J. of Mult.-Valued Logic & Soft Computing., Vol. 12, pp. 391–407
Reprints available directly from the publisher
Photocopying permitted by license only

ML-TIPN: An Algorithm for Automated Acquisition of Domain Models based on Time Interval Petri Nets

VADIM BULITKO[1] AND DAVID C. WILKINS[2]

[1]*Department of Computing Science, University of Alberta, Edmonton, AB T6G 2E8, Canada*
E-mail: bulitko@ualberta.ca
[2]*Center for the Study of Language and Information, Stanford University, Stanford, CA 94306*
E-mail: dwilkins@stanford.edu

Received: May 5, 2005, In Final Form: September 30, 2005

Extended Petri Nets have been applied to artificial intelligence reasoning processes, in areas such as planning, uncertainty reasoning, knowledge-based intelligent systems, and qualitative simulation. Creating Petri Net domain models faces the same challenges that confront all knowledge-intensive AI performance systems: model specification, knowledge acquisition, and refinement. Thus, a fundamental question to investigate is the degree to which automation can be used. This paper formulates the learning task and presents the first machine learning method for Time Interval Petri Net (TIPN) domain models. In a preliminary evaluation within a damage control domain, the method learned a nearly perfect model of fire spread augmented with temporal and spatial data.

Keywords: domain model learning, Petri net learning, spatial-temporal data series learning, real-time decision-making, automated damage control.

1 INTRODUCTION

Historically, Petri Nets and their extensions have been used to model concurrent processes in domains such as communication networks, operating systems, and job scheduling [1–5]. More recently, Petri Nets and their extensions, such as Time Interval Petri Nets (TIPNs), have been applied to artificial intelligence reasoning processes, in planning, uncertainty reasoning, intelligent systems [6, 7], qualitative simulation and modelling [8, 9]. TIPNs are especially suitable as fast qualitative-level models for concurrent

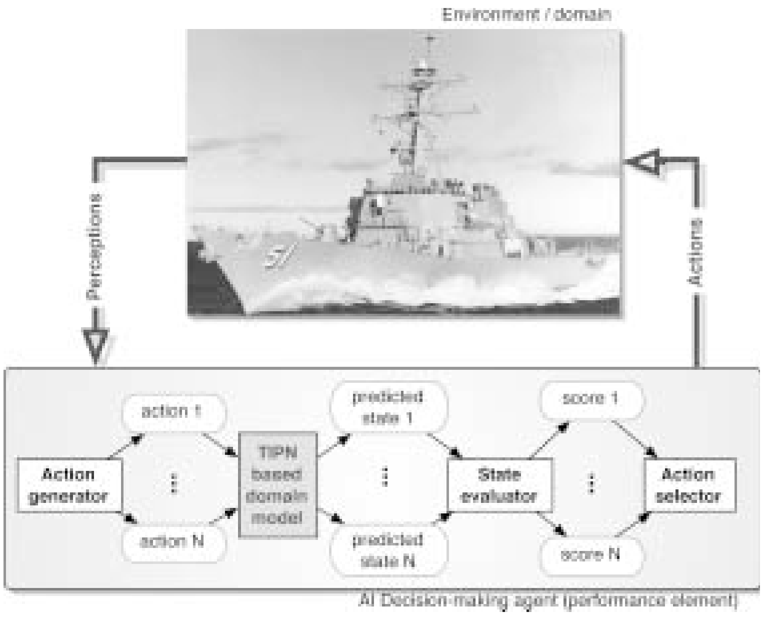

FIGURE 1
An AI decision-making agent in the shipboard damage control environment. A TIPN based domain model is used to predict effects of candidate actions toward selecting the best one.

temporally and spatially referenced processes such as the ones in real-time strategy games, traffic control, and crisis decision making. One example of a TIPN application is a decision-making system for the DC-Train real-time shipboard damage control training environment [9]. On 160 simulated difficult scenarios that involved fire, smoke, flooding, and machinery failure, the TIPN-based decision-making system saved 117 ships (73.1%) [9]. This is a substantial improvement over the performance of human experts, who saved 28 of 160 (17.5%) ships. The system employed TIPNs for qualitative simulation of effects of the proposed crisis responses and used them to select the most appropriate course of action (Figure 1). [10] has been exploring applications of TIPNs to modelling genetic regulatory networks in biology. The previously used model [11] is based on Bayesian networks and does not account for temporality.

At the present, Time Interval Petri Net based models need to be hand-crafted by subject matter experts. Thus, the ability to learn TIPNs automatically from historical data and domain theory would greatly improve applicability of the formalism. To illustrate: records of past scenarios are available in the domain of shipboard damage control. Likewise, microarray time series data can be used for TIPN learning in the study of biological genetic regulatory networks.

In this exploratory paper, we review the arguments as to why Time Interval Petri Nets are an interesting and useful representation for domain models, formulate the learning task, discuss applicability of the existing machine learning techniques, present the first method of learning Time Interval Petri Nets, conduct an initial empirical evaluation, and outline an agenda for the future research. This paper is an extended version of [12].

