# The BATmobile: Towards a Bayesian Automated Taxi

**Jeff Forbes, Tim Huang, Keiji Kanazawa, Stuart Russell**
Computer Science Division
University of California
Berkeley, CA 94720, USA
jforbes,tthuang,kanazawa,russell@cs.berkeley.edu

## Abstract

The problem of driving an autonomous vehicle in normal traffic engages many areas of AI research and has substantial economic significance. We describe work in progress on a new approach to this problem that uses a decision-theoretic architecture using dynamic probabilistic networks. The architecture provides a sound solution to the problems of sensor noise, sensor failure, and uncertainty about the behavior of other vehicles and about the effects of one's own actions. We report on advances in the theory of inference and decision making in dynamic, partially observable domains. Our approach has been implemented in a simulation system, and the autonomous vehicle successfully negotiates a variety of difficult situations.

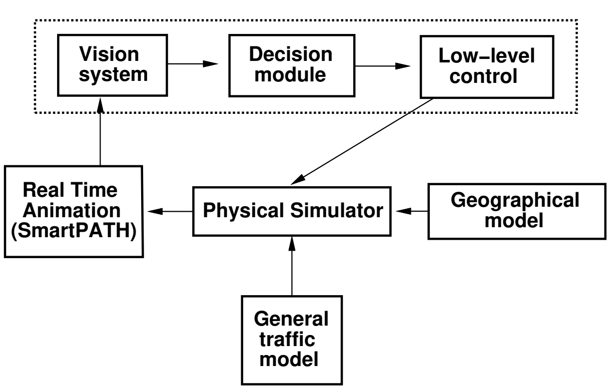


Figure 1: The basic components of the BAT project.

## 1 The BAT Project

Several government agencies and corporations in Europe, Japan, and the US are currently undertaking research in IVHS (Intelligent Vehicle and Highway Systems) with the aim of substantially reducing congestion and accidents, which cost $500 billion/year and 100,000 lives/year, respectively. In the near future, several research projects expect to demonstrate prototype systems for automated highways in which vehicles travel in segregated lanes under centralized control using inter-vehicle negotiation and only minimal sensing. For example, PATH (Partners for Advanced Transit and Highways), an agency of the State of California, has designed a "platooning" scheme in which large groups of automated vehicles travel together with minute inter-vehicle spacing, thereby quadrupling highway capacity. This scheme forms the basis for a nationwide consortium funded by the US Government.

The BAT (Bayesian Automated Taxi) project, although funded by PATH, takes an entirely different approach. The aim is to introduce autonomous vehicles into normal highway traffic. On the one hand, this approach:

- Eliminates the need for extensive highway rebuilding.
- Allows a gradual, evolutionary shift to fully automated highways.
- Eliminates the risk of widespread system failure.
- Improves integration with urban surface streets.

On the other hand, whereas driving in a restricted, instrumented lane is primarily a problem in control theory, driving in normal traffic on an uninstrumented highway is more difficult and engages many areas of AI research. This is either a disadvantage or an advantage, depending on one's viewpoint. Because the necessary low-level capabilities such as visual vehicle monitoring [Huang *et al.*, 1994] and lane-following [Dickmanns and Zapp, 1987; Pomerleau, 1993] are reaching maturity, we have decided to confront this challenge.

The first phase of the project is a feasibility study to establish the computational and sensing requirements for driving and to investigate the nature of the necessary decision algorithms. We use a 2-D physical simulation that generates moderately realistic 3-D rendered video output (SmartPATH), which is passed to the BAT (see Figure 1). Using this information, the BAT must understand the current traffic situation, select high-level actions such as braking, accelerating, and lane changing, and implement those actions using low-level control.

The AI problems involved in driving are legion: the BAT must make decisions in real-time; its sensors are noisy—position errors are significant, and some vehicles may not be detected, especially at night or in poor weather conditions; sensor inputs must be integrated, and some sensors may fail altogether; the world is only partially observable—vehicles may be occluded, and other drivers' intentions are invisible;

and the BAT has only a stochastic model of the results of its own actions. Finally, successful deployment requires a critical error rate below 1 in $10^8$ seconds, i.e., it must perform at least as well as a good driver. These considerations mean that the BAT *must*, at least at "compile time," weigh up quantitative risks and benefits. The only "safe" policy is to stay in one's garage.

Although there are dozens of projects worldwide with similar goals, descriptions of which can be found in the proceedings of many IVHS conferences, almost none have started from the premise that sensors and actuators are noisy and error-prone—a fact that has resulted in several deaths during testing of "Advanced Intelligent Cruise Control" (AICC) systems. One exception is the work of Niehaus and Stengel [Niehaus and Stengel, 1991], who have recently incorporated a somewhat *ad hoc* form of probabilistic reasoning into their rule-based driving controller. Since their system assumes worst-case outcomes in looking ahead, and does not integrate percepts over time in assessing the current state, its performance is limited.

