

Execution Monitoring with Quantitative Temporal Bayesian Networks

Dirk Colbry
colbrydj@eecs.umich.edu

Bart Peintner
bpeintne@eecs.umich.edu

Martha E. Pollack
pollackm@eecs.umich.edu

Computer Science and Engineering
University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48103

Abstract

The goal of execution monitoring is to determine whether a system or person is following a plan appropriately. Monitoring information may be uncertain, and the plan being monitored may have complex temporal constraints. We develop a new framework for reasoning under uncertainty with quantitative temporal constraints – Quantitative Temporal Bayesian Networks – and we discuss its application to plan-execution monitoring. QTBNs extend the major previous approaches to temporal reasoning under uncertainty: Time Nets (Kanazawa 1991), Dynamic Bayesian Networks and Dynamic Object Oriented Bayesian Networks (Friedman, Koller, & Pfeffer 1998). We argue that Time Nets can model quantitative temporal relationships but cannot easily model the changing values of fluents, while DBNs and DOOBNs naturally model fluents, but not quantitative temporal relationships. Both capabilities are required for execution monitoring, and are supported by QTBNs.

Introduction

The goal of execution monitoring is to determine whether a system or person is following a plan appropriately or is heading toward a failure state. Most execution monitoring systems build an internal model of the domain and use sensor inputs to update the model. Because sensors are error-prone, the execution monitoring process must be able to reason under uncertainty. Bayesian belief networks that support plan execution monitoring under uncertainty can be generated directly from the plans being monitored (Huber, Durfee, & Wellman 1994). However, previous work on this topic has generally been restricted to classical plans, which use only qualitative ordering and causal constraints. For many realistic domains, planning and execution systems must also model quantitative temporal constraints between actions. The execution monitor in such systems also needs to be able to model quantitative temporal constraints, and reason with them under conditions of uncertainty.

One existing method for temporal reasoning under uncertainty is the Time Net (Kanazawa 1991). Although Time Nets effectively model quantitative temporal relationships between actions, they cannot easily model the changing state of fluents that influence actions. Other approaches to modeling temporal reasoning under uncertainty include Dynamic Bayesian Networks (DBNs) and Dynamic Object Oriented Bayesian Networks (DOOBNs) (Friedman, Koller, & Pfeffer 1998). These approaches naturally model fluents, but because they utilize the Markov Assumption they give up the ability to model arbitrary quantitative temporal relationships. For execution monitoring, both capabilities are required. In this paper, we present a new framework to support execution monitoring, called Quantitative Temporal Bayesian Networks (QTBN), which combine the advantages of Time Nets and DBNs.

In the next section, we describe an execution-monitoring example that we will use throughout the rest of the paper. We then review the previous approaches to temporal reasoning under uncertainty. Next, we present QTBNs, and illustrate their use with our running example. Finally, we conclude by describing ongoing and future research issues on this topic.

Example

Execution monitoring is an important component of plan management (Pollack & Horta 1999). In this paper, we illustrate the QTBN framework using an example from the Autominder system (Pollack *et al.* 2002), which is being developed as part of the Initiative on Personal Robotic Assistants for the Elderly (Nursebot), a multi-university research effort ¹ aimed at investigations of robotic technology for the elderly. The Autominder system manages and monitors the execution of plans involving activities of daily living for elderly clients, providing appropriate reminders to clients and reports to caregivers. The plans used by the Autominder system are represented as Disjunctive Temporal Problems (DTPs) (Stergiou & Koubarakis 2000), an

¹The initiative includes researchers from the University of Pittsburgh, Carnegie Mellon University, and the University of Michigan.

extension of Simple Temporal Problems and Temporal Constraint Satisfaction Problems (Dechter, Meiri, & Pearl 1991). DTPs allow traditional causal relationships between actions as well as complex qualitative and quantitative temporal relationships. The DTPs differentiate between the start and end times of an action, allowing action durations to be specified as well. The following very simple example from the Autominder domain will be used throughout the rest of this paper:

An Autominder client must eat breakfast sometime after she wakes up in the morning and before 11:30am (after 11:30, the client is assumed to have missed breakfast and is eating lunch). The client must take her vitamins with a full stomach. This means that she needs to take them within a half hour after finishing breakfast.

The Autominder system thus must observe the client's activities, and attempt to infer whether the client has initiated the action of eating breakfast on her own or whether she needs a reminder to do so. This inference is in large part on information received from a sensor about whether and when the client enters the kitchen.² Additionally, once the system has inferred that the client has finished eating breakfast, it needs to update the client's plan to set the actual deadline for taking vitamins.

To achieve these goals, it is necessary for the system to generate a model of the expected behavior of the client directly from her plan, and to update that model as time passes and as new information is obtained. It is important to note the distinction between the client's plan and the client model that is used for execution monitoring. The client plan is a specification of daily activities and associated constraints; it is established by the client and her caregiver. The derived client model is a set of beliefs about the execution status of the actions in the client plan. The client model supports inference about what the client has done so far, what she is doing now, and at what times she is will likely perform other actions in the plan.

Thus, given our example plan, the execution monitor should first generate a model in which we expect that the client will eat breakfast before 11:30 and take her vitamins between 11:30 and noon. Default assumptions will initially need to be made about the exact time of the specified actions: for instance, we might assume that the probability that vitamins will be taken at time t is described by a probability distribution that is 0 outside of the time period 11:30-noon, and is uniformly distributed within that period. Over time, as the system observes the client's routines, it should learn a more accurate probability function, but that aspect of the system is outside the scope of the current paper.