2 TIME INTERVAL PETRI NETS AS DOMAIN MODELS

Time Interval Petri Nets have been used in AI as causal models of spatial and temporal dynamics in a system or domain of interest. One of their appeals lies with the ability to represent uncertainty along the temporal dimension. That is useful when timings of the events are not known precisely. Unlike Monte Carlo simulation based approaches, such as the ones used in Time Coloured Petri Nets by [2] and requiring mutliple simulation runs to establish a distribution of possible outcomes, Time Interval Petri Nets use temporal *intervals* as time stamps of the events modelled.

In order to make the machine learning contributions of this paper self-contained, the remainder of this section will introduce TIPNs informally via an example. Guidelines for TIPN use, comparisons to other qualitative reasoning models such as the Qualitative Process Theory (QPT) by [13], and an axiomatic introduction can be found in [9].

In order to model the process of fire spread with traditional modelling tools, one needs to set up a numeric simulator to compute combustible material distribution, gas zones, plumes, heat transfer, wall temperatures, ignition properties, and fire suppressant effects. This is the approach taken in [14]. Since fire fighting personnel are involved in the process, a behavioral simulator is needed to model the decision making of each crew member, and aspects of their fire-fighting performance such as temporal delays caused by travel time between places in the ship. The inherent complexity of these types of simulation prevents multiple alternatives extending 20 minutes into the future from being simulated in a matter of seconds at the resolution required, hence the motivation for the TIPN approach. As with all abstractions of first-principle models, the TIPN model is not as accurate, but it provides sufficient accuracy in a limited amount of time to support damage control decision making [9].

The process of constructing a TIPN begins with a model of qualitative-type patterns augmented with temporal information that underlie the phenomena in question. Using this approach, the complex phenomena outlined in the previous paragraph are abstracted into the following domain dependency (further simplified for illustration purposes):

"if compartment X is hot and the fire boundaries around X are not set then the fire will spread to neighbor compartment Y in 3 to 4 minutes."

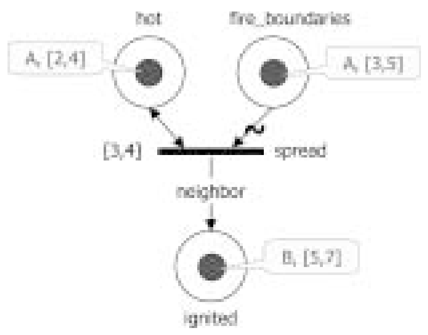

FIGURE 2
A simple Time Interval Petri Net (TIPN) qualitatively modelling fire spread with spatial and temporal information.

In the first order predicate calculus this model can be expressed as follows (we use Prolog notation henceforth):

```
ignited(Y,Tnew)  :-
   hot(X,Told),
   not fire_boundaries(X,Told),
   neighbor(X,Y),
   delay(Tnew,Told,3,4).
```

Here predicate `ignited(Y,Tnew)` holds when compartment `Y` is ignited at time `Tnew`. Likewise, predicate `hot(X,Told)` indicates the fact that compartment `X` is of a high temperature at time `Told`. Predicate `fire_boundaries(X,Told)` establishes that fire boundaries are set around compartment `X` at time `Told` (i.e., that the compartment walls are being cooled down to prevent fire spread). Predicate `neighbor(X,Y)` holds if and only if compartment `X` is adjacent to compartment `Y` thereby specifying the ship topology. Finally, predicate `delay(Tnew,Told,3,4)` tells us that `Tnew` and `Told` are between three to four minutes apart.

The corresponding TIPN is shown in Figure 2. It consists of three *places* (large circles) that correspond to predicates `hot`, `fire_boundaries`, and `ignited`. In general, TIPN places describe *attributes of system state*. The *transition* (horizontal bar) links them together and corresponds to a *change in system state*. The two input places (shown above the transition) and one output place (shown below the transition) are linked to the transition by *arcs* (shown as arrows) which express *causality of the system*. Note the tilded inhibitory arc that disables (inhibits) the transition from "firing" when the disabling condition (i.e., `fire_boundaries`) is *satisfied*.

While TIPN places represent state attributes and transitions correspond to state changes, TIPN *tokens* (shown as smaller filled circles in the diagram) reside in places and describe a particular *state* the modelled system is in. The assignment of tokens is called *marking* and evolves

TABLE 1
Learning Time Interval Petri Nets

Domain elements	**TIPN model elements**	**Role**
State attributes	Places	Provided
State changes	Transitions	Provided
Possible causality	Possible arcs	Provided
Actual causality	Actual arcs	Learned
Spatial dependencies	Output arc expressions	Learned
Temporal dependencies	Delay intervals	Learned

together with the system's state. Presence of a token in a place indicates that the particular attribute/condition is *satisfied*. For instance, the token `A,[2,4]` in the enabling place `hot` represents the fact that compartment `A` has become critically hot between 2 and 4 minutes scenario time. Likewise, token `A,[3,5]` in the disabling place `fire_boundaries` represents the fact that fire boundaries were set on compartment `A` between 3 and 5 minutes into the scenario thereby disabling the fire spread at that time. Under the firing logic assumptions adopted in [9], the transition will be enabled in the time window [2,3] and the fire will spread from compartment `A` to compartment `B`. Correspondingly, a new token is deposited in the output place `ignited` representing compartment `B` catching on fire between 2+3=5 and 3+4=7 minutes scenario time. Note that *output arc operator* `neighbor` changes the compartment label from `A` to `B`. The fire spread *delay* of [3,4] is shown to the left of the transition.