The BAT decision making architecture is multi-level, with a high level component in charge of overall trip planning and parameters such as desired cruising speed, which it passes down to a driving module in charge of the concrete task of driving the vehicle in traffic in real-time. This paper focuses on the latter task. The driving problem can be modelled formally as a POMDP (partially observable Markov decision process). In a POMDP, the optimal decision is a function of the current belief state—the joint distribution over all possible actual states of the world.[1] The problem can be divided into two parts: updating the current belief state, and making a decision based on that belief state. We begin this paper by showing how we use *dynamic probabilistic networks* to represent and update the belief state, and introduce *temporally invariant networks* and stochastic sampling as efficient methods for updating in real-time. We then describe three methods for making decisions: lookahead planning, explicit policy representations, and machine learning. Finally, we illustrate the effectiveness of an implemented BAT controller in a variety of scenarios designed to test the BAT's ability to make fast, effective decisions in difficult situations.

# 2 Maintaining the Current Belief State

In order to make appropriate control decisions, an agent must have accurate information about its own state and the state of its environment. For example, the BAT must know its own position, velocity, and intentions, and it must monitor those of its neighboring vehicles. It must also monitor road and weather conditions, since they may significantly affect the BAT's ability to drive.

The state of the BAT's environment is only partially observable. Sensor information for variables such as vehicle positions and velocities may be incomplete and noisy, while driver intentions and road conditions may not be directly measurable at all. Thus, the BAT cannot make decisions based merely upon the latest sensor readings. Rather, it must maintain estimates for the random variables that together represent the state of the world, and it must make its decisions based upon the joint probability distribution over all those variables. In the rest of this section, we outline our approach to maintaining the current belief state.

## 2.1 Dynamic probabilistic networks

To maintain the BAT's current belief state, we employ *dynamic probabilistic networks* (DPNs). Probabilistic networks are directed acyclic graphs in which nodes represent random variables (typically discrete) and arcs represent causal connections among the variables [Pearl, 1988]. Associated with each node is a CPT (conditional probability table) that provides conditional probabilities of the node's possible states given each possible state of its parents (or the prior probabilities if the node has no parents). Probabilistic networks offer a mathematically sound basis for making inferences under uncertainty. The conditional probability tables provide a natural way to represent uncertain events, and the semantics of the updated probabilities are well-defined. Knowledge of causal relationships among variables is expressed by the presence or absence of arcs between them. Furthermore, the conditional independence relationships implied by the topology of the network allow the joint probability distribution of all the variables in the network to be specified with exponentially fewer probability values than in the full joint distribution. When specific values are observed for some of the nodes in a probabilistic network, posterior probability distributions can be computed efficiently for any of the other nodes using a variety of inference algorithms [Pearl, 1988]. The extension to continuous variables is straightforward, and stochastic sampling provides a simple way to perform inference with such variables.

DPNs allow for reasoning in domains where variables take on different values over time [Dean and Kanazawa, 1988]. Figure 2 shows the general structure of a DPN. Typically, observations are taken at regular 'time slices,' and a given network structure is replicated for each slice. DPNs model their domains as partially observable Markov processes, so nodes can be connected not only to other nodes within the same time slice but also (and only) to nodes in the immediately preceding or immediately following slice. The Markov property states that:

$$P(State_{t+1}|State_t, State_{t-1}, State_{t-2}, \ldots) = P(State_{t+1}|State_t)$$

In other words, the future is independent of the past given the present. As long as the BAT's representation of the world conforms to this property, the BAT need not maintain the history of its percepts to predict the next state since the accumulated effect of its observations is captured in its current belief state.

The CPTs for the set of arcs proceeding from one time slice to the next form the BAT's state evolution model. This model quantifies how the BAT believes the actual state of the system will evolve over time. The CPTs for the set of arcs proceeding into nodes whose values are typically observed at each time slice form the BAT's sensor model. This model quantifies the likelihood of making a set of observations of the world given the actual state of the world.

Since a given variable may be measured by more than one sensor, the BAT must be able to integrate multiple sensor

[1] Because the belief state should reflect the integration of percepts over time in order to assess such unobservables as driver intentions, approaches such as Pomerleau's ALVINN that are based on feedforward neural networks cannot solve the full driving problem.

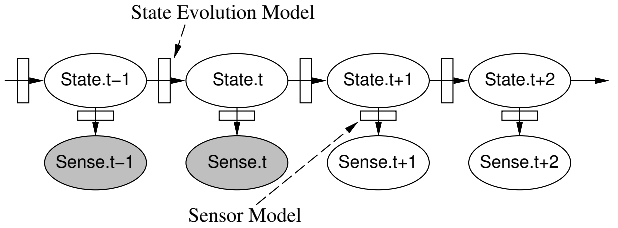


Figure 2: The structure of a dynamic probabilistic network. The ovals denote sets of state nodes or sensor nodes. The arcs going from one slice to the next form the state evolution model, and the arcs going into the sensor nodes form the sensor model. The shaded ovals denote observations available when predicting the state at time $t + 1$.

