Persistence and Probabilistic Projection

THOMAS DEAN AND KEIJI KANAZAWA

Abstract—**Predicting the future is an essential component of decision-making. In most situations, however, there is not enough information to make accurate predictions. A theory of causal reasoning for predictive inference under uncertainty is developed. A common type of prediction that involves reasoning about persistence is emphasized: whether or not a proposition once made true remains true at some later time. A decision procedure with a polynomial time algorithm for determining the probability of the possible consequences of a set events and initial conditions is provided. The integration of simple probability theory with temporal projection circumvents problems in dealing with persistence by nonmonotonic temporal reasoning schemes. These ideas have been implemented in a prototype system that refines a database of causal rules in the course of applying those rules to construct and carry out plans in a manufacturing domain.**

I. Introduction

WE ARE interested in the design of robust inference systems for generating and executing plans in routine manufacturing situations. We hope to build autonomous agents capable of dealing with a fairly circumscribed set of possibilities in a manner that demonstrates both strategic reasoning (the ability to anticipate and plan for possible futures) and adaptive reasoning (the ability to recognize and react to unanticipated conditions). In this paper, we develop a computational theory for temporal reasoning under uncertainty that is well suited to a wide variety of dynamic domains.

The domains that we are interested in have the following characteristics: 1) things cannot always be predicted accurately in advance, 2) plans made in anticipation of pending events often have to be amended to suit new information, and 3) the knowledge and ability to acquire predictive rules is severely limited by the planner's experience. Reasoning in such domains often involves making choices quickly on the basis of incomplete information. Although predictions can be inaccurate, it is often worthwhile for a planner to attempt to predict what conditions are likely to be true in the future and generate plans to deal with them.

Our theory concerns the development of efficient mechanisms to support this type of reasoning. It includes a polynomial-time decision procedure for probabilistic inference about temporally dependent information, a space and time efficient method for refining probabilistic causal rules, and a mechanism to support planners in recognizing potential plan failures. Simple decision theoretic [1], [2] methods are employed in the course of generating plans.

Manuscript received March 3, 1988; revised August 10, 1988. This paper was partially presented at the Canadian Conference on Artificial Intelligence, Edmonton, AB, Canada, June 1988. Shorter versions of this paper were presented at CSCSI-88 and at the 1988 Workshop for Uncertainty in Artificial Intelligence. This work was supported in part by the National Science Foundation under grant IRI-8612644 and in part by an IBM faculty development award.

The authors are with the Department of Computer Science, Brown University, P.O. Box 1910, Providence, RI 02912.

IEEE Log Number 8927051.

II. Probabilistic Causal Theories

To explore some of the issues that arise in causal reasoning, we will consider some examples involving a robot foreman that directs activity in a factory. The robot has a plan of action that it is continually executing and revising. Among its tasks is the loading of trucks for clients. If our robot learns that a truck is more likely to leave than it previously believed, then it should consider revising its plans so that this truck will be loaded earlier. If, on the other hand, it predicts that all trucks will be loaded ahead of schedule, then it should take advantage of the opportunity to take care of other tasks that it did not previously consider possible in the available time.

To construct and revise its plan of action, the robot makes use of a fairly simple model of the world: a special-purpose theory about the cause-and-effect relationships that govern processes at work in the world (referred to as a *causal theory*). The robot's causal theory consists of two distinct types of rules, which we will refer to as *projection rules* and *persistence rules*. We will defer discussion of persistence rules for just a bit.

As an example of a projection rule, the robot might have a rule that states that if a client calls in an order, then, with some likelihood, the client's truck will eventually arrive to pick up the order. The consequent prediction, in this case the arrival of a client's truck, is conditioned on two things: an event referred to as the *triggering event*, in this case the client calling in the order, and an enabling condition corresponding to propositions that must be true at the time the triggering event occurs. For example, the rule just mentioned might be conditioned on propositions about the type of items ordered, whether or not the caller has an account with the retailer, or the time of day. The simplest form of a projection rule is PROJECT$(P_1 \wedge P_2 \cdots \wedge P_n, E, R, \kappa)$. This says that R will be true, with probability κ immediately following the event E, given that P_1 through P_n are true at the time E occurs. Restated as a conditional probability, this would be

$$p(\langle R, t+\epsilon\rangle | \langle P_1 \wedge P_2 \cdots \wedge P_n, t\rangle \wedge (\langle E, t\rangle) = \kappa. \quad (1)$$

0018-9472/89/0500-0574$01.00

For simplicity, in the present work we assume that $P_1 - P_n$ are independent. In [13], we outline methods by which restriction can be removed. Projection rules are applied in a purely antecedent fashion (as in a production system) by the inference engine we will be discussing. The objective is to obtain an accurate picture of the future in order to support reasoning about plans [3].

Our approach, as described up to this point, is fairly traditional and might conceivably be handled by some existing approach [4], [5]. What distinguishes our approach from that of other probabilistic reasoning approaches is that we are very much concerned with the role of time and, in particular, the tendency of certain propositions (often referred to as *fluents* [6]) to change with the passage of time. By adding time as a parameter to our causal rules, we have complicated both the inference task and the knowledge acquisition task. Complications notwithstanding, the capability to reason about change in an uncertain environment remains an important prerequisite to robust performance in most domains. We simply have to be careful to circumscribe a useful and yet tractable set of operations. In our case, we have allowed the computational complexity of the reasoning tasks and the availability and ease of acquisition of the data to dictate the limitations of our inference mechanism.

Our inference system needs to deal with the imprecision of most temporal information. Even if a robot is able to consult a clock in order to verify the exact time of occurrence of an observed event, most information the robot is given is imprecise (e.g., a client states that a truck will pick up an order at *around* noon, or a delivery is scheduled to arrive *sometime* in the next 20 minutes). One of the most important sources of uncertainty involves predicting how long a condition lasts once it becomes true (i.e., how long an observed or predicted condition is likely to *persist*). In most planning systems (e.g., [7]) there is a single (often implicit) default rule of persistence [8] that corresponds more or less to the intuition that a proposition once made true will remain so until something makes it false. The problem with using this rule is that it is necessary to predict a contravening proposition in order to get rid of a lingering or persistent proposition, a feat that often proves difficult in nontrivial domains. If a commuter leaves his newspaper on a train, it is not difficult to predict that the paper is not likely to be there the next time he rides on that train; however, it is quite unlikely that he will be able to predict what caused it to be removed or when the removal occurred.

