Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Refining Process Descriptions from Execution Data in Hybrid Planning Domain Models

Alan Lindsay,[1,3] **Santiago Franco,**[2] **Rubiya Reba,**[3] **Thomas L. McCluskey**[3]
[1]Heriot-Watt University, UK, [2]Royal Holloway, University of London, UK
[3]University of Huddersfield, UK
alan.lindsay@hw.ac.uk, santiago.francoaixela@rhul.ac.uk, {rubiya.reba, t.l.mccluskey}@hud.ac.uk

Abstract

The creation and maintenance of a domain model is a well recognised bottleneck in the use of automated planning; indeed, ensuring a planning engine is fed with an accurate model of an application is essential in order that generated plans are effective. Engineering domain models using a hybrid representation is particularly challenging as it requires accurately describing continuous processes, which can have complex numeric effects. In this work we consider the problem of the refinement of an engineered hybrid domain model, to more accurately capture the effect of the underlying processes. Our approach exploits the information content of the original model, utilising machine learning techniques to identify important situation and temporal features that indicate a variation in the original effect. We use the problem of modelling traffic flows in an Urban Traffic Management setting as a case study and demonstrate in our evaluation that the refined domain models provide more accurate simulation, which can lead to higher quality plans. The contribution of this work is a general approach to the automated refinement of hybrid planning domain models that reduces the knowledge engineering effort in producing a detailed process model. The approach can be used for refining the domain model during the initial stages of development, or for re-configuring the domain model when used in the same problem area but with a different scenario. We test out the approach within a real world case study.

Introduction

In automated planning, it is common to formulate all the knowledge required to solve a problem within a precise planning description language, independently of the planning engine that will be used to generate a solution plan. Capturing this knowledge is a well recognised bottleneck in the use of automated planning: the problem formulation (often split into a model of the dynamics of the domain, and a model of the problem instance) is an abstraction of some real planning problem, and if it lacks some details or is incorrect then the wrong planning problem will be attempted. In particular, for the planner to be able to generate accurate plans, the domain model must adequately capture both a declarative description of the dynamics of the application and its simulation during plan execution. Various lines of research have focused on the domain model accuracy problem, from as far back as Benson's thesis (Benson 1996) to more recent work on Space applications (Clement et al. 2011; Frank 2015), and the related area of model incompleteness (e.g. as referred to in (Zhuo, Nguyen, and Kambhampati 2013)).

Even if planning is restricted to discrete models, validating that the model is an accurate representation may be a challenge. If the model is hybrid, involving continuous changes to variables, the burden on the knowledge engineer is greater, since the encoding of continuous changes are typically more difficult than the encoding of discrete changes. This is particularly apparent in applications to physical systems where it can be necessary to use richer languages to capture the dynamics of continuously changing phenomena in an environment (Say et al. 2017). Further, it may be that modelled processes change their behaviour over time, and the model has to be constantly maintained. Or it may be that there are many similar processes in the domain, but each is a variant over the space that the planning function is aimed at. In these cases, producing a faithful hybrid planning domain model a priori, is indeed a challenge.

Our research is aimed towards creating a general method for the automated refinement of pre-engineered hybrid domain models, suitable for automated planning in real world environments. In this paper a method for the refinement of domain model *process* descriptions is described, utilising the PDDL+ language (Fox and Long 2002) for model encoding. PDDL+ has the advantage that it is usable by a range of planners, and it has a well defined declarative and operational semantics. The approach is designed so that the resulting refined domain model will retain (a) its original language - PDDL+ (b) its efficiency, in terms of plan simulation and plan generation (c) its readability so that changes can be explained in terms of high level features. We illustrate and evaluate the approach using a case study from the domain of Urban Traffic Management. A PDDL+ model of traffic flow is utilised based on earlier work in this area, and an industry standard, proprietary simulator called AIMSUN (Barceló and Casas 2005) is used to provide process training data. Our evaluation demonstrates that simulation using the refined domain model is closer to the behaviour of the actual processes being modelled, and when using expert selected

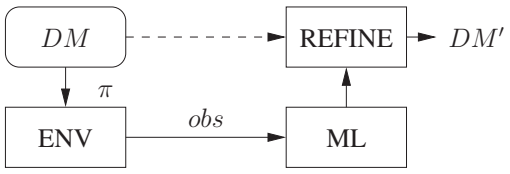

Figure 1: The system operates from a domain model, DM. A plan, π, can be observed as it is executed in the environment (ENV), resulting in observation traces (obs). A predictive model is learned to capture the difference between the observations (obs) and expected results (with respect to DM) using machine learning (ML).

features the plans produced with the refined domain are of higher quality than the original. Further, we show that the simulation time efficiency of the model increases only linearly with the complexity of the refinement step.

Preliminaries

Consider applications involving both plan generation and plan execution, where in execution mode sensors are available that can supply the current state of the system. State information may be obtained from the interpretation of a stream of such sensor information from the environment, or be supplied by virtual sensors from a simulator which represents the real environment (Clement et al. 2011). This sensor feedback is often required in planning and execution so that the planner can monitor a generated plan's execution, and check it is having the expected effect, with the possibility of re-planning if actual and expected diverge. Additionally, in our approach, this state information is used to drive refinement of the domain model.