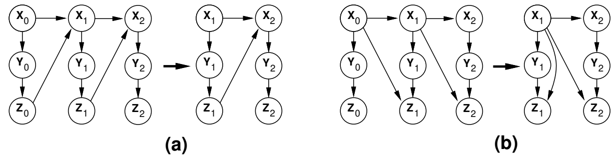


Figure 3: (a) A temporally invariant network: its structure is preserved after rollup. (b) A temporally variant network: its structure is altered after rollup.

readings in computing its estimate of the variable's value. Because sensors are usually conditionally independent of each other given the variable that they measure, Bayesian updating within a probabilistic network easily fuses the readings from several sensors, automatically returning an updated posterior probability distribution for the value of the measured variable.

## 2.2 Efficient Updating

As an agent goes about the world, performing actions, gaining sensor readings and going forward in time, it must efficiently update its model of the world. With DPNs, the agent must go through a constant process of incorporating percepts, adding new time slices at the "leading edge" of the network, and discarding old time slices at the "trailing edge" to avoid a blow-up in the DPN structure.

Because of the Markov property, time slices corresponding to the past can be removed as new slices are added to the network. Before a past slice is removed, its influence must be absorbed into the remaining part of the network by revising probability tables for nodes in the slice immediately following the slice to be removed. By *rolling up* the network in this fashion, evidence accumulated over time is always integrated into the current probabilistic network model. In general, the model for the current time slice along with the current percepts completely determines the current belief state.

Clearly, real-time temporal inference requires efficient rollup. Any exact method for rolling up one slice of a network is equivalent to performing a sequence of *node eliminations* [Kjaerulff, 1992]. Node elimination may introduce additional links into the network; thus the network structure may change as a result of rollup. This complicates maintaining the belief state in three ways. First, different node elimination sequences can result in drastically different connectivity in the resulting networks. Connectivity in turn affects the time needed for inference using a network. Second, modification of the network structure may force some inference algorithms, such as those based on junction trees, to perform expensive computations to modify or completely recreate internal data structures. Third, modification requires more complex and computation-intensive code to manipulate the network.

To address these issues, we have developed two methods for efficient rollup: *temporally invariant networks* and stochastic simulation. We first introduce the temporally invariant network, and then briefly describe our approach to stochastic simulation in DPNs, which is described in more detail in [Kanazawa *et al.*, 1995].

### Temporally Invariant Networks

A temporally invariant network is a DPN for which there exists a node elimination sequence which induces the same structure as the original network. Figure 3(a) shows a temporally invariant network. Although we do not have the space for a full exposition of the node elimination operations involved, it is easy to see that in this network, all information about a time slice is conveyed to a single node, the top node, in the next slice. Thus, intuitively, rollup should affect only this top node and not affect the other parts of the network.

By contrast, Figure 3(b) shows a temporally variant network. In this network, information about each slice is conveyed to the next slice through two nodes. Effectively, this means that those two nodes become dependent on each other, since knowing something about one of them implies something about the past, which in turn implies something about the other. Thus, after rollup, the new leading edge slice contains an arc joining the two nodes.

A temporally invariant network has obvious advantages for real-time temporal inference. Since the rollup operations are known in advance, there is no need to perform a potentially expensive run-time search for an elimination order. This can be done offline, as can open-coding of the CPT computations performed in the rollup.

One thing to notice is that in Figure 3(b), we could have added the arc introduced by rollup to every slice from the beginning. Then the resulting network would have been temporally invariant. Thus, we can take a temporally variant network and convert it into a temporally invariant one. There is a cost associated with this conversion. While the example only involves the addition of one arc, this arc also makes every slice completely connected. In fact, it is easy to show that a network with completely connected nodes in each slice is always temporally invariant. However, a network with completely connected subcomponents may be very expensive to represent and to compute with, and it lacks many of the advantages of using probabilistic networks in the first place.

For a given temporally variant network, finding an optimal conversion that adds a minimal cost set of edges to create a temporally invariant network is likely to be an expensive operation. Nevertheless, by precompiling it into a temporally invariant network, and by devoting significant resources to finding a good node elimination sequence, we may ease the run-time computation requirements of the network.

**Stochastic Simulation in DPNs**

In much of our past work, we have taken advantage of a commercial implementation of the exact clique tree algorithm (the HUGIN system). In our applications, we have found that the clique tree algorithm is too expensive and that exact probabilities are not needed. Furthermore, DPNs seldom conform to the structural requirements required for the clique tree to handle continuous variables. We have therefore investigated the use of stochastic simulation algorithms, which often provide fast approximations to the required probabilities and can be used with arbitrary combinations of discrete and continuous distributions. Even more importantly, the use of stochastic simulation makes unnecessary the expensive CPT manipulations involved in exact rollup.

In the context of DPNs, stochastic simulation methods attempt to approximate the current belief state using a collection of samples representing "simulated realities," each describing one possible evolution of the environment. Because the samples are a complete representation of our estimate of the joint distribution, there is no need to recompute CPTs. The sample population and associated weighting factors integrate and reflect all available evidence, and they are all we need to form the estimate of any marginal or joint probability in a DPN.