When McDermott first proposed the notion of persistence as a framework for reasoning about change [9], he noted that persistence might be given a probabilistic interpretation. That is exactly what we do here. We replace the single default rule of persistence used in most planning systems with a set of (probabilistic) rules: one or more for each fluent that the system is aware of. Our robot might use a persistence rule to reason about the likelihood that a truck driver will still be waiting at various times following his arrival at the factory. The information derived from applying such a rule might be used to decide which truck to help next or how to cope when a large number of trucks are waiting simultaneously. Each persistence rule has the form PERSIST(P, g), where P is a fluent and g is a function of time referred to as a *survivor* function [10]. In our implementation, we consider only two types of survivor functions: exponential decay functions and piecewise linear functions. Piecewise linear functions are described in Appendix II. Exponential decay functions are of the form $e^{-\lambda t}$, where λ is the constant of decay. Persistence rules referring to exponential decay functions are simply notated PERSIST(P, λ). Such functions are used, for example, to indicate that the probability of a truck remaining at the dock decreases by 5 percent every 15 minutes. The persistence rule PERSIST(P, λ) encodes the fact that

$$p(\langle P, t\rangle | \langle P, t - \Delta\rangle) = e^{-\lambda(t-\Delta)}$$

where Δ is a positive number indicating the length of an interval of time. Exponential decay functions are insensitive to changes in the time of occurrence of events that cause such propositions to become true, and hence are easy to handle efficiently.

There are a number of issues that every computational approach to reasoning about causality must deal with. One such issue involves reasoning about dependent causes [4] (e.g., the application of two probabilistic causal rules that have the same consequent effects, both of which appear to apply in a given set of circumstances but whose conditions are correlated). Another issue concerns handling other forms of incompleteness and nonmonotonic inference [11], [12] (e.g., the robot might have a general rule for reasoning about the patience (persistence) of truck drivers waiting to be served and a special rule for how they behave right around lunch time or late in the day). While we agree that these problems are important, we do not claim to have any startling new insights into their solution. There is one area, however, in which our theory does offer some new insights, and that concerns the form of probability functions used in causal rules, how they are acquired and modified in the course of assimilating new information, and finally how the system uses these functions to support reasoning about plans. The next three sections treat each of these issues in turn.

III. Probabilistic Projection

In this section, we will try to provide some intuition concerning the process of reasoning about persistence, which we will refer to as *probabilistic projection*. A planner is assumed to maintain a picture of the world changing over time as a consequence of observed and predicted events. This picture is formed by extrapolating from certain observed events (referred to as *basic facts*) on the basis of rules believed to govern objects and agents in a particular domain. These governing rules are collectively referred to as a *causal theory*.

Fig. 1 depicts a sample causal theory. Predicates (ATDOCK), and constants (TRUCK14) are in upper case,

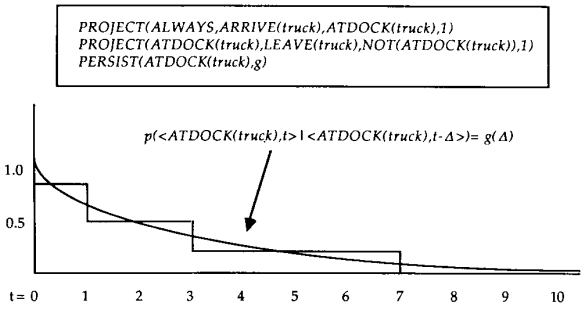

Fig. 1. Simple causal theory illustrating use of survivor functions.

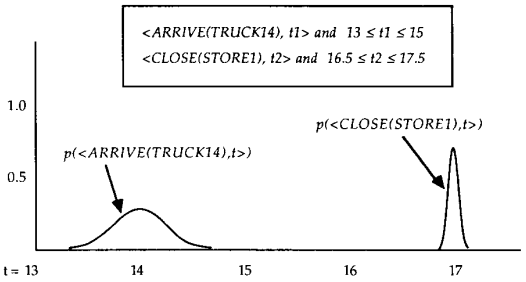

Fig. 2. Set of basic facts and their probabilistic interpretation.

while functions (p, g) and variables (t,truck) are in lower case. We refer to an instance of a fact (type) being true over some interval of time as a *time token*, or simply *token*. For example, ARRIVE(TRUCK14) denotes a general type of event whereas $\langle$ARRIVE(TRUCK14), $t\rangle$ denotes a particular instance of ARRIVE(TRUCK14) becoming true. The predicate ALWAYS is timelessly true (i.e., for all $t\langle$ALWAYS, $t\rangle$). The function g, a survivor function, describes how certain types of propositions are likely to persist in lieu of further supporting or contravening information.

Fig. 2 shows a set of basic facts corresponding to two events assumed in our example to occur with probability 1.0 within the indicated intervals. The system assumes that there is a distribution describing the probability of each event occurring at various times, and uses some default distribution if no distribution is provided.

Evidence concerned with the occurrence of events and the persistence of propositions is combined to obtain a probability function π for a proposition Q being true at various times in the future by convolving the density function f for an appropriate triggering event with the survivor function associated with Q:

$$\pi(t) = \int_{-\infty}^{t} f(z)e^{-\lambda(t-z)}\,dz. \tag{2}$$

We can explain this equation as follows: for Q to hold at time t, it must have become true at some time $z \leqslant t$, and it must not have become false in the interim. The degree to which the probability of Q being true has decayed by t is given by multiplying the probability that Q was true at z, $f(z)$, with the decay parameter given by $e^{-\lambda(t-z)}$. Summing this product over all points of time less than t provides us with the desired probability.[1]

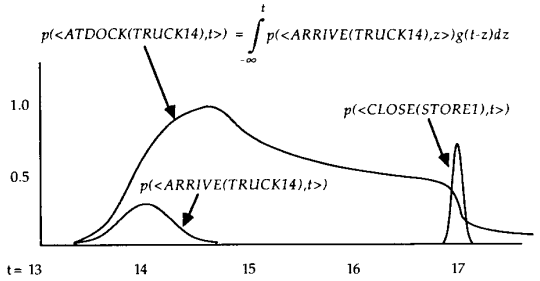

Fig. 3. Example of simple probabilistic inference about persistence.

Fig. 3 illustrates a simple instance of this kind of inference. Note that the range of the resulting probability function is restricted; after the point in time labeled 17, the persistence of ATDOCK(TRUCK14) is said to be *clipped*, and thereafter its probability is represented by another function not shown.

All probability computations are performed incrementally in our system. Each token has associated with it a vector which is referred to as its *expectation vector* that records the expected probability that the proposition corresponding to the token's type will be true at various times in the future.

The system updates the expectation vectors every time new propositions are added to the database, and also at regular intervals as time passes. In the update, a single pass sweep forward in time is made through the database. There is, according to the domain and granularity of data, a fixed *time step*, or a quantum by which we partition time. Starting at the "present time," we compute for each proposition its expected probability for the time step according to the causal theory governing that type of proposition, and record it in the expectation vector. We compute the probability for all propositions, before moving on to the next time step. The process is repeated for some finite number of time steps.

