Bayesian Models for Keyhole Plan Recognition in an Adventure Game

DAVID W. ALBRECHT, INGRID ZUKERMAN and ANN E. NICHOLSON
School of Computer Science and Software Engineering, Monash University
Clayton, VICTORIA 3168, AUSTRALIA
{*dwa,ingrid,annn*}*@cs.monash.edu.au*

Abstract. We present an approach to keyhole plan recognition which uses a dynamic belief (Bayesian) network to represent features of the domain that are needed to identify users' plans and goals. The application domain is a Multi-User Dungeon adventure game with thousands of possible actions and locations. We propose several network structures which represent the relations in the domain to varying extents, and compare their predictive power for predicting a user's current goal, next action and next location. The conditional probability distributions for each network are learned during a training phase, which dynamically builds these probabilities from observations of user behaviour. This approach allows the use of incomplete, sparse and noisy data during both training and testing. We then apply simple abstraction and learning techniques in order to speed up the performance of the most promising dynamic belief networks without a significant change in the accuracy of goal predictions. Our experimental results in the application domain show a high degree of predictive accuracy. This indicates that dynamic belief networks in general show promise for predicting a variety of behaviours in domains which have similar features to those of our domain, while reduced models, obtained by means of learning and abstraction, show promise for efficient goal prediction in such domains.

Key words: Plan recognition, Bayesian Belief Networks, language learning, abstraction, performance evaluation.

1. Introduction

To date, research in plan recognition has focused on three main areas: (1) inferring plans during cooperative interactions, (2) understanding stories, and (3) recognizing the plans of an agent who is unaware that his or her plans are being inferred (Raskutti, 1993). In the first two areas, the plan recognition process is *intended*, since a user/writer is attempting to convey his or her plan to the system. In addition, during cooperative interactions, a plan recognition system can interrogate the user when confronted with ambiguous or incomplete information, e.g., (Allen and Perrault, 1980; Litman and Allen, 1987; Raskutti and Zukerman, 1991). The third area is called *keyhole* plan recognition because the information available to the plan recognizer is gleaned from non-interactive and often incomplete observations of a user (as though one was looking into a room through a keyhole). In the past, the use of hand-crafted plan libraries in systems that perform plan recognition imposed heavy restrictions on the size of their application domain, and hence on their usefulness (Charniak, 1997; Charniak, 1993, Preface). However, recently several researchers

have applied machine learning techniques to the acquisition of information about planning in an effort to overcome this problem, e.g., (Lesh and Etzioni, 1995; Forbes et al., 1995) (Section 2).

The mechanism described in this paper is part of this trend. Our approach to keyhole plan recognition uses a Dynamic Belief Network (DBN) to represent features of the domain needed to identify users' plans and goals. Our current domain is the "Shattered Worlds" Multi-User Dungeon (MUD), an adventure game which resembles real world applications in its complexity and size (Section 3). The MUD is a text-based virtual reality game where players compete for limited resources in an attempt to achieve various goals. The MUD has over 4,700 locations and 20 different quests (goals); over 7,200 actions were performed by players. The objective of the plan recognition mechanism is to determine, as early as possible, which quest a player is attempting, and to predict which action a player will perform in the next move and which location a player will visit next. To achieve this, the system must first learn which actions and locations or sequences of actions and locations tend to lead to a particular quest, and which actions and locations normally follow each other. This information is obtained from previous instances of completed quests during a training phase and modeled by means of several DBNs (Section 4). During the testing phase, the different DBNs are used to predict a player's quest, next action and next location. To this effect, for each DBN, every time a player performs an action, the system updates the probability that the player is trying to achieve each of the quests, perform each of the actions and move to each of the locations. The empirical results obtained for each DBN are described in Section 5. Section 6 discusses the application of simple abstraction and learning techniques to speed up the performance of the most promising of the DBNs described in Section 5 without a significant change in the accuracy of goal predictions. These enhancements are performed along two dimensions: (1) learning significant actions in the domain; and (2) abstracting the locations visited by MUD players. Finally, Section 7 discusses the implications of the results presented in this paper, and Section 8 presents ideas for future work.

2. Related Work

In recent times there has been a shift from systems that rely heavily on hand coded domain knowledge for plan recognition towards systems that apply machine learning techniques to automatically acquire domain knowledge. This has allowed a shift in domain size, where later systems deal with hundreds of actions in realistic domains.

The systems described in (Cañamero et al., 1992) and (Wærn and Stenborg, 1995) rely on domain knowledge for keyhole plan recognition. Cañamero *et al.* use an abstraction/specialization plan hierarchy to perform plan recognition from noisy input representing sequences of observations of an evolving situation in traffic monitoring. In particular, their system aims to recognize a driver's plan,

the manner in which the plan is being executed and the possible motivation for a particular plan from observations regarding the speed of the car, its acceleration, the distance from the car in front, etc. Wærn and Stenborg use a hierarchy of actions in conjunction with "compiled" plans in order to anticipate a user's intentions in domains where users exhibit reactive rather than plan-based behaviour, e.g., news reading. They perform simple probabilistic calculations to match a user's actions in a particular time window to those in the domain plans.

The system described in (Bauer, 1996) uses a plan hierarchy to represent the actions in the domain, and it applies decision trees (Quinlan, 1983) in combination with the Dempster-Shafer theory of evidential reasoning to assess hypotheses regarding a user's plans in the context of a user's actions. In particular, the Dempster-Shafer theory takes into account the reliability of the data obtained so far in order to moderate the probability mass assigned to the hypotheses postulated by means of the decision trees. The Dempster-Shafer theory is also applied in (Carberry, 1990), where a threshold plausibility and different levels of belief are used to distinguish among competing hypotheses.

The plan recognition mechanism described in (Lesh and Etzioni, 1995) works on a graph which represents the relations between the actions and possible goals of the domain. The system iteratively applies pruning rules which remove from the graph goals that are not in any consistent plan. In later work, Lesh and Etzioni use plan and goal *biases* – assumptions about what types of plans and goals people have – to automatically construct a plan library from primitive actions and goal predicates (Lesh and Etzioni, 1996). There are three important differences between the operating assumptions of our plan recognition system and those of the system developed by Lesh and Etzioni (in addition to the fact that our system reasons under uncertainty): (1) they assume that any action performed by a user pertains to one of the goals in their plan library, while our mechanism admits extraneous actions; (2) they assume that the actions in a plan must be executed in a particular order, while we place no restrictions on the order of actions; and (3) they assume that every action is known to the system during plan recognition, while we admit previously unseen actions. In addition, at present, a user's goals in our system (MUD quests) correspond to single predicates, while Lesh and Etzioni's system admits conjunctive goals. The extension of our mechanism to such goals is the subject of future research (Section 8).

Belief (or Bayesian) networks (BNs) (Pearl, 1988) have become a popular representation for reasoning under uncertainty as they integrate a graphical representation of causal relationships with a sound Bayesian foundation. In particular, BNs have been applied in several areas of User Modeling, such as knowledge assessment, plan recognition and prediction of user responses (for an overview of these applications see (Jameson, 1996)).

Belief networks – a brief overview. BNs are directed acyclic graphs where nodes correspond to random variables. The relationship between any set of state variables can be specified by a joint probability distribution. The nodes in the network are

connected by directed arcs, which may be thought of as causal or influence links; a node is influenced by its parents. The connections also specify independence assumptions between nodes, which allow the joint probability distribution of all the state variables to be specified by exponentially fewer probability values than the full joint distribution. A *conditional probability distribution* (CPD) is associated with each node. The CPD gives the probability of each node value for all combinations of the values of its parent nodes. The probability distribution for a node with no predecessors is its prior distribution. Given these priors and the CPDs, we can compute posterior probability distributions for all the nodes in a network, which represent *beliefs* about the values of these nodes. The observation of specific values for nodes is called *evidence*. Beliefs are updated by re-computing the posterior probability distributions given the evidence. Belief propagation for singly-connected networks can be done efficiently using a message passing algorithm (Pearl, 1988). When networks are multiply-connected (i.e., when there is a loop in the underlying undirected graph), simple belief propagation is not possible; informally, this is because we can no longer be sure that evidence has not already been counted at a node having arrived via another route. In such cases, inference algorithms based on clustering, conditioning or stochastic simulation may be used (Pearl, 1988).

Belief networks have been used both in static and dynamic applications. In static applications the nodes and links in a BN do not change over time. Hence, in principle, hand-crafting BNs for these applications is possible. However, in order to increase the flexibility and extensibility of the resulting system, the automatic construction of BNs from other knowledge representations is preferred, e.g., (Charniak and Goldman, 1993; Conati et al., 1997). Charniak and Goldman (1993) use BNs for plan recognition in the framework of story understanding. They automatically generate a BN from a sequence of observations by applying rules which use plan knowledge to instantiate the network. The incorporation of prior probabilities into this network supports the selection of plausible explanations of observed actions. Similarly, Conati *et al.* (1997) apply the mechanism described in (Huber et al., 1994) to automatically construct a BN from the output of a rule-based physics problem solver that generates all the possible solutions to a given physics problem. This BN is then used to identify a student's problem-solving strategy and predict his or her next step.

Dynamic applications are characterized by a constantly changing world. In order to model this change, temporal reasoning must be incorporated into BNs (Dean and Wellman, 1991; Dagum et al., 1992; Nicholson and Brady, 1994). This is done by allowing a BN to grow over time, and representing the state of each domain variable at different times by a *series* of nodes. Typically, for these Dynamic Belief Networks (DBNs), the connections over time are Markovian, and a temporal 'window' is imposed to reduce the state space. Such DBNs provide a more compact representation than the equivalent Hidden Markov Model (Russell et al., 1995). Two applications of DBNs are described in (Forbes et al., 1995;

Pynadath and Wellman, 1995). Pynadath and Wellman (1995) use a DBN for plan recognition in traffic monitoring. Their DBN is composed of loosely connected sub-networks, where each sub-network captures an intermediate structure based on one of the following factors: the context in which a plan was generated, the mental state and planning process of the agent, and the consequences of the agent's actions in the world. Like Conati, they also apply the mechanism described in (Huber et al., 1994) to map planning actions to a DBN. Forbes *et al.* (1995) emphasize issues that pertain to sensor noise or failure, and to uncertainty about the behaviour of other vehicles and about the effects of drivers' actions. Finally, Russell *et al.* (1995) use a gradient-descent algorithm to learn the conditional probability tables for BNs and DBNs with hidden variables, i.e., variables whose values are not observable (surveys of research on learning belief networks appear in (Heckerman, 1995; Buntine, 1996)).

The mechanism described in this paper resembles most closely the system described in (Forbes et al., 1995), but there are several important differences: (1) we infer a user's longer term goals, i.e., quests, in addition to the locations and actions inferred by Forbes *et al.*; (2) our data was collected prior to the undertaking of this project, hence we have had no choice in the view of the world that we are modeling, rather than being allowed to select the observations we wish to make; (3) we observe the world only from the perspective of a single user (without knowing whether observed changes in the world are caused by the user's actions or by other agents' actions); and (4) we have no information regarding the quality of our observations, while they have information about sensor uncertainty and hence are able to model it.

3. The Domain

The domain of our implementation is the "Shattered Worlds" Multi-User Dungeon (MUD), which is a text-based virtual reality game where players compete for limited resources in an attempt to achieve various goals. As stated in Section 1, the MUD has over 4,700 locations and 20 different quests (goals); more than 7,200 actions were observed. The plan recognition problem is further exacerbated by the presence of spelling mistakes, typographical errors, snippets of conversations between players, newly defined commands and abbreviations of commands.*

The MUD also has reactive agents controlled by the system (non-player characters), and contains a number of items which may be acquired and used by characters in order to achieve some effect within the game. Despite the fact that the MUD is a game, only a minority of the players log-in to play. Many users log-in with other goals, such as socializing with other players, crashing the MUD, or engaging in

* There is a class of users (wizards) who can add new commands and new locations to the MUD at any time. In addition, users can define "aliases" which represent sequences of commands. The command definitions and aliases are handled in a pre-processing step of the DBN-training procedure, and will not be discussed further in this paper. Spelling mistakes, typographical errors and snippets of conversations are handled as described in Section 6.1.

socially aberrant behaviour. However, at this stage of our project, we are interested in recognizing only one type of goal, namely quests. Examples of the simplest quests in the MUD are the "Teddy-bear rescue", which involves locating and retrieving a teddy bear lost by a non-player character called Jane, and the "Wood chop", where a player must chop wood in the market place, after first acquiring an axe and eating food to obtain enough energy to carry out the wood-chopping task. More complex quests may involve solving non-trivial puzzles, interacting with various non-player characters, e.g., monsters, shopkeepers or mercenaries, or achieving a number of sub-goals, e.g., obtaining potions. Players usually know which quest or quests they wish to achieve, but they don't always know which actions are required to complete a quest. In addition, they often engage in activities that are not related to the completion of a specific quest, such as chatting with other players or fighting with MUD agents. As a result, players typically perform between 25 and 500 actions until they complete a quest, even though only a fraction of these actions may actually be required to achieve this quest.