The simplest simulation algorithm is *logic sampling* [Henrion, 1988]. Logic sampling stochastically instantiates the network, beginning with the root nodes and using the appropriate conditional distributions to extend the instantiation through the network. Because logic sampling discards trials whenever a variable instantiation conflicts with observed evidence, it is ineffective for DPN-based monitoring where evidence is observed throughout the temporal sequence.

*Likelihood weighting* [Shachter and Peot, 1989] attempts to overcome the general problem with logic sampling. Rather than discarding trials that conflict with evidence, each trial is *weighted* by the probability it assigns to the observed evidence. Probabilities on variables of interest can then be calculated by taking a weighted average of the values generated in the population of trials. It can be shown that likelihood weighting produces an unbiased estimate of the required probabilities.

The use of likelihood weighting in DPNs reveals some problems that require special treatment. The difficulty is that a straightforward application generates simulations that simply ignore the observed evidence and therefore become increasingly irrelevant. Consider a simple example: tracking a moving dot on a 2-D surface. Suppose that the state evolution model is fairly weak—for example, it models the motion as a random walk—but that the sensor is fairly accurate with a very small Gaussian error. Figure 4 illustrates the difficulty. The samples are evolved according to the state evolution model, spreading out randomly over the surface, whereas the object moves along some particular trajectory that is unrelated to the sample distribution. The weighting process will assign extremely low weights to almost all of the samples because they disagree with the sensor observations. The estimated distribution will be dominated by a very small number of samples that are closest to the true state, so the effective number of samples diminishes rapidly over time. This results in large estimation errors. All this occurs despite the fact that *the sensors can track the object with almost no error*! In the case of traffic monitoring, we have discovered that a naive application of likelihood weighting results in a sample population of more or less imaginary traffic scenes that bear no relation to what is actually happening on the road.

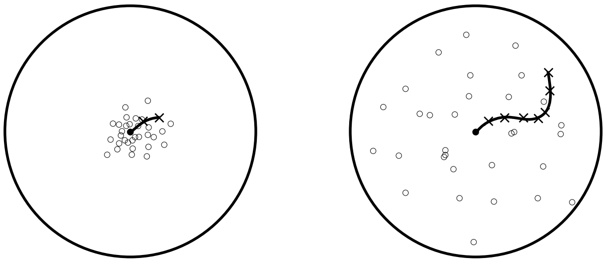

Figure 4: A simple 2-D monitoring problem. An object starts in the centre of the disc and follows the path shown by the solid line. Sensor observations are shown by crosses. The small circles show a snapshot of the population of samples generated by a naive application of likelihood weighting. Snapshots for $t = 2$ and $t = 7$ are shown.

We have developed two methods that use the current sensor values to reposition the sample population closer to reality rather than allowing it to evolve as if no sensor values were available. The problems with likelihood weighting typically arise in situations (such as often happens with DPNs) where nodes representing observed evidence have parents. *Evidence reversal* is a method that restructures each time slice of a DPN so that evidence nodes have no unobserved parents. This method ensures that the sample population remains close to reality when extending it using the current evidence. *Survival of the fittest sampling* is a method that uses the likelihood weights to preferentially propagate the most likely samples. This is related to genetic algorithms (except that there is no crossover). Our experimental results confirm that our methods perform better than likelihood weighting in DPN-monitoring applications. The best results are obtained by combining both methods [Kanazawa *et al.*, 1995].

## 2.3 Network structure

As implemented, the BAT monitors each vehicle tracked by the sensor system with a separate DPN. Each network contains nodes for sensor observations, such as vehicle position and velocity, as well as nodes for predicting driver intentions, such as whether the driver intends to make a lane change or to slow down.

Like a Kalman filter, each network computes probability distributions for a vehicle's position and velocity based on both its latest observations and its previous state estimate (which reflects the influence of all previously observed evidence). Unlike a Kalman filter, which is limited to Gaussian distributions, the network predictions can be arbitrarily distributed. For example, if a vehicle were approaching some debris directly in front of it, the network could predict that the vehicle would move either to the right or to the left (but not straight) in order to avoid the debris. Also, the network could easily incorporate additional sensor information. If the sensor system recognized that a vehicle was flashing its right turn signal, the network could make predictions that biased the vehicle's position towards the right.

