those predictions from the policy models that have been learned over time. Confidence values of favourable policy predictions are easily compared to determine which candidate to delegate the task to. In our case, confidence values range from 0 to 1, with 0 being *no confidence* in the prediction, and 1 being *complete confidence*.

The delegating agent explores the candidate space to identify suitable candidates to whom it can delegate a task. In these terms, our delegation problem is concerned with finding potential candidates whose policies permit it to perform the delegated task, and thereafter, selecting the most promising candidate from the pool of eligible candidates. Borrowing ideas from economics, we assume that some payment will be made to an agent for performing a delegated task (e.g. payment for the provision of a service).

Example 1 *Consider a situation where an agent x is collaborating with a number of agents, y_1, y_2, y_3, and y_4, to solve an emergency response problem. Let us assume that agent x does not have a helicopter in its resource pool, and that each of agents y_1, y_2, y_3, and y_4 can provide helicopters, jeeps, vans, bikes, fire extinguishers, and unmanned aerial vehicles (UAVs).*

Agent x in Example 1 has to decide which of the potential providers, y_1, y_2, y_3, and y_4 to approach to provide the *helicopter*. Let us assume that the four providers advertise similar services. Agent x, at this point, attempts to predict the policy of the providers with respect to task delegation (or resource provision). This prediction is based on policy models built from past experience with these providers (or similar agents). Assuming the predictions are as follows: (i) y_1 *will accept to provide the helicopter with 0.6 confidence*; (ii) y_2 *will accept with 0.9 confidence*; (iii) y_3 *will accept with 0.7 confidence*; and (iv) y_4 *will decline with 0.8 confidence*. If the decision to choose a provider is based on policy predictions alone, then y_2 is the best candidate.

3 Learning agent policies

The framework we propose here enables agents to negotiate and argue about task delegation, and use evidence derived from argumentation to build more accurate and stable models of others' policies. The architecture of our framework, sketched in Figure 1, enables agents to learn the policies and resource availabilities of others through evidence derived from argumentation, and improve those models by exploiting domain knowledge. The dialogue manager handles all communication with other agents. The learning mechanism uses machine learning techniques to reason over the dialogue and attempts to build models of other agents' policies and resource availabilities based on arguments exchanged during encounters. The arguments include the features that an agent requires in order to make a decision about accepting a task delegation or not. The agent attempts to predict the policies of others by reasoning over policy models (built from past experience). Such reasoning is further improved by exploiting background domain knowledge and concept hierarchies in an ontology (see Section 3.4).

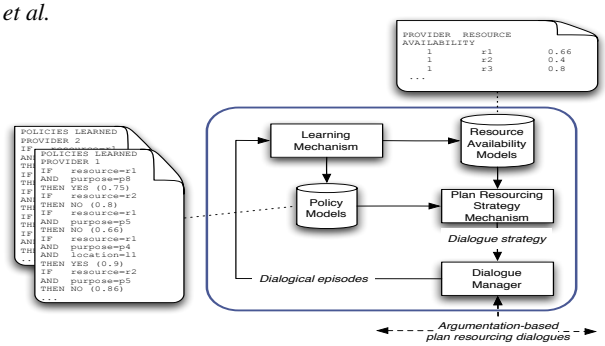

Fig. 1. Agent reasoning architecture.

3.1 Policies

In this framework, agents have policies that govern how resources are deployed to others. In our model, policies are conditional entities (or rules) and so are relevant to an agent under specific circumstances only. These circumstances are characterised by a set of features, e.g., vehicle type, weather conditions, etc.

Definition 1 (Features). *Let $\mathcal{F}$ be the set of all features such that $f_1, f_2, \ldots \in \mathcal{F}$. We define a feature as a characteristic of the prevailing circumstance under which an agent is operating (or carrying out an activity).*

Our concept of policy maps a set of features into an appropriate policy decision. In our framework, an agent can make one of two policy decisions at a time, namely (1) *grant*, which means that the policy allows the agent to provide the resource when requested; and (2) *deny*, which means that the policy prohibits the agent from providing the resource.