3 LEARNING TASK

As motivated in the previous sections, a natural and important application of machine learning lies with *automating synthesis and verification* of Time Interval Petri Net models. The resulting benefits include: (i) much reduced requirements for human expertise, (ii) accelerated design cycle and therefore improved cross-domain portability of the enveloping decision-making system, and (iii) greater confidence in the models produced.

Frequently, the designer of an AI decision-making system knows the relevant attributes to describe domain states (e.g., compartment identifier, compartment state, etc.) and knows possible state changes (e.g., engulfment, fire spread, explosion, etc.). This information forms a *domain theory* for TIPN learning.

On the other hand, a complete causal model (e.g., the fact that lack of fire boundaries and high temperature cause fire to spread), spatial information (e.g., likely fire spread trajectories), and temporal data (e.g., the fact that such a spread typically takes three to four minutes) are often unavailable as summarized in Table 1. In this table, "possible arcs" specifies a set of possible connections between places and transitions; and "actual arcs" have the additional information of directionality (unidirectional or bidirectional) and whether the arc is enabling or inhibitory. This information is recovered from past scenario traces (i.e., the *historical data*) during the learning process.

The learning task for TIPNs can be thus defined as automatic synthesis of a causal domain model expressed as a collection of Time Interval Petri Nets from the available domain theory and historical data. The desired attributes of a TIPN learning process include the ability to harness expert knowledge as well as noisy and incomplete historical data; and a facility for specifying the learning bias (e.g., accuracy of TIPN models vs. model size). In the following, we will use these attributes to discuss shortcomings of the existing methods available for learning Petri Nets and then introduce a novel algorithm designed to address the difficulties.

4 DISCUSSION OF EXISTING METHODS

Due to the recent nature of Time Interval Petri Nets, there are presently no methods for TIPN learning other than the one proposed in this paper. Thus, we discuss related research in the following order: (i) synthesis methods developed for *other flavors* of Petri Nets; (ii) inductive logic programming (ILP); (iii) automated first-order theory refinement methods. For each of them we indicate the difficulties with respect to TIPN model learning.

Main-stream Petri Net synthesis research centers around recovery of the Petri Net given its reachability graph [15–18]. The problems with using these methods for TIPN model learning include: (i) required consistency (i.e., no noise tolerance) and completeness of the learning input (i.e., no generalization is attempted), (ii) applicability to propositional Petri Nets only (i.e., no first-order representation), (iii) user's inability to control the resulting TIPN topology.

Time Interval Petri Nets have a natural connection to first-order representations such as predicate logic clauses and Prolog programs as demonstrated by [9]. Therefore, one can use machine learning methods for such representations to induce them first, followed by a conversion to TIPNs. Our study of TIPN learning via the Inductive Logic Programming tool called Progol [19] revealed considerable difficulties with: learning recursive predicates that are a natural representation of chain events such as progressive fire spread, and user's inability to explicitly supply their domain knowledge as a starting point of the TIPN model to be learned.

Automated first-order theory refinement methods such as in [20] address the last problem by taking a user-supplied and possibly incorrect initial theory and refining it with historical data. The following are the challenges of using this approach for TIPN model learning: (i) consistency of the historical data set is assumed but is frequently unavailable in real-life scenario logs due to noise and operator errors. Thus, a domain theory perfectly fitting the noisy data may not exist at all; (ii) the theory produced is made similar to the user-supplied theory through a number of minor syntactic changes which may not translate to similarity of the learned TIPN and the user-supplied TIPN; (iii) the trade-off between the accuracy and the size of the theory produced is not explicitly user-controlled; (iv) the single best theory is maintained while the user often wishes to select from several good TIPNs at the end; (v) qualitative models are learned as single conjunctive clauses with no temporal or spatial *quantitative* support.

Grammar based first-order theory refinement methods such as the one by [21] would face similar issues if applied to TIPN learning. Additionally, while their learning input requires a grammar to define the search space *and* an initial domain theory as a starting point of the refinement process, we combine these two together in an intuitive incomplete TIPN model. Finally, we make the refinement process fully autonomous by concentrating the user's problem-solving expertise in the learning input.

5 ML-TIPN LEARNING METHOD

In light of the difficulties with the use of existing methods, we have developed a novel approach specifically for learning TIPN-based domain models. The method addresses all of the problems discussed above as it: (i) takes a user-supplied domain theory, (ii) refines it using noisy and inconsistent historical data, (iii) allows the user to control the properties of the model produced, (iv) learns fully fledged TIPNs including the temporal and spatial causality, and (v) produces more than one TIPN model should the user so request. We will refer to the algorithm as ML-TIPN.

The outline of ML-TIPN is presented in Figure 3. We will describe its inputs in Section 5.1, its scoring metric in Section 5.2, and the algorithm itself in Section 5.3. Operation of ML-TIPN is illustrated with a hand-traceable example in Section 6.1.