An alternate, perhaps preferable, approach to vehicle monitoring would utilize one large scene network for all relevant ve-

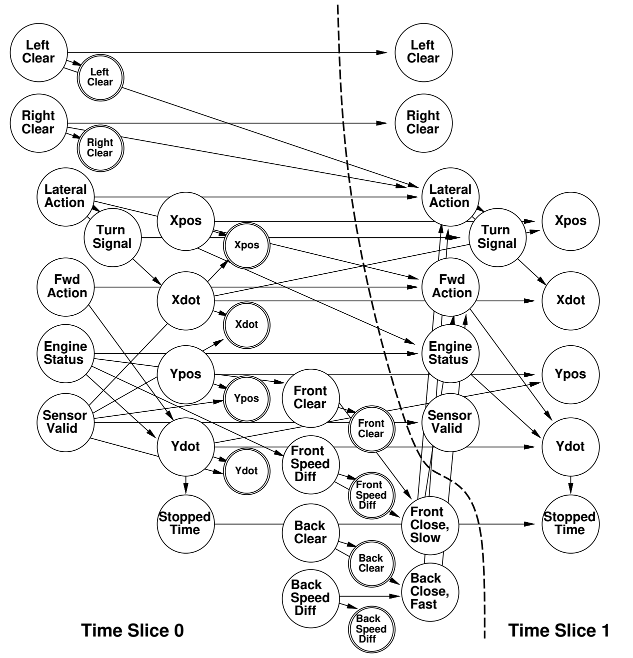


Figure 5: Dynamic probabilistic network for one vehicle, including inter-slice arcs. The smaller nodes with thicker outlines denote sensor observations.

hicles. Because this greatly increases the computational complexity of inference operations and requires run-time modifications to the network structure, we chose to use separate networks for each vehicle. To incorporate the influence of nearby vehicles, each network contains nodes corresponding to those vehicles. For example, the Front Clear and Front Speed Diff nodes in Figure 5 refer to "the space between this vehicle and the vehicle in front," and "the speed difference between this vehicle and the vehicle in front," respectively. Since the vehicle in front of or behind a given vehicle may change, these *indexical* nodes do not correspond to a specific vehicle. Instead, a preprocessing step using sensor data determines the spatial relationships among the vehicles and then sets the node states accordingly. Figure 5 shows an example vehicle network for one time slice, along with the inter-slice links to the next time slice.

### 2.4 Sensor models and sensor failure

Since adverse weather, road conditions, and extended use may cause the BAT's sensor systems to degrade or fail, it must be able to dynamically estimate the reliability and accuracy of its sensors. By quantifying a sensor's expected performance under various operating conditions, we have been able to construct belief networks for sensor validation.

Figure 6 shows an example of a network for monitoring the state of a single sensor (such as a video camera) based on observations for two cars. The Sensor Status node at each time slice specifies whether the sensor is performing well, performing at a degraded level, or completely inoperative. The CPT for each observation node specifies a distribution for the observation of a variable given its actual value and given the operating mode of the sensor. As the variation between predicted values and observed values increases, the sensor

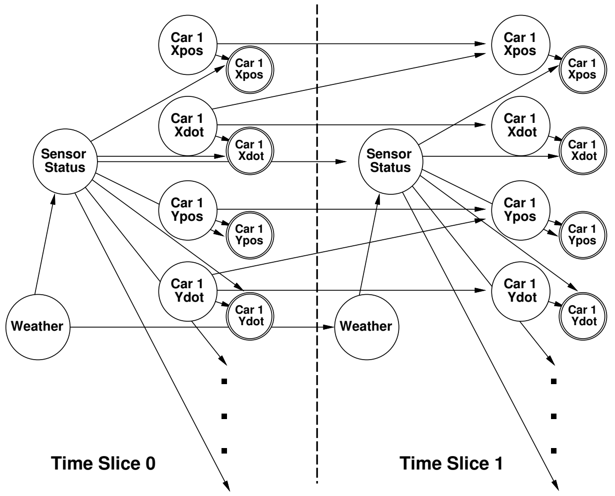


Figure 6: An example sensor validation network. Sensor readings are entered at observation nodes, the smaller nodes with thicker outlines. The Sensor Status node can take on one of several different values to indicate whether the sensor is broken or is operating in some degraded mode. Increased variance between predicted values and observed values may increase the belief that the sensor is performing less reliably, perhaps due to adverse weather conditions.

validation network will correspondingly change its estimate of the sensor's reliability and accuracy, perhaps attributing it to adverse weather conditions. Very high variance between observed and predicted values or no observed values at all may increase the belief that the sensor has failed completely. Finally, the link going from the sensor status node in one slice to its instantiation in the next slice allows quantification of the persistence of the various sensor operating modes.

## 3 Decision Making in the BAT

In this section, we describe our approach to real-time BAT decision making. As we noted earlier, the decision problem corresponds to a POMDP. Computing the policy for POMDPs is PSPACE-complete, and exact solutions can be obtained only for tiny state spaces. Our approach to real-time BAT decision making is to find approximate solutions. We are undertaking three separate approaches. They are (1) bounded lookahead using dynamic decision networks, which incorporate action nodes and an explicit utility function; (2) hand-coded, explicit policy representations, such as decision trees, that take as input the joint probability distribution encoded in the DPN; and (3) supervised learning and reinforcement learning methods for solving the POMDP, in which we learn a policy representation, a utility function on belief states, or an action-value function on belief-state/action pairs.

