Trading off Granularity against Complexity in Predictive Models for Complex Domains

Ingrid Zukerman, David W. Albrecht, Ann E. Nicholson and Krystyna Doktor

School of Computer Science and Software Engineering
Monash University, Clayton, VICTORIA 3800, AUSTRALIA
{ingrid,dwa,annn,krys}@csse.monash.edu.au

Abstract. The automated prediction of a user's interests and requirements is an area of interest to the Artificial Intelligence community. However, current predictive statistical approaches are subject to theoretical and practical limitations which restrict their ability to make useful predictions in domains such as the WWW and computer games that have vast numbers of values for variables of interest. In this paper, we describe an automated abstraction technique which addresses this problem in the context of Dynamic Bayesian Networks. We compare the performance and computational requirements of fine-grained models built with precise variable values with the performance and requirements of a coarse-grained model built with abstracted values. Our results indicate that complex, coarse-grained models offer performance and computational advantages compared to simpler, fine-grained models.

1 Introduction

It has long been recognized in the Artificial Intelligence community that problem reformulations are central to the ability of systems to reason effectively in complex domains. A commonly used type of reformulation is abstraction, which involves ignoring or combining parts of the state space to overcome computational intractability. Of course, this reduction in complexity comes at a cost; using an abstraction, rather than the complete state-space, usually means that the computed solution is less accurate than the solution obtained with the complete state space. However, good abstractions achieve a state space reduction without significantly compromising the quality of the solution. Abstraction techniques have been used in a variety of problem-solving settings, such as automatic programming, design, diagnosis, planning and theorem proving [6].

In this paper, we focus on the use of abstraction techniques in Bayesian Networks (BNs) [11]. In particular, we utilize BNs in predictive statistical models for plan recognition – the area of user modeling which endeavours to predict a user's plans and goals from observations of the user's state. The recent application of predictive statistical models to realistic application domains, such as the WWW, computer games or interactive systems, has placed increased demands on these models to handle vast numbers of values for the variables of interest. For instance, the WWW has millions of locations, and computer games may have thousands of possible actions and locations. However, BNs cannot handle efficiently such application domains. This may be due to theoretical boundaries or restrictions imposed by currently available systems and operating conditions. For instance, a theoretical limitation of statistical prediction models in general and BNs in particular pertains to the amount of data that needs to be collected in order to perform meaningful predictions when there is a large number of variables or variable values. Further, the belief-update algorithms of BNs are exponential in their state space (Section 2). Current BN software packages, such a Netica [10] and Hugin [8], are subject to memory restrictions that limit the number of variables and variable values they can handle. This problem is exacerbated in Java-based WWW applications, where executing programs must reside in the client's site.

In this paper, we describe an automated abstraction technique to address the "large state space" problem in the context of Dynamic Bayesian Networks (DBNs) [4] – a variant of BNs used in a variety of applications, e.g., [7, 12]. We use DBNs to predict a user's quest (goal) in a Multi-User Dungeon (MUD) adventure game with 2140 locations where 1311 actions were performed by players when attempting to achieve 24 different quests (Section 3). In our previous work, we addressed the large state space problem by removing low-probability values from the state space and taking advantage of domain features [1]. This involved deleting from the state space events that were not found during training, ignoring commands that contain typographical errors, and taking advantage of the hierarchical structure of the domain to merge specific locations into regions. While these approaches to abstraction often yield good results, they frequently do not generalize across domains with different features. In this paper, we offer a general clustering approach to abstraction which uses automatically learned categories, instead of precise variable values, to reduce the size of the state space (Section 5).

In the next section, we review abstraction methods applied to prediction models similar to ours. We then describe the features of our domain, our basic (fine-grained) prediction models and our abstraction process. In Section 6, we compare the performance and computational requirements of a coarse-grained prediction model with the performance and requirements of our fine-grained models. We then present concluding remarks and outline our plans for future work.