For event causation, the update is straightforward; in the simplest cases, it is just a table lookup and copying of the density function into the vector. For the convolution, it is necessary to take steps to avoid computing the convolution integral afresh at each time step. We compute the convolution as a Riemann sum, successively summing over the time axis with a mesh of fixed size (the time step). By using the exponential decay form of survivor functions, it is possible to compute the convolution at a time step by looking only at the value for the last time step, independent of the time at which the proposition of interest

[1]A proposition Q is an instance of some fact type; hence the distribution f never sums to more than 1.

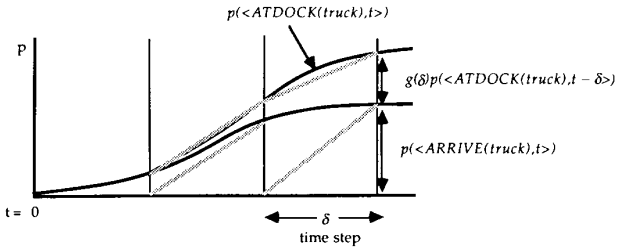

Fig. 4. Computing convolution integral incrementally.

became true. All that is required is to multiply the last value by the constant decay rate, and add it to any contribution from the causal distribution for that time step. The process is illustrated graphically in Fig. 4.

There are many details concerned with indexing and applying projection rules that will not be mentioned in this paper (see [8]). The details of probabilistic projection are described in Appendix I of this paper for those interested. Our update algorithm is polynomial in the product of the number of causal rules, the size of the set of basic facts, and the size of the mesh used in approximating the integrals. For many practical situations, performance is closer to linear in the size of the set of basic facts.

The convolution equation can be easily extended to handle the case of clipping. We add to (2) a term, the function g, corresponding to the distribution of an event that contradicts the proposition Q:

$$\pi(t) = \int_{-\infty}^{t} f(z)e^{-\lambda(t-z)}\left[1 - \int_{z}^{t} g(w)\,dw\right]dz. \quad (3)$$

The cumulative distribution of g defines the degree to which it becomes unlikely that the proposition remains true. It is easy to interpret this equation in terms of (2): the quantity to integrate, the product of the likelihood that Q occurred at time z, and the amount of decay between z and t is modified now by the degree to which we expect Q to have become false during that time. The latter is given by the integral of the disabling distribution g between z and t. What we are interested in is the probability that Q remains true; therefore we multiply the product term by 1 minus the probability that Q has become false. We see that under certain conditions, (3) describes exactly what we desire. Unfortunately, there will be a tendency for the exponential decay term and g to count the same effects twice. We describe in [13] methods by which this, and other issues about dependent causes might be handled in a different probabilistic framework.

IV. Acquiring Probabilistic Causal Theories

Ideally, the component of a planner corresponding to our system should engage in a cycle of activity: not merely to predict but to predict and observe, and modify its predictions in the future according to its observations. Our system does not depend on *a priori* data in order to make its predictions. It is capable of routine acquisition and update of its probability data in the course of its everyday operation.

The general problem of generating causal rules about the world from experience is very difficult. However, it is straightforward to acquire default persistence rules of the kind which our system uses for probabilistic projection. To acquire persistence rules, we need to collect data about the duration of time tokens. From information that a particular state has become true, the system must periodically execute actions to determine whether or not it remains true. Duration data for each type of fact is gathered and used to construct a survivor function for that type.

To illustrate, suppose that the system has seen n instances of a given proposition P, and that the system has recorded how long each instance lasted. If the shortest instance lasted for k minutes, then the expectation for P given a new instance is 1.0 up to k minutes, and then it drops by $1/n$. Sweeping forward along the time axis, the expectation drops by $1/n$ for each instance, until it drops to 0.0 after the longest instance. In our implementation, we assume that the system already knows the general form of survivor function for a given fact type and that all it has to learn is the parameters that determine the type's specific persistence behavior (see Appendix III for details concerning the update algorithms).

There are several obvious opportunities in the process of assimilating new data to refine survivor functions. If the frequency information can be clustered and then partitioned so as to indicate two or more distinct survivor functions, then it is quite likely that the persistence rule refers to a class of propositions. Since as a matter of course it would be prohibitive to retain enough information to actually support the necessary distinctions once the need is detected, all this really tells the planner is that it should explicitly plan to gather additional information in the future. For example, a human foreman will eventually learn that all truck drivers do not behave the same, and will undoubtedly develop a means of identifying truck drivers with particular survivor functions. Abrupt changes in a survivor function (relative discontinuities in the piecewise linear approximations) are a good indication of events that have not been accounted for, and warrant increased vigilance and the addition of new projection and persistence rules. The foreman might note that a 15-minutes-til-closing message broadcast over the factory's public address system will precipitate a certain number of departures, whereas an announcement of free coffee and donuts will increase some truck drivers' reserves of patience.

We are in the process of designing a planner that is capable of managing its own causal theory in a cycle of planning and execution. We hope to extend existing information gathering schemes used in handling timeless inferences [14], and take advantage of certain specialized temporal techniques such as those proposed for handling multistep predictions [15]. Appendix IV describes some preliminary results on using probabilistic causal theories for planning in the warehouse domain.

V. Monitoring the Reasons for Planning Decisions

In the traditional approach to planning in AI, a *protection* is a constraint on a plan that a particular consequence of an action be preserved over some interval in order that the action achieve its intended effect. If some other action or event intervenes to undo this intended effect, then the protection is said to be violated. Noticing and responding to protection failures have become important issues in automated planning research, but protections are just a specific case of a more general concept having to do with the reasons for decisions made during planning. In general the planning process might be characterized by the following steps: 1) make a decision to commit to particular actions or schedules, 2) determine the reasons why those decisions appear to be sound, 3) set up a process to monitor those reasons and to respond if those reasons appear to weaken sufficiently to necessitate reassessment of the commitments originally suggested-by those reasons.

In the case of our factory robot, one reason for choosing to serve a set of trucks in a particular order might be that they arrived in that order and that, according to current projections, if they are handled as planned, there is a good chance that they will actually be loaded rather than a truck driver becoming impatient and leaving. The decision to commit to a plan can depend upon some notion of utility [16]. Using utilities and probabilistic projection one can compute a standard decision theoretic expected utility measure.

If the robot is delayed in handling the trucks, then the utility associated with the proposed schedule will change. Computing this change is simple and is performed routinely by an incremental update algorithm. The basic idea is that in committing to a plan, the planner explicitly notes the reasons (often couched in terms of expected utility estimates) for initially making the commitment. In addition, the planner explicitly states the conditions for reconsidering its decision (usually couched in terms of a threshold that the expected utility must drop below). A protection fails when the expected utility drops below the specified threshold.

VI. Conclusion

In this paper, we have sketched a theory of reasoning about change that extends previous theories [9], [17]. In particular, we have shown how *persistence* can be modeled in probabilistic terms. Probabilistic projection is a special case of reasoning about continuously changing quantities involving partial orders and other sorts of incomplete information, and as such it represents an intractable problem. We have tried to identify a tractable core in the inferences performed by probabilistic projection.