Analysis of the MUD yields the following features.* (1) it is extremely difficult to obtain a perspicuous representation of the domain (for example in the MUD there is a vast number of actions whose effects and preconditions are not fully known); (2) there may be more than one way to achieve a goal; (3) some sequences of actions may lead to more than one eventual goal; (4) some actions leading to a goal may need to be performed in sequence, while other actions are order-independent; (5) users may interleave actions performed to achieve two or more goals or may perform actions that are unrelated to any domain goal (e.g., socializing) while trying to achieve a goal; (6) the states of the MUD are only partially observable: the only information available at present is a user's actions (obtained from the user's keyboard commands), a user's locations (obtained from the system), and a few system messages, e.g., notification that a quest was completed and that a user has entered the MUD (the first and last "actions" in Table I respectively); and (7) the outcome of a user's actions is uncertain, i.e., the performance of an action is not a sufficient condition for the achievement of the action's intended effect (e.g., due to the presence of other agents who affect the state of the system).

The MUD software collects the actions performed by each player and the quest instance each player completed. In the current implementation, each data point is composed of: (1) a time stamp, (2) the name of the player, (3) the number of the log-in session, (4) the location where the action was executed, and (5) the name of the action.** A DBN is then constructed on the basis of the collected data as described in Section 4. Table I illustrates some of the 62 actions performed by a player to achieve the Avatar quest (the number of the log-in session is not shown). Without domain knowledge, it is extremely difficult to determine by inspection

* Other domains which we intend to investigate, viz WWW and Unix, have most of these features.

** At present, the MUD software does not record keyboard commands regarding an agent's movements on the horizontal plane, i.e., North, South, East and West. In addition, only the first word of each command is considered during training and testing.

Table I. Sample data for the Avatar quest.

Action No.	Time	Player	Location	Action
1	773335156	spillage	room/city/inn	*ENTERS*
12	773335264	spillage	players/paladin/room/trading_post	buy
17	773335291	spillage	players/paladin/room/western_gate	bribe
28	773335343	spillage	players/paladin/room/abbey/guardhouse	kill
37	773335435	spillage	players/paladin/room/abbey/stores	search
40	773335451	spillage	players/paladin/room/shrine/Billy	worship
54	773335558	spillage	players/paladin/room/brooksmith	give
60	773335593	spillage	players/paladin/room/shrine/Dredd	avenger
62	773335596	spillage	players/paladin/room/abbey/chamber	*Avatar quest*

which of these actions (if any) are necessary to complete the quest, the order of the necessary actions, or whether an action had the intended outcome.

4. Dynamic Bayesian Networks

In the next four subsections, we develop DBN models for the MUD domain. To this effect, we perform the following actions: (1) identify the interesting domain variables which become the network nodes (Section 4.1); (2) consider dependencies between the domain variables and the manner in which the domain variables change over time (these dependencies correspond to several alternative network structures, which we investigate experimentally) (Section 4.2); (3) describe how the CPDs are constructed from the collected MUD data and how we handle actions, locations and quests which do not occur in the training data (Section 4.3); and (4) present the data processing algorithm and belief update equations (Section 4.4).

4.1. NETWORK NODES

Based on the data we have for the MUD domain, the domain variables, which are represented as nodes in the DBNs, are as follows:

Action (A): This variable represents the possible actions a player may perform in the MUD, which we take to be the first string of non-blank characters entered by a user, plus the special `other` action, which includes all previously unseen actions. The results given in Section 5 were obtained with a state space size, $|A|$, of 4,904 actions. This state space is reduced from the original space of 7,259 actions, since this research takes into account only those runs where a quest was completed. In Section 6.1 we consider the effect of taking into account only significant actions.

Location (L): This variable represents the possible locations of a player, plus the special `other` location, which includes all previously unseen locations. The results given in Section 5 were obtained with a state space size, $|L|$, of 3,369 locations. As for actions, this state space is reduced from the original space of 4,722 locations. In Section 6.2, we consider the effect of abstracting location information using the hierarchical structure of the locations in the MUD.

Quest (Q): This variable represents the 20 different quests a player may undertake, plus the `other` quest, which includes all quests not seen in the training data, and the `null` quest. The variable representing the previous quest achieved is set to `null` if the user has just started a session.

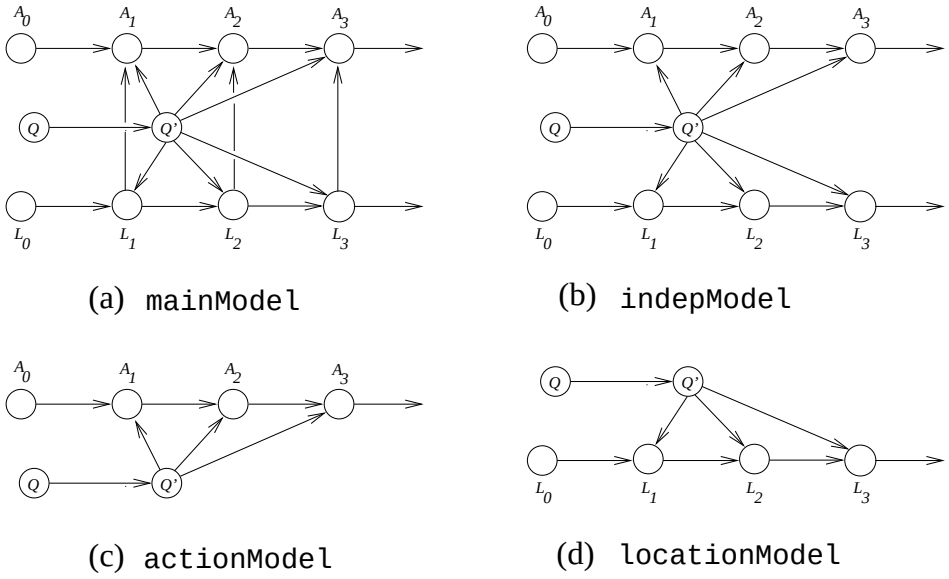
4.2. NETWORK STRUCTURE

Several DBN models involving these nodes were investigated (Figure 1). These models are not pure DBNs; the changes in actions and locations over time are represented, but we have made the simplifying assumption that a player's current quest, Q', does not change during a run and depends on the previous quest, Q (the relaxation of this assumption will be addressed in the future (Section 8)).*

Figure 1(a) shows the most complex of these models (called `mainModel`). This model stipulates that the location L_i at time step i depends on the current quest, Q', and the previous location at time step $i-1$, and that the action A_i depends on the previous action, the current quest and the current location. These dependencies are based on the following observations and assumptions. The dependence of a location on the previous location reflects the physical limitations of the domain, whereby most movement is to a topologically adjacent location (although teleporting is also possible). The dependence of an action on the previous action reflects the assumption that there is some correlation between pairs of actions; clearly, there may be longer sequences of actions which are connected, but including these dependencies in the model would defeat the Markovian assumption inherent in DBNs, which in turn would cause an explosion in the state space of the problem. The dependence of an action on a location reflects the observation that certain actions are mainly performed in certain locations, e.g., objects are usually bought at the market and food consumed at the inn.** The dependence of both location and action on the current quest reflects the assumption that most quests are completed in a particular subset of locations by undertaking particular actions.

* In practice, we always have evidence as to the previous quest, so we could fold node Q into the network, but it is clearer for expository purposes to maintain a separate node and the CPD for the $Q \rightarrow Q'$ link.

** Clearly, there are actions that change locations, hence in principle one may consider models where locations depend on actions. However, in BNs it is not possible to have both action–location and location–action links. Thus, when choosing one of these options, we preferred a model which features a dependence of actions on locations, because, as stated above, the MUD software keeps only partial records of actions which cause changes in locations.



(a) `mainModel` (b) `indepModel`

(c) `actionModel` (d) `locationModel`

Figure 1. Dynamic Belief Networks for the MUD: (a) `mainModel`; (b) `indepModel`; (c) `actionModel`; (d) `locationModel`.

The model in Figure 1(b) (called `indepModel`) relaxes the direct dependence between actions and locations, assuming that given the current quest, the current action and location are independent. Finally, the models in Figure 1(c) and 1(d) (called `actionModel` and `locationModel` respectively) are simpler still; they take into consideration either actions or locations in relation to quests (but not both).

4.3. PROBABILITIES

The CPDs are constructed from the collected MUD data as follows. The data is pre-processed to take the following form:

Previous Quest	Current Quest	Current Action	Current Location	Next Action	Next Location
`null`	`teddy`	`scream`	room/sewer/sewer20	`u`	room/city/alley1

A frequency count is maintained for each entry in the CPDs that was observed during training. These entries represent action, location and quest combinations that are relevant to the belief update formulas presented in Section 4.4 and were seen during training. In order to account for the possible actions, locations and quests that do not occur in the training data, we adjust the frequencies so that the resulting CPDs include some probability that the `other` value may occur. This adjustment consists of adding a small number that corresponds to Good's *flattening*

constant (Good, 1965) or Heckerman's *fractional updating* (Heckerman, 1995) to the non-zero frequencies. A factor of 0.5, which is computed by the Minimum Message Length theory (Wallace and Boulton, 1968; Wallace, 1990) assuming the prior is constant on seen events, was used for the results obtained in this paper (other flattening constants are briefly discussed in Section 5.4). The frequencies are then converted into CPDs.*

Thus, unseen actions, locations and quests are treated differently from unseen combinations of actions, locations and quests. That is, values of the domain variables that were not seen during training are put in the `other` category, while combinations of values of the domain variables that were not seen during training are assigned a probability of 0, i.e., they will not be predicted during testing. For example, if action a_i was seen in training, but was never performed at location l_j after action a_k, then during testing, action a_i will be predicted with probability 0 at location l_j if the previous action was a_k. This distinction was introduced mainly to make the belief updating process computationally feasible, especially for `mainModel`, which is the most computationally expensive among our models.** The introduction of the `other` category for unseen values of domain variables is essential for the operation of the system, since without it, if an action, location or quest unseen during training is reached during testing, our DBNs will assign a probability of 0 to all the values of next action, next location and current quest (since the unseen value is not represented in the CPDs). In contrast, as seen in the above example, if a particular combination of domain variables was unseen during training, then during testing a predicted domain variable will have a particular value, e.g., a_i, with 0 probability, but most other values will have non-zero probabilities, hence the DBNs can continue making predictions.

4.4. BELIEF UPDATING

Once a DBN is constructed, new data from a user is added to the network as evidence, and belief is updated regarding that user's next action and next location and the current quest being undertaken.

A *run* is a sequence of action-location pairs, beginning either after a player enters the MUD or after a player completes the previous quest, and ending when a new quest is achieved. The belief update algorithm applied for processing a run is given in Figure 2. If the run begins when a player enters the MUD, `PreviousQuest`, `PreviousAction` and `PreviousLocation` are set to `null`. In this case, a value of `null` is added as evidence for time-step 0 to nodes Q, A_0 and L_0. Otherwise (the run begins upon completion of a quest), the last quest completed, last action performed and last location visited are used. The evidence nodes for the domain at

* Since we are dealing with nodes with very large state spaces, we use hash tables of hash tables to store the CPD entries, and do not explicitly store zero probabilities.

** One of the reviewers suggested the investigation of canonical models of multi-causal interactions (Pearl, 1988) to address this problem. This investigation is left for future research.

1. Receive initial data:
   `PreviousQuest`, `PreviousAction`, `PreviousLocation`.
2. Add data as evidence for nodes Q, A_0 and L_0.
3. Update belief on nodes Q', A_1 and L_1.
4. Loop from n=1 until quest is achieved
   - 4.1 Receive new data: `Action`, `Location`.
   - 4.2 Add data as evidence for nodes A_n and L_n.
   - 4.3 Update belief on nodes Q', A_{n+1} and L_{n+1}.
   - 4.4 n=n + 1.

Figure 2. Belief update algorithm for processing a run.

time-step n + 1 are: the last completed quest, Q, the previous actions, $A_0, \ldots, A_n$, and the previous locations, $L_0, \ldots, L_n$.

There are underlying loops in the network structures shown in Figure 1, such as the loop between A_i, A_{i+1} and Q' in Figures 1(a), 1(b) and 1(c) and the loop between L_i, L_{i+1} and Q' in Figures 1(a), 1(b) and 1(d). This would seem to indicate that we must use an inference algorithm based on clustering, conditioning or stochastic simulation (Pearl, 1988). However, further analysis of these structures, together with the location of the evidence nodes, identifies d-separations (Pearl, 1988), indicating that certain nodes are conditionally independent (see Appendix A for details of this analysis). Using these independence relations, we obtain the following belief update equations for `mainModel` corresponding to Steps 3 and 4.3 in the belief update algorithm (the simplified update equations resulting from the analysis of the d-separations for `indepModel`, `actionModel` and `locationModel` appear in Appendix A; the actual analyses may be found in (Albrecht et al., 1997)).