5.1 Representation of Learning Input

ML-TIPN's input consists of domain theory P_0 and historical data $\{e_i\}$, $\{e'_j\}$, here and below we follow the notation of Figure 3. Domain theory is encoded as one or more *incomplete* Time Interval Petri Nets. Such representation enables the user to specify the lexicon and known causality information (if any) while leaving the unknown parts for ML-TIPN to learn. Technically, an incomplete TIPN is a Time Interval Petri Net where certain arcs are labeled “unknown” and can be of any standard type (e.g., inhibitory or enabling). ML-TIPN will “refine” such possible arcs into one of the basic types. Additionally, operators on the output arcs can be left open. ML-TIPN will then attempt to fill such an operator with a table of records of the

```
ML-TIPN
INPUT:
        - domain theory: incomplete TIPNs: $P_0$
        - historical data events:
            - training: $e_1, \ldots, e_N$
            - validation: $e'_1, \ldots, e'_{N'}$
        - scoring metric
        - desired number of final TIPNs ($M$)
OUTPUT: $M$ completely specified TIPNs
 1   initialize the beam width to $B_1$
 2   for $t = 1, \ldots, N$ do
 3       reset the new population of TIPNs $P_t = \emptyset$
 4       retrieve event $e_t$ from the historical data
 5       for each TIPN $X$ in the previous population $P_{t-1}$ do
 6           if event $e_t$ is external then
 7               update TIPN $X$'s marking to reflect $e_t$
 8               add TIPN $X$ to the new population $P_t$
 9           else
11               refine TIPN $X$ according to event $e_t$
12               add the resulting TIPNs to the population $P_t$
13           end if
14       sort population $P_t$ by the scoring metric on $e'_1, \ldots, e'_{N'}$
15       retain the top $B_t$ TIPNs in population $P_t$, discard the rest
16       update the beam width: $B_t$ to $B_{t+1}$
17   end for
18   fully refine TIPNs in population $P_N$, put them in $P_{N+1}$
19   output the $M$ top-ranked TIPNs in $P_{N+1}$
```

FIGURE 3
The ML-TIPN learning algorithm.

type "input token label $\rightarrow$ output token label" (as illustrated in the bottom boxes of Figures 5 and 6). Finally, the temporal delay intervals can be left for ML-TIPN to fill in as well.

Historical data are represented as a series of event records. Each record describes an actual event in terms of token introduction or removal from a particular place in the TIPN model. Specifically, an event is represented as a 5-tuple `<place, token, time interval, +/-, ext/int/unknown>` where `place` is the TIPN place the `token` was put in or removed from, `time interval` indicates when `token` is believed to acquire its current attributes and place, `'+'` denotes token introduction and `'-'` stands for removal, and finally `ext/int/unknown` indicates whether the event is *external* (i.e., the token came from outside of the TIPN), or *internal* (i.e., the token introduction/removal is a result of TIPN operation), or *unknown* which may be either of the two. As an example, consider the following event: `<fire_fighting, [compartment, 'A'], [team, R5], [3:27, 3:45], +, ext>`. It represents the fact that the token `[compartment, 'A'], [team, R5]` with the timestamp of `[3:27,3:45]` was *introduced* into the place `fire_fighting` and it came from *outside* of the given TIPN. In English the record reads:

"A firefighting team R5 started to fight fire in compartment A sometime between 3:27 and 3:45".

ML-TIPN takes two sets of historical data: a training set $e_1, \ldots, e_N$ of N events and a validation set $e'_1, \ldots, e'_{N'}$ of N' events. The training set is used to update TIPN models (line 11) which are then evaluated on the validation set (line 14).

5.2 Scoring Metric

Usually TIPN models are used within larger AI systems. Therefore, one of the most important attributes is whether the model learned by ML-TIPN improves the system performance. While being conceptually simple, such a wrapper approach would not be practically feasible as frequent evaluations of candidate TIPN models via running the entire performance element are cost-prohibitive. Therefore, we adopt a filter approach and assess the quality of a TIPN model with the following computationally less expensive scoring metric.

ML-TIPN's score of a Time Interval Petri Net is inversely proportional to a weighted sum of the following five terms: (i) the total number of known arcs in the TIPN, (ii) the total duration of all transition delays, (iii) the total size of all output arc operator tables, (iv) the total number of false positives (i.e., the events that the TIPN predicts but which were not recorded as a part of the historical data), (v) the total number of false negatives (i.e., the events that the TIPN does not predict but which were indeed recorded as a part of the historical data). According to the scoring metric, larger TIPN models are penalized more as well as the models that are less accurate with respect to the validation set of historical data. Additionally, the user has control over the weights of the individual terms in the scoring metric. This provides a way of specifying the learning bias. For instance, increasing the weight of component (i) above will encourage ML-TIPN to seek more compact models even if they are less accurate.

5.3 Learning Process

In this section we focus on the details of ML-TIPN important for replicating this work. We will use the line numbers of Figure 3 throughout the section. ML-TIPN learns by conducting a heuristic beam search in the space of all TIPNs that are refinements of the domain theory (i.e., of the incompletely specified initial TIPNs P_0). The learning process is carried out in N iterations (line 2) where N is the number of events in the training data $e_1, \ldots, e_N$. At each iteration t, event e_t is retrieved from the historical data in chronological order (line 4). Then the population P_{t-1} of B_t partially specified TIPN models is refined into the new population P_t (lines 5-12).