The method used in this paper is summarized as follows, and outlined in Figure 1: a planning application is being undertaken in which knowledge engineers have developed a hybrid domain model DM to the point where an available planning engine can generate plans for the application, but the planning engine's simulation does not accurately match the state information extracted from the sensors due to inaccurate process descriptions. Given that a process P in the DM changes a set of continuous variables V, for each v in V, assume the effect of P's corresponding domain process is dependent on some subset of state variables captured in DM. From the environment ENV (see Figure 1), observation traces of external states obs are obtained. A learning method uses them to discover an expression, incorporating a combination of state variables from DM, which best predicts the effects on v as found in the external states, and replaces the original expression with the discovered expression in the process specification. The DM is then replaced with the new version DM'.

To make this method concrete, and evaluate it on a real planning application, we have used the PDDL+ encoding (Fox and Long 2002) for hybrid domain models, and the ENHSP planner (Scala et al. 2016) to generate plans when input with a task and a domain model. In a PDDL+ encoding, a hybrid planning model involves time-dependent discrete-continuous changes in the numeric resources which can be encoded with three main components: processes, actions and events. A process simulates continuous changes on numeric variables, which can be initiated or terminated by an action/event whereas an event can be triggered by the external environment to bring about discrete changes.

Motivating Example Domains: Consider as a simple example a model of the Bouncing Ball domain[1] with no friction and where the ball has perfect elasticity i.e. no energy is lost upon bouncing. The original process describing the ball movement is called *ball-movement*, and may be engineered using the known gravitational equation. If it were possible to collect accurate and representative data on the position and velocity of the ball, the model could be refined to better fit reality - in this case we assume by finding the appropriate friction coefficients. The model would be refined in accordance with the data by learning scaling factors that act on the increase in velocity. This can be achieved by splitting the ball-movement process into (e.g.,) two new processes by adding as preconditions whether $velocity >= 0$ or < 0 and treating each case individually. For example, the falling friction might be observed as 0.8. However, when the ball is rising, both gravity and friction pull the ball back, resulting in a rising friction coefficient bigger than 1, e.g. 1.2.

As another example, consider a fixed angle solar panel used for charging,[2] where the original engineered model assumes a constant rate of charge. The amount of electricity it receives, however, is a function of the time of the day, weather etc. Given a set of training data containing time, weather, etc, and the charging effect, the PDDL+ domain model can be refined into multiple charge processes to improve accuracy. This illustrates a potential trade-off between accuracy vs computational complexity: the more accurate, the more processes need to be checked for applicability.

As an in-depth case study, for the rest of this paper, the domain of Urban Traffic Management (UTM) is used. (Vallati et al. 2016) introduced a hybrid planning approach using a PDDL+ representation to solve traffic management problems involving congestion in both unexpected and regular road traffic. In their macroscopic model of UTM, a road network is represented by a directed graph where the edges and vertices denote the road links and junctions (the entry/exit point of the road links) respectively. Each connected entry/exit pair is called a turn and the traffic flow through a turn (here called the *turn rate*) is measured in standardized vehicles (PCU) per second. Flow across a junction is organised by grouping these turns into *stages* (representing the stages of traffic signals) and then selecting durations for each of these stages. Each road link has a maximum *capacity*, and the *occupancy* of a road link denotes the current number of PCUs within it. This approach was used in feasibility trials and incorporated a simple description of the flow process (called *flowrun_green*), which took no account of the chang-

[1] Similar to the example in Coles's tutorial http://cognitive-robotics17.csail.mit.edu/docs/tutorials/Tutorial8_Planning_in_Hybrid_Domain.pdf

[2] A good example is the satellite cooled domain, see https://nms.kcl.ac.uk/planning/software/colin.html

ing features of the road network (McCluskey and Vallati 2017). The use of this case study in this paper is to demonstrate the utility of learning an improved specification for the *flowrun_green* processes, which leads to more accurate simulation and better solution plans.

Process Refinement

Key to the method is to replace a domain model process with a set of more targeted processes that will each better capture the effects of the observed process in specific situations. Another important feature is that *effect modifying factors* are learned, which instead of overwriting the original specified effects, adjusts them. The motivation here is that in focusing on refining the process descriptions we exploit the existing structure and information content within the existing model.

For example, a process in the world being modeled might require a warm up period, which can be better represented by separating the process's first n seconds of operation and reducing the process's effect in that initial period (e.g., queuing PCUs getting up to speed when signals have changed). The method learns this by constructing a set of processes that each define the process's effect for a more specific situation. This requires that the new processes have extended preconditions, as illustrated by the `refinement-condition` predicates in Figure 3. The new process also defines factor terms (highlighted in red), which are specified for each of the continuous effects of the process.

The context that we have selected for process refinement includes both temporal and situational aspects, which can both impact on the effect of continuous transitions.

Temporal Features: process effects can change depending on how long the process has been active, e.g., waiting PCUs getting up to speed when a stage becomes active. In order to allow the refined processes to reflect this change, the context includes the time since the process became active. Notice that this part of the context is treated in the same way as the functions in the state features (described next).