When sensors provide the model with additional information, the model is updated accordingly. For exam-

ple, if the sensors report that the client is in the kitchen at 9:30, the probability that the client has begun eating breakfast should be increased. As additional information arrives, and the probability of breakfast having begun and then ended goes over some threshold, the client *plan* should be updated to represent that fact that the vitamins must be taken by, say 10:15 (if the system infers that breakfast ended at 9:45). This change in the client plan will then be reflected back in an updated client model.

This example illustrates how the execution monitor must perform temporal reasoning in uncertain environments. We next consider several approaches to such reasoning.

Existing Approaches

A lot of research has been done on temporal reasoning without uncertainty. Most of this work extends first-order logic and uses similar inference methods to reason about time. (For a survey, see (Vila 1994)). Because these methods have first-order logic at their core, incorporating uncertainty into them is at least as difficult as incorporating uncertainty into first-order logic. This difficulty has led researchers to work from the other direction: to incorporate time into methods for reasoning under uncertainty. This section reviews three approaches that incorporate time into Bayesian Networks.

Time Nets

Time Nets (Kanazawa 1991) model temporal information by assigning time values to the nodes in a standard Bayesian Network. Each node in a Time Net represents an event or a property of the environment. Each value of a node N represents an interval of time during which the event or property represented by N may occur. Taken together, the values of a node provide a probability distribution that signifies the belief that the event has occurred or will occur at particular times. The arcs between nodes represent causal influences and temporal constraints between events and properties. Associated with each node is a Bayesian update function that specifies how to update the node's value based on the current values of its parent nodes in the network. Time Nets often make use of continuous update functions, but these can be discretized and represented with conditional probability tables (CPTs) that associate values with distinct time intervals. For example, the CPT for End(Breakfast) might record the probabilities that that event will occur between 7 and 8, between 8 and 9, and between 9 and 10, contingent upon the time at which Start(Breakfast) occurred. Figure 1 shows how our example is represented by a Time Net; for ease of presentation, we omit the CPTs.

Because we do not show the CPTs, it may be easy to mistake Figure 1 as a diagram of the client plan. It is not this, but rather is a diagram of the client model. As such, it is descriptive, and represents the system's beliefs about the execution status of the plan and about

²Autominder is currently deployed on a mobile robot (Baltus *et al.* 2000), which has a variety of sensors including a camera, microphone, and a laser rangefinder.

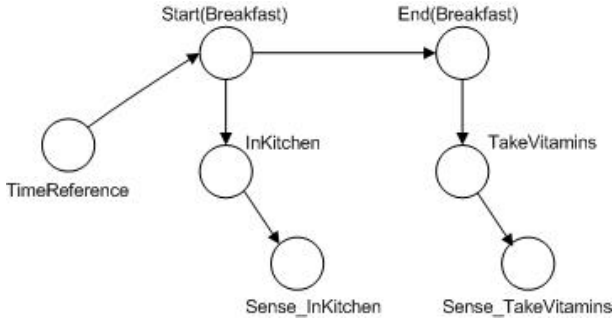


Figure 1: Example Time Net (CPTs omitted)

sensor information. This contrasts with the client plan, which is prescriptive, and represents the actions that the client should (is obliged to) perform. All the diagrams in this paper represent client models.

Time Nets effectively model quantitative temporal constraints: for example, it can represent the belief that the probability that the client will take her medicine within 30 minutes of finishing breakfast is, say, 95%. However, these networks have difficulty modeling changes in the value of a fluent over time. For example, the sensors may report that the client is in the kitchen multiple times over the course of the day. Unfortunately, there is a single *Sense_InKitchen* node in the network, and it is thus impossible to represent multiple occurrences of the event that triggers such a sensor report.

Consider the case in which the client moves to the kitchen at 7:30 and again at 9:00, and imagine for a moment that we have a perfect sensor. When the system receives the sensor data at 7:30, it encodes the fact that *Sense_InKitchen* occurred at 7:30 with probability 1, and sets to 0 the probability that this event occurred at any other time. This evidence then propagates backward in the Time Net, changing the beliefs about *InKitchen* and *Start(Breakfast)*. What happens at 9:00 when another perfectly accurate report is received from the sensor? Either this new report must be thrown out, because the system is already “certain” that the client entered the kitchen at 7:30, or the earlier, equally certain report must be discarded, to enable the *Sense_InKitchen* node to have its value to set to be 1 at 9:00. To correctly model the actual situation, a different node would be required for each time a client enters the room. Dynamic Bayes Nets provide a way of modeling fluents without introducing a large number of copies of each node.

Dynamic Bayes Nets

Dynamic Bayes Nets (DBN) are an extension to standard Bayes Nets. DBNs work by maintaining two copies of a standard Bayes Net: one representing the beliefs at the current time (T), and the other representing beliefs about the “next” time ($T+1$). These two copies are referred to as *time slices*, and should not to be confused

with the time intervals in the CPTs for a Time Net. A time slice in a DBN does not represent a fixed duration of time: rather, a transition between time slices occurs whenever a new piece of evidence arises. In the monitoring setting, this corresponds to the occurrence of an action, or the observation of such via the sensors.

DBNs make a first-order Markov assumption, modeling the next state of the system as depending only on the current state. Given a DBN, when new evidence arises, it is added to time slice T , values for nodes in the second ($T+1$ st) time slice are inferred, and “roll up” then occurs. During roll-up, slice T is deleted, slice $T+1$ becomes the new T slice, and a new copy of slice $T+1$ is created: essentially, the “next” time becomes the new “current” time slice, and a new “next” time slice is generated. In this way, a DBN can model changes in a world state over time. DBNs have been successfully used to model causal relationships in planning (Boutilier, Dean, & Hanks 1999), execution monitoring (Albrecht, Zukerman, & Nicholson 1998) and plan recognition (Huber, Durfee, & Wellman 1994), for plans without quantitative temporal constraints.