2 Related Research

DBNs have been used for knowledge representation and reasoning in domains where the world changes and the focus is reasoning over time. In these domains, the DBN grows over time, and the state of each domain variable at different times is represented by a series of nodes. In addition to having the same computational problems as those experienced by ordinary BNs during belief updating (both exact and approximate inference are NP-hard [2, 3]), DBNs incur additional complexity as the number of time-slices increases. In order to constrain the state space, the DBN connections over time are typically Markovian, and a temporal 'window' is imposed. For example, our DBN for predicting a user's goals in the MUD is limited to a two-time-slice window [1].

In this paper, we focus on the use of abstraction to reduce the size of the state space when the DBN nodes represent domain variables that have vast numbers of values. Abstraction encompasses a number of techniques: (1) ignoring variables of low relevance; (2) ignoring some of the less relevant values a variable may take; and (3) modulating the precision (or granularity) of the variables by combining values.

In a BN framework, ignoring a variable of low relevance is equivalent to pruning it from the network. An example of this is the work of Jitnah [9], who uses a relevance measure based on mutual information to prune past nodes from a DBN. In the related area of decision-theoretic planning using Markov Decision Processes, Dearden and Boutilier [5] remove completely variables of low relevance in terms of their impact on utilities of actions. Merging or ignoring individual values, as proposed in this paper, is not an option for the problems considered by these researchers, as the variables are either Boolean propositions or have very few values.

An example of the second form of abstraction – ignoring some of the less relevant values a variable may take – is the removal of low-probability values from the action state space, as described in [1].

An example of the third approach is Wellman and Liu's use of abstraction for the approximate evaluation of BNs when the state space is prohibitive or when a real-time response is required [14]. They trade off accuracy in the result for computational efficiency by varying the granularity of the variable state spaces. This is done by merging values that are adjacent in the enumeration of the variable's state space. This approach can be effective when such values are "similar", but is not suitable for many domains. Jitnah [9] investigates measures for choosing which values to merge based on similarities of their entries in the network's Conditional Probability Tables (CPTs). Another example of this clustering form of abstraction is our previous exploitation of hierarchical aspects of the domain to merge individual locations into regions [1]. As mentioned in the previous section, although this method gives reasonable results, it is not generalizable across domains with different features. In this paper, we describe a machine learning classification method for abstraction that automatically groups values that have similar features.

3 Domain

Our application domain is the "Shattered Worlds" Multi-User Dungeon (MUD) – a text-based virtual reality game where players compete for limited resources in an attempt to achieve various quests. As stated in Section 1, the MUD has 24 different quests, and 2140 locations where 1311 actions were observed. As shown in [1], the MUD is a complex domain whose features challenge traditional plan recognition systems. This motivated our use of DBNs to develop models that predict users' actions, locations and goals. In this paper, we extend our previous results by providing a generally applicable abstraction method that supports the development of DBNs in complex domains.

The MUD software collects information about the *runs* performed by each player. Each run is composed of a sequence of data points collected from the time a player enters the MUD or completes a quest until s/he achieves a new quest. Each data point contains information regarding the state of the player when an event happens (either an action is performed or a quest is achieved). Table 1 shows the following information for a subset of the data points collected during a sample run: a time stamp, the name of the player, the location where the action was executed, and the name of the action (or quest achieved). This is the information used to build our DBNs (Section 4).

Table 1. 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*

4 Knowledge Representation

In this section, we discuss briefly three simple DBNs developed for the MUD domain [1]. These models, whose predictive power was investigated in our previous research, were selected since they enable us to compare the predictive performance and computational requirements of our fine- and coarse-grained modeling techniques under otherwise equivalent conditions.

The domain variables represented as DBN nodes are actions, locations and quests.

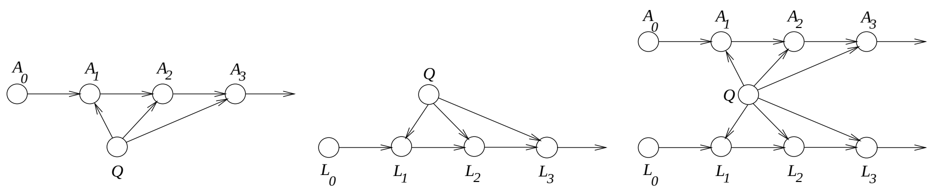
Action (A) – represents the possible actions a player may perform in the MUD while trying to achieve a quest ($|A|$=1311 actions in our current trials), plus the special `other` action, which includes all previously unseen actions. In our current implementation, we take an action to be the first string of non-blank characters entered by a user.