State Features: a process's effect may also be different when certain state relationships exist (e.g., turning across a busy road will reduce turnrate). The context that we use for learning includes all of the predicates, both numeric and propositional, which can be referenced by the parameters of the process. The parameters of the original process are used to create the context, as naturally the factors that impact on a processes' behaviour are most likely related to the parameters of the process. Moreover, by focusing the context it should allow less data to be used, while still learning process specific phenomena. For example, there will be fewer correlations with external features that might be specific to the gathered data. Another reason is that including additional parameters further increases the computational complexity of the model.

Feature Space: Using the temporal and state features of the PDDL+ encoding of the domain as specified above, the feature space is constructed automatically. In order to support a general method of refinement we have designed a language for generating numeric features and propositional terms that supports a variety of common relationships. Propositional terms are included directly from the context, while numeric features are built from any function in the context (denoted $f[y_0, \ldots, y_n]$) using the free variables of the process header (X). The language is specified as follows:

$$T[X] ::= (T[X]\, R\, T[X]) \mid f[y_0, \ldots, y_n], y_i \in X$$
$$R ::= + \mid - \mid * \mid /$$

We define the depth of a numeric term $depth(T[X])$, as the count of the number of function terms in T[X], e.g., $depth$[(+ (occupancy ?r1) (occupancy ?r2))]$= 2$. The set of features are then generated by fully expanding the expressible terms in the language up to a defined depth. In this case this constructs expressions that build from the functions of the domain using the typical binary relationships: $\{+, -, *, /\}$.

Refinement as a Learning Problem

Multi-target regression methods are used on learning problems where a model that relates the set of input features to a set of target outputs is sought. The approach is used here with one target output for each of the process's continuous effects. This allows a single model to be learned for each process, which can be more concise than if a model is learned for each target variable, as well as being more practical for encoding back into PDDL+. The learned model can be used to identify the appropriate factor for each effect at any state. The approach which relates state observations to active processes is similar to the deictic representation developed for learning action models in (Mourão, Petrick, and Steedman 2010).

The input data is modified into a process orientated form, for a process P, as illustrated in Figure 2. For each time point, t, and active process, p (an instantiation of P), a learning example, $\langle p.X, t_{active}, s\rangle$ describes: $p.X$, a process header, t_{active}, the active time of the process and s, the state. Figure 2 (a) and (b) illustrates this for $t = x + \delta$: the process `flowrun_green` is active from x and has been active for δ at time t. Each of these learning examples is used to construct a single data point. The state is generalised by using the header $p.X$ to replace the constants of the state predicates (numeric and propositional) with the process parameters (Figure 2 (b) and (c)). Any predicates with parameters that are not replaced (i.e., that have parameters that are not in $p.X$) are discarded. This gives a single frame that has been generalised from its instantiated process, p.

The feature space (as described above) defines lifted terms (e.g., `(/(occupancy ?r1)(capacity ?r1))`). The terms' values are then evaluated in the generalised states. For example, in Figure 2 (d) the inserted row records the values for the occupancies of in and out links and the process active time are reported for the frame in (c).

The aim for learning is to capture an effect modifying factor for each of the processes's effects. The targets are therefore computed for each data point, d, and process effect, e,

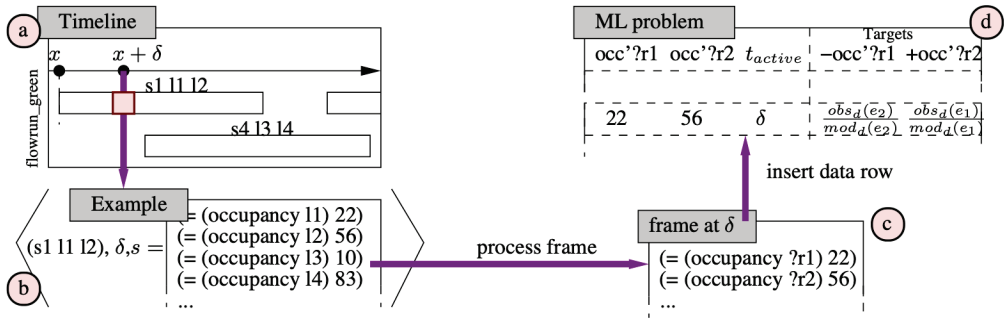

Figure 2: For a process that becomes active at time x we have an observation at each observed time point, δ, until it is inactive. For each time point we extract the relevant information for the process (parameters) and write it in terms of the parameters. Each process frame becomes a data point for our learning process.

```
(:process flowrun_green-leaf-i
   :parameters (?p - stage ?r1 ?r2 - link)
   :precondition
   (and
      (> (occupancy ?r1) 0.0) (active ?p)
      (>= (turnrate ?p ?r1 ?r2) 0.0)
      (< (occupancy ?r2) (capacity ?r2))
      (refinement-condition-1 ?p ?r1 ?r2)
      ...
      (refinement-condition-n ?p ?r1 ?r2))
   :effect
   (and
      (increase (occupancy ?r2)
         (* (#t (* refinement-factor-i-1 (turnrate ?p ?r1 ?r2))) )
      (decrease (occupancy ?r1)
         (* #t (* refinement-factor-i-2 (turnrate ?p ?r1 ?r2))))))
```

Figure 3: An example of process refinement using the *flowrun_green* process. A set of conditions (bold) identify sets of states where it is appropriate to modify the process's effect using a factor (red).

as $factor = \frac{obs_d(e)}{mod_d(e)}$, where $obs_d(e)$ is the observed effect and $mod_d(e)$ is the modelled effect. In Figure 2 (d) this is illustrated for the two *flowrun_green* effects: increasing the occupancy of the out link (+occ'?r2) and decreasing the occupancy of the in link (-occ'?r1).