Definition 2 (Policies). *A policy is defined as a function $\Pi : \boldsymbol{\mathcal{F}} \rightarrow \{grant, deny\}$, which maps feature vectors of agents, $\boldsymbol{\mathcal{F}}$, to appropriate policy decisions.*

In order to illustrate the way policies may be captured in this model, we present an example. Let us assume that f_1 is resource, f_2 is purpose, f_3 is weather report (with respect to a location), f_4 is the affiliation of the agent, and f_5 is the day the resource is required, then $\mathbb{P}_1$, $\mathbb{P}_2$, and $\mathbb{P}_3$ in Figure 2 will be interpreted as follows: $\mathbb{P}_1$**:** You are **permitted** to release a *helicopter* (h), to an agent if the $helicopter$ is required for the purpose of transporting relief materials (trm); $\mathbb{P}_2$**:** You are **prohibited** from releasing an *aerial vehicle* (av) to an agent in bad weather conditions - e.g. volcanic clouds (vc); $\mathbb{P}_3$**:** You are **permitted** to release a *jeep* (j) to an agent.

Policy Id	f_1	f_2	f_3	f_4	f_5	Decision
$\mathbb{P}_1$	h	trm				grant
$\mathbb{P}_2$	av		vc			deny
$\mathbb{P}_3$	j					grant
$\mathbb{P}_4$	c		vc	xx		grant
...	...	...	...	...	...	...
$\mathbb{P}_n$	q	yy	w	xx	z	deny

Fig. 2: An agent's policy profile.

In the foregoing example, if *helicopter* is intended to be deployed in an area with volcanic clouds then the provider is forbidden from providing the resource but might offer a ground vehicle (e.g. *jeep*) to the seeker if there is no policy prohibiting this and the resource is available. Furthermore, whenever a seeker's request is refused, the seeker may challenge the decision, and seek justifications for the refusal. This additional

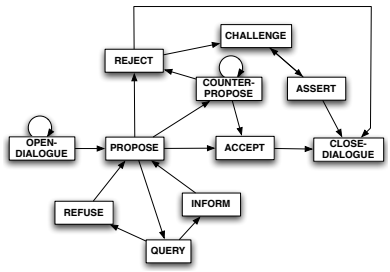

Fig. 3. The negotiation protocol.

evidence is beneficial, and could be used to improve the model, hence, the quality of decisions made in future episodes.

3.2 Argumentation-based Negotiation

Figure 3 illustrates the protocol employed in this framework, which guides dialogical moves. Our approach in this regard is similar to the dialogue for resource negotiation proposed by McBurney & Parsons [3]. To illustrate the sorts of interaction between agents, consider the example dialogue in Figure 4. Let x and y be seeker and provider agents respectively. Suppose we have an argumentation framework that allows agents to ask for and receive explanations (as in Figure 4, *lines* 11 *and* 12), offer alternatives (counter-propose in Figure 3), or ask and receive more information about the attributes of requests (*lines* 4 *to* 9), then x can gather additional information regarding the policy rules guiding y concerning provision of resources.

#	**Dialogue Sequence**	**Locution Type**
1	x: Start dialogue.	OPEN-DIALOGUE
2	y: Start dialogue.	OPEN-DIALOGUE
3	x: Can I have a *helicopter* for $0.1M reward?	PROPOSE
4	y: What do you need it for?	QUERY
5	x: To transport relief materials.	INFORM
6	y: To where?	QUERY
7	x: A refugee camp near Indonesia.	INFORM
8	y: Which date?	QUERY
9	x: On Friday 16/4/2010.	INFORM
10	y: No, I can't provide you with a *helicopter*.	REJECT
11	x: Why?	CHALLENGE
12	y: I am not permitted to release a *helicopter* in volcanic eruption.	ASSERT
13	x: There is no volcanic eruption near Indonesia.	CHALLENGE
14	y: I agree, but the ash cloud is spreading, and weather report advises that it is not safe to fly on that day.	ASSERT
15	x: Ok, thanks.	CLOSE-DIALOGUE

Fig. 4. Dialogue example.