### 3.1 Dynamic decision networks

Decision networks (or influence diagrams) extend probabilistic networks by including distinguished node types for actions and utility functions. They are solved to obtain optimal decisions by maximizing the utility function over the possible instantiations of the action nodes, given the available evidence. Dynamic decision networks (DDNs) resemble DPNs, but have an action node for each time slice. If the utility function is time-separable (that is, the utility of a sequence of states can be computed by combining separate rewards for each state in the sequence), then the DDN can include a reward node with each slice. For example, the reward function might include positive components for progress made towards the destination and negative components for jerky motions, illegal actions, crashes, etc. A DDN can represent a finite-horizon decision problem by projecting forward the appropriate number of time steps, or can approximate an infinite-horizon problem (such as driving) by projecting up to an artificial horizon and then using an approximate utility function on the final state to estimate the expected reward for the rest of time. In this respect, DDNs are similar to algorithms used in game-playing. In Figure 7, we show a generic DDN with a three-step horizon. The sensor model in a DDN is similar to that in a DPN; the state evolution model, on the other hand, now includes the action node as a parent, and therefore represents the effects of the agent's actions. Note also that we do not show the "informational links" that are used in influence diagrams to show the evidence available to the agent when each decision is made. Instead, we enforce the convention that a decision at time *t* is made with all evidence gained up to and including time *t*, and we identify a particular set of the chance nodes as evidence nodes that will be instantiated at each time step.

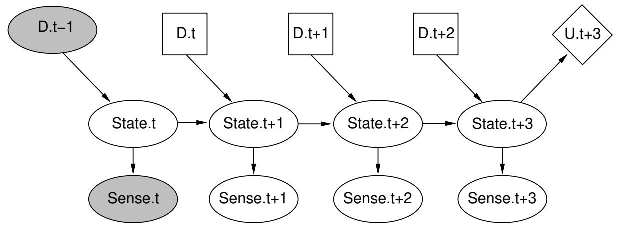


Figure 7: A generic dynamic decision network (DDN).

Tatman and Shachter [Tatman and Shachter, 1990] provide an algorithm for computing the policy for a DDN. In the case of driving, which is a partially observable problem, only a subset of the variables in a given time slice will be instantiated as evidence. This means that as well as maximizing over action sequences, the algorithm must average over all possible percept sequences as well. Although this enables generation of intelligent policies such as "moving over a bit to look around the car in front," it is expensive. It is possible to avoid generating an explicit representation of the policy (which resembles an enormous "conditional plan"), but the time complexity still makes the process infeasible for a real-time agent using current hardware. Using metalevel control and adjusting the horizon can mitigate this to some extent, and some success has been achieved in simple environments. Currently, however, we envisage using DDN lookahead mainly for offline generation of optimal actions in large sets of training examples. The belief-state/action pairs can then be generalized using inductive methods to provide an efficiently executable policy representation.

### 3.2 Decision tree policy representation

Our second approach combines the DPN state evolution model with a decision tree. The decision tree is a tree of binary if-then-else constructs where the test predicates are computed from the joint distribution computed by the DPN. Each leaf of the tree is a decision. This obviously yields an effective, real-time policy, but constructing the decision tree is a difficult task. Other researchers, for example Lehner and Sadigh [Lehner and Sadigh, 1993], have examined the creation of decision trees from influence diagrams, but only for static problems. In such cases, the decision tree nodes test fully determined evidence variables. If this method is applied to dynamic problems, one may be forced to test the entire percept sequence.

Our approach involves testing the current belief state instead. Although this is potentially much smaller, optimality in a POMDP requires that the tests define regions in the joint probability space rather than regions in the marginal probability space for each variable. We have found this extremely unintuitive, and so have used tests on marginals of individual variables as an approximation. To date, this has been reasonably effective. We have implemented several hand-constructed decision trees, which have the following general structure (each "predicate" here is actually a complex set of probability thresholds on specific variables, and each "action" a subsidiary decision tree):

```
if changing lane
    if safe to continue
        Continue lane change
    else
        Abort lane change
else if not in target lane and can change to target lane
    Initiate lane change to target lane
else if vehicle in front
    Maintain safe following or Pass
else
    Maintain target speed
```

Figure 8: General decision-tree structure.

Experimental results with our current decision tree are described later on.

### 3.3 POMDP Policy Learning

The last approach to decision making takes advantage of an observation we made earlier, namely, that the POMDP policy is a function from the current belief state to the optimal action, and that the current belief state is encoded in the DPN. In the preceding section, we proposed decision trees over the DPN distribution as one possible function to approximate the POMDP policy. Here, we briefly describe how to learn the optimal value function for each action directly.

Assume that the world is represented as a temporally invariant DPN. In the DPN, the belief state is determined by the CPTs, evidence values, and topology of the current slice. However, since the topology and some of the CPTs are fixed due to temporal invariance, we need pay attention only to the time-varying CPTs and the evidence when learning the action-value function. The time-varying CPT entries are identified during the compilation of the temporally invariant network. In this way, we can project down the complete joint probability space onto a much smaller subspace in which to learn the action-value function. Since the decision problem is Markov in this subspace, we can use standard reinforcement learning techniques such as Q-learning to learn a generalized action-value representation. The details of this technique are described in a forthcoming paper.