Step 3 (time-step 0):

$$\Pr(Q'{=}q'|q,a_0,l_0) = \Pr(Q'{=}q'|q),$$

$$\Pr(L_1{=}l_1|q,a_0,l_0) = \sum_{q'} \Pr(L_1{=}l_1|l_0,q') \Pr(q'|q,a_0,l_0),$$

$$\Pr(A_1{=}a_1|q,a_0,l_0) = \sum_{q',l_1} \Pr(A_1{=}a_1|a_0,l_1,q') \Pr(l_1|l_0,q') \Pr(q'|q,a_0,l_0).$$

Step 4.3 (time-step n+1):

$$\Pr(Q'{=}q'|q,a_0,l_0,\ldots,a_{n+1},l_{n+1}) =$$
$$\alpha \Pr(l_{n+1}|l_n,q') \Pr(a_{n+1}|a_n,l_{n+1},q') \Pr(Q'{=}q'|q,a_0,l_0,\ldots,a_n,l_n),$$

$$\Pr(L_{n+1}{=}l_{n+1}|q,a_0,l_0,\ldots,a_n,l_n) =$$
$$\sum_{q'} \Pr(L_{n+1}{=}l_{n+1}|l_n,q') \Pr(q'|q,a_0,l_0,\ldots,a_n,l_n),$$

$$\Pr(A_{n+1}{=}a_{n+1}|q, a_0, l_0, \ldots, a_n, l_n) = \sum_{q', l_{n+1}} \Pr(A_{n+1}{=}a_{n+1}|a_n, l_{n+1}, q') \Pr(l_{n+1}|l_n, q') \Pr(q'|q, a_0, l_0, \ldots, a_n, l_n),$$

where α is a normalizing constant.

The update equations for time-step $n+1$ show that the new belief for the current quest (Q'), the next action (A_{n+1}) and the next location (L_{n+1}) can be computed from the previous beliefs in the values for the current quest and the CPD entries for the latest evidence received. The evidence before this, i.e., the evidence for action nodes $A_0, \ldots, A_{n-1}$ and location nodes $L_0, \ldots, L_{n-1}$, does not have to be considered explicitly, having been "folded" into the beliefs for A_n and L_n.

5. Experimental Results for DBN Models

In this section we present empirical results showing how the DBN models described in Section 4 predict current quest, next action and next location. We begin by showing the quest, action and location predictions for a single test run for a single character, and a selection of single runs showing typical quest predictions (Section 5.1). In Section 5.2, we present two quantitative methods which compare the different models in terms of their quest, action and location predictions: the average predicted probability, and the average of a scoring function based on the ranking of the actual quest, action and location. In Section 5.3 we consider the effect of varying the size of the training set on quest predictions, and in Section 5.4 we consider the effect of different flattening constants on quest predictions.

As stated in Section 3, in the current research we are interested in users' plans and goals while trying to achieve a quest. Thus, all the results presented in this section were obtained by choosing randomly a certain percentage of the 3,017 quest-achieving runs in our corpus as training data, and using the remaining runs as test data.* During each test run, we used the belief update algorithm shown in Figure 2.

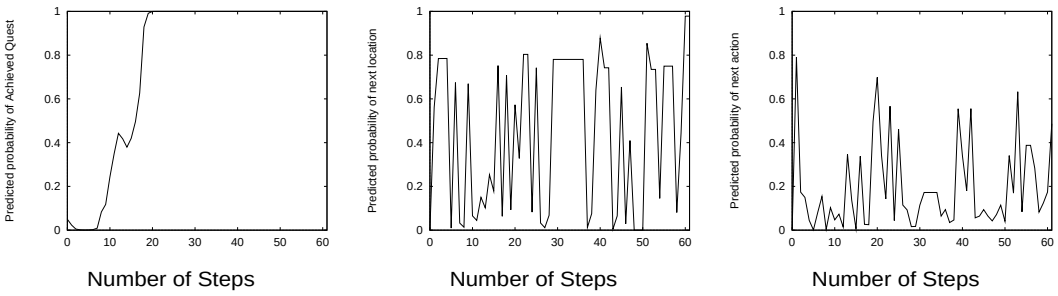
5.1. SINGLE RUNS

The output of `indepModel` (trained on 80% of the data) for the sample run where the character `spillage` achieves the `Avatar` quest (Table I) is shown in the graphs in Figure 3(a-c). The x-axes for these graphs show the number of time steps in the DBN, which correspond to the number of actions performed by the user. The y-axes show the current beliefs for the user's current quest (Q'), next location (L_{n+1}) and next action (A_{n+1}) respectively.**

Figure 3(a) shows that initially the system predicts a nearly zero probability that the `Avatar` quest is being attempted. This reflects the prior probability that the

* During the testing phase, a value which was not seen in the training data gets classified as `other`.

** Graphs of similar shape are obtained for individual runs with the other models.



(a) quest (b) location (c) action

Figure 3. Predictions for `spillage` (80% training with `indepModel`): (a) quest, (b) location, and (c) action.

`Avatar` quest follows the `null` quest; the CPD entry for $\Pr(Q'$=`Avatar`$|Q$=`null`$)$ is 0.04985. The predicted probability begins to rise after about 7 steps, reaching a value close to 1 around step 20, and remaining there until the quest is completed in step 62. The shape of this graph is typical of the more successful predictions performed for individual runs. Less successful runs take longer for the prediction to increase (Figure 4(a,d)), exhibit more fluctuations (Figure 4(b,c,f)), and a small percentage of the runs fail to predict the quest being attempted (Figure 4(e)). Such failures reflect the difficulties with quest prediction in this domain: a character may be performing actions that are unrelated to any quest (e.g., socializing), or for a while may be attempting a quest other than the quest that was actually completed.

The absolute probabilities of the actual next location and next action (Figure 3(b,c)) are not as high as those of the current quest. This is to be expected in light of the large number of possible actions and locations. Nonetheless, the probabilities of the actual location and actual action predicted by our models are generally among the highest predicted probabilities (see Section 5.2.1 for descriptions of two measures for the evaluation of the predictive ability of our models).

5.2. Model Comparison

In this section we compare the performance of the four models using two measures: *average prediction* and *average score*. These measures provide different views of the behaviour of our models (an intuitive interpretation of these measures is given in Section 5.2.4 together with the considerations for selecting the best model). Each measure is used to evaluate each model when trained on 80% of the data and tested on 20% with cross-validation using 20 different splits of the data (Sections 5.2.2 and 5.2.3).

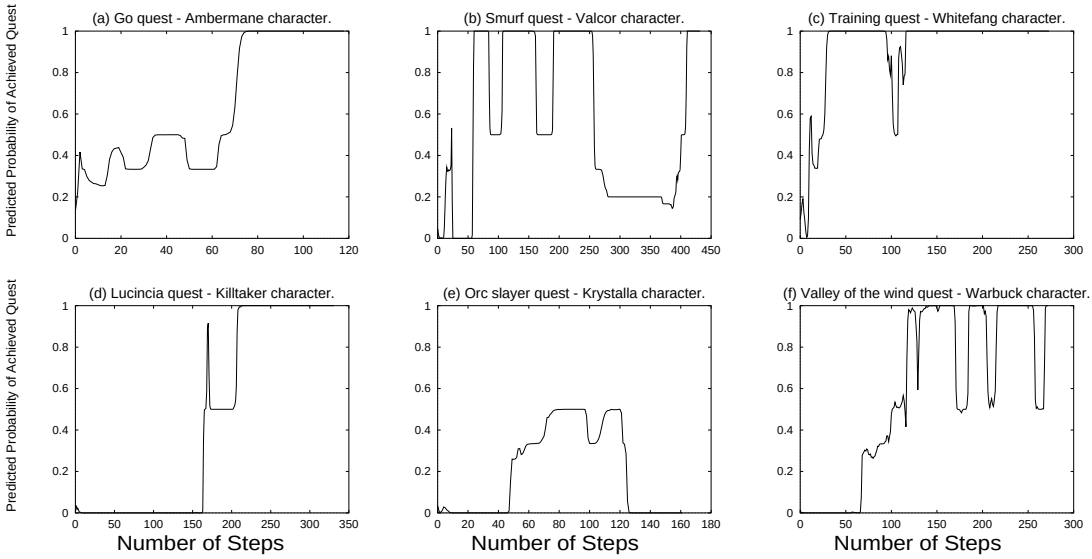


Figure 4. Typical quest prediction curves based on 80% training.

5.2.1. *Comparison measures: average prediction and average score*

– **Average prediction** is the average across all test runs of the predicted probability of a domain variable, i.e., the actual quest, next action or next location, at each point during the performance of a quest:

$$\textit{average prediction} = \frac{1}{n}\sum_{i=1}^{n} \Pr(\text{actual value of } \textit{variable} \text{ in the } i\text{-th test run}),$$

where *variable* may be either current quest, next action or next location, and n is the number of test runs performed.

– **Average score** consists of using the following function to compute the score of a prediction, and then computing the average of the scores at each point during the performance of a quest:

$$\textit{score} = \begin{cases} \frac{1}{|\text{top predicted values}|} & \text{if Pr(actual value of } \textit{variable}\text{)=Pr(top prediction)} \\ 0 & \text{otherwise} \end{cases}$$

$$\textit{average score} = \frac{1}{n}\sum_{i=1}^{n} \textit{score} \text{ in the } i\text{-th test run.}$$

The *score* function measures the percentage of times where the probability of the actual value of a domain variable is the highest, while taking into account the possibility that other values for this variable may have been assigned an

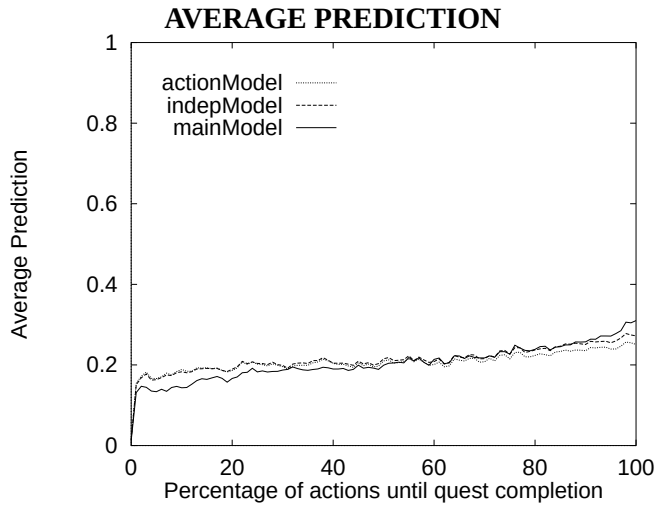
equally high probability. For example, this happens when the actual action is assigned a probability of 0.3, and there are two other actions with the same probability. In this case, a single action cannot be selected among these equiprobable ones. Thus, in order to measure accurately the predictive ability of a model, we must divide the score of 1, which indicates a successful prediction, by the number of equally successful candidates (3 in our example). An interesting situation presents itself when we have a prediction of `other`, and the actual action, location or quest is indeed `other`. We cannot count this as a correct prediction, since in this case, a model that is untrained and constantly predicts `other` will always be correct. However, we also cannot say that this is an incorrect prediction, since if we only have a few variable values that were unseen during training, we know that a prediction of `other` must be one of these values. Thus, a top prediction of `other` that matches an actual occurrence of `other`, i.e., an action that was not seen during training, is divided by the number of unseen actions prior to applying the scoring function. For example, if during testing we find k actions that were not seen during training, and `other` is predicted with probability p, we assign to the prediction a probability of $\frac{p}{k}$, and then apply the scoring function. If this modified probability is still the highest, then our prediction receives a positive score (equal to the reciprocal of the number of actions predicted with probability p/k). Otherwise, our prediction is wrong, and it receives a score of 0.

For both measures, in order to compare across runs where the number of recorded actions varies, we use the percentage of actions taken to complete a quest. That is, we apply our measures of performance at 0.1%, 0.2%, 0.3%, . . ., 1%, 1.1%, . . ., 2%, 2.1%, . . ., 100% of quest completion. These percentages are computed so that there is at least one data point for the quest with the largest number of actions; the quests with only a few actions will have a single action that corresponds to several data points.