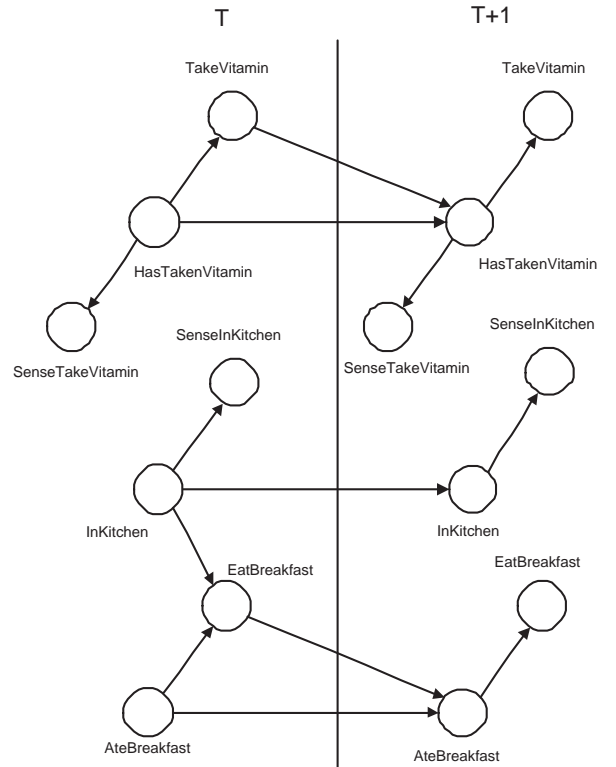


Figure 2: Example DBN

Figure 2 shows a basic DBN representation of our example; again, we omit the CPTs for clarity. Note that unlike the nodes in a Time Net, whose values are time intervals, the nodes in a DBN take more standard values. In our example, the nodes are all Boolean, and indicate the belief that a action is occurring or a prop-

erty is true either “now” (in time slice T) or in the next state (in $T+1$). Most of the nodes are directly derived from the client plan or from the sensor model and are standard types of DBNs nodes:

- TakeVitamin and EatBreakfast are associated with the two actions in our example
- InKitchen represents a precondition, or property, of the action EatBreakfast.
- The sensor nodes, SenseTakeVitamin and SenseInKitchen map to the sensors of the system and are influenced by the properties they represent.

However, there is also another class of nodes that we add to the DBN:

- Each action has an associated cumulative belief influencing it. In our example, these are the nodes HasTakenVitamin and AteBreakfast. A cumulative belief in time slice T represents the belief that the associated action either has already occurred (prior to the current time slice T) or is currently occurring. Similarly, such a node in $T+1$ represents the belief that the action has occurred, is currently occurring, or will occur next.

Cumulative belief nodes weaken the force of the Markov assumption, and are essential in plan execution monitoring because this is not easily modeled as a straightforward Markovian process. The probability that a client will eat breakfast during the next time slice depends not only on the probability that she is currently eating breakfast, but also on the probability that she has eaten breakfast at any previous time (during the current day).

Although DBNs can model change in fluent values over time, and with the inclusion of cumulative belief nodes, can be applied to plan execution monitoring for classical plans, they are inadequate for monitoring plans with quantitative temporal constraints. A DBN cannot, for example, model the belief that lunch normally occurs 3-4 hours after breakfast. One conceivable way to incorporate such constraints would be to assign clock times and durations to each time slice and construct networks with many time slices. With this approach, a quantitative temporal constraint can be represented with an arc that spans the correct number of time slices. However, assigning appropriate times and interval sizes to each time slice is very difficult. Large intervals provide too coarse of a model, making it impossible to represent constraints that are smaller than the interval size. On the other hand, small intervals increase the size of the network and make inference intractable.

Another problem with DBNs in general is drift in the probability values. Not all events modeled in a DBN evolve at the same rate. In our domain, for example, we want to monitor a client’s vital statistics as well as provide day-to-day action monitoring. However, a client’s temperature will tend to change much less frequently than her location. But every time the client is observed to have moved to a new room, all belief

node values will be updated during rollup. Suppose that at time slice T , we observe a normal temperature, and further assume that the probability is 98% that the temperature is normal at time slice $T+1$ if it is normal at slice T . If the client does not perform any observable actions for the next hour, $p(\text{temp}=\text{Normal})$ remains at 98%. On the other hand, if the client is observed to move to five different locations in the next hour, then $p(\text{temp}=\text{Normal})$ is only 90% ($= .98^5$) by the end of the hour. Thus, because of the mechanics of roll-up, the belief about the client’s temperature ends up depending upon her actually independent movements around the house.

In summary, DBNs effectively model changes in fluent values without creating multiple nodes for each fluent, but they fail to model quantitative temporal constraints between actions in a computationally tractable way, and they suffer from the problem of unwarranted probability drift.

DOOBNs

A third approach to handling temporal information in a Bayes Net is using Dynamic Object Oriented Bayes Nets (DOOBN) (Friedman, Koller and Pfeffer 1998). DOOBNs combine the benefits of object-oriented techniques with uncertainty reasoning to model temporal situations similar to those handled by DBNs. This object-oriented structure mainly facilitates software engineering. However, by enhancing objects with additional information, DOOBNs can also directly address the problem of drift, and can lead to more efficient roll-up.