We believe that the inferential and causal rule refinement capabilities designed into our system are essential for robots to perform robustly in routine manufacturing situations. We hope that our current investigations will yield a new view of strategic planning and decisionmaking under uncertainty based on the idea of continuous probabilistic projection.

Appendix I
Algorithmic Details

Probabilistic causal theories are composed of two types of rules: projection rules

$$\text{PROJECT}(P_1 \wedge P_2 \cdots \wedge P_n, E, R, \kappa) \tag{4}$$

and persistence rules

$$\text{PERSIST}(Q, \lambda) \tag{5}$$

where P_1 through P_n, R, and Q are all fact types, and E is an event type. We assume (statistical) independence of fact types so that, if we are interested in the conjunction $P_1 \wedge P_2 \cdots \wedge P_n$, we can assume that

$$p(\langle P_1 \wedge P_2 \cdots \wedge P_n, t\rangle) = \prod_{i=1}^{n} p(\langle P_i, t\rangle). \tag{6}$$

We define a relation $\prec_C$ on fact types so that $Q \prec_C R$ just in case there exists a rule of the form PROJECT($P_1 \wedge P_2 \cdots \wedge P_n, E, R, \kappa$), where $P_i = Q$ for some i. For any given set of causal rules, the graph $\mathcal{G}_{\prec_C}$ whose vertices correspond to fact types and whose arcs are defined by $\prec_C$ is likely to have cycles; this will be the cause of a small complication that we will have to resolve later. In this paper we distinguish between fact types corresponding to propositions that hold over intervals and event types corresponding to instantaneous (point) events. For each occurrence (token) of a point event of type E, we will need its density function $p(\langle E, t\rangle)$. Probabilistic projection takes as input a set of initial events and their corresponding density functions. Given the restricted format for projection rules, the only additional point events are generated by the system in response to the creation of new instances of fact types. For each token of fact type P, we identify a point event of type E_P corresponding to the particular instance of P becoming true. In the process of probabilistic projection, we will want to compute the corresponding density function $p(\langle E_P, t\rangle$. In addition to computing density functions, we will also want to compute the mass functions $p(\langle P, t\rangle)$ for instances of propositions persisting over intervals of time.

In order to describe the process of probabilistic projection, we will divide the process into two different stages: *deterministic causal projection* and *probabilistic causal refinement*. The actual algorithms are more integrated to take advantage of various pruning techniques, but this simpler staged process is somewhat easier to understand. Deterministic causal projection starts with a set of tokens and a set of projection rules and generates a set of new tokens T by scanning forward in time and applying the rules without regard for the indicated probabilities. This stage can be carried out using any number of simple polynomial algorithms (see [8], [18]) and will not be further detailed here. Probabilistic causal refinement is concerned with computing density and mass functions for tokens generated by deterministic causal projection. In the following, all den-

sity and mass functions are approximated by step (i.e., piecewise constant) functions. We represent these functions of time using vectors (e.g., mass(T) denotes the mass function for the token T and mass(T)(i) denotes the value of the function at $t = i$). For each fact token T_P, we create a corresponding event token T_{E_P} and define a vector mass(T_P). For each event token T_E, we define a vector density(T_E). We define an upper bound Ω on projection and assume that each mass and density vector is of length Ω.[2] Initially, we assume that

$$\forall T \in \mathrm{T}: 1 \leqslant i \leqslant \Omega:$$
$$(\mathrm{density}(T)(i) = 0) \wedge (\mathrm{mass}(T)(i) = 0).$$

Event tokens are supplied by the user in the form

$$\kappa = \int_{\mathrm{est}}^{\mathrm{lst}} p(\langle E, t\rangle)\, dt$$

where "est" and "lst" correspond, respectively, to the earliest and latest start time for the token and κ is the probability that the event will occur at all. We assume that the density function for such an event is defined by a Gaussian distribution over the interval from est to lst. For a token T_E corresponding to a user-supplied initial event, it is straightforward to fill in density(T_E). Probabilistic causal refinement is concerned with computing mass(T_P)(i) and density(T_{E_P})(i) for all fact tokens T_P and all event tokens T_{E_P}. We partition the set of tokens T into fact tokens $\mathrm{T_F}$ and event tokens $\mathrm{T_E}$. Probabilistic causal refinement can be defined as follows:

Procedure: refine(T)
  for $i = 1$ to Ω:
  for $T \in \mathrm{T_E}$: density-update(T, i);
    for $T \in \mathrm{T_F}$: mass-update(T, i).

Of course, all of the real work is done by "density-update" and "mass-update." Each token has associated with it a specific *derivation* that is used in computing its mass or density. For a token T_{E_R}, this derivation corresponds to a rule of the form

$$\textsc{project}(P_1 \wedge P_2 \cdots \wedge P_n, E, R, \kappa)$$

and a set of antecedent tokens $\{T_E, T_{P_1}, T_{P_2} \cdots T_{P_n}\}$ used to instantiate the rule and generate the consequent token T_R. Given that

$$p(\langle E_R), t\rangle = \kappa * p(\langle E, t\rangle) * p(\langle P_1 \wedge P_2 \cdots \wedge P_n, t\rangle)$$

and, assuming independence (6), we have

Procedure: density-update(T_{E_R}, i)
density(T_{E_R})(i) ←

$$\kappa * \mathrm{density}(T_E)(i) * \prod_{j=1}^{n} \mathrm{mass}(T_{P_j})(i).$$

[2] There are some obvious optimizations to be made here.

There is one problem with this formulation: it relies on all the mass and density functions for the antecedent conditions already being computed for the instant i. In the present algorithm, "refine" takes no care in ordering the tokens in T. There are a number of ways of ensuring that the updates are performed in the correct order. The easiest is to partially order T according to $\prec_C$ and insist that $\mathcal{G}_{\prec_C}$ be acyclic, but this would preclude the use of most interesting causal theories. A more realistic method is to partition T with respect to an instant i into those tokens that are open and those that are closed. Deterministic causal projection defines an earliest start time (est) for each token; for event tokens a latest start time (lst) is specified. An event token is open throughout the interval est to lst and closed otherwise. For fact tokens, we modify probabilistic causal refinement so that it closes a fact token T_P as soon as mass(T_P)(i) drops below a fixed threshold. A fact token is open from its est until it is closed. All we require then is that for any i the set of tokens that are open define an acyclic causal dependency graph using $\prec_C$. This restriction still allows for a wide range of causal theories. To get "refine" to do the right thing, we would have to apply "refine" only to open tokens and either sort the tokens using $\prec_C$, or (as is actually done) define "refine" so that if in the course of updating a consequent token, "refine" finds an antecedent token that has not yet been updated, it applies itself recursively.