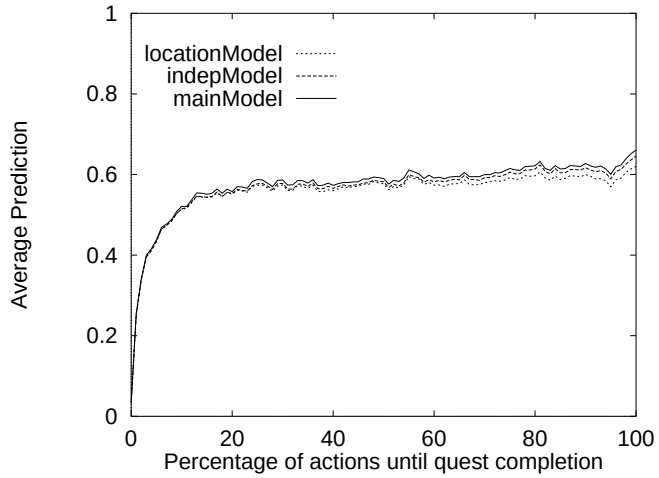
We believe that these measures are more informative regarding the performance of a plan recognizer than a measure such as that used in (Lesh, 1997), which gives the average percentage of quest completion when the following conditions are satisfied: (1) the top-predicted quest is the current quest, and (2) the probability of this prediction reaches some probability threshold. Lesh's measure requires the pre-selection of thresholds, which may vary between different domains. Further, it assumes that once a threshold is reached, the plan recognizer will not change its mind (as seen in Figure 4 this is not so in our domain). Finally, this measure is not applicable to the prediction of actions and locations, since it implicitly assumes that there will be a single top-predicted event only.

5.2.2. *Average prediction results*

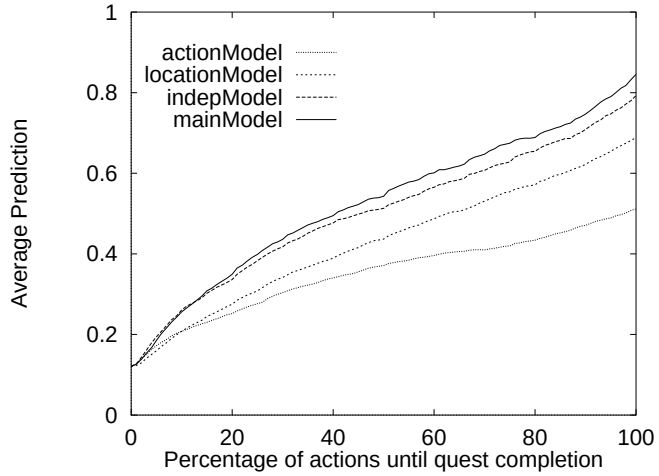
Figure 5(a-c) shows the average predictions for actual actions, locations and quests for the four models. Figure 5(a) shows that given the previous quest, current action



(a) Action prediction



(b) Location prediction



(c) Quest prediction

Figure 5. Performance comparison of models. Average prediction for (a) actions, (b) locations, and (c) quests.

and current location, we can predict the next action with an average probability around 0.2. Thus the average odds for predicting the next action, which are computed using the following formula:

$$\frac{P(\text{actual next action}|\text{evidence})}{P(\neg\text{actual next action}|\text{evidence})},$$

have improved from about $1 : 4,900$ to $1 : 4$.* Figure 5(a) also shows that the average predictions of `indepModel` are virtually indistinguishable from those of `actionModel` until about 46% of the actions have been completed, and that both of these models perform better than `mainModel` until 52% of the actions have been done. The null hypothesis that there is no significant difference between the models' predictions was tested using a T-test with 38 d.f. for each set of predictions and each pair of models.** For action predictions, the T-test confirmed our observations at the 0.5% significance level. In addition, when between 52% and 60% of the actions have been performed, there is no significant difference between the average predictions of `actionModel` and `mainModel` (at the 5% significance level), while `indepModel` gives better predictions than both of these models (at the 0.5% significance level). After 60% of the actions have been performed, `actionModel` gives worse predictions than the other two models at the 0.5% significance level. Finally, after 92% of the actions for any quest have been performed, the average action predictions of `mainModel` are higher than those of the other models at the 0.5% significance level.

In Figure 5(b) we see that given the previous quest and current location, we can predict the next location with an average probability of about 0.6 (so the average odds of predicting the next location have improved from about $1 : 3,370$ to $3 : 2$). The best predictions are produced by `mainModel`, with the predictions produced by `indepModel` being marginally lower, and those produced by `locationModel` being slightly lower again. The T-tests show at the 0.5% significance level that after 50% of the actions have been performed, the average location predictions obtained by `mainModel` are better than those of the other models.

Figure 5(c) shows that given the previous quest, current action and current location, the average prediction for the current quest rises steadily from 0.12 to about 0.83 for `mainModel`, which gives the best average predictions. Thus, for `mainModel` the average odds of predicting the next quest have improved from $1 : 19$ to $3 : 22$ at the start of a quest, and to nearly $5 : 1$ near the end of a quest. The average predictions obtained with `indepModel` are slightly lower, while those obtained with `actionModel` and `locationModel` are significantly

* The initial odds for next action, next location and current quest predictions are based on a naive calculation involving the number of possible actions, locations and quests respectively.

** We have 38 degrees of freedom because each of the two averages and standard deviations are computed from 20 splits; one degree of freedom is deducted because we use the sample standard deviation when calculating the difference between the two models, yielding $2\times(20\text{ splits} - 1\text{ standard deviation}) = 38$.

lower. The T-tests show at the 0.5% significance level that the average predictions of `mainModel` are significantly better than those of the other models after only 20% of the actions have been performed.

5.2.3. *Average score results*

Figure 6(a-c) shows the average score for actual actions, locations and quests for the four models. Figure 6(a) shows that given the previous quest, current action and current location, the average score obtained by `indepModel` and `actionModel` for the next action is around 0.35. Figure 6(a) also shows that given the previous quest, current action and current location, the average action scores obtained with `indepModel` and `actionModel` are virtually indistinguishable for the first 72% of a run; between 72% and 92% of a run, `indepModel` achieved better average scores than `actionModel`. In addition, for the first 93% of a run, `indepModel` and `actionModel` obtained better average scores than `mainModel`, while after 95% of a run, `mainModel` obtained better average scores than these two models. As for average predictions, the null hypothesis that there is no significant difference between the average scores obtained by the models was tested using a T-test with 38 d.f. for each set of predictions and each pair of models. For action predictions, the T-tests confirmed all these observations at the 0.5% significance level.

In Figure 6(b) we see that given the previous quest and current location, we can predict the next location with an average score of about 0.7. The average scores for the next location obtained by all the relevant models, viz `mainModel`, `indepModel` and `locationModel`, are virtually indistinguishable at the 5% significance level throughout most of a run. An exception occurs for the actions performed between 68% and 74% of a run, where the average scores obtained by `mainModel` are better than those obtained by `indepModel` at the 5% significance level.

Figure 6(c) shows that given the previous quest, current action and current location, the average score for the current quest rises steadily from about 0.22 to about 0.88 both for `indepModel` and `mainModel`. The average score obtained with `locationModel` is slightly worse than that obtained with the two best models, unlike the average score obtained with `actionModel`, which is significantly worse. The T-tests confirm these observations, showing that after 44% of a run, `mainModel` achieves better average scores than `indepModel` at the 0.5% significance level; after 88% of a run, `indepModel` achieves better average scores than `locationModel` at the 0.5% significance level; and after 7% of a run, `locationModel` obtains better scores than `actionModel` at the 0.5% significance level.

5.2.4. *Selecting the best model*

The selection of a model is based on the quality of its predictions of the variables of interest, that is, current quest, next action and next location. There are two ways

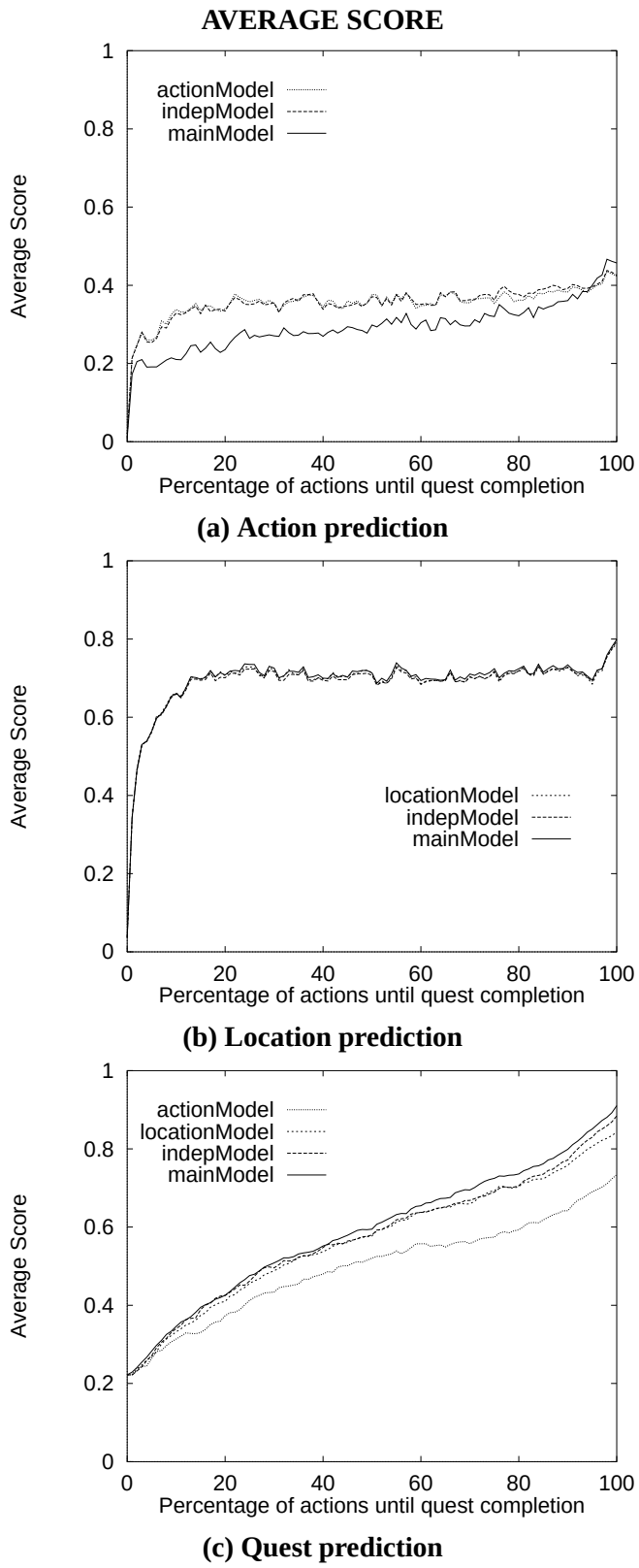


Figure 6. Performance comparison of models. Average score for (a) actions, (b) locations, and (c) quests.

of making a prediction for a variable: (1) selecting a value from a distribution, and (2) choosing the value with the highest probability (taking into account the fact that sometimes this value will be randomly selected among several equiprobable values which have the highest probability). These two methods correspond to the average prediction and average score measures respectively.

The meaning of the average prediction measure is as follows. Given that a model has yielded an average prediction probability p for the actual value of a particular variable at a certain point in time, if we perform our prediction for this variable at that point in time by randomly selecting a value based on the calculated probability, then on average our prediction will be correct $100p\%$ of the time. For example, according to Figure 5(c), if we are asked to predict the current quest based on the average prediction obtained by `mainModel` after 80% of the actions have been performed, then our prediction will be correct on average about 69% of the time.

The average score measure may be interpreted as follows. Given that a model has yielded an average score s for the actual value of a particular variable at a certain point in time, if we perform our prediction for this variable at that point in time by selecting the value with the highest probability, then on average our prediction will be correct $100s\%$ of the time. For example, according to Figure 6(c), if we are asked to predict the current quest based on the score obtained by `mainModel` after 80% of the actions have been performed, then our prediction will be correct on average about 74% of the time. The average score generally yields prediction percentages that are higher on average than the average prediction, because a singleton value, e.g., actual quest = `Avatar`, with the highest probability is assigned a score of 1, regardless of the absolute value of this probability.