The DOOBN formalism categorizes nodes into two types: persistent and transient. During rollup, the conditional probability tables (CPT) of the persistent nodes are updated to reflect the newly inferred information. Thus, the information persists into the future time slices of the DBN. This allows the property information to be maintained inside the persistent node. In contrast, transient nodes do not maintain state. When rollup occurs, the CPTs of the transient nodes retain their original values. This distinction will become relevant when we discuss the DBN portion of our QTBN.

Additionally, the DOOBN formalism allows nodes in a DBN to be grouped into objects. To each object, one can attach a parameter that represents the frequency in which to update the nodes in the object. This granularity parameter allows the DOOBN to propagate only information about nodes that are important at any particular time. This limited propagation can greatly decrease the required inference time compared to a standard DBN because of the reduced number of nodes that need to be queried during each rollup. It also addresses the problem of drift because nodes are updated at a rate that coincides with their rate of change.

QTBNs

We now present a new approach designed to model both the change in fluent values over time as well as the

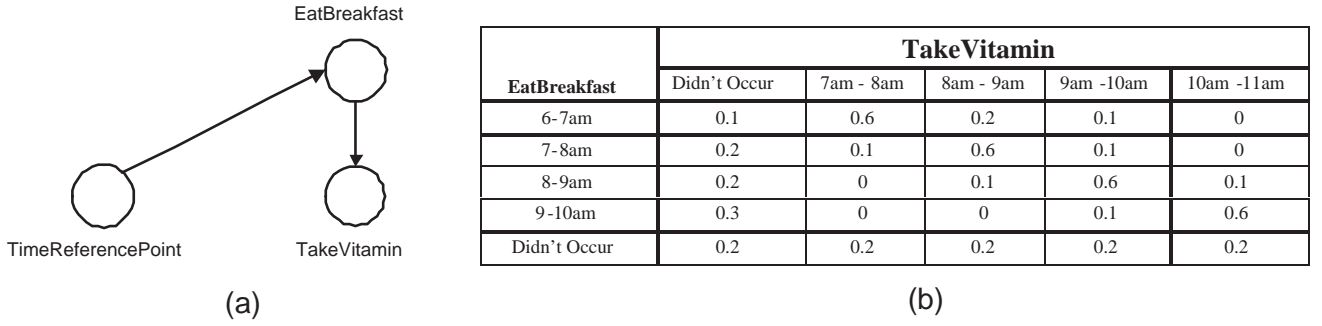


Figure 3: (a) A Time Net designed to represent temporal dependencies of actions.
(b) the conditional probability table for the TakeVitamins action.

causal and quantitative temporal relationships between actions. As described above, DBNs are well suited for the task of modeling causal relationships and the change in fluent values over time, while Time Nets effectively model quantitative temporal relationships. We define *Quantitative Temporal Bayes Nets* (QTBNs), which use interface functions to combine a simplified version of the Time Nets with an enhanced version of the DBNs. We first describe the structure of the individual components (Time Nets and DBNs) and how they are automatically derived from the plan to be monitored. Then we describe the interface mechanism that shares temporal information between the Time Net and DBN components of the model.

Components of a QTBN

We build a QTBN using information from the plan it monitors and from available sensors. As is typical in the planning literature, we model plans as a set of actions, their pre- and postconditions, causal links between actions, and temporal constraints between actions. Although we use the term “action” for consistency with the DBN literature, we are referring to action instances—steps of the plan. Each action occurs only once; if the plan contains two instances of the same type, then it contains two distinct actions, each of which is separately modeled in the QTBN. In our small example, TakeVitamin is an action, and occurs only once during the scope our model. In the Autominder system, the scope of a QTBN is one full day; the QTBN is reinitialized each morning. In a larger example, in which the client takes vitamins twice a day, the QTBN would model two different actions: TakeVitamin1 and TakeVitamin2.

In contrast, state properties—specifically the pre- and postconditions of actions—are modeled in the QTBN as fluents whose values change over time. InKitchen may become true and false multiple times during the day, but it can be represented as a single fluent. We treat actions and properties differently to minimize the size of the network: in order to perform execution monitoring, we need to be able to separately track each distinct action in the plan. But we do not need copies of the

state properties: it suffices to reason about the truth of each property at particular times.³

Name	Location	Node Values
$ATIMENODE$	$TimeNet$	$\{setof\ TimeIntervals\}$
$ATRANSIENT$	$DBNTimesliceT$	$\{true,false\}$
$ACUMULATIVE_T$	$DBNTimesliceT$	$\{true,false\}$
$ACUMULATIVE_{T+1}$	$DBNTimeslice_{T+1}$	$\{true,false\}$
$ATIN$	$DBNTimesliceT$	$\{true,false\}$

Table 1: Nodes required to model each action

Table 1 shows the set of nodes that are required by the QTBN to model each action in the plan. We will explain the purpose of each in the paragraphs below.

Time Net Component

The Time Net component of the QTBN models all temporal relationships that exist between events in the domain. It maintains probability distributions for each event that describe when each event is likely to occur. It can answer queries such as, “What is the probability that event A will occur in a interval I?” and “When is event A most likely to occur?”. This component is used to give the DBN component information about the likelihood of events occurring during particular intervals.

The Time Net component is automatically constructed by building a single node (identified as $ATIMENODE$ in Table 1) for every action in the plan and an arc between the nodes for every Temporal Constraint associated with the plan. One additional node, the TimeReferencePoint, indicates an arbitrary starting time (e.g., midnight) to ground quantitative temporal references. Figure 3a shows the Time Net for our example. Note that it is much simpler than the one shown in Figure 1 because it does not need to represent the whole

³This is actually a simplification. Sometimes it is necessary to distinguish among multiple instances of properties when they participate in quantitative temporal constraints. However, in general the number of such properties will be very small.

plan. To reason about quantitative constraints amongst actions (e.g., “X should happen now because Y happened 10 minutes ago”) it is sufficient for the Time Net to contain only the actions and temporal constraints amongst them.