The derivation of a token T_P corresponds to a rule of the form PERSIST(P, λ) where λ is the constant of decay for the fact type P, and an event token T_{E_P}. The procedure "mass-update" is a bit more difficult to define than "density-update" since it depends upon the type of decay functions used in persistence rules. In the case of exponential decay functions, the operation of "density-update" is reasonably straightforward. Recall the basic combination rule for probabilistic projection:

$$h(t) = \int_{-\infty}^{t} f(x) g(t - x)\, dx$$

and suppose that g is of the form $e^{-\lambda z}$, where λ is some constant of decay, and that f can be approximated by a step function as in

$$f(x) = \begin{cases} C_1 & s_0 \leqslant x < s_1 \\ C_2 & s_1 \leqslant x < s_2 \\ \cdots & \\ C_n & s_{n-1} \leqslant x < s_n. \end{cases}$$

We will take advantage of the fact that

$$\int_{s_j}^{s_k} f(x)\, dx = \sum_{i=j}^{k-1} \int_{s_i}^{s_{i+1}} f(x)\, dx$$

and

$$g(s_{k+1} - x) = e^{-\lambda\delta} g(s_k - x)$$

where $\delta = s_{k+1} - s_k$.

Making appropriate substitutions, we have

$$\begin{aligned} h(s_{k+1}) &= \sum_{i=j}^{k} \int_{s_i}^{s_{i+1}} f(x)g(s_{k+1}-x)\,dx \\ &= \sum_{i=j}^{k-1} \int_{s_i}^{s_{i+1}} f(x)g(s_{k+1}-x)\,dx \\ &\quad + \int_{s_k}^{s_{k+1}} f(x)g(s_{k+1}-x)\,dx \\ &= e^{-\lambda\delta} \sum_{i=j}^{k-1} \int_{s_i}^{s_{i+1}} f(x)g(s_k-x)\,dx \\ &\quad + \int_{s_k}^{s_{k+1}} f(x)g(s_{k+1}-x)\,dx \\ &= e^{-\lambda\delta} h(s_k) + \int_{s_k}^{s_{k+1}} f(x)g(s_{k+1}-x)\,dx. \end{aligned}$$

It should be clear that updates depending upon such simple survivor functions can be performed quite quickly. Integration is approximated using Riemann sums with a mesh of fixed size (roughly) corresponding to δ. We define the procedure "mass-update" as

Procedure: mass-update (T_p, i)

$$\text{mass}(T_P)(i) \leftarrow e^{\lambda_P \delta}\text{mass}(T_P)(i-1) + \text{density}(T_{E_P})(i).$$

The actual algorithms are complicated somewhat by the fact that the choice of mesh size may not coincide precisely with the steps in the step functions approximating survivor functions and distributions. We compensate for this by using a somewhat finer mesh in the update algorithms. The fact that we employ a fixed mesh size still causes small errors in the accuracy of the resulting mass and density functions, but these errors can be controlled. We have tried to make a reasonable tradeoff, taking into account that the finer the mesh the larger the mass and density vectors. Given that the step functions used for encoding survivor functions and distributions are only approximations, there is a point past which employing a finer mesh affords no additional information. We have found that a mesh size of half the smallest step in any step function works quite well in practice.

Appendix II
Linear Decay Survivor Functions

The exponential decay function used in our previous examples have many features which make it particularly attractive for temporal probabilistic projection. It is computationally tractable and is especially suited to incremental computation. It is conceptually simple and occurs in many forms in nature.

However, there are also problems with the exponential decay function. Many of the types of processes whose persistence behavior we would like to track with probabilistic projection have the following sort of general behavior: our expectation of their persistence is initially very high and remains so for some time; then there is a gradual and accelerating descent, with the fastest descent around the mean time of persistence; and finally a gradual tapering off, until it levels off in the end.

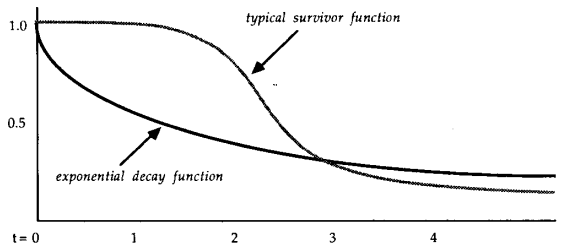

Fig. 5. Exponential decay and other functions as persistence functions.

The exponential decay function is not a good approximation to such processes. Exponential decay is characterized by the opposite behavior; steep descent in the beginning, with a relatively quick leveling off. To contrast the two types of functions, when the first type of function is almost constant at unity, the exponential decay function will be experiencing its fastest descent while, conversely, by the time the first type of process begins its descent, the exponential decay is past its major descent, perhaps leveling off at a relatively low probability (see Fig. 5).

It should be clear that any inference procedures relying upon probabilistic information derived from such an approximation will often draw unsound or, at best, unexpected conclusions. For example, a planner attempting to maximize expected utility values using decision theoretic heuristics may actually make the decisions opposite from those we would like. Consider a situation in which two trucks having similar persistence functions arrive at the loading dock. As usual, the planner's task is to figure out the best order to load the trucks. With the exponential decay function, the expected persistence for each truck drops very rapidly in the beginning, and they become virtually indistinguishable in a very short period of time. Therefore, no matter which truck is loaded first, the expected success for the second truck will be about the same as that for the first. However, for the first truck the expected success is higher for the truck which arrived later; therefore the planner will decide to load the second truck first, which, of course, is probably not a good choice. (Does this show up too much weakness?)

In our current implementation, this has been countered with the introduction of a new quantity, which we call the (persistence) *delay*. Intuitively, the delay is a simple entity: it is an initial time duration, starting from a token's "est," during which the expected persistence of the token experiences no decay. The probabilistic update algorithms will refrain from applying the token's persistence function until the delay is over. While the introduction of the delay tends to counter the ill effects of the exponential decay function, it seems worthwhile to consider other functions which might better describe the kinds of processes we are likely

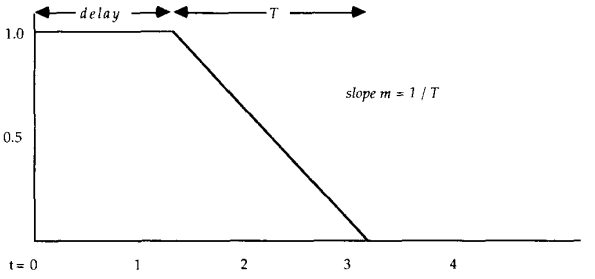

Fig. 6. Examples of piecewise linear (linear decay) functions.

to encounter in real life, and of those functions, those which are amenable to incremental computation.