Finally examples and features are pruned: i) Features are pruned where any evaluation leads to an undefined value (i.e., the result of the expression is not defined). This would suggest that the feature might lead to undefined values during planning. ii) Propositional features are pruned if their value is the same across all examples. iii) Examples are pruned where the target is not defined (i.e., where the modelled effect is 0).

The result is a consistent set of features and targets for examples across all of the instantiations of the process that can be addressed as a general machine learning with optimisation function:

$$\arg\min_{\mathbf{M}} \sum_{d \in DB} \sum_{e \in EFF(p)} [obs_d(e) - \mathbf{M}_d(e) * mod_d(e)]^2$$

function `Reprocess`$(obsTrn, obsVal, P, DM)$:
  $F =$ `chooseFeatures`(DM)
  $trnData =$ `makeProcessOrientated`$(trnObs, P)$
  $valData =$ `makeProcessOrientated`$(valObs, P)$
  $t \leftarrow Root()$
  `GrowTree`$(trnData, t, F)$
  `PruneTree`$(valData, t)$
  `RevalueTree`$(trnData + valData, t)$
  $P+ =$ `extractProcesses`(t, P)
  `extendModel`$(DM, P+)$
end function

Figure 4: Pseudo code for the `Reprocess` approach, which refines a process description. The `Reprocess` function organises the data, grows the tree and replaces the original process with the generated refinements and supporting extensions (e.g., for maintaining active time features).

Regression Tree Learning

The hypotheses for the process model are represented by regression trees, as they can be used to approximate complex functions, and algorithms exist that can grow them efficiently and effectively from observation data. Moreover it is possible to encode the learned model in PDDL+, as presented at the end of this section. This contrasts with several of the alternative representations used for machine learning. In this section we describe the tree learning approach and focus on its important aspects.

Hypothesis Construction

A high level description of the `Reprocess` method is presented in Figure 4. The method uses both training and validation data sets, which are both pre-processed into process orientated data points as described in the previous section (see Figure 2).

At the heart of the approach (i.e., `GrowTree`), a multi-target regression tree learning approach based on earlier work (De'Ath 2002) is used to learn a multi-target regression tree. The important difference that extends standard tree learning approaches to multi-target regression is that the cost function sums error terms for each of the targets. The ap-

proach greedily identifies conditions that lead to the largest reduction in the error for the training data. A condition is used to split the data into two parts: those for which the condition holds and those for which it does not. A child is made for each of the data sets and the `GrowTree` process is repeated at each child with the associated (and therefore reduced) data set. The recursive process is stopped when there are too few data points at a node, or when splitting has a small change on the error.

As tree learning approaches can lead to over-fitting, a validation data set is used in order to inform a tree pruning process (`PruneTree` in Figure 4). Starting from the leaf nodes, the process re-evaluates each of the branchings of the tree and decides whether it provides a sufficient information gain, given the new data set. If it does not then the node becomes a leaf and the branch is pruned. The tree is completed by re-evaluating the values at the leaf nodes using the combined data sets (`RevalueTree`).

Feature Selection As the feature space defined above is potentially large, a pre-processing step is used to identify a reduced subset of the features that appear most relevant for use during learning. Two approaches are demonstrated here:

- The first is hand selection. A transport expert who was involved in the initial engineering of the domain model indicates which features are relevant to the target.
- The second is to exploit *attribute selection*, a common pre-processing step in machine learning applications. A filter method is adopted which identifies features that are correlated with the targets, yet uncorrelated with each other (Hall 1998).

Describing Tree Node Conditions The system must also generate possible division points for the domains of numeric functions (e.g., `(<=(/(occupancy ?r1)(capacity ?r1))` **0.8**`)`). A standard approach for regression trees is used: calculate the value of a feature at each data point (at the current node), order the values and propose splitting the feature's domain in between each pair of adjacent values.

The Model Prediction The value of a tree leaf is calculated for each of the processes' effects separately ($e \in EFF(P)$) as the average of the factors between the modelled value ($mod_d(e)$) and the observed values ($obs_d(e)$) for the data set (DB) at the leaf node: $t.e = [\sum_{d \in DB} \frac{obs_d(e)}{mod_d(e)}]/|DB|$

Leaf Cost Function The evaluation of a hypothesis is defined as the sum of the error at each of the tree's leaves. This is based on the data at that leaf (DB) and calculates the squared differences between the observed values and the modified modelled value (using the tree's value) for each data point. In order to use a consistent measure of error we evaluate the error as the error of the updated process's effect: $Err = \sum_{d \in DB} \sum_{e \in EFF(p)} [obs_d(e) - t.e * mod_d(e)]^2$

Extracting PDDL+ Processes From a Tree The final step is to modify the domain model with the refined process (Figure 4 `extractProcesses` and `extendModel`). The tree conditions themselves are expressible in PDDL. As a result the path to each leaf node describes a conjunction of conditions that should each hold for the leaf to be appropriate for the state. As such the tree is descended and conditions are recorded at each node (either $(<= X)$ or $(not X)$, for the left hand branch or $(> X)$ or (X) for the right hand branch) as illustrated in Figure 5. These conditions are added to the original conditions to form a larger conjunction. The effects are then each modified by multiplying the right hand side of the effect terms by the associated factors learned for the leaf node. A unique symbol is added to the process name to distinguish it from the processes constructed for the other leaves. These processes replace the existing process description.