Specifically, if event e_t is external (i.e., is not to be modeled by the TIPN being learned) then ML-TIPN merely updates the marking of every TIPN in population P_{t-1} (line 7) and copies them all to the new population P_t (line 8). If the event e_t is internal (i.e., is to be modeled by the TIPN being learned) then ML-TIPN uses the set of the refinement operators to modify each TIPN in the population P_{t-1} in an attempt to make them able to model the event (line 11). Specifically, four types of refinement operators can be applied to a TIPN: (i) converting arc's type from 'unknown' to one of the standard types, (ii) extending the time delay of a transition, (iii) adding a record to an output arc operator table, and (iv) leaving the TIPN intact.

For instance, if an event indicates a fire spread from compartment A to compartment B, all TIPNs in the population that model fire spread can get their output arc operator updated with the record $A \rightarrow B$. Another example: suppose there is an unknown-type arc from the place "Hot" to transition "Engulfment" and the latter leads to the place "On Fire". Suppose also that compartment C is known to be hot (i.e., there is a corresponding token in place "Hot"). Then if ML-TIPN observes an internal event of compartment C catching on fire, it can refine the arc's type from unknown to enabling.

Note that there may be several ways of refining a particular TIPN so that it models the event at hand. All such refinements will be performed in line 11, each producing a refined TIPN. All of these refinements are put in the new population P_t (line 12). Then ML-TIPN will sort the new population P_t using the scoring metric and the validation set $e'_1, \ldots, e'_{N'}$ (line 14) and retain B_t top-ranked ones (line 15). The beam search width B_t is then updated (line 16). Similar to simulated-annealing search, the beam width is gradually decreased as higher quality TIPNs are expected to reside in P_t.

After N iterations, the training data $e_1, \ldots, e_N$ will be exhausted. ML-TIPN will then refine the final population P_N in all possible ways (line 18), sort the resulting TIPNs on the validation set using the scoring metric, and output the M top-ranked of them (line 19). Note that the number M is supplied by the user giving him/her an opportunity to hand-pick from several learned TIPNs.

There are three substantial differences between genetic algorithms [22], there are three substantial differences. First, our search is deterministic while genetic algorithms use stochastic operators of mutation and cross-over. Second, while the search in genetic algorithms is guided by a fitness function, our search is controlled by a fitness function (i.e., ML-TIPN's metric) *and* the training data. Finally, in ML-TIPN the initial domain theory not only constrains the search space but also defines a better non-random starting point of the search.

6 EMPIRICAL EVALUATION

In this section, three different experiments are described. We begin by illustrating the learning process with a simplified fire spread phenomenon.

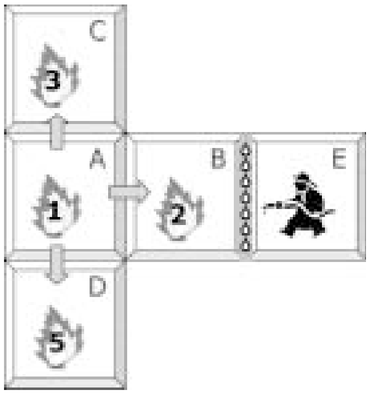

FIGURE 4
A simple fire spread scenario. The numbers correspond to the ignition times.

TABLE 2
Event record for the simple fire spread scenario

Event	Description
`<hot,A,[1,1],+,external>`	primary damage in compartment `A`
`<fbs,B,[2,2],+,external>`	fire boundaries set on `B` prevent fire spread from `B` to `E`
`<hot,B,[2,2],+,internal>`	fire spreads from `A` to `B` with a 1-min delay
`<hot,C,[3,3],+,internal>`	fire spreads from `A` to `C` with a 2-min delay
`<hot,D,[5,5],+,internal>`	fire spreads from `A` to `D` with a 4-min delay

Then ML-TIPN is evaluated on a more complex fire spread learning task. The section is concluded with a comparison between ML-TIPN and an inductive logic programming approach.

6.1 Learning a Simple Fire Spread Model

The first test of the ML-TIPN learning algorithm illustrates learning a TIPN model of greatly simplified fire spread in the domain of ship damage control. The input scenario involves a fire spread in five adjacent compartments (`A`, `B`, `C`, `D`, `E`) as illustrated in Figure 4. The corresponding event record ($e_i = e_i'$ for all i), which is the input to the ML-TIPN algorithm, is shown in Table 2. For the sake of simplicity and space, we will start with the prior domain theory P_0 in the form of the two-element TIPN set shown in Figure 5. Additionally, we will set the beam width B_t to ∞ for all t. The left TIPN in the figure represents the hypothesis that fire boundaries have no effect on fire spread while the right TIPN supposes that they disable fire spread. Note that both candidates are incompletely specified in terms of the temporal information and arc operators. Namely, the arc operator is extendable and already has one record in it: `B` $\rightarrow$ `E` meaning that the user

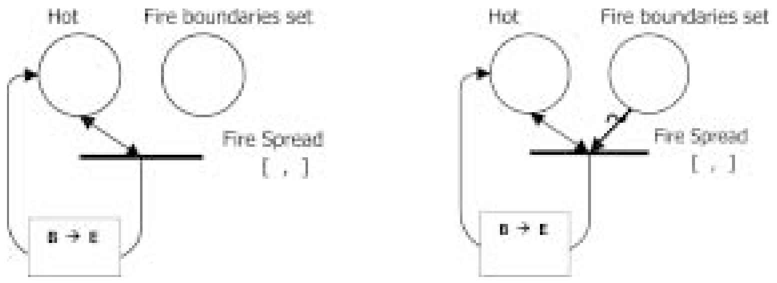

FIGURE 5
Prior domain theory used for the ML-TIPN demonstration shown encoded as two TIPNs. Note that the TIPNs are incomplete as their delay intervals are empty and the output arc table contains one record only.