Location (L) – represents the possible locations visited by a quest-achieving player ($|L|$=2140 locations), plus the special `other` location, which includes all previously unseen locations.

Quest (Q) – represents the 24 different quests a player may undertake.

Figure 1 shows our three DBNs: (a) `actionModel`, (b) `locationModel`, and (c) `indepModel`. `actionModel` infers a user's quest from his/her observed actions only; `locationModel` infers a user's quest from observed locations only; and `indepModel` considers both actions and locations. The arcs in the DBNs reflect the different influences between the domain variables. For example, the next action in `actionModel` depends on the previous action and the quest being undertaken. Each node in a DBN is associated with a CPT that quantifies its relationship with its parents. For example, the CPT associated with node A_{i+1} in `actionModel` represents the conditional probability Pr$(A_{i+1}|A_i, Q)$.

The main CPT used by `actionModel` has $(|A|+1)^2 \times Q$ entries (= $(1311+1)^2 \times 24 \cong 41 \times 10^6$); the main CPT used by `locationModel` has $(|L|+1)^2 \times Q$ entries (= $(2140+1)^2 \times 24 \cong 110 \times 10^6$); and `indepModel` requires both of these CPTs, which have a total size of $\cong 41 \times 10^6 + 110 \times 10^6$ (151×10^6). In our previous work, the size of the CPTs was substantially reduced by not storing zero-probability events, e.g., for `indepModel`, this yielded CPTs whose total size was 1.1×10^6 on average [1]. As



(a) `actionModel` (b) `locationModel` (c) `indepModel`

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

a result, the real-time belief updating process was computationally feasible. However, this approach is effective only if the CPTs are sparse.

5 Automated Abstraction

The primary purpose of the models considered in this paper is to predict which quest a user is doing. Consequently the abstractions we considered involve using features of the state space which are quest related. These abstractions are based on the observation that the locations/actions which are more frequently visited/performed in the process of doing a quest are more important for that quest.

The abstraction of the locations involves clustering the locations into regions. To do this we first calculated for each location L_i 24 attributes, a_{ij} for $j = 1, \ldots, 24$, one attribute for each quest, where each attribute counts the number of times location L_i was visited by all users while trying to accomplish quest Q_j. For each location L_i:[1]

$$a_{ij} = \sum_{\text{all users}} \sum_{L_k} [\text{\# of times each user moved from } L_k \text{ to } L_i \ (L_k \neq L_i) \text{ while doing } Q_j]$$

Once all the attributes were calculated, we applied the classifier Snob [13] to these attributes in order to automatically determine the regions. The classifier identified 116 regions consisting of locations with similar attributes. One region consisted of all the locations which were never visited and hence were not important for the prediction of any quest. The other regions consisted of groups of locations, such that each group comprised locations with similar hierarchical descriptions. However, not all locations with a similar hierarchical description were found in the same region.

The abstraction of actions into action classes was learned in a manner similar to the learning of regions for locations:[2] 24 attributes were calculated for each action, one for each quest, representing how often an action was performed by all users doing a particular quest. Once all the attributes were calculated, we used the Snob classifier to automatically cluster the actions into classes. Snob found 85 actions classes, including one with all the unused actions.

[1] Note that the attribute value is not incremented when a user stays in the same location over several actions.

[2] However, repeated action occurrences were counted, since several actions may be performed in the same location.

Having obtained the abstractions of the locations and actions, we investigated `abstractModel` – a DBN which has the same structure as `indepModel`, but with regions instead of locations, and action classes instead of actions. The total size of the main CPTs in `abstractModel` is $(116+1)^2 \times 24 + (85+1)^2 \times 24$ ($\cong 50 \times 10^4$), compared with 151×10^6 for `indepModel`. This corresponds to a reduction by 99.7% in the size of the models. Further, the size of `abstractModel` is 99.6% smaller than that of `locationModel` and 98.8% smaller than that of `actionModel`.

In the future, we intend to investigate using other attributes to classify locations and actions, such as the number of users who visited a location or performed an action. This would enable the classifier to distinguish between locations/actions which are visited/performed often by one user and those which are visited/performed only a few times by many different users.