Figure 3b shows the CPT for one of the TimeNodes, TakeVitamin. Each column in the CPT represents a specific interval of time. These TimeIntervals are sized based on the granularity that the action requires. Therefore, different actions can have different sets of possible values, as long as the intervals cover a continuous interval of time. Taken as a whole, the values assigned to each node represent a belief distribution over time. From these belief distributions, we can extract information about what has happened in the past and predict what will happen in the future. We can also extract information about the current time slice for use in our DBN component.

DBN Component

The DBN Component of the QTNB models the causal relationships between actions, domain properties and sensors. At any given time, the information in the DBN represents and reasons about a small interval of time, called the CurrentTimeInterval. The DBN retrieves information about the CurrentTimeInterval from the Time Net and then operates like a standard DBN.

To construct the DBN component, it is necessary to construct a structure containing both transient and persistent nodes for each action A in the plan, as illustrated in Figure 4. The node structure is centered on $A_{TRANSIENT}$, which represents the belief that action A is occurring in the current time slice. This node is influenced by the persistent node $A_{CUMULATIVE}$, representing the cumulative belief that A has or is currently occurring. Also influencing $A_{TRANSIENT}$ is the Temporal Influence Node (A_{TIN}), which summarizes all of the quantitative temporal beliefs about A for the CurrentTimeInterval. Unlike traditional DBNs, we take advantage of the observations made with DOOBNS and only copy persistent nodes into time slice T+1, i.e. only $A_{CUMULATIVE}$.

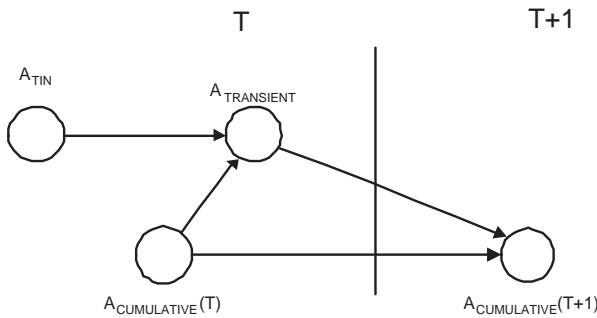


Figure 4: Generic DBN action structure

Once the nodes for each action have been created, we can add arcs to specify casual relationships between

actions and their pre- and postconditions. Quantitative temporal constraints are not modeled here; they are already represented in the Time Net.

Figure 5 shows the DBN for our example plan. It includes instantiations of the generic structure of Figure 4 for both of the actions in the plan. The node InKitchen does not represent an action, but rather a property of the current state (i.e., a fluent), so it is treated as a normal DBN node.

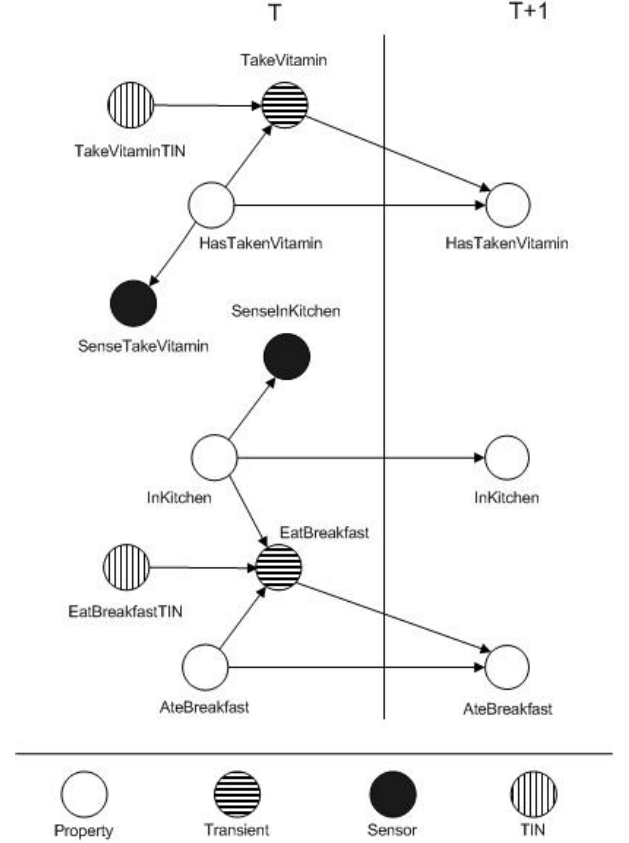


Figure 5: Example DBN Component

Every property node that represents a perceivable feature of the client’s environment can influence a sensor node. When our system is given sensory information, the sensor nodes are set in evidence, which influences the action’s transient node through the property nodes.

As stated earlier, a TIN can be considered a summary of all temporal information related to the action it influences and to the CurrentTimeInterval. A TIN is a binary node whose values are determined by querying the Time Net. The result of the query, “What is the probability that this event will occur in the CurrentTimeInterval?” is set to the TRUE value of the TIN. This query is performed by probing the associated TimeNode in the Time Net and extracting the portion of the returned probability distribution that corresponds to the current time interval. This process is

described more formally in the next section.