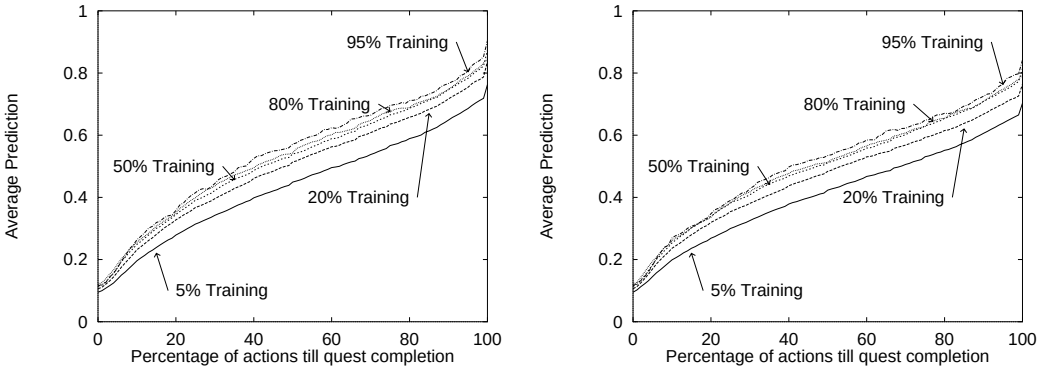
As seen from Figures 5 and 6, both measures of performance produce generally consistent assessments of the various models: `mainModel` is consistently worse at action predictions for most of a run, and consistently better at quest predictions, while `indepModel` performs well for all three predictions. The assessment produced by the average score accentuates some differences in performance compared to the assessment produced by the average prediction, while it blurs certain other differences. For example, for the first 50% of a run, the average action predictions of `mainModel` are quite close to those of `locationModel` and `indepModel` (Figure 5(a)), while the average action scores obtained by `mainModel` are much lower than the scores obtained by the other two models (Figure 6(a)). This is because with an average prediction of probability less than 0.2 for the actual action, such as that yielded by `mainModel`, it is quite possible that there are other actions predicted with the same probability or with a higher probability. In the first case, the score assigned to the actual action is reduced (it is divided by the number of equiprobable actions), and in the second case, it is 0. As seen in Figure 5(a), the three models yield relatively low average action predictions. However, since the average prediction obtained by `mainModel` for the actual action is the lowest, `mainModel` is more likely than the other models to assign to other actions the same probability or a higher probability than that assigned to the actual action.

The average score tends to blur the differences between `actionModel` and `indepModel` for action predictions, and the differences between `locationModel`, `indepModel` and `mainModel` for location predictions. This is because different models may have assigned different probabilities to the actual value of a particular variable. However, if all the models assign the highest probability to n values for this variable (including the actual value), then all the models will obtain the same score. In addition, for quests the average scores of `locationModel` are quite close to those of `mainModel` and `indepModel`, while the average quest predictions obtained by `locationModel` are much lower than the average predictions obtained by the other two models. This is because the probabilities assigned to the current quest by `locationModel` are lower than those assigned by these two models, but on average the current quest is still assigned the highest probability among its competitors.

Despite the relatively good performance of `locationModel` we have decided to retain only `indepModel` and `mainModel` for the remainder of our analysis. This is because `locationModel` cannot be used for action predictions, hence we must run two models in tandem: `actionModel` for action predictions and `locationModel` for location and quest predictions, which takes more computation time than running `indepModel` on its own, since each model must update the states of the other.

5.3. VARYING THE SIZE OF THE TRAINING SET

In this section we examine the effect of varying the size of the training set on the predictive power of the best two DBN models: `mainModel` and `indepModel`. Figure 7 shows the average quest predictions obtained after training with 5%, 20%, 50%, 80% and 95% of the data for `mainModel` and `indepModel`. These results were obtained with 20 different splits of the data. The null hypothesis that there is no significant difference between the average predictions obtained with each training set was tested using a T-test with 38 d.f.; the results are at the 0.5% significance level. As expected, training with 5% of the data produces the worst results for both models. As the size of the training set increases, the average predictions improve, with the best results being obtained when training is performed with 95% of the data. For `mainModel`, after 36% of a run these results are significantly better than those obtained when trained with 80% of the data, and after only 10% of a run they are better than the results obtained when trained with 50% of the data. Training with 80% of the data yields better average predictions than training with 50% of the data throughout a run. Similar results were obtained for `indepModel`, but the average predictions made after training with the different data sets became distinguishable from each other slightly later in a run.



(a) `mainModel` (b) `indepModel`

Figure 7. Average quest predictions with different training set sizes: (a) `mainModel`, and (b) `indepModel`

5.4. USING DIFFERENT FLATTENING CONSTANTS

The flattening constant is a small number which is added to frequencies to account for possible events which do not occur in training. This constant is the result of assuming a Dirichlet prior distribution or Jeffrey's non-informative prior distribution (Box and Tiao, 1973) when calculating the posterior distribution for the probabilities of events. As indicated in Section 4.3, due to computational efficiency considerations, in our implementation this constant is not added to zero frequencies corresponding to events which involve domain variables seen in training.

The three main flattening constants used in the literature (Good, 1965) are 1, 0.5, and $1/k$, where k is the number of possible values. The results presented thus far have used a flattening constant of 0.5, which is obtained by using Jeffrey's non-informative prior distribution for the probabilities, and is also implied by the Minimum Message Length criterion (Wallace and Boulton, 1968; Wallace, 1990).

Figure 8 shows the average quest predictions obtained with flattening constants 0.5, 1 and $1/k$ for `mainModel`, where k is the number of observed quests plus 1 (for the `other` quest). We selected `mainModel` for this analysis, since its CPD is the biggest among the CPDs of all the models. Hence, the results obtained with this model should highlight any potential benefits that may be obtained by using different flattening constants.

According to Figure 8, a flattening constant of 0.5 yields better average quest predictions than a flattening constant of 1 throughout a run, and better average predictions than $1/k$ for most of a run (between 5% and 85%). The average predictions obtained with a flattening constant of 1 are better than those obtained with $1/k$ during short portions of a run (between 13% and 36% and between 69% and 78%), but most of the time these flattening constants yield similar average predictions.

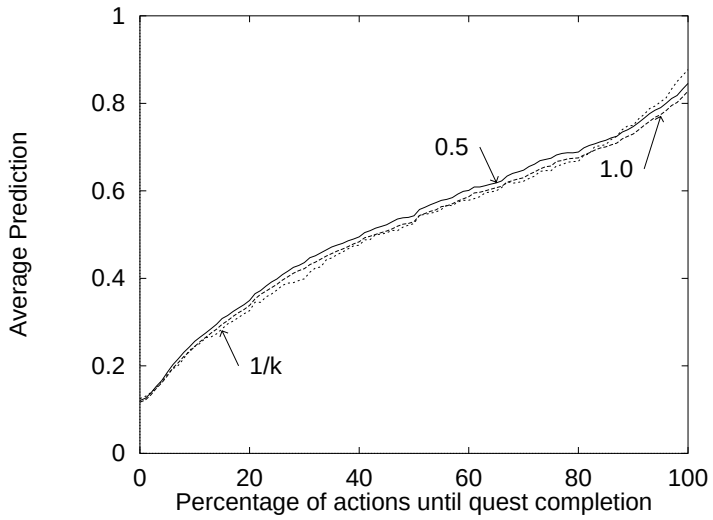


Figure 8. Average quest predictions for `mainModel` with a flattening constant of 0.5, 1 and $1/k$.

The average quest predictions obtained with $1/k$ rise over those obtained with a flattening constant of 1 after 86% of a run, and then over those obtained with 0.5 after 91% of a run. These results were obtained using 20 different splits of the data, each comprising a training set of 80% and a test set of 20%. The T-tests confirm these results at the 0.5 significance level.

6. Screening and Abstraction

We use simple models and update equations because the size of the domain compounds the complexity of the plan recognition problem. Another type of simplification involves reducing the size of the state space representation. We consider two approaches for this task: ignoring non-significant actions in the domain, and reducing the granularity of the location state space. In this section we describe these approaches, and present the experimental results obtained with them.

6.1. Screening Non-significant Actions

As indicated in Section 3, the plan recognition problem in the MUD is exacerbated by the presence of typographical errors and spelling mistakes, which increase the number of actions that must be dealt with during training and testing without actually having an impact on the states of the MUD. In order to overcome the difficulties caused by these extraneous actions we used a Minimum Message Length (MML) classifier (Wallace and Boulton, 1968; Wallace, 1990), which performs

unsupervised learning of the language that is understood by the MUD (details on how the classifier is used are given in (Albrecht et al., 1997)). We then considered only the actions in this language both during training and testing of our DBN models. The results obtained with this classifier were used in the two best models discussed in Section 4: `indepModel` and `mainModel`.

6.1.1. *Classification*

The classifier was run with all our data, and it was given the following attributes: (1) how many times each action was performed, and (2) how many players performed it. We found it appropriate to learn the language of the MUD using the entire data set (rather than using a portion of the data set as a training set) because of the way in which we treat unseen actions. That is, actions that are not seen during language training are simply ignored when making predictions during testing. Thus, if we learned the language using only a small percentage of the data, then we would obtain a reduced action set and location set (the number of locations would be reduced because we consider only those locations where an action that is in the language was performed). This would artificially increase our chances of making successful predictions simply because there are less actions and locations to choose from, unlike standard learning techniques where training with a reduced data set yields poorer performance during testing.

Our classifier identified five classes. Table II shows sample actions in these classes together with the percentage of the actions contained in each class, the ranges which contain most of the attribute values in each class (for each of the attributes), and the percentage of the elements in each class accounted for by these ranges.* For example, class C9 contains 65.6% of all the actions typed in by players; these actions were normally typed in only once or twice (99.16% of the actions in this class were typed in once or twice), and were executed by one player only (all the actions in this class were executed by a single player). As can be seen in Table II, classes C8 and C11 contain the most widely used actions that were typed in by the largest number of players. The actions in these classes are 'sensible' MUD domain actions and communication actions. In contrast, classes C9 and C6 contain infrequent actions used by a few players. These are typographical and spelling errors, personal aliases, infrequent numerical combinations, and words used in snippets of conversations. Class C10 contains a mixture of actions of the type found in C8 and C11 (but less frequently used), and actions of the type found in C6 and C9 (but more frequently used). Thus, the classifier has identified two classes of actions that are unlikely to be in the MUD language, namely C6 and C9, and two classes of actions that are very likely to be in the MUD language, namely C8 and C11. However, the situation is not so clear with respect to C10.

* The classifier we used generates numerical identifiers for its classes as it creates them. When classes are merged, they are not re-numbered. This results in non-consecutive numerical identifiers for the resulting classes.

Table II. Classes of actions in the MUD

Class	**% of total**	**# of times action was performed**		**# of players who performed action**		**Sample commands**
		Range	% of class	Range	% of class	
C9	65.6%	1-2	(99.16%)	1	(100.00%)	-guard ..will 1- 23e
C6	21.9%	2-7	(97.01%)	1-2	(100.00%)	101:tell 1move I've
C10	6.7%	7-54	(97.55%)	2-7	(98.70%)	abuse alis copy killl
C11	4.5%	20-1,096	(97.33%)	20-54	(95.20%)	break dance free pray
C8	1.3%	148-22,026	(98.94%)	54-403	(98.26%)	answer climb sell shout

In order to determine whether this screening process is useful in general, and whether C10 should be included in the MUD language, we trained and tested `indepModel` and `mainModel` with two candidate MUD languages: the language composed of the actions in C8 and C11 (called C8.11), and the language composed of the actions in C8, C10 and C11 (called C8.10.11). Language C8.11 reduces the action space from 4,904 actions to 415, while C8.10.11 reduces the action space to 926 actions. In order to obtain a preliminary indication of the validity of the learned languages, we checked how many consecutive non-significant actions (i.e., actions outside these languages) are typically performed. This test is based on the notion that if the ignored actions are mainly typographical and spelling errors, typically there should be short sequences of these actions, since a player would immediately correct an erroneous command with a correct (significant) command. Indeed, the average number of consecutive non-significant actions is 1.89 for C8.11 and 1.24 for C8.10.11, thereby supporting our hypothesis that the actions excluded from C8.11 and C8.10.11 are not significant.

6.1.2. *Screening results*

As stated above, DBN training and testing was performed with the actions in the language, rather than with the entire action set. Training was performed on 80% of the data, and testing on 20% with cross-validation over 20 splits of the data (while language learning was performed with the entire data set). The null hypothesis that there is no significant difference between the models' predictions was tested using a T-test with 38 d.f. for each set of predictions and each pair of models. All the results reported in the analysis below are at the 0.5% significance level.