Negotiation for resources takes place in a turn-taking fashion. The dialogue starts, and then agent x sends a request (propose in Figure 3) to agent y, e.g. line 3, Figure 4. The provider, y, may respond by conceding to the request (accept), rejecting it, offering an alternative resource (counter-propose), or asking for more information (query) such as in line 4 in Figure 4. If the provider agrees to provide the resource then the negotiation

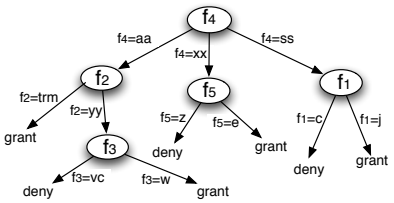

Fig. 5. Example decision tree.

ends. If, however, the provider rejects the proposal (line 10, Figure 4) then the seeker may challenge that decision (line 11), and so on. If the provider suggests an alternative then the seeker evaluates it to see whether it is acceptable or not. Furthermore, if the provider agent needs more information from the seeker in order to make a decision, the provider agent would ask questions that will reveal the features it requires to make a decision (query, inform/refuse). The negotiation ends when agreement is reached or all possibilities explored have been rejected.

3.3 Learning from past experience through dialogue

When an agent has a collection of experiences with other agents described by feature vectors (see Section 3.1), we can make use of existing machine learning techniques for learning associations between sets of discrete attributes (e.g. $f_1, f_2, \ldots f_n \in \mathcal{F}$) and policy decisions (i.e., *grant* and *deny*). In previous research we investigated three classes of machine learning algorithms: (i) instance-based learning (using k-nearest neighbours); (ii) rule-based learning (using sequential covering); and (iii) decision tree learning (using C4.5). Figure 5 shows an example decision tree representing a model of the policies of some other agent learned from interactions with that agent. Nodes of the decision tree capture features of an agent's policy, edges denote feature values, while the leaves are policy decisions.

The machine learning algorithms were chosen to explore the utility of different classes of learning techniques. Instance-based learning is useful in this context because it can adapt to and exploit evidence from dialogical episodes incrementally as they accrue. In contrast, decision trees, and rule learning are not incremental; the tree or the set of rules must be reassessed periodically as new evidence is acquired.[3] Learning mechanisms such as sequential covering, decision trees do have a number of advantages over instance-based approaches; in particular, the rules (or trees) learned are more amenable to scrutiny by a human decision maker.

The training examples used in each learning mechanism are derived from plan resourcing episodes (or interactions), which involves resourcing a task t using provider y and may result in $(\boldsymbol{F}_y, grant)$ or $(\boldsymbol{F}_y, deny)$. In this way, an agent may build a model of the relationship between observable features of agents and the policies they are operating under. Subsequently, when faced with resourcing a new task, the policy model can

[3] For these algorithms we define a *learning interval*, ϕ, which determines the number of plan resourcing episodes (or interactions) an agent must engage in before building (or re-building) its policy model.

be used to obtain a prediction of whether or not a particular provider has a policy that permits the provision of the resource. In this paper, we take this aspect of the research further by investigating *semantic-enriched decision trees* ($STree$), which extend C4.5 decision trees using ontological reasoning to explore how much domain knowledge can improve learning.

3.4 Learning from domain knowledge

In this paper, we argue that domain knowledge can be used to improve the performance of machine learning approaches. Specifically, in this section, we will describe how we can exploit domain knowledge to improve C4.5 decision trees.

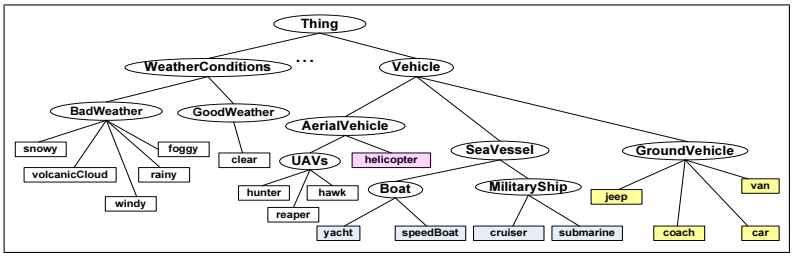

Fig. 6. A simple ontology for vehicles and weather conditions. Ellipsis and rectangles represent concepts and their instances respectively.