6 Results

In this section, we present empirical results showing how the predictive performance of the DBN models described in Section 4 compares with that of `abstractModel` (Section 5). The measure of performance used to compare these models is *average prediction*, which assesses how well a model predicts the actual quest [1]. This measure computes the average across all test runs of the predicted probability of the actual quest at a particular point t during the performance of a quest:

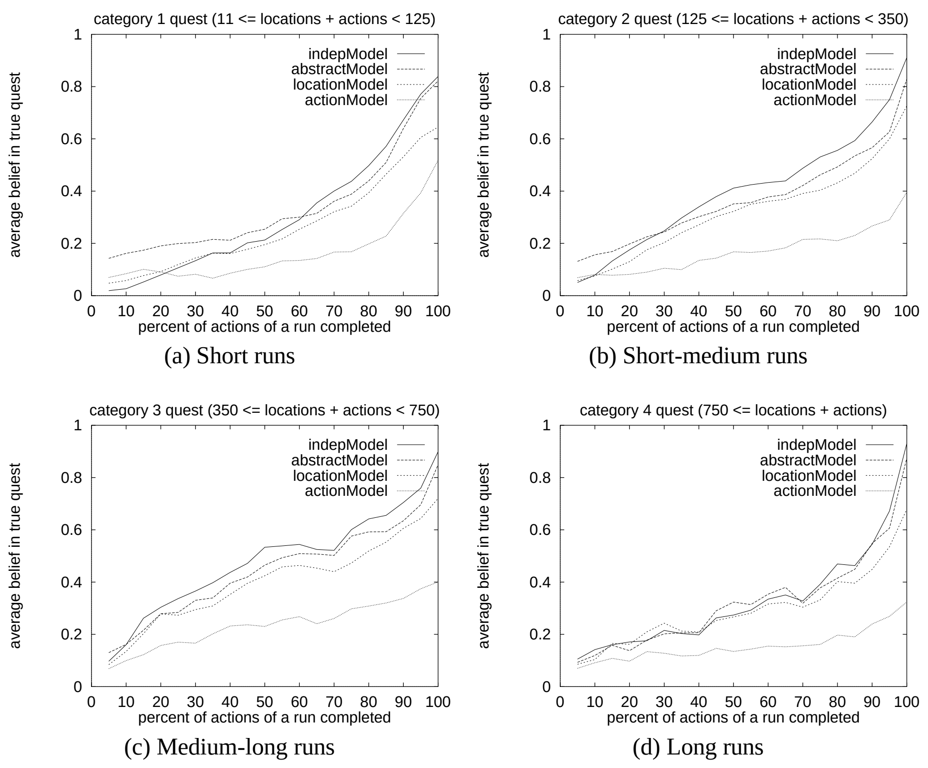
$$\textit{average prediction}_t = \frac{1}{n}\sum_{i=1}^{n} \Pr(\text{actual value of } \textit{quest} \text{ at point } t \text{ in the } i\text{-th test run}) \ ,$$

where n is the number of test runs. Our results were obtained by training each model on 80% of the data and testing on 20% with cross-validation using 5 different splits of the data.

In order to determine whether the length of a run (in terms of number of actions performed and locations visited) affects the relative performance of the different models, we divided the test runs into four categories as follows: *short* – between 11 and 124 locations and actions; *short-medium* – between 125 and 349 locations and actions; *medium-long* – between 350 and 749 locations and actions; and *long* – more than 750 locations and actions. Each of these categories contained approximately 300 runs.

Figure 2 shows the average predicted probability of the actual quest for each of the four DBNs considered in this paper (`actionModel`, `locationModel`, `indepModel` and `abstractModel`) for each of the four categories of runs. In order to compare the performance of the different models across runs where the number of recorded actions varies, the x-axis was chosen to represent the percentage of the actions in a run that have been performed. Specifically, the plots in Figure 2 show the average prediction after 5%, 10%, 15%, . . ., 100% of the actions in a run have been performed.