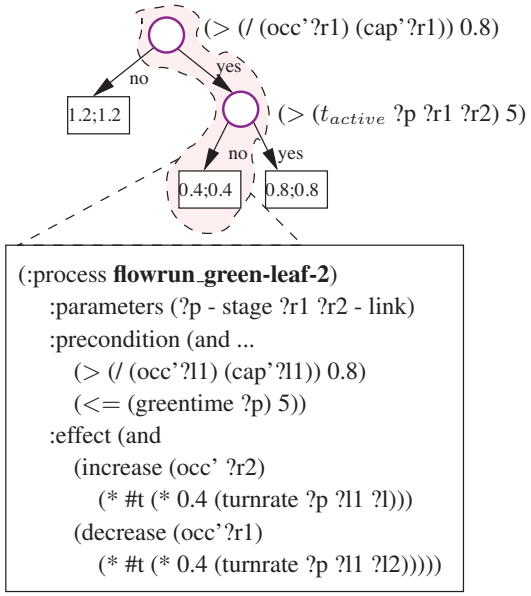

Figure 5: An example decision tree and the resulting refined process generated from one path from the tree root to a leaf node. In this example the leaf found by going down the right branch at the root, i.e., `(>(/(occupancy ?r1)(capacity ?r1))0.8)` and then the left branch, i.e., `(<=(`t_{active} `?p ?r1 ?r2) 5)`. The factors at this leaf are both 0.4. These conditions are used to refine the original process and control when it is applied.

The tree hypothesis is further examined to determine if it exploits any features that relate to the active time of the process. If the original model does not already represent the active time of the process (and the tree uses the features) then the model is further extended. In particular, for a process, p, we define a new set of functions (one for each instantiation of the process): $\langle p.\texttt{name} \rangle$`_counter`. These functions are maintained by events that start and end the timer and a new process that records the active time.

Evaluation

Here we report results from experiments with two domains: the Bouncing Ball domain, which demonstrates the application of our framework in a familiar scenario with a constrained learning problem, and the UTM domain, which is extracted from industrial trials of automated planning to

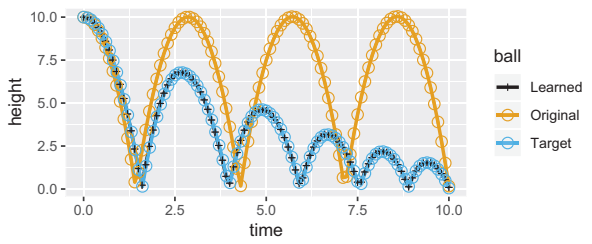

Figure 6: Height vs. graph for a ball modelled using the original, target and learned models in the Bouncing Ball domain.

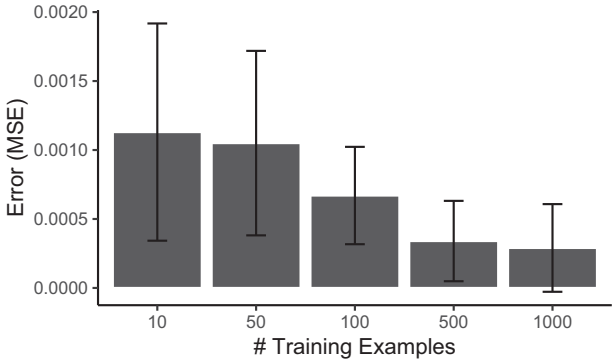

Figure 7: Plotting error (mean squared error) against number of training examples for learning factors to modify height and velocity effects of the *ball-movement* process (to capture air resistance) in Bouncing Ball.

regulate traffic flow through large urban areas. Whereas in the Bouncing Ball domain we create our own *ground truth* model representing the target of learning,[3] in the UTM domain no such explicit model is known. In particular with the UTM domain, we investigate whether the `Reprocess` procedure improves simulation accuracy and leads to better plans in this industrial-strength application.[4]

Bouncing Ball: Capturing Air Resistance

Experiments with this domain, introduced in Preliminaries, have been used to investigate the impact of the data set size, and in particular how the number of training examples can lead to higher accuracy in the learning problem. Two domain models are used: an original model where the moving process captures frictionless motion, and the target model, which captures a simple form of air resistance. A collection of problems with balls starting from various heights was generated. Plans for these problems were simulated using a PDDL+ simulator, which was modified to output state observations at regular intervals. Figure 6 plots height against time for a ball modelled using the original frictionless model (where the balls continue bouncing to the same height) and the target model.

The `Reprocess` method was used in order to refine the *ball-movement* process. The process was repeated using different sizes of training sets and 10 times for each training set size. The error (mean squared error) on a separate testing set is presented for each training set size in Figure 7. As expected this illustrates that as the number of examples in the training set increases the mean error and standard deviation decrease. Figure 6 also plots the ball simulated using the learned model (1000 examples).