## 4 Scenarios and Results

In this section, we describe the result of testing the decision-tree-based BAT controller on several sample traffic scenarios. We have built a working simulator to test various decision making modules for the BAT. For each test, the simulator reads a scenario description file, which describes the volume of traffic and the behaviors of other vehicles traveling along the highway. At each simulator "clock tick", the simulator determines the trajectories of all the vehicles until the next tick; it passes current state information in the form of sensor readings (adding noise as necessary using sensor models) to each vehicle's controller, which in turn outputs its decision for the current time step. The simulator uses the vehicle's decision and a physical model to plot trajectories and to detect collisions and other significant events.

We have predefined a set of controllers for vehicles other than the BAT (called "drones") that engage in a variety of behaviors simulating good drivers, antisocial, incompetent, and unsafe drivers, stalled vehicles, and so on. These controllers are configurable in terms of when and how often they undertake such behaviors, their speed, and so on; furthermore, the system is easily extensible to add more types of driving behavior.

In the results described here, the system is limited to having only one BAT at a time. This is due largely to the current inference architecture, which uses the HUGIN belief network system. Although our networks are not very large (the total number of nodes is on the order of 300), they are highly connected, which slows down the HUGIN system. A newer, implemented system uses stochastic simulation which is considerably faster for our problem, and we plan to have multiple BATs driving at the same time.

The goal of the BAT controller is to maintain a target speed in a target lane. When other vehicles interfere, the controller makes appropriate acceleration/deceleration and lane-changing maneuvers. We show five such situations: passing a slow-moving vehicle (Figure 9), reacting to unsafe drivers (Figure 10), avoiding a stalled car (Figure 11), aborting a lane change maneuver (Figure 12), and merging into another lane (Figure 13). We show the situations as discrete sequences of 2-D pictures, although of course they are actually continuous 3-D video sequences. In the figures, the BAT is the shaded vehicle.

## 5 Summary and future work

We have described the overall structure and early theoretical and practical results of a long-term project on intelligent vehicles. In a short paper we cannot do full justice to either type of result, but we hope to have given something of the flavour of the AI problems involved and their solutions. Specific contributions include the following:

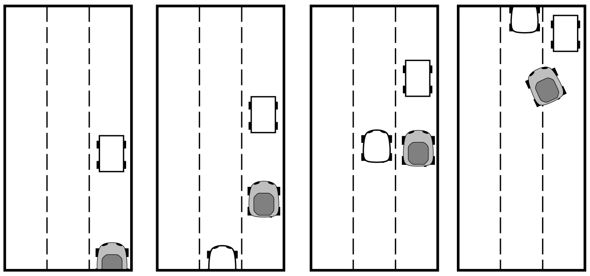

Figure 9: Passing a slower car: This scenario demonstrates the BAT's ability to pass slow cars. As the BAT approaches a slower vehicle, it decides to pass to the left so that it can maintain its target speed. Because of another vehicle in that lane, the BAT first maintains a safe following distance behind the slower car until the left lane is clear and then performs a left lane change maneuver and accelerates back to its target speed.

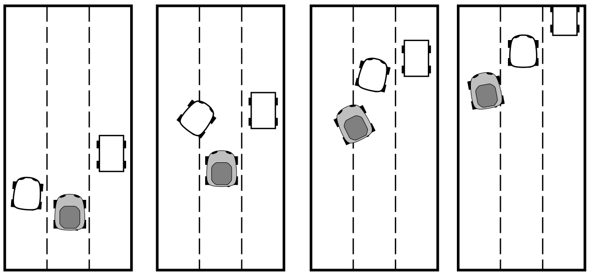

Figure 10: Reacting to unsafe drivers: This scenario shows the BAT's ability to deal with aggressive and unsafe drivers. A car cuts in front of the BAT and proceeds to slow down. When the probability of the other car making a lane change passes a threshold, the BAT initiates a defensive lane change even before the other car is fully in the BAT's lane. The BAT first slows down to avoid the car and then accelerates back to its target speed.

- The use of dynamic probabilistic networks (DPNs) to solve the problems of noise and partial observability that arise in driving.
- The decomposition of the overall DPN into separate vehicle networks linked by indexical variables in order to improve performance.
- Automatic diagnosis and accommodation of sensor degradation and failure.
- The use of temporally invariant networks and stochastic simulation to achieve real-time updating.
- The use of decision trees based on current belief state, and their successful application in a variety of difficult driving scenarios.