Domain Knowledge Domain knowledge consists of such background information that an expert (in a field) would deploy in reasoning about a specific situation. Semantic Web technologies allow software agents to use ontologies to capture domain knowledge, and employ ontological reasoning to reason about it [2]. Figure 6 shows a part of a simple ontology about vehicles and weather conditions. The hierarchical relationships between terms in an ontology can be used to make generalisation over the values of features while learning policies as demonstrated in Example 2. Policies are often specified using numerical features (e.g., vehicle price) and nominal features (e.g., vehicle type). Each nominal feature may have a large set of possible values. Without domain knowledge, the agent may require a large training set containing examples with these nominal values. However, domain knowledge allows agents to reason about to the terms unseen in the training set and learn more general policies with fewer number of training examples.

Example 2 *Suppose agent x in Example 1 has learned from previous interactions with agent y_1 that there is a policy that forbids y_1 from providing a helicopter when the weather is rainy, foggy, snowy, or windy. In addition, suppose agent x has learned from previous experience that agent y_1 is permitted to provide a jeep in these conditions. This information has little value for x if it needs a helicopter when the weather is not rainy, foggy, snowy, or windy but volcanic clouds are reported. On the other hand, with the help of the ontology in Figure 6, agent x can generalise over the already experienced weather conditions and expect that "agent y_1 is prohibited from providing helicopters in bad weather conditions". Such a generalisation allows x to reason about y_1's behavior for the cases that are not experienced yet. That is, with the help of the domain knowledge, agent x can deduce (even without having training examples involving*

volcanic clouds directly) that agent y_1 may be prohibited from providing a helicopter if there is an evidence of volcanic clouds in the region.

C4.5 Decision Tree Algorithm In this section, we shortly describe the induction of C4.5 decision trees. Then, in the following section, we describe how domain knowledge can be exploited during tree induction.

The well-known C4.5 decision tree algorithm [6] uses a method known as divide and conquer to construct a suitable tree from a training set S of cases. If all the cases in S belong to the same class C_i, the decision tree is a leaf labeled with C_i. Otherwise, let B be some test with outcomes $\{b_1, b_2, \ldots, b_n\}$ that produces a partition of S, and denote by S_i the set of cases in S that has outcome b_i of B. The decision tree rooted at B is shown in Figure 7, where T_i is the result of growing a sub-tree for the cases in S_i. The root node B is based on an attribute that best classifies S. This attribute is determined using information theory. That is, the attribute having the highest *information gain* is selected.

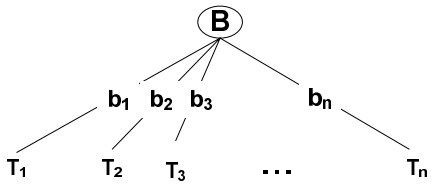

Fig. 7: Tree rooted at the test B and its breaches based on its outcomes.

Information gain of an attribute is computed based on *information content*. Assume that testing an attribute A in the root of the tree will partition S into disjoint subsets $\{S_1, S_2, \ldots, S_t\}$. Let $RF(C_i, S)$ denote the relative frequency of cases in S that belong to class C_i. The information content of S is then computed using Equation 1. The *information gain* for A is computed using Equation 2.

$$I(S) = -\sum_{i=1}^{n} RF(C_i, S) \times log(RF(C_i, S)) \tag{1}$$

$$G(S, A) = I(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} \times I(S_i) \tag{2}$$

Once the attribute representing the root node is selected based on its information gain, each value of the attribute leads to a branch of the node. These branches divide the training set used to create the node into disjoint sets $\{S_1, S_2, \ldots, S_t\}$. Then, we recursively create new nodes of the tree using these subsets. If S_i contains training examples only from the class C_i, we create a leaf node labeled with the class C_i; otherwise, we recursively build a child node by selecting another attribute based on S_i. This recursive process stops either when the tree perfectly classifies all training examples, or until no unused attribute remains.