Process Refinement in UTM

A microscopic model of a region of the Manchester (UK) city centre road network was obtained from the local transport authority and was captured in AIMSUN (Barceló and Casas 2005): professional modelling and simulation software. AIMSUN uses a micro model of the road network, which has been heavily parameterised to represent individual PCUs (e.g., aggression level of drivers, bus stops) and has been validated against real world data. This level of detail cannot be captured in a macro model; however, our expectation is that our base line model can be refined in order to better characterise the flow. The model contains c.300 road links and over one hundred junctions, and provides a challenging test case for this approach. Two subnetworks at either side of this road network were extracted for the experiments – we denote them RHS and LHS. For each run, data (including turnrates, active signals and occupancies) is collected from the simulator at regular intervals. For the experiments we have generated 8, 1 hour long, data sets for each of the networks. These data sets each start from different initial states and have been generated by first planning from that initial state using the original model and then simulating the original model's plan. Each plan provided around 10,000 data points. For each network, the observations for 3 plans were divided into training (75%) and validation (25%) sets. The remaining 5 simulations were used as the test set.

The starting (original) PDDL+ domain model is the representation of the UTM problem domain presented in (McCluskey and Vallati 2017), as introduced above. The turnrates used to model flow in the original model are specified for each green time stage and link pair and approximate maximum average flow.

The Refined Models Two trees were learned for each of the networks using the training and validation sets described above. Each uses a subset of the feature set generated by our system. The first (Expert Assisted) uses features that experts believed would determine in some way the real traffic flow rates and the second (Auto) uses the automated process, described as follows.

There are 201 features generated (for max depth $\phi = 2$) from the original domain. For the Expert Assisted approach the following features were selected: in-link and out-link saturation, proportion of process time (with respect to maximum stagetime) and approximated maximum turnrate.

For the Auto approach we used a filtering method for feature selection, which results in a subset of 8 features for the RHS network and 7 features for the LHS network. The auto

[3] Ground truth models are often used in planning domain model acquisition research to assess the quality of a learning technique

[4] In this context we interpret a *better* plan as one that concludes in a state that is closer to the goal (during execution).

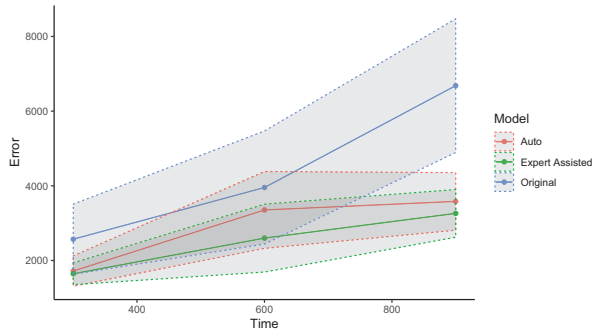

Figure 8: A comparison of the RHS network occupancies between modelled (Expert Assisted, Auto and Original) and those observed in AIMSUN after 300s, 600s and 900s of simulation.

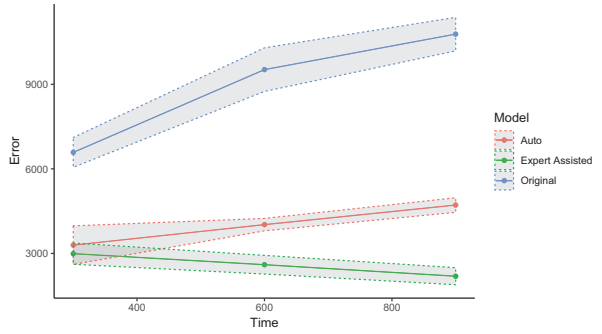

Figure 9: A comparison of the LHS network occupancies between modelled (Expert Assisted, Auto and Original) and those observed in AIMSUN. Dotted areas indicate standard deviation.

selected features included 2 (RHS) and 3 (LHS) of the 4 expert selected features.

Accuracy Through Simulation

In this experiment we examine the accuracy of the model during plan simulation. During data collection in AIMSUN we have periodically recorded the occupancies of the links across the network. The plans that were used to generate the test data were simulated using the learned and original models. At each time point we compared the occupancies in the modelled states against the occupancies in AIMSUN.

We have plotted the squared error observed (between planning model and AIMSUN) for the Expert Assisted, Auto and Original models in the two networks (RHS in Figure 8 and LHS in Figure 9). The results indicate that the refined model using expert selected features leads to less error across the network in each of the networks. In the case of the automatic features, the refined model improves simulation accuracy in each of the subnetworks, although not as much as for the expert picked features.

Efficiency of Refined Models The more refined the model the more processes in the PDDL+ domain and consequently

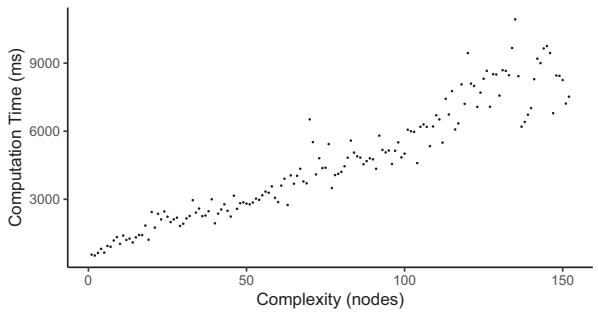

Figure 10: As the complexity of the refinement is increased, the computational effort of simulating the model increases.