A possible candidate is the class of piecewise linear functions (see Fig. 6). Another is a reverse normal distribution (we model the persistence of a proposition in terms of the occurrence, with a normal distribution, of the event of the proposition becoming false; in other words, the complement event of the proposition being true is considered as the cumulative probability of a normal distribution). While piecewise linear functions, or linear decay functions, are perhaps no less a simplification of real processes than the exponential decay function; they avoid the major shortcoming of exponential decay, namely, the significant distortion in the initial stages. Conceptually, the linear decay function is just as simple, and when the initial delay is factored in, intuitively more satisfying. Furthermore, the linear decay function is no less amenable to acquisition than the simple exponential decay function. For this reason, in this section we will illustrate how a simplified version of the linear decay function might be used as an alternative in temporal probabilistic projection.

The linear decay function is defined by

$$g(t) = \min(1, \max(0, 1 - m(t - \theta)))$$

or

$$g(t) = \begin{cases} 1, & \text{if } 0 \leqslant t < \theta; \\ 1 - m(t - \theta), & \text{if } \theta \leqslant t < \theta + 1/m; \\ 0, & \text{otherwise} \end{cases}$$

where m is the slope, and θ is the delay. Recall that the convolution is

$$h(t) = \int_{-\infty}^{t} f(z) g(t - z)\, dz.$$

We would like to be able to compute $h(t)$ in terms of $h(t - \delta)$ and f. In this paper, we will show the derivation for the simpler function will make such derivation for the simpler function

$$h(t) = \begin{cases} 1 - mt, & \text{if } 0 \leqslant t < 1/m; \\ 0, & \text{otherwise} \end{cases}$$

without the delay. The case with the delay is a relatively simple extension of this case:

$$h(t) = \int_{t-(1/m)}^{t} f(z)(1 - m(t - z))\, dz$$

$$= (1 - mt) \int_{t-(1/m)}^{t} f(z)\, dz + m \int_{t-(1/m)}^{t} z f(z)\, dz$$

and similarly,

$$h(t - \delta) = (1 - m(t - \delta)) \int_{t-(1/m)-\delta}^{t-\delta} f(z)\, dz + m \int_{t-(1/m)-\delta}^{t-\delta} z f(z)\, dz.$$

We will use the approximate identities

$$\int_{t-(1/m)}^{t} f(z)\, dz \approx \int_{t-(1/m)-\delta}^{t-\delta} f(z)\, dz + \delta f(t) - \delta f\left(t - \frac{1}{m}\right)$$

and

$$\int_{t-(1/m)}^{t} z f(z)\, dz \approx \int_{t-1/m-\delta}^{t-\delta} z f(z)\, dz + \delta t f(t) - \delta\left(t - \frac{1}{m}\right) f\left(t - \frac{1}{m}\right)$$

and the preceding equations to derive the expression we desire:

$$\begin{aligned} h(t) &= h(t - \delta) + \delta(1 - mt)\left(f(t) - f\left(t - \frac{1}{m}\right)\right) \\ &\quad + m\delta\left(t f(t) - \left(t - \frac{1}{m}\right) f\left(t - \frac{1}{m}\right)\right) \\ &\quad - m\delta \int_{t-(1/m)-\delta}^{t-\delta} f(z)\, dz \\ &= h(t - \delta) + \delta(1 - mt + mt) f(t) \\ &\quad - \delta\left(1 - mt + m\left(t - \frac{1}{m}\right)\right) f\left(t - \frac{1}{m}\right) \\ &\quad - m\delta \int_{t-(1/m)-\delta}^{t-\delta} f(z)\, dz \\ &= h(t - \delta) + \delta f(t) - m\delta \int_{t-(1/m)-\delta}^{t-\delta} f(z)\, dz. \end{aligned}$$

With the exponential decay function, we had

$$h(t) = e^{-\lambda\delta} h(t - \delta) + \delta f(t)$$

so the main difference is the need for computing the integral

$$\int f(z)\, dz.$$

Recall that f is some function describing the probability distribution of the occurrence of an event. The preceding integral is the cumulative mass probability that the event has occurred, and in the recursive definition, it is the cumulative probability that the event occurred between two points of time. It is then possible to compute mass probabilities via the convolution with linear decay persistence functions, by a relatively simple and painless expedient—the addition of a token for the cumulative probabil-

ity of the event being convolved. It should be clear that many simple inference tasks might actually require this fact token anyway, so this new requirement of probabilistic projection may not add unnecessarily to a set of tokens.

Appendix III
Acquiring Persistence Rules

Statistical methods have not seen particularly wide application in AI. This is largely due to problems concerning the availability of the data necessary to employ such methods. Data provided from experts has been labeled as unreliable. The use of priors in Bayesian inference has been much maligned. An alternative to expert judgements and estimating priors is to integrate the data acquisition process into the system: have it gather its own data. In such a scheme, all predictions made by the system are conditioned only upon what the system has directly observed. Of course, this is unrealistic in many cases (e.g., diagnostic systems whose decisions could impact on the health or safety of humans). In the industrial automation applications considered in this paper, however, not only is it practical but it also appears to be crucial if we are to build systems capable of adapting to new situations.

In this section, we describe a system for continually refining a database of probabilistic causal rules in the course of routine planning and execution. Given the focus of this paper, we will concern ourselves exclusively with the acquisition (or refinement) of persistence rules. Our warehouse planner keeps track of how long trucks stay around and uses this information to construct survivor functions for various classes of trucks. The system must be told which quantities it is to track and how to distinguish different classes of trucks, but that given, the rules it acquires are demonstrably useful and statistically valid in the limit.

The survivor function for a given class of trucks is computed from a set of data points corresponding to instances of trucks observed arriving and then observed leaving without being loaded.[3] It should be clear that, in general, a collection of data points will not define a survivor function uniquely. There are many ways in which to derive a reasonable approximation for such a function. For example, we might employ some form of curve fitting based on an expected type of function and the sample data. While such methods may yield more accurate approximations in some cases, for our application there are simpler and more efficient methods.

Our system derives two parameters for constructing survivor functions. The first is an estimate of the delay (i.e., the initial interval of time during which the function remains constant), and the second corresponds to the rate of decay during the function's period of descent. We simply use the arithmetic mean of the samples to compute the rate of decay. With both of the simple classes of functions we have considered, the exponential decay and the linear decay functions computing the persistence parameter (λ) and the slope, respectively, is trivial. In the case of an exponential decay, we use the mean as the half-life of the function.

Computing the delay interval is also quite simple. Recall that in our examples the delay corresponds to the interval of time during which no trucks are likely to leave. Each data point is represented as an integer corresponding to how long a particular truck stayed. Keeping in mind that there will be occasional aberrations, we choose to ignore some percentage of the data points corresponding to those that are far from the mean. There are more sophisticated means of doing this, but we simply sort the data points for each class of trucks in increasing order and set the delay to be the length of time corresponding to some data point in the kth percentile of the resulting sorted list, where k defaults to 5. This provides a reasonable approximation to the actual functions and it is very fast to compute.