Clearly, much remains to be done. We are currently investigating the application of new learning algorithms to construct DPNs automatically from time series data [Russell *et al.*, 1995]. Combined with our computer vision subsystem, this will allow the construction of human driver models from videotapes. In our network representation, we are working to incorporate continuous variables, thereby avoiding the combinatorial explosion caused by discretization. One topic requiring significant empirical research is the design of appropriate utility functions. Even if the assumptions of time-separability

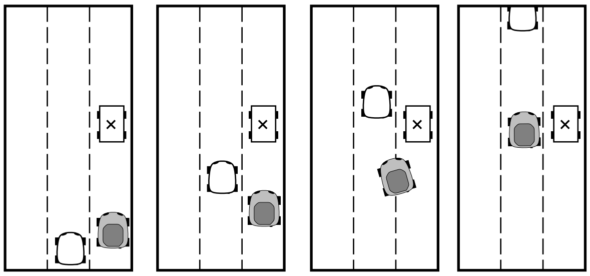

Figure 11: Avoiding a stalled car: This scenario demonstrates the BAT's ability to avoid a stalled car. The BAT detects a stalled car in front based on the difference in their forward velocities. The BAT realizes that the car is stalled, rather than stuck in traffic, because there are no cars in front of it. However, the lane to the left of the BAT is not clear, so the BAT slows down. Once it determines that the left lane is clear, it initiates a lane change and begins accelerating up to its target speed.

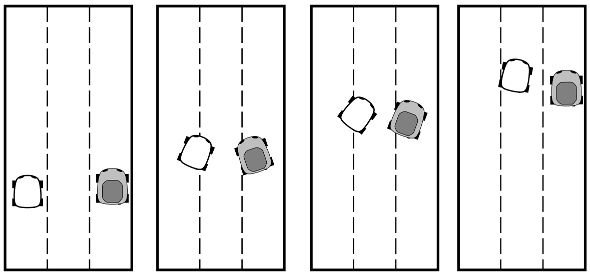

Figure 12: Aborting a lane change: This scenario describes an aborted lane change. Initially, the BAT is changing lanes to the left lane, but at the same time a car two lanes over tries to change lanes to the right. When the BAT detects this by thresholding the probability that the other car is changing lanes, it decides to abort its own lane change by slowing down and going back to its original lane.

and additivity are appropriate, we still need to assess relative weights of the various components. Such research can of course be done independently of any specific design for a vehicle controller and will be of use in all such projects. Finally, one of the most important topics in intelligent vehicle design is the question of validation—demonstrating that the vehicle will be sufficiently safe for deployment in the real world. As well as improving and assessing the verisimilitude of our simulation, we need to gather a vast library of real-world video footage to test the BAT's response to real situations.

## References


[Dean and Kanazawa, 1988] T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *Proceedings AAAI-88*, pp. 524–528. Minneapolis.

[Dickmanns and Zapp, 1987] E. D. Dickmanns and A. Zapp. Autonomous high speed road vehicle guidance by computer vision. In R. Isermann, editor, *Automatic Control—World Congress, 1987*, pp. 221–226, Munich.

[Henrion, 1988] M. Henrion. Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In *Un-


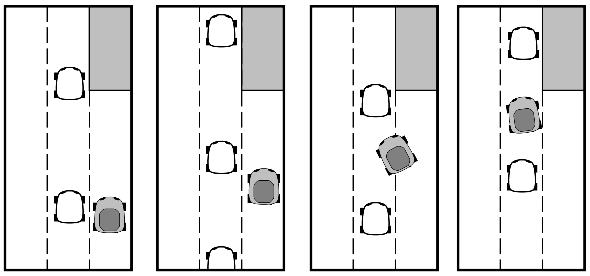

Figure 13: Merging into traffic: This scenario shows the BAT entering the flow of traffic as its own lane ends. It first slows down and waits for an opening in the traffic before accelerating and changes lanes to the left.


*certainty in Artificial Intelligence 2*, pp. 149–163. Elsevier, Amsterdam.

[Huang *et al.*, 1994] T. Huang, D. Koller, J. Malik, G. Ogasawara, B. Rao, S. Russell, and J. Weber. Automatic symbolic traffic scene analysis using belief networks. In *Proceedings AAAI-94*. Seattle.

[Kanazawa *et al.*, 1995] K. Kanazawa, D. Koller, and S. J. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. Submitted for publication.

[Kjaerulff, 1992] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings UAI-92*, pp. 121–129. Stanford.

[Lehner and Sadigh, 1993] P. Lehner and A. Sadigh. Two procedures for compiling influence diagrams. In *Proceedings UAI-93*. Washington, D. C..

[Niehaus and Stengel, 1991] A. Niehaus and R. F. Stengel. Rule-based guidance for vehicle highway driving in the presence of uncertainty. In *Proceedings of ACC*, volume 3, pp. 3119–24. Boston.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California.

[Pomerleau, 1993] D. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. Kluwer, Dordrecht, The Netherlands.

[Russell *et al.*, 1995] S. J. Russell, J. Binder, D. Koller, and K. Kanazawa Adaptive probabilistic networks. To appear, IJCAI-95.

[Shachter and Peot, 1989] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proceedings UAI-89*. Windsor, Ontario.

[Tatman and Shachter, 1990] J. A. Tatman and R. D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):365–379.