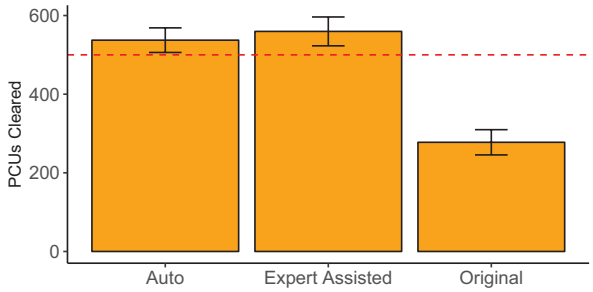

Figure 11: PCUs cleared from the goal link (in AIMSUN) after simulating plans for the three models in the LHS network; goal to clear 500 PCUs (red line).

the longer the simulation time. In Figure 10 we plot the simulation time (15 minutes simulating a plan on RHS network) with increasingly complex models (from 0 to 150 decision tree nodes). It shows that computation time grows in a linear fashion as a function of the number of nodes in the tree. The trade-off between accuracy and computation time is application specific and will depend on the type of search that will be used.

Accuracy in Planning

In this experiment we examine the accuracy of the models in the context of planning tasks in the UTM domain. In this case we ran AIMSUN three times from each initial state in our test set. In the first case we generated a plan using the original model and in the other cases, we generated a plan using the refined models. Each plan was simulated in AIMSUN and stopped when the plan completed. At this stage the model had predicted that the goal would be achieved. We then analysed this final state in the simulator in order to determine whether the goal had been achieved.

For this experiment the goal was to clear 500 PCUs from a specific link. This was repeated 5 times in each network (LHS and RHS). We have plotted the number of PCUs that were cleared from the link at the end of the plan in the case of each model. The graphs in Figure 11 and Figure 12 show that the original model over predicts the number of PCUs that are cleared from the goal link in both networks. In compari-

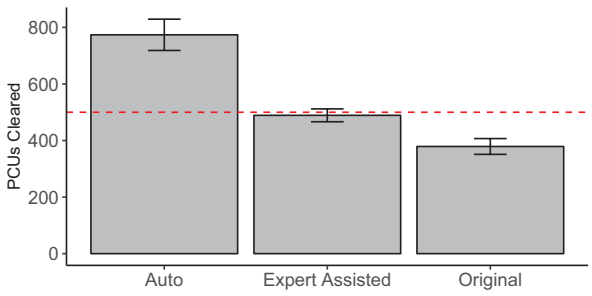

Figure 12: PCUs cleared from the goal link (in AIMSUN) after simulating plans for the three models in the RHS network; goal to clear 500 PCUs (red line).

son, the expert assisted model is able to more accurately predict the number of cleared PCUs in both networks and was particularly effective in the RHS network. The Auto model performs well in the LHS network (Figure 11). In this case the selected features set included the saturation of the outlink (e.g., `(/(occupancy ?l2)(capacity ?l2))`). In the RHS the Auto model predicts that the clearance will be slower than was observed in AIMSUN. We observed that the stage times around the goal link are significantly different in the generated plans for this experiment from the training set used to learn the model. Also the selected features do not include the saturation of either the in- or out-links. It is therefore possible that the selected features did not allow an effective model to be learned, but instead a model that is overfitted to the plan distribution of the training data.

Related Work

While much of the *motivation* of this work aligns with recent work on the challenges of domain model construction in Space applications, and in particular the ideas underlying the Interactive Model Development Environment (Clement et al. 2011), related work is centred around learning from execution data, an early example being Benson's TRAIL method, which used ILP on success and failure to learn action models in robot simulation environments (Benson 1996). In (Say et al. 2017) they learn a deep network model of a process from observations and compile it into a Mixed Integer Linear Program problem. This provides a complementary approach for use in domains where a utility function and horizon can be defined. A key benefit of our approach is that it supports representing the learned model in PDDL+, allowing any PDDL+ planner to be used.

Most related to this paper's refinement method is the progressive development of appropriate representations for concept learning, e.g., (Martin and Geffner 2000) and selection of the appropriate contexts for learning for control knowledge (Lindsay 2015) and heuristic correction (Yoon, Fern, and Givan 2007). In domain model acquisition (DMA) it has been common to assume accurate input data and this has allowed inductive learning approaches to be exploited, e.g., (Cresswell and Gregory 2011). In recent work, researchers have examined noisy data, exploiting clustering (Lindsay et al. 2017), machine learning (Zhuo and Kambhampati 2013) and deep learning (Asai and Fukunaga 2018) as part of their processes. DMA has progressively considered richer target fragments of the PDDL language, from propositional (Wu, Yang, and Jiang 2007; Cresswell and Gregory 2011), including ADL (Zhuo et al. 2010); to learning action costs (Gregory and Lindsay 2016) and numeric constraints (Segura-Muros, Pérez, and Fernández-Olivares 2018). As the richness of the language is increased, the space of possible models that explain the data vastly increases and has led to the DMA problem being set as either a search or learning problem (e.g., subtype selection in LOCM2; CP model to identify cost relevant parameters in NLOCM). We are not aware of any work in DMA that supports modelling of continuous transitions in PDDL+, although there are related works that consider numeric fragments, such as the approach in (Lanchas et al. 2007), which learns relational decision trees to appropriately estimate situation specific action durations from observational data.