Intuitively, the value of a TIN can be viewed in the following way. Suppose you are trying to decide if it is time to start cooking breakfast. You check your clock, consider all the actions that must be performed before it is time to eat breakfast, and end up with a decision about whether to start breakfast now or wait. At the end of this reasoning, you may be 90% sure you want to start breakfast. This would be analogous to a value of 0.9 being assigned to the TIN influencing the ‘Start-Breakfast’ action. The TIN summarizes the result of reasoning with the temporal information in the Time Net, so the DBN does not have to consider the various temporal constraints. Given the TIN, the DBN can instead just reason about what is happening now.

TINs are updated differently from all the other nodes in a DBN or DOOBN. They are neither transient nor persistent. Rather than being rolled up or reinitialized each time there is a time slice transition, a TIN has its CPT updated by an interface function that relates the Time Net and DBN.

The Interface

The Time Net/DBN Interface provides an information channel between the DBN and Time Net through the various action nodes. This interface has two functions:

(1) UpdatePredictions, which extracts the summary temporal data from the Time Net to be used to influence reasoning in the DBN. (2) RecordHistory, which extracts data from the DBN for use as historical data in the Time Net.

The main algorithm for updating the model is shown in Figure 6. It is driven by two possible events: the addition of evidence (i.e. a new sensor value) or a time change. As with any DBN, the addition of evidence occurs any time the model receives new information; at such a point, Bayesian update and rollup are performed.

```

Main loop(evidence or time change)
  if evidence
    Record evidence in the DBN
    RollupDBN
  else if time change
    RecordHistory()
    UpdatePredictions()
    RollupDBN
  End if

```

Figure 6: Main Algorithm

A time change only occurs when the actual clock time crosses the boundary of two values in one of the TimeNodes in the Time Net. For example, if ‘EatBreakfast’ in the Time Net has the values (time intervals) 7-8am, 8-9am, and 9-10am, then time changes would occur at 7, 8, 9, and 10. If another Time Net node had time intervals 7-8:30 and 8:30-10, then an additional time changes would occur at 8:30.

Whenever a time change occurs, information flows between the TIN and the DBN. It flows from the DBN

to the Time Net via the RecordHistory interface function, which updates the TimeNodes’ CPTs to reflect beliefs about what has occurred since the last time change. The UpdatePredictions interface function carries information from the Time Net to the DBN—specifically to a TIN—so that the TIN always reflects the summary information the actual current time interval.

Let us consider this information transfer in a bit more detail. Every time a time-interval boundary is crossed for some action, a different column of the CPT for that action’s TimeNode (in the Time Net) becomes valid. For example, if we use the Time Net CPT from Table 3(b), then from 7:00 until 8:00, the relevant probability distribution for the occurrence of TakeVitamin is the second column of the table. At 8:00, the third column becomes relevant. The TIN in the DBN encodes the result of quantitative temporal reasoning done using the relevant distribution. If we assume that client ate breakfast at 7:45 and that we have not yet observed the client taking a vitamin, then when 8:00 arrives, the probability that the client will take a vitamin is 60%. This value can be extracted from the Time Net using the algorithm in Figure 7, which queries each TimeNode, obtains the belief distribution for the represented action, selects the value corresponding to the current clock time, and sets the TIN CPT in the DBN accordingly.

```

UpdatePredictions()
  For each A ∈ Actions
    ProbDistribution ← bayesNetQuery(ATIMENODE)
    p ← getProbForValue(ProbDistribution, CurrentTimeInterval)
    ATIN ← {p, 1-p}

```

Figure 7: Update Predictions

However, a one-way flow of information is not sufficient. Before we call UpdatePredictions to update the TIN values at the beginning of the new time interval, we need to execute the RecordHistory algorithm, which records the information inferred by the DBN about the current time interval as history in the Time Net. For example, if an action’s Cumulative Belief node has increased from 0.2 to 0.35 throughout the 7-8am time interval, we need to update that action’s TimeNode to reflect the following fact: we believe that the event occurred in the interval 7-8am with probability 0.15 (0.35-0.2). After this update, any query of the event’s TimeNode will show 0.15 in the 7-8am interval. The algorithm for RecordHistory is shown in Figure 8.

```

RecordHistory()
  For each A ∈ Actions
    ProbDistribution ← DBN.Query(APROPERTY(T+1))
    p ← getProbForValue(ProbDistribution, True)
    ATIMENODE ← adjustCPTwithNewBelief(ATIMENODE, p)

```

Figure 8: RecordHistory Algorithm

The function adjustCPTwithNewBelief replaces each

value in the CPT column representing the current time interval with the value p . This ensures that any query to the TimeNode will return a distribution with the value p for that time interval. However, by adjusting the time interval probability, the sum of the row will change and will violate the requirement that all values in each row must sum to 1. (This is a requirement in all Bayes Nets). We address this by adjusting the remaining values (i.e. the future time intervals) in the row to restore the sum to 1. Essentially, we distribute the difference between p and the original value over the remaining time intervals. In our system, we distribute this difference proportionally to the original beliefs. However, one can imagine other ways of distributing this difference. We hope to evaluate other methods in future work.

Performance

Bayesian inference (and thus inference in Time Nets, DBNs, DOOBNs, and QTBNs) is known to be an NP-hard problem. However, we hope that in “reasonable” domains, inference will be tractable and we will be able to obtain feasible computation times. We consider the Autominder domain to be a reasonable one. To study performance issues, we implemented a system to construct and update QTBNs in Java running on a Pentium4 1.7GHz processor with 256Megs of RAM. Experimental testing of the time and space needs of the algorithm involved developing plans for our domain and then testing our model based on these plans. For analyzing performance, these test plans do not necessarily have to be valid plans, so we used random methods to build our plans and then constructed our QTBN based on the random plans. The random methods were parameterized by the following conditions that encompass the plan’s complexity with respect to our model:

- NumTimeIntervals- The number of time intervals used by the actions in the Time Net.
- NumActions - The number of actions in the plan.
- ClperAction - The average number of causal links per action.
- TcperAction - The average number of temporal constraints per action

Using these variables, our plan generator creates a plan with the given number of actions and divides its required time of performance into the number of intervals specified. Pairs of actions are then randomly picked to form the causal links and temporal constraints. Each of these links is connected to two of the existing actions. We pick which actions they are connected to at random to give some variation in the trials. From these parameters, we developed the following base case that is plausible for the Autominder domain.