The general trends emerging from the results shown in Figure 2 are: (1) the performance of all the models improves as quest completion draws near; (2) `indepModel` has the best performance overall, followed by `abstractModel`, `locationModel` and `actionModel`; (3) `abstractModel` outperforms the other models during some time span (often at the beginning of a quest); and (4) the performance of `actionModel`



(a) Short runs

(b) Short-medium runs

(c) Medium-long runs

(d) Long runs

Fig. 2. Predictive performance of precise versus abstracted models.

is substantially worse than that of the other models.[3] This last (seemingly counter-intuitive) result may be explained by the observation that MUD players often interleave quest-related actions with other actions that do not contribute to quest achievement. We now consider the first three of these trends in turn.

The performance of all the models improves as quest completion draws near. At first glance, it appears that in the initial stages of a run, the predictive accuracy of all the models is low in absolute terms. However, even after only 10% of the actions in a run have been performed, the average predictive accuracy of the best performing model (which is either `indepModel` or `abstractModel`) is approximately 16%. This is significantly better than randomly selecting a quest from 24 uniformly distributed quests (which has an average predictive accuracy of 4%). When 70% of the actions in a run have been performed, the average predictive accuracy of `indepModel`, which has the best performance at that stage, is 40% for short runs, 49% for short-medium runs, 53% for medium-long runs and 33% for long runs. These probabilities indicate that the actual quest is often assigned the highest probability mass among the candidate quests (indeed, at this point in a run `indepModel` assigns the highest probability to the actual

[3] T-tests performed for each set of predictions and each pair of models confirmed these results at the 5% significance level.

quest 35% of the time for short runs, 51% for short-medium runs, 48% for medium-long runs and 32% for long runs). The predictive accuracy of the three best models, `indepModel`, `abstractModel` and `locationModel`, climbs sharply from that point, with `indepModel` reaching an average predictive accuracy of approximately 90% upon quest completion for all run categories.

`indepModel` *has the best performance overall, followed by* `abstractModel`, `locationModel` *and* `actionModel`. The superior overall performance of `indepModel` is expected, as it is both complex and precise. The fact that `abstractModel` outperforms `locationModel` and `actionModel` indicates that it is worth trading off simple but fine-grained DBN models against more complex but coarse-grained DBN models. This trade off yields both substantial computational savings as well as improved performance.

`abstractModel` *outperforms the other models during some time span.* As seen in Figure 2, for the three shorter run categories this time span occurs at the beginning of a run: the first 60% of a run for short runs, the first 25% for short-medium runs, and the first 10% for medium-long runs. In contrast, for the long runs, `abstractModel` has the best performance between 45% and 65% of the actions in a run (while `locationModel` outperforms the other models between 25% and 40% of the actions in a run).

The superior performance of `abstractModel` during the initial stages of the shorter runs may be attributed to the following factors.

Vague predictions are better at ruling out unlikely quests. This is a result of the relative sparseness of the CPTs for the fine-grained DBNs. Many entries in the $L \times L \times Q$ CPT or the $A \times A \times Q$ CPT contain a rather low probability or 0. Thus, the distinction between events seen in training and those not seen in training is quite low, resulting in a low discrimination between likely and unlikely quests. In contrast, each entry in the CPTs of the coarse-grained model collates information from several locations (which are grouped according to the quests achieved while visiting these locations) or several actions (which are also grouped in this manner). Thus, the differences between the various entries in these CPTs are more pronounced than the differences between the entries of the fine-grained CPTs, leading to a greater ability to rule out unlikely quests, and assigning a higher probability mass to the remaining quests.

Precise predictions may be misleading at the beginning, but are useful at the end. This follows from the observation that the precise information with the most discriminatory power is often collected close to quest completion. In the MUD, as in many other domains, a particular event (e.g., action or location) observed at the beginning of a quest may not be particularly indicative of the quest being attempted, but the same event observed later on may be a powerful indicator of the quest under consideration. Since the frequency counts for the CPTs are collected throughout the performance of a quest (without distinguishing between its beginning or its end), the information in CPT entries with higher conditional probabilities may be more representative of the later stages of a run, rather than its earlier stages. Nonetheless, these CPTs are used to make predictions at all stages of a run, thereby yielding lower quality predictions at the beginning of a run.