During both training and testing, the non-significant actions are simply ignored, i.e., they are removed from the data set. This means that the DBN does not learn to predict the occurrence of non-significant actions during training, and that the

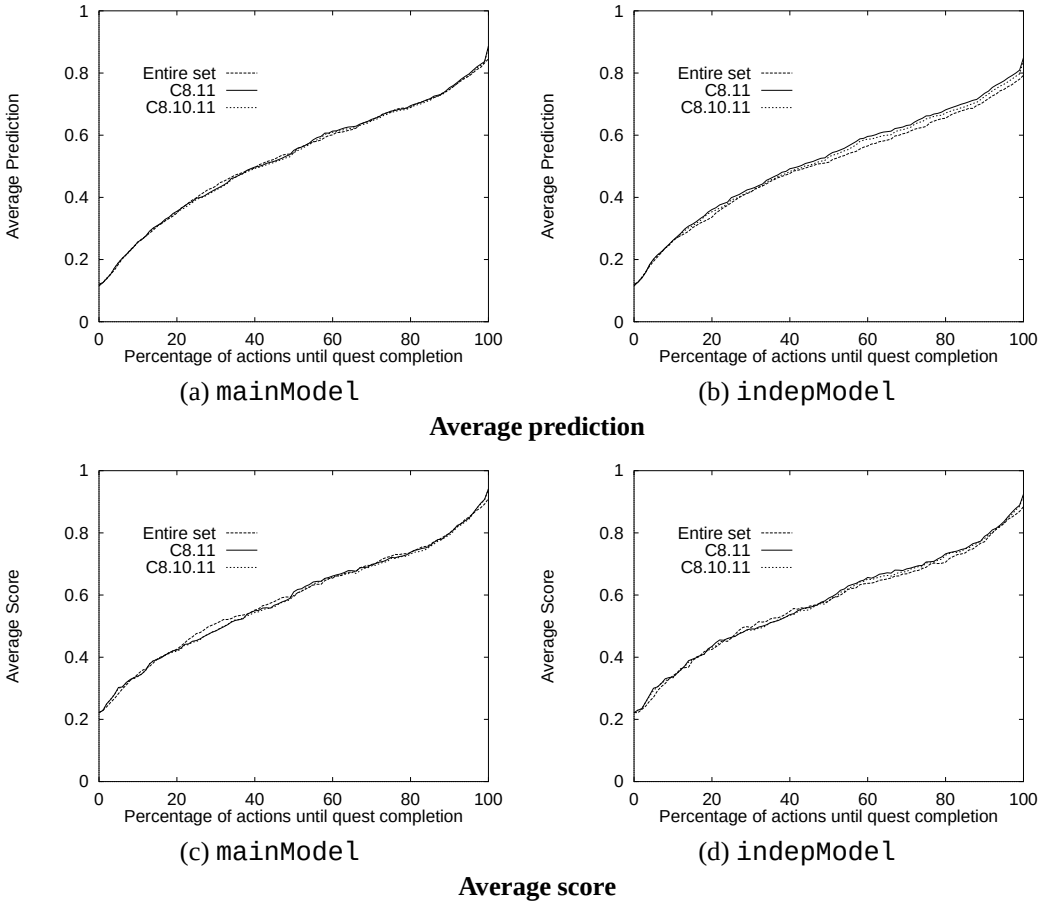


Figure 9. Average quest predictions and average quest scores for `mainModel` and `indepModel` when trained with C8.11, C8.10.11 and the entire action set.

performance of non-significant actions by a player does not affect the predictions made by the plan recognizer during testing. This causes difficulties when trying to compare the performance of DBNs which use different languages, such as C8.11, C8.10.11 and the entire action set, for action and location predictions. Hence, we compare the performance of these DBNs only for quest predictions. Figure 9(a-d) shows the performance of `mainModel` and `indepModel`, the two best models, when trained and tested on C8.11, C8.10.11 and the entire action set. Figure 9(a-b) shows the average predictions, and Figure 9(c-d) shows the average scores.

According to Figure 9, the performance of `mainModel` when trained with each of the training sets is slightly better than that of `indepModel`. `indepModel` obtained the best average predictions when trained and tested with C8.11. After 38% of a run the average predictions obtained with C8.11 are significantly better

than those obtained with C8.10.11, and after only 13% of a run the average predictions obtained with C8.11 are significantly better than those obtained with the entire data set. The average scores for `indepModel` are largely consistent with the average predictions, with the exception that the entire data set yields significantly higher average scores between 27% and 41% of a run. The average predictions obtained by `mainModel` when trained and tested with the entire data set, C8.11 and C8.10.11 are virtually indistinguishable from each other for most of a run. For small portions of a run the average predictions obtained with the entire data set are better (between 26% and 33% and between 41% and 49%) and for other portions they are worse (between 56% and 65%). As for `indepModel`, the average scores obtained by `mainModel` largely mirror the average predictions. These results indicate that training and testing with C8.11 had more impact on the results obtained with `indepModel` than on those obtained with `mainModel`. This can be explained by the observation that in `mainModel` the link between locations and actions lowers the probabilities of non-significant actions, which in turn reduces their contribution to quest predictions. Therefore, the removal of non-significant actions does not substantially change quest predictions. The absence of this link in `indepModel` (which results in higher probabilities for non-significant actions) means that non-significant actions interfere more with quest predictions. Hence, their removal has a higher impact on quest predictions.

Interestingly, the memory requirements of `mainModel` (which involve representing the CPDs) were reduced only by about 8% when trained and tested with the smallest language, C8.11. The reason for this relatively small reduction is that the CPDs for `mainModel` were initially very sparse (and zero probability events were not explicitly represented). Further, action screening introduces previously unseen action-action combinations, which must be represented in the CPDs. The reduction in memory requirements for `indepModel` was about 15%. It is worth noting that the reduction in memory requirements for both models was similar in absolute terms, indicating that similar information was removed from both models when trained with C8.11. However, in terms of percentages, the reduction is higher for `indepModel` since its CPDs are smaller than those of `mainModel`. In contrast, there was a considerable reduction in computation time during training and testing for both `mainModel` and `indepModel` (training and testing for `mainModel`, which has the highest computational requirements, went down from about one day to about half a day). The reductions obtained during training took place when constructing the CPD tables which involve actions and locations; the reductions obtained during testing took place when making quest predictions.

Due to the advantages of using C8.11 both in terms of quest prediction performance and in terms of computational requirements, we decided to use this language for the subsequent abstraction-based simplification process. Although in absolute terms `mainModel` performs slightly better than `indepModel` when using this language, both models were retained for the next stage of our analysis.

6.2. ABSTRACTION OF LOCATIONS

The abstraction of locations consists of identifying sets of related locations in the MUD. This can be done in two ways: (1) *Path abstraction* – involves abstracting a specific location to a larger location which includes it, e.g., keeping track of the building a player is visiting, rather than a particular room in that building; or (2) *Room abstraction* – abstracting a specific location to all locations of the same type, e.g., the reception at any inn in the MUD is considered a single type of room, namely "reception". These abstractions are implemented as follows:

Path – involves using the entire hierarchical description of a location in the MUD except for the last word, e.g., "players/paladin/room/abbey/guardhouse" in Table I is represented as "players/paladin/room/abbey". This reduces the size of the location state space, $|L|$, from 3,369 to 181.

Room – involves using only the last word in the hierarchical description of a location in the MUD, e.g., "players/paladin/room/abbey/guardhouse" in Table I is represented as "guardhouse". This leads to a reduction in the location state space size, $|L|$, from 3,369 to 3,079.

6.2.1. *Location abstraction results*

DBN training and testing was performed with the abstracted locations and the actions in language C8.11, rather than with the entire location set and action set. This means that during both training and testing, the specific locations are not taken into account, which in turn affects directly the location predictions performed, and indirectly the action predictions (since actions may depend on locations). Thus, as for action screening, we compare the performance of DBNs trained and tested on different location sets for quest predictions only.

Figure 10(a-d) shows the performance of `mainModel` and `indepModel` when trained and tested on the following data: entire location and action set; actions in C8.11 and entire location set (C8.11); actions in C8.11 and locations in the path abstraction (called C8.11-path); and actions in C8.11 and locations in the room abstraction (called C8.11-room). Figure 10(a-b) shows average quest predictions, and Figure 10(c-d) shows average quest scores. These results were obtained when the models were trained on 80% of the data and tested on 20% with cross-validation over 20 splits of the data. The null hypothesis that there is no significant difference between the models' predictions was tested using a T-test with 38 d.f. for each set of predictions and each pair of models. All the results reported in the analysis below are at the 0.5% significance level.

According to Figure 10, the performance of `mainModel` when trained with each of the training sets is slightly better than the performance of `indepModel`. The average predictions obtained when `indepModel` was trained and tested with C8.11 and with C8.11-room are virtually indistinguishable, and after about half a run they

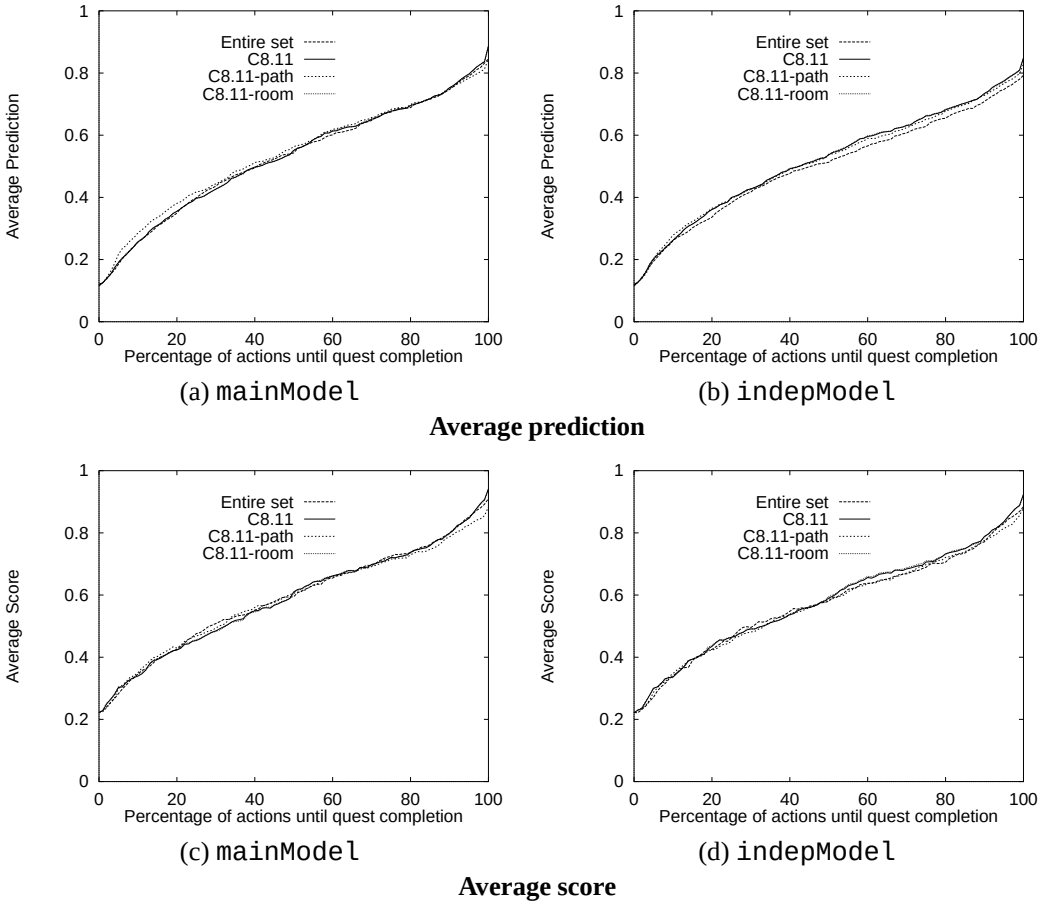


Figure 10. Average quest predictions and average quest scores for `mainModel` and `indepModel` when trained with C8.11, C8.11-path, C8.11-room and the entire action-location set.

rise to be the best among the average predictions obtained with all the training sets. Training and testing with C8.11-path yields slightly lower average predictions for the second half of a run. In addition, after only 12% of a run, the average predictions obtained when the entire data set was used for training and testing are worse than those obtained with the other three data sets. The average scores obtained when `indepModel` was trained and tested with C8.11 and with C8.11-room are also indistinguishable. However, unlike the average predictions, they are higher than the average scores obtained with the entire data set only after about half a run. Further, the average scores obtained with C8.11-path are relatively lower than the average predictions obtained with this data set, and drop off significantly below the average scores obtained with the other data sets after 93% of a run.

In contrast to `indepModel`, for most of a run (until 93% has been completed), `mainModel` yields the best average predictions when trained and tested on C8.11-path. For the first 53% of a run C8.11-path yields the highest average predictions. Between 70% and 93% of a run, the predictions obtained when trained and tested with each of the data sets become virtually indistinguishable, at which point the average predictions obtained with the entire data set and with C8.11-path drop off slightly. As for `indepModel`, the average predictions obtained with C8.11 and with C8.11-room are virtually indistinguishable throughout a run, and so are the average scores obtained with these data sets. In addition, the average scores obtained when `mainModel` was trained and tested with C8.11-path are relatively lower than the average predictions obtained with this data set, indicating a performance that is commensurate with that obtained with the other data sets for most of a run (but not better). Further, the relative drop in the average scores obtained with C8.11-path starts after only 75% of a run (compared to 93% for the average predictions).