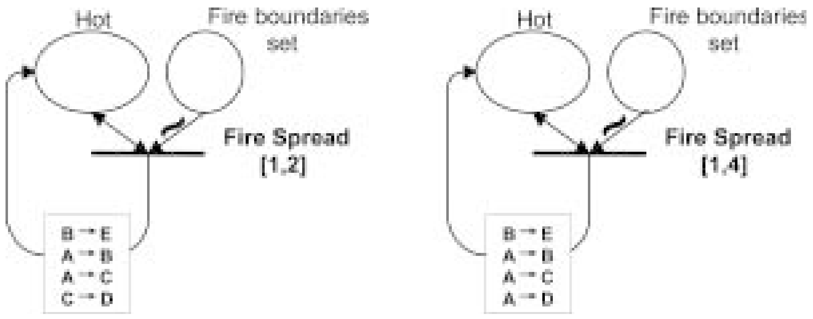

FIGURE 6
Two top-ranked models learned by ML-TIPN. The TIPN on the right is accurate.

knows *a priori* that fire can spread from compartment `B` to compartment `E`. The first two external events (e_1 = `<hot,A, [1,1],+, external>`, e_2 = `<fbs,B, [2,2],+, external>`) and one internal event e_3 = `<hot,B, [2,2],+, internal>` result in four TIPNs in the population P_3. Then ML-TIPN processes the internal event e_4 = `<hot,C, [3,3],+, internal>` thereby generating nine TIPNs in P_4. After the final event e_5 = `<hot,D, [5,5],+, internal>`, the total number of TIPNs in the population becomes 27. The two TIPNs ranked highest with respect to $e_1, \ldots, e_5$ are shown in Figure 6.

6.2 Learning a More Complex Fire Spread Model

The second experiment involved larger test and training sets, and the goal was to learn a more complex fire spread model. *Primary damage* ignitions were randomly set in one or more of 476 ship compartments, and physical and intelligent agent simulators propagated the fire to other compartments. Fire would spread to a neighbor compartment if the firefighting efforts were delayed by more than five minutes. Otherwise, a *fire out* event was recorded.

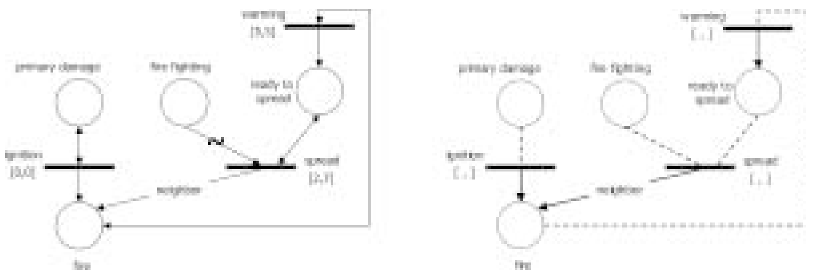

FIGURE 7
Left: Manually designed TIPN. **Right:** Prior domain theory P_0 expressed as an incomplete TIPN. Dashed lines represent possible/unknown arcs. Temporal interval delays, arc directionality, and enablement/disablement conditions are left to be learned by ML-TIPN.

Chain fires were defined as fires in compartments other than the ones with primary damage. Table 2 shows a primary damage event in compartment `A`, and three chain fires of length 1 in compartments `B`, `C`, and `D`. If the fire spreads to compartment `E`, then this would create a chain fire of length 2 (`A to B to E`). Chain fires are particularly challenging to learning as mispredicting one fire usually renders the rest of the fire spread chain causally inexplicable.

Each output event of the simulation became a part of the historical data with a 10% probability of one of the following three errors (i.e., training data noise): (i) its time interval was randomly altered, (ii) the compartment identifier was randomly modified, and (iii) the event was left out of the event record completely. Below is a small fragment of an actual event log $\{e_i\}$:

```
primary_damage('3-142-1',[9,9]).
fire_detected('3-142-1',[9,9]).
fire_fighting('3-142-1',[16,17]).
fire_detected('3-142-0',[19,19]).
primary_damage('1-54-2',[32,32]).
fire_detected('1-54-2',[32,32]).
fire_fighting('1-54-2',[36,37]).
fire_out('1-54-2',[40,41]).
```

Here `[t1,t2]` represents the time interval of the event and `'D-F-P'` is the deck-frame-position compartment identifier. For instance, the first two lines indicate a fire in compartment '3-142-1' at time 9 caused by primary damage there at the same time.