Figure 8 lists 10 training examples, where $Type$, Age, and $Price$ are the only features. C4.5 decision tree algorithm makes induction only over numerical attribute val-

ues. However, it could not make induction or generalisation over the nominal attribute values (i.e., terms). For instance, a decision node based on the *price* test in Figure 9 can be used to classify a new case with price $\$250,000$, even though there is no case in the training examples with this price value. However, a new case with an unseen type, for instance a *submarine*, cannot be classified using the decision node based on the attribute $Type$.

#	Type	Age	Price	Class
1	Van	10	10,000	$grant$
2	Van	5	20,000	$grant$
3	Car	8	5,000	$grant$
4	Car	15	1,000	$grant$
5	Coach	2	200,000	$grant$
6	Yacht	20	300,000	$deny$
7	Yacht	2	500,000	$deny$
8	Speedboat	4	8,000	$deny$
9	Speedboat	15	2,000	$deny$
10	Cruiser	10	100,000	$deny$

Fig. 8: Training examples.

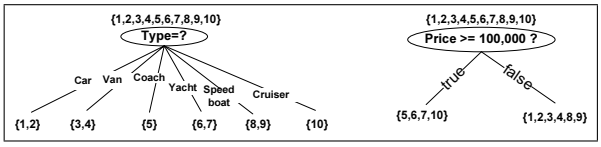

Fig. 9. Decision nodes created using the tests on $Type$ (on left) and $Price$ (on right).

Semantic-enriched decision trees Here, we propose semantic-enriched decision trees ($STree$) built upon the subsumptions relationships between terms in the ontology. These relationships can be derived automatically using an off-the-shelf ontology reasoner [2]. The main idea of $STree$ is to replace the values of nominal attributes with more general terms iteratively during tree induction, unless this replacement results in any decrease in the classification performance.

Algorithm 1 Generalising values of nominal attribute A in training set S.

1: **Input :** S, A
2: **Output:** T
3: $g = G(S, A)$
4: $banned = \{Thing\}$
5: $T = getAttributeValues(S, A)$
6: **while** $true$ **do**
7: **if** $\exists t$ **such that** $t \in T \wedge t \notin banned$ **then**
8: $t' = generalise(t)$
9: $T' = replaceWithMoreSpecificTerms(T, t')$
10: $s = replaceAttributeValues(S, A, T')$
11: **if** $G(s, A) = g$ **then**
12: $S = s$ **and** $T = T'$
13: **else**
14: $banned = banned \cup \{t\}$
15: **end if**
16: **else**
17: **break**
18: **end if**
19: **end while**

Algorithm 1 summarises how the values of A are generalised for S. First, we compute the original gain $G(S, A)$ (line 3). Second, we create a set called $banned$, which contains the terms that cannot be generalised further (line 4). Initially, this set contains only the top concept $Thing$. Third, we create the set T that contains A's values in S (line 5). While there is a generalisable term $t \in T$ (lines 6-18), we compute its generalisation t' using ontological reasoning (line 8) and create the set T' by replacing more specific terms in T with t' (line 9). If this term is an instance of a concept, then the

generalisation of the term is the concept, e.g., $Boat$ is generalisation of $Yacht$. If the term is a concept, its generalisation is its parent concept, e.g., $SeaVessel$ is generalisation of $Boat$. For instance, let S be the data in Figure 8, then T would contain $Yacht$, $Speedboat$, $Cruiser$, Van, Car, $Coach$, and $Cruiser$. If Car is selected as t, t' would be $GroundVehicle$. In this case, T' would contain $Yacht$, $Speedboat$, $Cruiser$, and $GroundVehicle$. Next, we check if the generalisation leads to any decrease in the information gain. This is done by creating a temporary training set s from S by replacing A's values in S with the more general terms in T' (line 10) and then comparing $G(s, A)$ with the original gain g (line 11). If there is no decrease in the information gain, S and T are replaced with s and T' respectively; otherwise t is added to $banned$. We iterate through until we cannot find any term in T to generalise without any decrease in the information gain.