- NumTimeIntervals - 100
- NumActions - 25
- ClperAction - 0.5

- TcperAction - 0.5

Once the QTBN has been constructed, we force a time change event, which causes both interface functions to run as well as a DBN rollup. We then record the running time of each function for each trial.

Experiment 1 - Base Case

In the first experiment, we ran the base case trial 50 times to establish an average run time. The average run time of the 50 trials is 2.5 minutes with a standard deviation of .46 minutes. This average may be acceptable for some Autominder situations-after all, change tends to occur very slowly in this application. However, more work is clearly needed to increase the efficiency of QTBN update.

Experiment 2 - Time Complexity Profile

To better understand where we can make efficiency gains, we conducted an experiment in which we varied NumActions, and we profiled the performance of the three main processes: DBN Rollup, RecordHistory, and UpdatePredictions (which includes propagation in the Time Net). We varied the number of actions per plan from 0 to 50, and used base-case levels for the other conditions. The results are shown in Figure 9 (UpdatePredictions takes a trivial amount of time and is not shown). Not surprisingly, inference in the DBN component dominates the total time. This is because our DBN is more complex than our Time Net and rollup is more costly than simple Bayes Net updating. Also note that the interface algorithms that we have implemented are very cheap compared to the DBN rollup time.

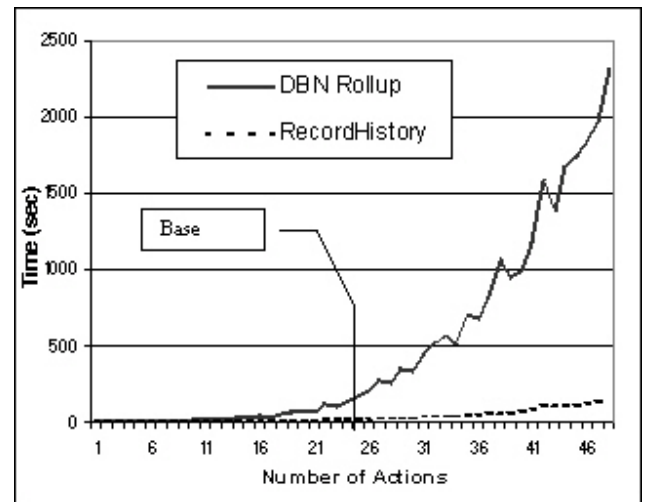


Figure 9: Number of Actions vs. Inference time.

Theoretically, growth of DBN inference time is exponential with respect to the number of actions in the plan. In the preliminary experiment shown here, growth is only polynomial, but still very costly.

Experiment 3 - Other Factors on Growth

Experiment 2 varied the number of actions in the plan. In Experiment 3, we fixed the number of actions to the baseline (25 actions) and varied the number of temporal constraints. This variance has no effect on the inference time of the DBN rollout or RecordHistory because temporal constraints only add complexity to the Time Net component. Figure 10 shows the relationship between the TcperAction condition and the UpdatePredictions running time. Unsurprisingly, there is large order increase in the inference time as TcperAction increases. Notice, however, that the inference time for UpdatePredictions remains insignificant (less than 1 second) compared to both DBN rollout and RecordHistory.

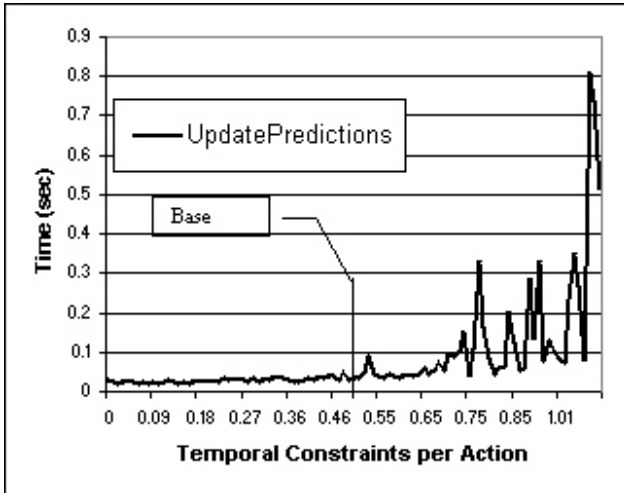


Figure 10: Number of Temporal Constraints per Action vs. Inference time.

Discussion

As just shown, the computational time increases for rollout and the interface functions as the number of actions and the complexity of the plan structure increased. Memory usage can also be an issue when the client model is initially built from the client plan. Although most of the time the base case was not a problem, even with these settings we occasionally ran out of memory. The frequency of this problem increased as the NumTimeIntervals and/or Tcperaction increased. This occurred because the size of the CPTs associated with the Time Net actions grow exponentially with the number of temporal constraints in the actions. Thus, our initial experiments show that while QTBNs can be used to solve some problems, especially in domains in which change is not rapid, more work on increased efficiency is required before the framework can be used as widely as we would like. The efficiency challenge we face is not unique to QTBNs: the bulk of processing time is consumed by DBN update. Current research on making DBNs more efficient, such as the use of DOOBNS,

is thus directly relevant to our effort. We discuss this a bit more in the next section.