The machine learning objective of the ML-TIPN algorithm was to produce a model for `fire` events in terms of `primary_damage`, `fightfire`, `fire_out`, and other `fire` events. Figure 7 shows a TIPN manually designed to model the simulated crisis phenomena *without the noise*. The initial domain theory P_0 given to ML-TIPN is shown in Figure 7 as an incomplete TIPN. We then generated eight training and eight testing data sets of 2, 5, 7, 10, 15, 20, 30, and 50 events. A cross-validation study was carried out

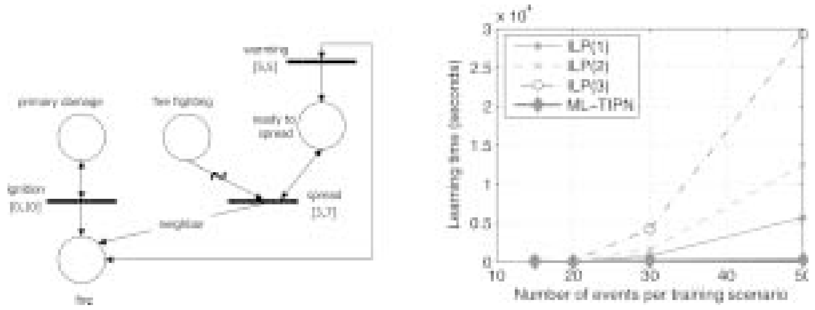

FIGURE 8
Left: TIPN learned by ML-TIPN from 30 training events. **Right:** Learning times for ILP with fire spread chains limited to 1, 2, and 3 fires as well as ML-TIPN.

by training the algorithms on each of the eight training sets and testing them on each of the eight testing sets. ML-TIPN was run with an initial beam width B_1 of 150. It was decreased by the factor of 0.9 at each iteration until it reached 5 after which it was kept constant. The scoring metric (Section 5.2) weights were fixed at $(3, 1, 2, 7, 10)$.

The three types of noise in the data present a considerable difficulty to learning as many events cannot be causally explained. Figure 8 shows the TIPN model learned from the training scenario of 30 events. Similar models were learned from 50-event scenarios. Comparing it to the hand-engineered TIPN in Figure 7, we note that all causal conditions are properly learned and the transition delay intervals have only minor differences. This is a result of noise as well as small size of the training data set.

6.3 Comparison between ML-TIPN and ILP

One of the strengths of the TIPN representation is the ability to convert between TIPNs and predicate calculus statements. This allowed us to conduct an experiment to learn a first-order logic model of the fire spread phenomena of Section 6.2 using inductive logic programming (ILP). On the ILP side, we used C-Progol [19]. Four versions of ILP learning code were run for each of the 64 combinations of the training and test scenarios. The first three versions capped the length of fire spread chains at 1, 2, and 3 fire events. The fourth version allowed for an arbitrarily long chain fire spreads via the use of recursive calls to the learned predicate `fire`. We do not report results of the fourth version as achieving its convergence within the available amount of memory and time is a topic for future research.

The first finding of this comparative learning experiment is that there is a significant difference in the learning times (Figure 8). We believe this is because ML-TIPN is searching the space of valid TIPNs, and this significantly restricts the search space. By contrast, the ILP algorithm is searching the space of valid clauses, which is much larger. Because there is no way to make the ILP search TIPN-specific, there is no natural way to restrict the search space further. On average, the number of false negative plus false positives was 45.1% higher with ILP. We believe the larger search space and the noise in the data prevented C-Progol from converging on a high-quality fire spread model.

7 SUMMARY AND RESEARCH OUTLOOK

This paper presents results of the first attempt to apply machine learning to the construction of Time Interval Petri Nets (TIPNs) for an AI task. Using an incomplete domain theory and noisy training data, an algorithm called ML-TIPN successfully constructed a fire spread TIPN in the domain of ship damage control. Petri net models of concurrent processes have been widely used in computer science because they have the advantages of providing an intuitive graphical representation of the processes being modeled and because of their strong theoretical foundation. This paper strengthens their relevance to AI problems by demonstrating that parts of TIPN construction can be automated.

Given the positive results of this exploratory study, an important question to address is directions of future research. What are the limits of machine learning for TIPNs?

One future research area is to understand how TIPN learning is affected by different characteristics of the training data. Accurate TIPNs were successfully learned from a relatively small set of training data with the noise rate of 10%. Of interest is how a greater level of noise in the training data and different volumes of training data impact the accuracy of the learned model and learning times.

Another area of future research is to understand how TIPN learning is affected by the quality and size of the background domain theory. The vocabulary used in the present experiments is provided at the correct level of abstraction. Thus, it is an open question of how much more difficult learning would become should the given vocabulary been more general or more specific than the target level of abstraction. Another issue relates to the scope of the background domain theory. Presently, the domain theory (P_0 in Figure 3) includes a specification of TIPN places and transitions (Table 1). As the models scale-up, automated ways of acquiring internal places and transitions are likely to become more important.

Yet another area of future research is the development of refinement operators and different scoring metrics that evaluate the quality of the refined TIPNs. The operators and metric described in this paper have been demonstrated to produce an accurate TIPN, but this may not be the case in a different domain. A related research goal is automatic learning of the best weights to use in the scoring metric as well as the optimal "cooling scheme" for the beam width. Related recent work includes search with and automated acquisition of meta-models [23].