The output of the algorithm would be $\{SeaVessel, GroundVehicle\}$ for the examples in Figure 8, because any further generalisation results in a decrease in information gain. Hence, a decision node based on $Type$ attribute would be as shown in Figure 10 (left hand side). A new test case (11, $Submarine$, $40years$, $\$800,000$) would be classified as $deny$ using this decision node, because a submarine is a sea vessel and all known sea vessels are labeled as $deny$. If the actual classification of the case is $grant$ instead of $deny$, the decision node would be updated as seen in Figure 10 (right hand side), because generalisation of $Submarine$ or $Cruiser$ now results in a decrease in the information gain.

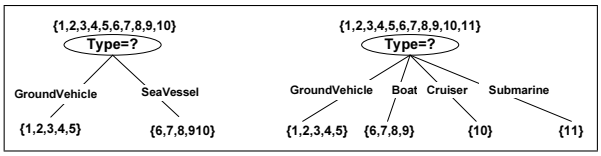

Fig. 10. Decision nodes using the generalisation of cases in Figure 8 (left hand side) and after the addition of a new case (11, $Submarine$, 40, 800, 000, $grant$) (right hand side).

4 Evaluation

In evaluating our approach, we employed a simulated agent society where a set of seeker agents interact with a set of provider agents with regard to resourcing their plans over a number of runs. Each provider is assigned a set of resources. Providers also operate under a set of policy constraints that determine under what circumstances they are permitted to provide resources to seekers. In the evaluation reported in this section, we demonstrate that it is possible to use domain knowledge to improve models of others' policies, hence increase their predictive accuracy, and performance. To do this, we consider two experimental conditions (i.e. closed and open). There are five features that are used to capture agents' policies, namely *resource type*, *affiliation*, *purpose*, *location*, and *day*. In the open scenario, each feature can have up to 20 different values, whereas only 5 different values are allowed in the closed scenario. In each scenario, six agent configurations (RS, SM, $C4.5$, kNN, SC, and $STree$) are investigated. In configuration RS, random selection is used. In SM, simple memorisation of outcomes is used. In $C4.5$, C4.5 decision tree classifier is used. In kNN, k-nearest neighbour algorithm

is used. In SC, sequential covering rule learning algorithm is used. Lastly, in $STree$, agents use semantic-enriched decision trees to learn policies of others.

Seeker agents were initialised with random models of the policies of providers. 100 runs were conducted in 10 rounds for each case, and tasks were randomly created during each run from the possible configurations. In the control condition (random selection, RS), the seeker randomly selects a provider to approach. In the SM configuration, the seeker simply memorises outcomes from past interactions. Since there is no generalisation in SM, the *confidence* (or prediction accuracy) is 1.0 if there is an exact match in memory, else the probability is 0.5.

Figure 11 gives a graphical illustration of the performance of six algorithms we considered in predicting agents' policies in the closed scenario. The results show that $STree$, SC, kNN, $C4.5$ and SM consistently outperform RS. Furthermore, STree, SC and kNN consistently outperform C4.5 and SM. It is interesting to see that, with relatively small training set, SM performed better than C4.5. This is, we believe, because the model built by C4.5 overfit the data. The decision tree was pruned after each set of 100 tasks and after 300 tasks the accuracy of the C4.5 model rose to about 83% to tie with SM and from then C4.5 performed better than SM. Similarly, STree performed much better than SC with relatively small training set. We believe, this is because STree takes advantage of domain knowledge and so can make informed inference (or guess) with respect to feature values that do not exist in the training set. After 400 tasks the accuracy of SC reached 96% to tie with STree. We believe, at this point, almost all the test instances have been encountered and so have been learned (and now exist in the training set for future episodes).