Thus, for both `indepModel` and `mainModel` (and according to both average prediction and average score) the path abstraction performs as well as or better than the other data sets for about the first half of a run, at which point the relative performance of the path abstraction starts deteriorating, becoming significantly worse than the performance obtained with the other data sets towards the end of a run. This may be explained by the observation that initially information about the general whereabouts of a player may give a good indication of his or her intent. However, as quest completion draws near, more detailed information is required to make a precise prediction. In addition, training with the room abstraction (and the screened action set) yielded the same performance as training with the screened action set and all the rooms. This result is consistent with our expectation that players perform the same types of actions in rooms of the same type. Finally, the discrepancies in the performance assessments obtained with average prediction and average score suggest that further investigation is required into the distribution of scores and predictions over different runs.

The memory requirements for `mainModel` (which involve representing the CPDs) were further reduced (in addition to the 8% obtained from action screening) by about 44% when trained and tested with C8.11-path, and only by 1% for C8.11-room. Similarly, the reductions in memory size for `indepModel` were 39% and 1% for C8.11-path and C8.11-room respectively. The path abstraction yielded a considerable reduction in computation time during training and testing for both `mainModel` and `indepModel` (training and testing for `mainModel`, which has the highest computational requirements, went down from half a day, which was achieved by using C8.11, to about three hours). The reductions obtained during training took place when constructing the CPD tables which involve actions and locations; the reductions obtained during testing took place when making quest predictions. This indicates that a fruitful strategy for achieving reductions in computation time during plan recognition may consist of starting the plan recognition process with the models trained with the path abstraction, and switching to the

models trained with the more comprehensive data sets later in the plan recognition process. In contrast to the path abstraction, the room abstraction yielded no substantial gains in computation time due to the relatively small difference between the number of rooms in the entire location set and the number of rooms in the room abstraction. Nonetheless, if such small gains are necessary, our results indicate that the room abstraction can fully replace the entire location set without causing any change in predictive power.

7. Discussion

We have presented and compared four Dynamic Belief Networks which predict a user's next location, next action and current quest based on a training corpus. We do not learn the structure of the Bayesian networks. Instead, we have proposed four basic network structures that model different underlying dependencies in the domain. Simple models are required since the number of possible values for each node makes network training and evaluation a computationally complex task.

We have used two different measures to compare the different DBN models, viz average prediction and average score. Both measures compute averages across all the test runs, and both are used from the start of a run until quest completion. This type of calculation is required because of the nature of the MUD domain, where current quest, next action and next location predictions may fluctuate within the course of a single run. Using our performance measures we can show gradual prediction improvement as quest completion nears.

The comparison between the four presented models gives some insight into the underlying dependencies in the domain. The accuracy of quest predictions obtained when using both a user's locations and a user's actions is significantly higher than the accuracy of the predictions obtained when using actions alone, and somewhat higher than the accuracy obtained when using locations alone. As seen in Section 5.2.4, the average quest predictions produced by `locationModel` are significantly lower than those produced by `mainModel` and `indepModel`. However, the average scores obtained by these three models are rather close. This indicates that when the current quest is predicted by randomly choosing a quest with the highest probability, the performance of the three models will be quite close. Still, both performance measures clearly indicate that `mainModel` is the best of all the models for quest prediction. In addition, our results indicate that the system's belief regarding which quest a user is attempting affects both location and action predictions. Interestingly, `mainModel` performs the worst on action predictions for most of a run, as measured by both the average prediction and the average score. As indicated in Section 6.1.2, this may be due to the link from locations to actions in `mainModel`, which reduces the probability of non-significant actions (these actions form a large percentage of the observed actions). Another contributing factor may be the increased size of the CPD table used in `mainModel` and the resulting sparseness of that table; the lack of a sufficient number of data

points results in undue emphasis being placed upon a relatively small number of observations. There is very little difference between the location predictions of `locationModel`, `mainModel` and `indepModel`. Therefore, if the focus is on location predictions only, the simplest model, i.e., `locationModel`, should be used.

All the performance results for quest predictions presented in this paper are based on the assumption that at all stages until a quest is completed a player is intending to complete that quest. However, one of the features of our domain is that a player may undertake actions towards another quest that is completed in a later run, may execute actions that are not related to any quest, or may have abandoned an attempt at a quest that was intended for a while. In each case we have no way of knowing this has occurred. Thus, our quest prediction results may be considered an underestimate of the actual outcomes.

An important feature of our approach is that due to the probabilistic training, predictions are based on actions that are *normally* performed to achieve a goal, rather than on actions that necessarily advance a user towards the achievement of a goal. Thus, actions that are necessary to achieve a goal (and hence performed by a large number of users to satisfy a particular goal) have a relatively significant effect on predictions. On the other hand, actions that are performed across many goals and extraneous actions (i.e., those which do not contribute to the achievement of any goals, such as typographical errors), have little effect on the prediction of a particular goal.

In an extension to the basic approach, we have attempted to screen out these extraneous actions using an MML classifier. By learning the language of the MUD, we have substantially reduced the computation time required by our DBNs while gaining quest predictive power (the reductions in memory requirements were rather modest). We have shown that further computational reductions may be obtained by using an abstraction based on the hierarchical structure of the location variable. The simplest abstraction, room abstraction, obtained little reduction in both computation time and memory requirements, and its prediction results were virtually indistinguishable from those obtained without the abstraction. The coarser abstraction, i.e., path abstraction, achieved significant reductions in both computation time and memory requirements at the expense of a slight reduction in the predictive performance of `mainModel` towards the end of a run.

To summarize, if we wish to predict all the domain variables of interest, that is, current quest, next action and next location, the best models are `mainModel` or `indepModel`. However, if we are interested only in quest predictions, `mainModel` obtains both the highest average predictions and the highest average scores when trained on the actions in the MUD language C8.11. Training with the locations in the path abstraction leads to an improved performance during the early stages of a run, while training with the room abstraction yields the same performance as training with all the locations in the MUD (these training sets also reduce the computational requirements of `mainModel`).

8. Future Work

In this section, we discuss ideas for future research along several dimensions.

Model comparison

The model comparisons presented in this paper use measures which are averages across all the test runs. A more in-depth comparison of our DBNs may be obtained by looking at differences in their performance for individual predictions in individual runs. To this effect, we intend to extend the model comparison analysis to pairwise combinations of the predictions made by different DBNs.

Learning the MUD language

There are some caveats to our results showing how the MML classifier can be used to learn the language of the MUD. Firstly, we have compared only quest prediction performance across languages; in future research we intend to devise a suitable method for comparing action and location predictions across languages. In addition, a possible problem with our approach for learning the MUD language is that it may ignore actions that are performed only for quests that are very infrequently attempted, such as Arachnophobia, Demon and Mantis, which are performed 15 times, 17 times and 13 times respectively in a corpus of 3,017 quest-achieving runs. Such actions would not be considered part of the MUD language, even though they are significant for the quests in question. In order to overcome this problem, we propose to include the following 20 additional attributes when activating the MML classifier. For each action a_j in the data set we include qa_{ij} i=1, . . . , 20 (one attribute for each quest), where the value of qa_{ij} is:

$$qa_{ij}=\frac{\text{number of instances of quest } Q_i \text{ in which action } a_j \text{ was performed}}{\text{number of completed instances of quest } Q_i} .$$

However, in this case, the nature of the learned language differs, since what is being learned are the actions that are significant for each quest, rather than the actions that are significant for the MUD as a whole. In order to test the impact of this approach, we would need to split the evaluation process so that each quest is evaluated separately.

Relaxing simplifying assumptions

The results in this paper were obtained under certain user-related and domain-related simplifying assumptions. Examples of the former are: all users complete a quest, all users have similar profiles (i.e., behaviour patterns), all users attempt one quest at a time, and the interactions between users can be ignored. Among the latter we have: the domain has certain independence relations, and only certain types of data are available.

The first two assumptions will be relaxed simultaneously by including non-quest runs into our observations, and using a classification mechanism to build user profiles which reflect the kinds of activities performed by different types of users. A DBN which incorporates a user's class will then be built and trained from this data. The plan recognition task will involve the identification of a user's profile on the basis of his or her actions and visited locations, and the prediction of the actions, locations and objectives of this user in the context of the identified profile. The relaxation of the third and fourth user-related assumptions requires the extension of our system so that it can handle conjunctive goals and goals that change over time, and also so that it can keep track of the actions and locations of all the users playing the game at the same time.

The data for our domain originally provided at the beginning of this research were limited. Recently we have started collecting additional data, e.g., horizontal movements and the health and wealth state of the players. These data will allow us to develop more detailed models, and to test them against the baseline results obtained with our current models. We also plan to investigate whether we can improve the performance of the system by including the object of an action in our models.

Applying our approach to other domains

In addition to extending our models to handle richer MUD data, we are interested in applying our approach to other domains. The four simple models explored in this paper for the MUD domain show promise for application to other domains with similar features, e.g., the WWW and Unix. If we consider the location variable to be a typical state variable, and the quest variable to correspond to the accepted notion of a goal, then the set of node types encompassing action, state and goal is very general.

Like the MUD, the WWW has a hierarchical location structure, but the WWW has a very limited number of actions (dictated by the Web browser). This indicates that `locationModel` has potential for predicting the next web-page a user may fetch on the basis of the previous pages visited.* The Unix domain also has a hierarchical location structure, and like the MUD, it contains a large number of possible actions, though probably less than 7,259. Further, it appears that the object of an action is of importance in Unix, hence the intended extension to incorporate objects into the models used for the MUD will also be applicable to the Unix domain. Finally, Unix goals such as sending a file to a printer loosely correspond to MUD quests. However, the Unix domain highlights the importance of extending our approach to conjunctive goals, since a typical Unix goal may be to print on a double sided printer that is also a color printer and that is also on the fourth floor

* A related application of keyhole plan recognition to the WWW is in the design of filtering agents which unobtrusively infer a user's preferences from observations of his or her behaviour, e.g., (Joachims et al., 1997; Moukas and Maes, 1998; Balabanović, 1998).

(from (Lesh and Etzioni, 1996)). It is difficult to determine similar goals in the MUD, which limits the applicable models to those containing a single quest node. However, our current models may be extended to DBNs which have one node for each goal conjunct. Initially, we intend to apply our models to the recognition of simple Unix goals, and then consider extensions to handle conjunctive goals.

Acknowledgments

This research was supported in part by grant A49600323 from the Australian Research Council. The authors are indebted to Michael McGaughey for writing the data collection programs for the MUD and for his assistance during the initial stages of this project. The authors would also like to thank Ariel Bud for valuable contributions throughout this project.

References

Albrecht, D. W., Zukerman, I., and Nicholson, A. E. (1997). Bayesian models for keyhole plan recognition in an adventure game (extended version). Technical Report 328, Department of Computer Science, Monash University, Victoria, Australia.

Allen, J. and Perrault, C. (1980). Analyzing intention in utterances. *Artificial Intelligence*, 15:143–178.

Balabanović, M. (1998). Exploring versus exploiting when learning user models for text recommendation. *User Modeling and User-adapted Interaction*, this issue.

Bauer, M. (1996). Acquisition of user preferences for plan recognition. In *UM96 – Proceedings of the Fifth International Conference on User Modeling*, pages 105–112, Kona, Hawaii.

Box, G. E. and Tiao, G. C. (1973). *Bayesian Inference in Statistical Analysis*. Addison-Wesley Publishing Company, Philippines.

Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):195–210.

Cañamero, D., Delannoy, J., and Kodratoff, Y. (1992). Building explanations in a plan recognition system for decision support. In *ECAI92 Workshop on Improving the Use of Knowledge-Based Systems with Explanations*, pages 35–45, Vienna, Austria.

Carberry, S. (1990). Incorporating default inferences into plan recognition. In *AAAI90 – Proceedings of the Eight National Conference on Artificial Intelligence*, pages 471–478, Boston, Massachusetts.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts.

Charniak, E. (1997). Personal communication.

Charniak, E. and Goldman, R. P. (1993). A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):50–56.

Conati, C., Gertner, A. S., VanLehn, K., and Druzdzel, M. (1997). On-line student modeling for coached problem solving using Bayesian Networks. In *UM97 – Proceedings of the Sixth International Conference on User Modeling*, pages 231–242, Sardinia, Italy.

Dagum, P., Galper, A., and Horvitz, E. (1992). Dynamic network models for forecasting. In *UAI92 – Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 41–48, Stanford, California.

Dean, T. and Wellman, M. P. (1991). *Planning and control*. Morgan Kaufmann Publishers, San Mateo, California.

Forbes, J., Huang, T., Kanazawa, K., and Russell, S. (1995). The BATmobile: Towards a Bayesian automated taxi. In *IJCAI95 – Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1878–1885, Montreal, Canada.

Good, I. J. (1965). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. Research Monograph No. 30. MIT Press, Cambridge, Massachusetts.

Heckerman, D. (1995). A tutorial on learning Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research.

Huber, M. J., Durfee, E. H., and Wellman, M. P. (1994). The automated mapping of plans for plan recognition. In *UAI94 – Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 344–350, Seattle, Washington.