The superior performance of `abstractModel` around the middle of the long runs may be explained by the observation that initially players may not be attempting the quest they end up doing. Thus, one could argue that they are actually starting this quest only later in a run. Further, in long runs players perform more actions that do not contribute to quest completion than in shorter runs. These two observations account for the lower predictive accuracy of all the DBNs for the long runs, compared to their performance in the shorter runs (as indicated above, after 70% of the actions have been performed, `indepModel` – the best performing model – predicts the actual quest with an average probability of 0.35 for the long runs, compared 0.4, 0.48 and 0.53 for the other run categories).

7 Conclusions and Future Work

We have offered an automated abstraction technique which addresses the large state space problem in the context of DBNs. The comparison of the predictive performance and computational requirements of fine-grained models with the performance and requirements of the coarse-grained model built with our abstraction technique leads to the following conclusions: (1) fine-grained DBN models generally have a higher predictive accuracy than coarse-grained models with the same structure (this is corroborated by our experiments with coarse-grained versions of `actionModel` and `locationModel`); (2) coarse-grained DBN models perform better than fine-grained models with a simpler structure; (3) coarse-grained DBN models generally perform better than fine-grained models with the same structure during the initial stages of a task; (4) the space requirements of the coarse-grained DBN model built with our abstraction technique are approximately 1/100 of the space requirements of the fine-grained models; and (5) the space savings achieved by coarse-grained DBN models are twice as large as the savings achieved by ignoring zero-probability events.

These conclusions clearly point towards complex, coarse-grained models as a viable alternative to fine-grained, simpler models for domains with large data spaces. The coarse-grained complex models not only perform better than their simpler fine-grained counterparts, but also incur very large savings in space. In the future, we intend to extend this idea to include additional variables, e.g., MUD players, who may be classified according to the length of their sessions and their quest completions. This will enable us to determine whether the insights obtained from this research regarding the granularity-complexity trade-off extend to models of a higher complexity than those investigated here.

As indicated in Section 5, the results presented in this paper were obtained by performing abstractions with respect to one type of variable, i.e., quests. In the future, we intend to investigate using other attributes to classify locations and actions, such as the number of users who visited a location or performed an action. In addition, we propose to study the computational and performance implications of an abstract DBN that relies on the joint classification of actions and locations (rather than their separate classification, as done in this paper), and the computational implications of combining coarse-grained models with sparse CPTs that do not represent zero-probability events.

Finally, an interesting result of our work pertains to the good performance of the coarse-grained DBN during the initial stages of a quest. This points to the need to investigate a dynamic model selection policy, which can change from a coarse-grained model to a fine-grained model at some point during task performance.

References

1. D.W. Albrecht, I. Zukerman, and A.E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8(1-2):5–47, 1998.
2. G.F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.
3. P. Dagum and M. Luby. Approximating probabilistic inference in belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.
4. T. Dean and M.P. Wellman. *Planning and control*. Morgan Kaufmann Publishers, San Mateo, California, 1991.
5. R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997.
6. T. Ellman. Symposium on abstraction, reformulation and approximation. *http://www.cs.vassar.edu/∼ellman/sara98/sara98.html*, 1998.
7. J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The BATmobile: Towards a Bayesian automated taxi. In *IJCAI95 – Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1878–1885, Montreal, Canada, 1995.
8. Hugin. HUGIN: Expert. *http://www.hugin.dk*, 2000.
9. N. Jitnah. *Using Mutual Information for Approximate Evaluation of Bayesian Networks*. PhD thesis, Monash University, School of Computer Science and Software Engineering, 1999.
10. Netica. Norsys: Software corp. *http://www.norsys.com/netica.html*, 2000.
11. J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers, San Mateo, California, 1988.
12. D.V. Pynadath and M.P. Wellman. 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, 1995.
13. C.S. Wallace. Classification by minimum-message-length inference. In G. Goos and J. Hartmanis, editors, *ICCI '90 – Advances in Computing and Information*, pages 72–81. Springer-Verlag, Berlin, 1990.
14. M.P. Wellman and C.L. Liu. State-space abstraction for anytime evaluation of probabilistic networks. In *UAI94 – Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 567–574, Seattle, Washington, 1994.