We can now sketch the simple algorithm utilized in our system. As noted, we need to collect data for each class of interest. The data for each class is collected in a data structure along with various intermediate quantities used by the update algorithm (e.g., since the algorithm calls for the arithmetic mean of the data points, it is convenient to incrementally compute the sum of the elements of the collection). The *class* data type has the following accessor functions associated with it (c is an instance of class):

type(c):	the type of the associated survivor function: linear or exponential,
lambda(c):	the rate or slope,
delay(c):	the delay,
history(c):	a vector corresponding to the sorted collection of data points (individual data points are referenced using history(c)(i), where i is an integer index),
instances(c):	the number of data points in the collection,
sum(c):	the sum of the items in the collection,
offset(c):	a percentile indicating the bottom n data points in the sorted collection to be ignored in computing the delay (defaults to 5 percent).

Assuming that c is an instance of *class* and p is a new data point, the acquisition algorithm can be described as follows:

```
Procedure: acquire(c, p)
  begin
    history(c) ← insert(p, history(c));
    instances(c) ← instances(c) + 1;
    sum(c) ← sum(c) + p;
    lambda(c) ← rate(c, ((sum(c)/instances(c)) −
    delay(c)));
    delay(c) ← history(c)([instances(c) * offset(c)])
  end.
```

[3] There is actually more information to be had. For example, instances of trucks observed arriving and subsequently observed to be absent (the exact time of leaving unknown) are presumably relevant to the problem at hand, and in fact it is possible to make use of such information by making various additional assumptions. We will not, however, consider such complications here.

The function "insert" is assumed to insert a data point into a sorted collection. The function "rate" depends on the type of survivor function used:

```
Function: rate(c, μ)
  if μ = 0
    then +∞
    else if type(c) = linear
        then 0.5/μ
        else if type(c) = exponential
            then (ln 2)/μ.
```

Although we have tested our approach extensively in simulations and have found the acquired persistence data to converge very rapidly to the correct values (these results are described further in Appendix IV), we do not claim that the aforementioned methods have any wider application. The simplicity of the algorithm and its incremental nature is attractive, but the most compelling reason for using it is that it works well in practice. Probabilistic projection does not rely upon a particular method for coming up with persistence rules. As an alternative, the data might be integrated off line, using more complex (and possibly more accurate) methods.

It should be noted that our system is given the general form of the rules it is to refine. It cannot, on the basis of observing a large set of trucks, infer that trucks from one company are more impatient than those from another company, and then proceed to create two new persistence rules where before there was only one. The general problem of generating causal rules from experience is very difficult. We are currently exploring methods for distinguishing different classes of trucks based on statistical clustering techniques (e.g., Kolmogorov–Smirnov's D-statistic [19] and Shapiro–Wilk's W-statistic [20]). Using such methods, it appears to be relatively straightforward to determine that a given data set corresponds to more than one class, and even to suggest candidate survivor functions for the different classes. However, figuring out how to distinguish between the classes in order to apply the different survivor functions is considerably harder.

Appendix IV
Experiments

To help in evaluating temporal probabilistic projection for robot planning, we carried out a series of simple experiments. The experiments involved a simulated factory environment similar in many respects to the warehouse domain from which we have drawn most of the examples in this paper. The experiments were designed to test the ability of a system to make reasonable predictions in the face of uncertainty. Since the predictions made by our system are based on expected outcomes, it was necessary to run experiments involving hundreds of planning decisions and many times that many predictions. We measured the performance of a simulated robot engaged in loading trucks over the course of many (simulated) days. We considered several strategies for deciding which truck to unload next. Each of the strategies constituted a test "group." In this section, we will describe the experiments we conducted, the different strategies that we compared, and the results that we obtained.

In our simulated warehouse, a robot is charged with loading all trucks that arrive at a particular loading dock. If there is one truck waiting at the dock and the robot is otherwise unoccupied, then it will load that truck. If more than one truck is waiting, then the robot must decide which one to load first. How it makes this decision will determine how successful it is in carrying out its charge. Success is measured in terms of the number of trucks that the robot loads. We assume that trucks do not sit around waiting to be loaded forever. Trucks are not, however, completely arbitrary in their behavior; we assume that there are different classes of trucks, red, green, and blue, and that individuals in each class have similar persistence behavior. If all of the trucks were in the same class, then loading the trucks in the order of their arrival (first come first served) would be the best strategy. If there is more than one class and the robot has some idea of how members of each class are likely to behave, then it would seem that the robot could do somewhat better. In our experiments, we consider three basic strategies.

1) Temporal probabilistic projection (TPP)—the robot uses the techniques described in this paper to predict the consequences of various loading sequences and to refine the persistence rules used in the process of prediction.
2) No acquisition (NA): Same as the preceding but the robot does not acquire its persistence rules; it is given the same probability distributions as the simulator uses in determining when a truck actually will leave. The robot cannot, however, determine the simulator's outcome with certainty.
3) First come first served (FCFS)—the robot does no prediction at all; it simply loads the trucks in the order that they arrive.

It would seem that TPP should approach NA in the limit; how fast and how much performance suffers in the interim depend critically on how fast it learns the survivor functions that govern the various classes of trucks. It is relatively easy to concoct situations in which FCFS will do badly (simply have the longer persisting trucks arrive before the shorter persisting ones). On the other hand, it is difficult to conceive of FCFS doing any better; it should at best do only as good as TPP. Therefore, on the average, we would expect FCFS to perform somewhat less well than TPP. The interesting thing about the experiments is that TPP converges very quickly on an approximation to the actual survivor functions, allowing the TPP-guided robot to perform on a par with the NA-guided robot.

The robot's knowledge of its simulated environment is obtained through a *status board* that reports the arrivals and departures of trucks at the loading dock. When the robot has nothing to do, it checks this status board frequently. At other times it can only check the status board at irregular, possibly long, intervals between loading tasks. In our current simplistic world, the only kinds of events

TABLE I
EXPERIMENTAL RESULTS IN A SIMULATED RUN OF 300 DAYS

Experiment	Total	Success	Failure	Success Rate
TPP	5898	4669	1229	79.2%
NA	5830	4638	1192	79.6%
FCFS	6058	4123	1935	68.1%

TABLE II
ACQUIRED PERSISTENCE RATES AFTER 300 DAYS

Truck Class	Acquired λ	True λ	Difference in $e^{\lambda\delta}$
Red	0.400	0.346	3.7%
Blue	0.368	0.346	1.5%
Green	0.369	0.346	1.6%

TABLE III
ACQUIRED PERSISTENCE FOR A TRUCK CLASS AS A FUNCTION OF SAMPLES
(AGAINST REAL PERSISTENCE OF 0.346)

Samples	Acquired	Difference (percent)
0	0.000	29
5	0.495	9
10	0.533	12
15	0.335	0
20	0.347	0
25	0.361	1
30	0.378	2
35	0.398	3
40	0.375	1
45	0.371	1
50	0.369	1

TABLE IV
DIFFERENCE IN CUMULATIVE SUCCESS OF TPP AND NA STRATEGIES

Day	TPP	NA	Difference (percent)
25	0.7689	0.8162	4.73
50	0.7878	0.8015	1.37
75	0.7867	0.7957	0.90
100	0.7885	0.7990	1.05
125	0.7952	0.8038	0.86
150	0.7937	0.7990	0.52
175	0.7930	0.7972	0.41
200	0.7925	0.7978	0.53
225	0.7948	0.7980	0.32
250	0.7943	0.7927	0.15
275	0.7928	0.7929	0.01
300	0.7916	0.7955	0.39

that occur are those that are displayed on the status board. In other words, no events occur such that the robot must interrupt a task and respond. This makes the simulator quite simple; the world has to be updated only when the robot looks at the status board.