Jameson, A. (1996). Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interaction*, 5:193–251.

Joachims, T., Freitag, D., and Mitchell, T. (1997). Webwatcher: A tour guide for the world wide web. In *IJCAI97 – Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 770–775, Nagoya, Japan.

Lesh, N. (1997). Adaptive goal recognition. In *IJCAI97 – Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1208–1214, Nagoya, Japan.

Lesh, N. and Etzioni, O. (1995). A sound and fast goal recognizer. In *IJCAI95 – Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1704–1710, Montreal, Canada.

Lesh, N. and Etzioni, O. (1996). Scaling up goal recognition. In *Principles of Knowledge Representation and Reasoning*, pages 178–189.

Litman, D. and Allen, J. F. (1987). A plan recognition model for subdialogues in conversation. *Cognitive Science*, 11:163–200.

Moukas, A. and Maes, P. (1998). User modeling in an evolving multi-agent system. *User Modeling and User-adapted Interaction*, this issue.

Nicholson, A. E. and Brady, J. M. (1994). Dynamic belief networks for discrete monitoring. *IEEE Systems, Man and Cybernetics*, 24(11):1593–1610.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers, San Mateo, California.

Pynadath, D. and Wellman, M. (1995). Accounting for context in plan recognition with application to traffic monitoring. In *UAI95 – Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 472–481, Montreal, Canada.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R. S. and Carbonell, J., editors, *Machine Learning: an Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto, California.

Raskutti, B. (1993). *Handling Uncertainty during Plan Recognition for Response Generation*. PhD thesis, Monash University, Victoria, Australia.

Raskutti, B. and Zukerman, I. (1991). Generation and selection of likely interpretations during plan recognition. *User Modeling and User Adapted Interaction*, 1(4):323–353.

Russell, S., Binder, J., Koller, D., and Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. In *IJCAI95 – Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1146–1152, Montreal, Canada.

Wærn, A. and Stenborg, O. (1995). Recognizing the plans of a replanning user. In *Proceedings of the IJCAI-95 Workshop on The Next Generation of Plan Recognition Systems: Challenges for and Insight from Related Areas of AI*, pages 113–118, Montreal, Canada.

Wallace, C. (1990). Classification by minimum-message-length inference. In Goos, G. and Hartmanis, J., editors, *ICCI '90 – Advances in Computing and Information*, pages 72–81. Springer-Verlag, Berlin.

Wallace, C. and Boulton, D. (1968). An information measure for classification. *The Computer Journal*, 11:185–194.

Appendix A: Update Formulas for the Four Models

In a Bayesian network (Pearl, 1988) we say that an undirected path between a set of nodes X and another set of nodes Y is *blocked* by a set of nodes E, if there is a node Z on the path for which one of the following conditions holds:

1. Z is in E and Z has one arrow on the path leading in and one arrow out.
2. Z is in E and Z has both path arrows leading out.

3. Neither Z nor any of its descendants is in E, and both arrows lead into Z.

A set of nodes E is said to *d-separate* two sets of nodes X and Y if every undirected path between X and Y is blocked.

The d-separation condition. The set of nodes E d-separates X and Y if and only if X and Y are conditionally independent given E.

A.1 MAINMODEL

This model was constructed assuming that the next action depends only on the current action, the next location and the current quest, and the next location depends only on the current location and the current quest. Therefore:

$$\begin{aligned}\Pr(a_1|q', q, a_0, l_0, l_1) &= \Pr(a_1|q', a_0, l_1)\,,\\ \Pr(l_1|q', q, a_0, l_0) &= \Pr(l_1|q', l_0)\,,\end{aligned}$$

and for $n \geq 0$,

$$\begin{aligned}\Pr(a_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n, l_{n+1}) &= \Pr(a_{n+1}|q', a_n, l_{n+1})\,,\\ \Pr(l_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n) &= \Pr(l_{n+1}|q', l_n)\,.\end{aligned}$$

Also, in this model node Q d-separates node Q' from nodes $\{A_0, L_0\}$. So $\Pr(q'|q, a_0, l_0) = \Pr(q'|q)$. Therefore, we obtain the following update equations:

$$\begin{aligned}\Pr(a_1|q, a_0, l_0) &= \sum_{q', l_1} \Pr(a_1|q', q, a_0, l_0, l_1)\Pr(l_1|q', q, a_0, l_0)\Pr(q'|q, a_0, l_0)\\ &= \sum_{q', l_1} \Pr(a_1|q', a_0, l_1)\Pr(l_1|q', l_0)\Pr(q'|q)\,,\\ \Pr(l_1|q, l_0, a_0) &= \sum_{q'} \Pr(l_1|q', q, a_0, l_0)\Pr(q'|q, a_0, l_0)\\ &= \sum_{q'} \Pr(l_1|q', l_0)\Pr(q'|q)\,,\end{aligned}$$

and for $n \geq 0$,

$$\begin{aligned}&\Pr(a_{n+1}|q, a_0, l_0, \ldots, a_n, l_n)\\ &\quad= \sum_{q', l_{n+1}} \{\Pr(a_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n, l_{n+1}) \times\\ &\qquad\qquad \Pr(l_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n)\Pr(q'|q, a_0, l_0, \ldots, a_n, l_n)\}\\ &\quad= \sum_{q', l_{n+1}} \Pr(a_{n+1}|q', a_n, l_{n+1})\Pr(l_{n+1}|q', l_n)\Pr(q'|q, a_0, l_0, \ldots, a_n, l_n)\,,\\ &\Pr(l_{n+1}|q, a_0, l_0, \ldots, a_n, l_n)\\ &\quad= \sum_{q'} \Pr(l_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n)\Pr(q'|q, a_0, l_0, \ldots, a_n, l_n)\end{aligned}$$

$$= \sum_{q'} \Pr(l_{n+1}|q', l_n) \Pr(q'|q, a_0, l_0, \ldots, a_n, l_n) ,$$

$$\Pr(q'|q, a_0, l_0, \ldots, a_{n+1}, l_{n+1})$$

$$= \frac{\Pr(a_{n+1}, l_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n) \Pr(q'|q, a_0, l_0, \ldots, a_n, l_n)}{\Pr(a_{n+1}, l_{n+1}|q, a_0, l_0, \ldots, a_n, l_n)}$$

$$= \alpha \Pr(a_{n+1}|q', a_n, l_{n+1}) \Pr(l_{n+1}|q', l_n) \Pr(q'|q, a_0, l_0, \ldots, a_n, l_n) ,$$

where α is a normalizing constant.

A.2 ACTIONMODEL

This model was constructed assuming that the next action depends only on the current action and the current quest. Therefore:

$$\Pr(a_1|q', q, a_0) = \Pr(a_1|q', a_0) ,$$

and for $n \geq 0$,

$$\Pr(a_{n+1}|q', q, a_0, \ldots, a_n) = \Pr(a_{n+1}|q', a_n) .$$

Also, in this model node Q d-separates node Q' from node A_0. So $\Pr(q'|q, a_0) = \Pr(q'|q)$. Therefore, we obtain the following update equations:

$$\Pr(a_1|q, a_0) = \sum_{q'} \Pr(a_1|q', a_0) \Pr(q'|q) ,$$

and for $n \geq 0$,

$$\Pr(a_{n+1}|q, a_0, \ldots, a_n) = \sum_{q'} \Pr(a_{n+1}|q', a_n) \Pr(q'|q, a_0, \ldots, a_n) ,$$

$$\Pr(q'|q, a_0, \ldots, a_{n+1}) = \alpha \Pr(a_{n+1}|q', a_n) \Pr(q'|q, a_0, \ldots, a_n) ,$$

where α is a normalizing constant.

A.3 LOCATIONMODEL

This model was constructed assuming that the next location depends only on the current location and the current quest. Therefore:

$$\Pr(l_1|q', q, l_0) = \Pr(l_1|q', l_0) ,$$

and for $n \geq 0$,

$$\Pr(l_{n+1}|q', q, l_0, \ldots, l_n) = \Pr(l_{n+1}|q', l_n) .$$

Also, in this model node Q d-separates node Q' from node L_0. So $\Pr(q'|q, l_0) = \Pr(q'|q)$. Therefore, we obtain the following update equations:

$$\Pr(l_1|q, l_0) = \sum_{q'} \Pr(l_1|q', l_0) \Pr(q'|q) ,$$

and for $n \geq 0$,

$$\Pr(l_{n+1}|q, l_0, \ldots, l_n) = \sum_{q'} \Pr(l_{n+1}|q', l_n) \Pr(q'|q, l_0, \ldots, l_n),$$
$$\Pr(q'|q, l_0, \ldots, l_{n+1}) = \alpha \Pr(l_{n+1}|q', l_n) \Pr(q'|q, l_0, \ldots, l_n),$$

where α is a normalizing constant.

A.4 INDEPMODEL

This model was constructed assuming that the next action depends only on the current action and the current quest, and the next location depends only on the current location and the current quest. Therefore:

$$\Pr(a_1|q', q, a_0, l_0) = \Pr(a_1|q', a_0),$$
$$\Pr(l_1|q', q, a_0, l_0) = \Pr(l_1|q', l_0),$$

and for $n \geq 0$,

$$\Pr(a_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n) = \Pr(a_{n+1}|q', a_n),$$
$$\Pr(l_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n) = \Pr(l_{n+1}|q', l_n).$$

Also, in this model we have the following d-separations:

1. Node Q d-separates node Q' from nodes $\{A_0, L_0\}$.
2. Nodes $\{Q', Q, A_0, \ldots, A_n, L_0, \ldots, L_n\}$ d-separate node A_{n+1} from node L_{n+1}.

So, by the d-separation condition we have: $\Pr(q'|q, a_0, l_0) = \Pr(q'|q)$, and for $n \geq 0$,

$$\Pr(a_{n+1}, l_{n+1}|q', q, a_0, l_0, \ldots, a_n, l_n) = \Pr(a_{n+1}|q', a_n) \Pr(l_{n+1}|q', l_n).$$

Therefore, we obtain the following update equations:

$$\Pr(a_1|q, a_0, l_0) = \sum_{q'} \Pr(a_1|q', a_0) \Pr(q'|q),$$
$$\Pr(l_1|q, l_0, a_0) = \sum_{q'} \Pr(l_1|q', l_0) \Pr(q'|q),$$

and for $n \geq 0$,

$$\Pr(a_{n+1}|q, a_0, l_0, \ldots, a_n, l_n) = \sum_{q'} \Pr(a_{n+1}|q', a_n) \Pr(q'|q, a_0, l_0, \ldots, a_n, l_n),$$
$$\Pr(l_{n+1}|q, a_0, l_0, \ldots, a_n, l_n) = \sum_{q'} \Pr(l_{n+1}|q', l_n) \Pr(q'|q, a_0, l_0, \ldots, a_n, l_n),$$
$$\Pr(q'|q, a_0, l_0, \ldots, a_{n+1}, l_{n+1}) = \alpha \Pr(a_{n+1}|q', a_n) \Pr(l_{n+1}|q', l_n) \Pr(q'|q, a_0, l_0, \ldots, a_n, l_n),$$

where α is a normalizing constant.

Authors' Vitae

David Albrecht

Monash University, School of Computer Science and Software Engineering, Clayton, Victoria 3168, Australia

David Albrecht is a Research Fellow and Lecturer in Computer Science at Monash University. He received his B.Sc and Ph.D. degrees in Mathematics from Monash University, and has worked in several research areas including plan recognition, speech recognition, program extraction, linear logic, functional operator theory, general relativity, and optimal control theory.

Ingrid Zukerman

Monash University, School of Computer Science and Software Engineering, Clayton, Victoria 3168, Australia

Ingrid Zukerman is an Associate Professor in Computer Science at Monash University. She received her B.Sc. degree in Industrial Engineering and Management and her M.Sc. degree in Operations Research from the Technion – Israel Institute of Technology. She received her Ph.D. degree in Computer Science from UCLA in 1986. Since then, she has been working in the Department of Computer Science at Monash University. Her areas of interest are discourse planning, plan recognition, multi-media interfaces, agent modeling and speech recognition.

Ann Nicholson

Monash University, School of Computer Science and Software Engineering, Clayton, Victoria 3168, Australia

Ann Nicholson is a Senior Lecturer in Computer Science at Monash University. She received her B.Sc (Hons) and M.Sc. degrees in Computer Science from the University of Melbourne. In 1992 she received her D.Phil. in Engineering from the University of Oxford, where she was part of the Robotics Research Group. In 1994, after 2 years as a post-doctoral research fellow in Computer Science at Brown University, she took up a position in the Department of Computer Science at Monash University. Her areas of interest are reasoning under uncertainty, dynamic belief networks, stochastic planning, monitoring, scheduling, plan recognition, intelligent agents, robotics and sensor validation.