Machine Learning for 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
bulitko@ualberta.ca
[2] Center for the Study of Language and Information, Stanford University, Stanford, CA 94306
dwilkins@stanford.edu

Abstract. 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

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 [1], qualitative simulation and modelling [2]. TIPNs are especially suitable as fast qualitative-level models for concurrent 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 [2]. On 160 simulated difficult scenarios that involved fire, smoke, flooding, and machinery failure, the TIPN-based decision-making system saved 117 ships (73.1%) [2]. 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.

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.

2 ML-TIPN Learning Method

In light of difficulties with the use of existing methods, we have developed a novel approach specifically for learning TIPN-based domain models. The method (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.

ML-TIPN's input consists of domain theory P_0 and historical data $\{e_i\}, \{e'_j\}$. 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 type "input token label → output token label" (as illustrated in the bottom boxes of Figures 1 and 2). 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".

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 speci-

fying 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.

3 Learning Process

ML-TIPN learns by conducting a 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 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. Then the population P_{t-1} of B_t partially specified TIPN models is refined into the new population P_t.

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} and copies them all to the new population P_t. 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. 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, each producing a refined TIPN. All of these refinements are put in the new population P_t. Then ML-TIPN will sort the new population P_t using the scoring metric and the validation set $e'_1, \ldots, e'_{N'}$ and retain B_t top-ranked ones. The beam search width B_t is then updated. 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, sort the resulting TIPNs on the validation set using the scoring metric, and output the M top-ranked of them. Note that the number M is supplied by the user giving him/her an opportunity to hand-pick from several learned TIPNs.

4 Empirical Evaluation

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`). The corresponding event record ($e_i = e'_i$ for all i), which is the input to the ML-TIPN algorithm, is shown in Table 1. For the sake of simplicity and space, we will start with the prior domain theory

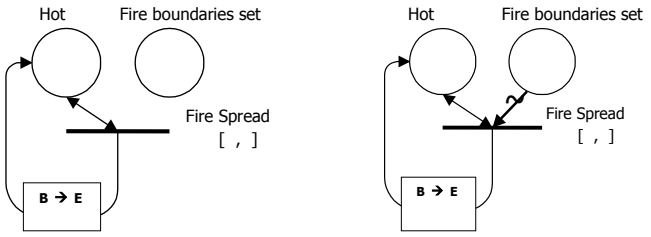

Fig. 1. Prior domain theory used for the ML-TIPN demonstration shown encoded as two TIPNs. Note that the TIPNs are incomplete as their delay intervals are empty and the output arc table contains one record only.

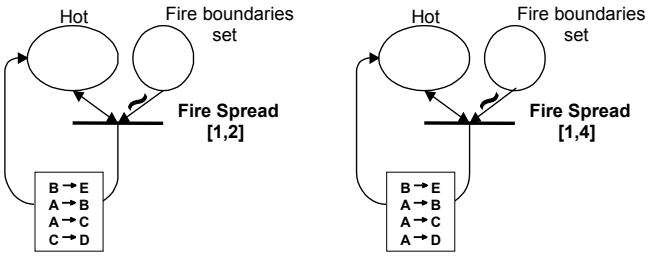

Fig. 2. Two top-ranked models learned by ML-TIPN. The TIPN on the right is accurate.

P_0 in the form of the two-element TIPN set shown in Figure 1. 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 knows *a priori* that fire can spread from compartment `B` to compartment `E`.

Table 1. 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

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 2.

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

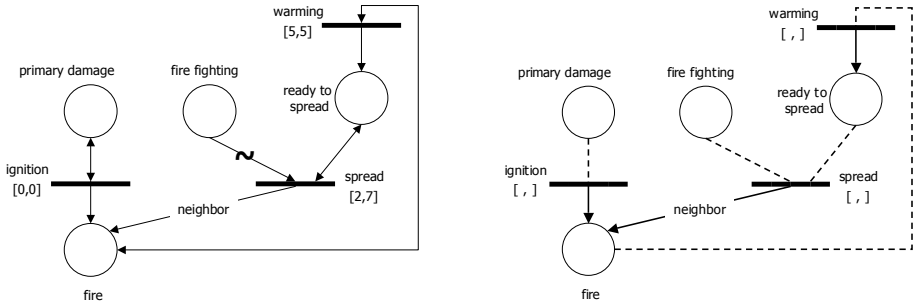

Fig. 3. Left: manually designed TIPN. **Right:** Prior domain theory P_0 expressed as an incomplete TIPN. Dashed lines represent possible/unknown arcs. Temporal interval delays, arc directionality, and enablement/disablement conditions are left to be learned by ML-TIPN.

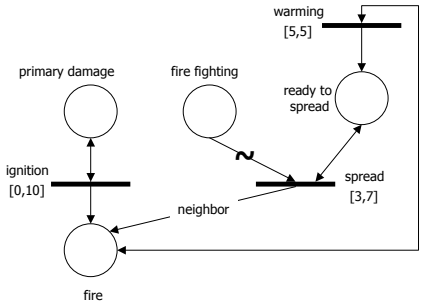

Fig. 4. Time Interval Petri Net learned by ML-TIPN from 30 training events.

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.

Chain fires were defined as fires in compartments other than the ones with primary damage. Table 1 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 3 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 3 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 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 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 4 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 3, 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.

5 Summary

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.

We appreciate programming by Tony Czupryna. 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, and the National Science and Engineering Research Council.

References

1. Costa Miranda, M.: 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 (1999) 59–70
2. Bulitko, V., Wilkins, D.: Qualitative simulation of temporal concurrent processes using Time Interval Petri Nets. Artificial Intelligence **144** (2003) 95 – 124