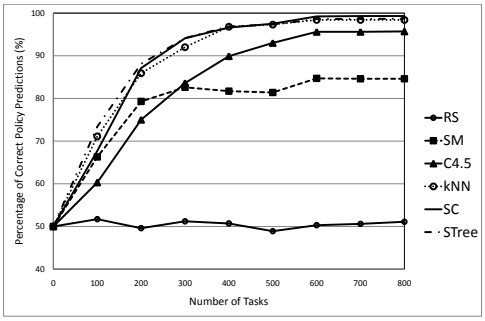

Fig. 11: The effectiveness of exploiting domain knowledge in learning policies (closed).

Figure 12 illustrates the effectiveness of four learning techniques (C4.5, kNN, SC, and STree) and SM in learning policies in the open scenario. The result shows that the technique that exploits domain knowledge (STree) significantly outperforms the other techniques that did not. The decision trees (i.e. STree and C4.5) were pruned after each set of 100 tasks and after 300 tasks the accuracy of the STree model had exceeded 82% while that of C4.5) was just over 63%. These results confirm that exploiting appropriate domain knowledge in learning policies mean that more accurate and stable models of others' policies can be derived more rapidly than without exploiting such knowledge.

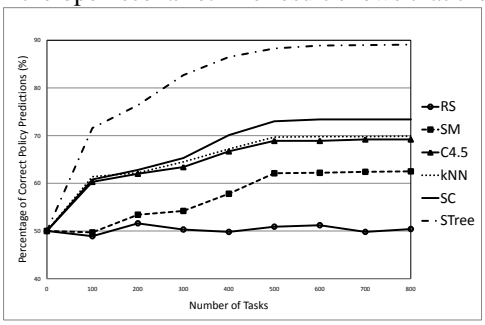

Fig. 12: The effectiveness of exploiting domain knowledge in learning policies (open).

5 Discussion

We have proposed an agent decision-making mechanism where models of other agents are refined through argumentation-derived evidence from past dialogues, and these models are used to guide future task delegation. Our evaluations show that accurate models of others' policies could be learned by exploiting domain knowledge. We believe that this research contributes both to the understanding of argumentation strategy for dialogue among autonomous agents, and to applications of these techniques in agent support for human decision-making. Sycara *et al.* [5] report on how software agents can effectively support human teams in complex collaborative planning activities. One area of support that was identified as important in this context is guidance in making policy-compliant decisions. This prior research focuses on giving guidance to humans regarding their own policies. Our work complements the approach of Sycara *et al.* by allowing agents to support humans in developing models of others' policies and using these in decision making. Our approach extends decision trees with ontological reasoning. Zhang and Honavar have also extended C4.5 decision trees with Attribute-value taxonomies [7]. Their approach is similar to $STree$, but it does not allow ontological reasoning during tree induction. Unlike their approach, our approach can directly incorporate existing domain ontologies and exploits these ontologies during policy learning.

In future, we plan to extend other matching learning methods with domain knowledge and explore how much this extension improves policy learning and enhances agents' support for human decision-making.

References

1. Emele, C.D., Norman, T.J., Parsons, S.: Argumentation strategies for plan resourcing. In: Proceedings of AAMAS 2011. p. to appear. Taipei, Taiwan (2011)
2. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
3. McBurney, P., Parsons, S.: Games that agents play: A formal framework for dialogues between autonomous agents. Journal of Logic, Language and Information 12(2), 315 – 334 (2002)
4. Norman, T.J., Reed, C.A.: Delegation and responsibility. In: Castelfranchi, C., Lesperance, Y. (eds.) Intelligent Agents VII, LNAI 1986, volume 1986 of Lecture Notes in Artificial Intelligence. pp. 136–149. Springer-Verlag (2001)
5. Sycara, K., Norman, T.J., Giampapa, J.A., Kollingbaum, M.J., Burnett, C., Masato, D., McCallum, M., Strub, M.H.: Agent support for policy-driven collaborative mission planning. The Computer Journal 53(1), 528–540 (2009)
6. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco, 2nd edn. (2005)
7. Zhang, J., Honavar, V.: Learning decision tree classifiers from attribute value taxonomies and partially specified data. In: Proceedings of the International Conference on Machine Learning (2003)