Lastly, it is of interest to investigate how learning larger TIPNs scales up. In [9] manually constructed TIPNs in the domain of ship damage control had up to 43 nodes and transitions, and the learning experiments to date have synthesized TIPNs with 11 nodes. Clearly it is advantageous if the learning of larger TIPNs can be decomposed into the learning of a set of interconnected smaller TIPNs, and the extent that this is usually possible needs to be explored. Also, the difficulty of directly synthesizing larger TIPNs needs to be investigated.

ACKNOWLEDGMENTS

ML-TIPN algorithm was designed by the first author and implemented by IRCL members. Feedback from anonymous reviewers, KBS members, Valeriy K. Bulitko, Dan Roth, and Stefan Wrobel has been invaluable. The research was supported in part by ONR Grant N00014-95-1-0749, ARL Grant DAAL01-96-2-0003, NRL Contract N00014-97-C-2061, NSERC, and the University of Alberta. Finally, we appreciate the rapid processing time on the publisher's side.

REFERENCES

[1] Peterson, J. (1981). Petri Nets Theory and Modeling of Systems. Prentice-Hall, Inc.

[2] Jensen, K. (1997). Colored Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Monographs in Theoretical Computer Science. Springer-Verlag.

[3] Merlin, P. and Faber, D. (1976) Recoverability of communication protocols. IEEE Trans. on Communication *24*, 1036–1043.

[4] van der Aalst, W. (1993). Interval timed colored Petri nets and their analysis. In Marsan, M., ed.: Applications and Theory of Petri Nets. Volume 691 of Lecture Notes in Computer Science., Berlin, Springer-Verlag 453–472.

[5] Murata, T. (1989). Petri nets: Properties, analysis, and applications. In: IEEE 77. 541–580.

[6] Sil, J. (1995). Intelligent Expert and Learning Systems Using Petri Nets. PhD thesis.

[7] Costa Miranda, M. (1999). Modeling and analysis of a multi-agent system using colored Petri nets. In Portinale, L., Valette, R., Zhang, D., eds.: Proceedings of the Workshop on Application of Petri Nets to Intelligent System Development, Williamsburg, USA 59–70.

[8] Medeiros, S., Xexeo, G. and de Souza, J. (1999). Fuzzy Petri nets for dynamic workflow in GIS environment. In Portinale, L., Valette, R., Zhang, D., eds.: Proceedings of the Workshop on Application of Petri Nets to Intelligent System Development. 38–46.

[9] Bulitko, V. and Wilkins, D. (2003). Qualitative simulation of temporal concurrent processes using Time Interval Petri Nets. Artificial Intelligence *144*, 95–124.

[10] Mandell, D. (2003). Private communication.

[11] Segal, E., Barash, Y., Simon, I., Friedman, N. and Koller., D. (2002). From promoter sequence to expression: A probabilistic framework. In: Proceedings of the 6th Int. Conf. on Research in Comp. Molecular Biology (RECOMB), Washington, DC.

[12] Bulitko, V. and Wilkins, D. C. (2005). Machine learning for Time Interval Petri Nets. In: Lecture Notes in Artificial Intelligence (LNAI), Proceedings of the 18th Australian Joint Conference on Artificial Intelligence, Sydney, Australia, Springer-Verlag in press.

[13] Forbus, K. D. (1984). Qualitative process theory. Artificial Intelligence *24*, 85–168.

[14] Shou, G., Wilkins, D., Hoemann, M., Mueller, C., Tatem, P. and Williams, F. (2001). Supervisory control system for ship damage control: Volume 2 – scenario generation and physical ship simulation of fire, smoke, flooding, and rupture. Technical report, NRL.

[15] Badouel, E. and Darondeau, P. (1998). Theory of regions. In: Proceedings of the Third Advance Course on Petri Nets, Dagstuhl Castle, Germany, Springer-Verlag.

[16] Cortadella, J., Kishinevsky, M., Lavagno, L. and Yakovlev, A. (1998). Deriving Petri nets from finite transition systems. IEEE Transactions on Computers *47*.

[17] Desel, J. and Reisig, W. (1996). The synthesis problem of Petri nets. Acta Inf. *33*, 297–315.

[18] Zhang, D. and Murata, T. (1996). Fixpoint semantics for a Petri net model of definite clause logic programs. Advances in the Theory of Comp. and Computational Math. *1*, 155–194.

[19] Muggleton, S. (1995). Inverse entailment and Progol. NGCJ *13*, 245–286.

[20] Richards, B. J. and Mooney, R. J. (1995). Automated refinement of first-order Horn-clause domain theories. Machine Learning *19*, 95–131.

[21] Todorovski, L. and Džeroski, S. (2001). Theory revision in equation discovery. LNCS *2226*.

[22] Holland, J. (1962). Outline for a logical theory of adaptive systems. Journal of the Association for Computing Machinery *3*, 297–314.

[23] Lee, G. and Bulitko, V. (2005). GAMM: genetic algorithms with meta-models for vision. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Washington, DC 2029–2036.