Conclusions

In this paper, we have presented an approach to monitoring the execution of plans with quantitative, as well as qualitative, temporal constraints. We argued that previous approaches to temporal reasoning under uncertainty are not adequate for this purpose, and we developed a new reasoning framework, QTBNs, which combine the ability of Time Nets to reason about arbitrary quantitative temporal constraints with the ability of DBNs to reason about the changing value of fluents. QTBNs consist of three components: a dynamic model that represents the evolution of events over time, a model that reasons about quantitative temporal relationships between events, and interface functions that manage the flow of temporal information between the two models.

We have fully implemented a working version of an execution monitor, and have integrated it with the other components of Autominder. Our preliminary testing indicates that the model is accurate, and is efficient enough for the Nursebot Autominder domain. QTBNs can also be applied to other domains that maintain plans with quantitative temporal relationships. For example, autonomous mobile robots need to reason about how the world evolves over time when choosing their actions and reasoning about the outcome of their actions.

Although designed specifically for execution monitoring, QTBNs are not limited to reasoning about plans. In general, they reason about causal and temporal relationships between events and domain properties, making them applicable to expert systems as well. One class of expert systems, Medical Decision-Support Systems, has a particular need to model time (Aliferis *et al.* 1997).

We have given three items priority in our plans for future work:

- Proving the validity of the model.
- Reducing the memory requirements of the Time Net.
- Increasing the efficiency of query processing.

Although we have encouraging experimental evidence that our system is correctly modeling our domain, we have no proof that the interface functions we defined correctly transfer information between the DBN and Time Net components of our model. Currently, we are looking for transformations between QTBNs and proven, yet inefficient techniques. We hope this exercise will uncover the extent (if any) of the information loss/distortion that occurs using our interface functions.

As discussed in the previous section, the memory requirements of the Time Net can be crippling. We are developing an alternate representation for the information in the Time Net that only requires space linear in the number of nodes, intervals, and temporal

constraints. The representation we have developed is equivalent to that of the Time Net, but an inference method has not yet been designed.

To improve efficiency, we are incorporating the granularity parameter of DOOBNS (Friedman, Koller, & Pfeffer 1998) as mentioned above. This additional parameter will help prevent drift, as well as increase the speed of inference by reducing the number of nodes that are propagated during each update of the dynamic part of the network. In our current implementation, each action node in the Time Net component contains the same values representing fixed intervals of time. Adding the granularity parameter will allow the designer to independently specify the divisions of time for each action. Because the memory requirements in the Time Net depend on the number of time intervals represented, this flexibility grants the designer some control over memory usage.

Koller et al. (Friedman, Koller, & Pfeffer 1998) also suggest another method for reducing the number of nodes: posting a guard on some objects. A guard is associated with an object and has a Boolean test that is made during the building of the model. An object whose guard is set to false is omitted from the model. In the Nursebot Autominder domain, there are many situations where it is impossible for an action to occur at a specific time. For example, we could easily say that a client cannot eat lunch before 10:00am. If she eats before 10:00am, she is eating breakfast or a snack. In this case a guard could be put on the lunch node to prevent it from being activated before 10:00am.

Our expectation is that by combining these new ideas, we will create a model that will only need to roll up a fraction of the modeled events at any one time. This would mean that our model would not be limited by the total number of events that must be modeled, but would depend only on the number of events that must be reasoned about at any specific time.

Acknowledgements

This research was supported by the National Science Foundation (IIS-0085796) and by the Air Force Office of Scientific Research (F49620-01-1-0066). The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR or the U.S. Government.

References

- Albrecht, D. W.; Zukerman, I.; and Nicholson, A. E. 1998. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction* 8(1-2):5–47.
- Aliferis, C. F.; Cooper, G. F.; Pollack, M. E.; Buchanan, B. G.; and Wagner, M. M. 1997. Representing and developing temporally abstracted knowledge as a means towards facilitating time modeling in medical decision-support systems. *Computers in Biology and Medicine* 27(5):411–434.
- Baltus, G.; Fox, D.; Gemperle, F.; Goetz, J.; Hirsch, T.; Magaritis, D.; Montemerlo, M.; Pineau, J.; Roy, N.; Schulte, J.; and Thrun, S. 2000. Towards personal service robots for the elderly. *Workshop on Interactive Robots and Entertainment (WIRE 2000)*.
- Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.
- Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.
- Friedman, N.; Koller, D.; and Pfeffer, A. 1998. Structured representation of complex stochastic systems. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)* 157–164.
- Huber, M. J.; Durfee, E. H.; and Wellman, M. P. 1994. The automated mapping of plans for plan recognition. *UAI94 - Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence* 344–350.
- Kanazawa, K. 1991. A logic and time nets for probabilistic inference. *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI)* 5:142–150.
- Pollack, M. E., and Horty, J. F. 1999. There's more to life than making plans: Plan management in dynamic, multi-agent environments. *AI Magazine* 20(4):71–84.
- Pollack, M. E.; McCarthy, C. E.; Ramakrishnan, S.; Tsamardinos, I.; Brown, L.; Carrion, S.; Colbry, D.; Orosz, C.; and Peintner, B. 2002. Autominder: A planning, monitoring, and reminding assistive agent. *7th International Conf. on Intelligent Autonomous Systems*.
- Stergiou, K., and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120:81–117.
- Vila, L. 1994. A survey on temporal reasoning in Artificial Intelligence. *AI Communications* 7(1):4–28.