Conclusion and Future Work

In this paper a novel approach for refining hybrid planning models by exploiting observation data from executions has been presented and evaluated. In order to exploit the information content of the original model, its effects are modified in order to better fit the observed data. Our approach relies on learning a decision tree for each process, which captures the relationship between state functions and propositions and the effects of continuous processes. A particular advantage of this approach is that it removes the knowledge engineering effort in producing and maintaining very detailed hand-crafted process models. We presented both a fully automated version, which selects its own features and a collaborative version, which takes advantage of expert selected features. We used our approach to refine a planning model for the Urban Traffic Management domain, which uses continuous processes to model the flow of traffic. We demonstrated that when expert selected features are used, the accuracy of simulation can be improved (e.g., the average reduction in error at the end of simulation in each road network of our case study was over 50%), resulting in a more accurate representation of occupancy of the network over time, as well as plans that are more effective during execution. The fully automated version is not as effective and highlights the main limitations of the approach: the feature set must be suitable and the dataset must be representative and sufficient. The current framework supports exploiting structure present in the original model, which is not always sufficient to capture important phenomena, e.g. a busy cross-flow turn can significantly alter turn rates if it ever gets full. In future work we will combine our approach with a framework for extending the planning model with additional features, e.g., based on derived predicates (de la Rosa and McIlraith 2011) or on identification of relevant features through structural analysis (Lindsay 2015).

References

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Barceló, J., and Casas, J. 2005. Dynamic network simulation with aimsun. In *Simulation approaches in transportation analysis*. Springer. 57–98.

Benson, S. 1996. *Learning Action Models for Reactive Autonomous Agents*. Ph.D. Dissertation, Stanford University.

Clement, B. J.; Frank, J. D.; Chachere, J. M.; Smith, T. B.; and Swanson, K. 2011. The challenge of grounding planning in simulation in an interactive model development environment. In *Proceedings of the ICAPS Workshop on Knowlege Engineering for Planning and Scheduling (KEPS)*.

Cresswell, S., and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

de la Rosa, T., and McIlraith, S. 2011. Learning domain control knowledge for tlplan and beyond. In *Proceedings of the ICAPS Workshop on Planning and Learning (PAL)*.

De'Ath, G. 2002. Multivariate regression trees: a new technique for modeling species–environment relationships. *Ecology* 83(4):1105–1117.

Fox, M., and Long, D. 2002. PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, volume 4, 34.

Frank, J. 2015. Reflecting on planning models: A challenge for self-modeling systems. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*.

Gregory, P., and Lindsay, A. 2016. Domain Model Acquisition in Domains with Action Costs. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Hall, M. A. 1998. *Correlation-based Feature Subset Selection for Machine Learning*. Ph.D. Dissertation, University of Waikato, Hamilton, New Zealand.

Lanchas, J.; Jiménez, S.; Fernández, F.; and Borrajo, D. 2007. Learning action durations from executions. In *Proceedings of the ICAPS Workshop on AI Planning and Learning (AIPL)*.

Lindsay, A.; Read, J.; Ferreira, J. F.; Hayton, T.; Porteous, J.; and Gregory, P. J. 2017. Framer: Planning models from natural language action descriptions. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Lindsay, A. 2015. *Problem Models for Rule Based Planning*. Ph.D. Dissertation, Department of Computer and Information Sciences, Strathclyde University, UK.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning using concept languages. In *Proceedings of the 7th International Conference of Knowledge Representation and Reasoning*.

McCluskey, T. L., and Vallati, M. 2017. Embedding automated planning within urban traffic management operations. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Mourão, K.; Petrick, R. P. A.; and Steedman, M. 2010. Learning action effects in partially observable domains. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*. IOS Press.

Say, B.; Wu, G.; Zhou, Y. Q.; and Sanner, S. 2017. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *IJCAI*, 750–756.

Scala, E.; Haslum, P.; Thiébaux, S.; et al. 2016. Heuristics for numeric planning via subgoaling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Segura-Muros, J. Á.; Pérez, R.; and Fernández-Olivares, J. 2018. Learning numerical action models from noisy and partially observable states by means of inductive rule learning techniques. In *Proceedings of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Vallati, M.; Magazzeni, D.; De Schutter, B.; Chrpa, L.; and McCluskey, T. L. 2016. Efficient macroscopic urban traffic models for reducing congestion: A pddl+ planning approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Wu, K.; Yang, Q.; and Jiang, Y. 2007. ARMS: An automatic knowledge engineering tool for learning action models for AI planning. *The Knowledge Engineering Review* 22(2):135–152.

Yoon, S.; Fern, A.; and Givan, R. 2007. Using learned policies in heuristic-search planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Zhuo, H. H., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence* 174(18):1540–1569.

Zhuo, H. H.; Nguyen, T. A.; and Kambhampati, S. 2013. Refining incomplete planning domain models through plan traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2451–2458.