The simulator consists of two parts: a set of *oracles*, which determine what type of events will occur and when, and an *agenda*, used to determine how those events impact on the robot's behavior. Whenever the simulator is called, it first runs each oracle, placing any events that are determined to occur onto the agenda. Then it makes a sweep of the agenda and returns to the robot (i.e., "displays" to the status board) all the events that were deemed to occur before the current time. Since the simulator is called at irregular intervals, and since some types of events are defined to occur at regular intervals (e.g., red trucks might arrive with a 3-percent probability every ten minutes), the simulator actually runs the oracles as often as is necessary. This is an important consideration in allowing for repeatable experiments.

In our experiments, two oracles were assigned to each class of truck, to determine when a truck of the class arrived, and once arrived, to determine how long it would stay if it failed to be loaded. The oracles were implemented in a way that enabled repeatable behavior so that the three different strategies could be tried on exactly the same sequence of events.

The results of the simulations were generally very encouraging. A typical simulation run consisted of 300 work days (see Table I). On the whole, the TPP strategy performed favorably against the NA strategy. While our expectation was that our methods should approach the ideal in the limit, in practice it did so very rapidly, and the survivor functions it learned converged very quickly towards the right numbers. For the typical run depicted in the table, it took less than 20 data points for the survivor function to come to within 1 percent of the correct function initially, and it took less than 1500 total trucks (about 100 data points for each class) for cumulative success rate to come to within 1 percent of NA.

By contrast, the FCFS strategy did relatively poorly, as expected. Although the FCFS strategy ran significantly faster (it cut out projection altogether), it is difficult to draw any conclusions about how such a speed difference might scale up in real systems.

ACKNOWLEDGMENT

The authors would like to thank Linda Mensinger Nunez for her suggestions on how to extend the rule acquisition algorithms using statistical clustering techniques.

REFERENCES

[1] H. Raiffa, *Decision Analysis: Introductory Lectures on Choices under Uncertainty*. Reading, MA: Addison-Wesley, 1968.
[2] R. D. Shachter, "Evaluating influence diagrams," *Operat. Res.*, vol. 34, pp. 871–882, Nov./Dec. 1986.
[3] T. Dean, "An approach to reasoning about the effects of actions for automated planning systems," *Ann. Operat. Res.*, vol. 12, pp. 147–167, 1988.
[4] J. Pearl, "A constraint propagation approach to probabilistic reasoning," in *Proc. 1985 AAAI/IEEE Sponsored Workshop on Uncertainty in Artificial Intelligence*, 1985.
[5] R. Duda, P. Hart, and N. J. Nilsson "Subjective Bayesian methods for rule-based inference systems," in *Readings in Artificial Intelligence*, B. Webber and N. Nilsson, Eds. Palo Alto, CA: Tioga, 1981.
[6] J. McCarthy and P. J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," *Mach. Intell.*, vol. 4, 1969.
[7] E. Sacerdoti, *A Structure for Plans and Behavior*. New York: American Elsevier, 1977.
[8] T. Dean and D. V. McDermott, "Temporal data base management," *Artific. Intell.*, vol. 32, pp. 1–55, 1987.
[9] D. V. McDermott, "A temporal logic for reasoning about processes and plans," *Cognit. Sci.*, vol. 6, pp. 101–155, 1982.

[10] R. Syski, *Random Processes*. New York: Marcel Dekker, 1979.
[11] M. Ginsberg, "Does probability have a place in non-monotonic reasoning?" in *Proc. IJCAI 9*, IJCAI, 1985, pp. 107–110.
[12] T. Dean and M. Boddy, "Incremental causal reasoning," in *Proc. AAAI-87*, AAAI, 1987, pp. 196–201.
[13] T. Dean and K. Kanazawa, "Probabilistic temporal reasoning," in *Proc. AAAI-88*, AAAI, 1988, pp. 524–528.
[14] J. R. Quinlan, "A task-independent experience-gathering scheme for a problem solver," in *Proc. IJCAI 1*, IJCAI, 1969, pp. 193–197.
[15] R. S. Sutton, "Learning to predict by the methods of temporal differences," Tech. Rep. TR87-5091, GTE Laboratories, Waltham, MA, 1987.
[16] V. Barnett, *Comparative Statistical Inference*. New York: John Wiley and Sons, 1982.
[17] Y. Shoham and T. Dean, "Temporal notation and causal terminology," in *Proc. Seventh Ann. Conf. Cognitive Science Society*, Cognitive Science Society, 1985, pp. 90–99.
[18] S. Hanks and D. V. McDermott, "Default reasoning, nonmonotonic logics, and the frame problem," in *Proc. AAAI-86*, AAAI, 1986, pp. 328–333.
[19] O. J. Dunn and V. A. Clark, *Applied Statistics: Analysis of Variance and Regression*. New York: John Wiley and Sons, 1974.
[20] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality," *Biometrika*, vol. 52, pp. 591–612, 1965.

Thomas Dean received the B.A. degree in mathematics from Virginia Polytechnic Institute and State University in 1982 and the Ph.D. in computer science from Yale University in 1986.

He is currently an Assistant Professor on the faculty of the Department of Computer Science at Brown University in Providence, RI. His general research interests include deductive retrieval methods for applications in artificial intelligence, logic programming, robot problem solving, and probabilistic inference. Current research is concerned with theories of temporal and spatial inference for reasoning about actions and processes. Recent work has led to the design and implementation of a temporal database system for applications involving mobile robots and factory automation. The main thread of the robotics work is concerned with decisionmaking under uncertainty. Of particular interest are problems in which the notion of risk is complicated by there being limited time for both deliberation and action.

Keiji Kanazawa received the B.A. degree in mathematics from Bennington College in 1985, and the M.S. in computer science from Brown University in 1988.

He is currently a doctoral candidate in the Department of Computer Science at Brown University in Providence, RI. His research interests include robot problem solving, probabilistic inference, and neuromorphic systems. Current research focuses on temporal inference, decisionmaking under uncertainty, and computational methods for probabilistic inference.