Journal of Artificial Intelligence Research 69 (2020) 765-806 Submitted 06/2019; published 11/2020

Using Machine Learning for Decreasing State Uncertainty in Planning

Senka Krivic SENKA.KRIVIC@KCL.AC.UK
King's College London
Bush House, 30 Aldwych, London
WC2B 4BG, United Kingdom

Michael Cashmore MICHAEL.CASHMORE@STRATH.AC.UK
University of Strathclyde
26 Richmond Street, G1 1XH, Scotland UK

Daniele Magazzeni DANIELE.MAGAZZENI@KCL.AC.UK
King's College London
Bush House, 30 Aldwych, London
WC2B 4BG, United Kingdom

Sandor Szedmak SANDOR.SZEDMAK@AALTO.FI
Aalto University
Aalto SCI Computer Science Konemiehentie 2, Finland

Justus Piater JUSTUS.PIATER@UIBK.AC.AT
University of Innsbruck
Technikerstr. 21a, 6020 Innsbruck, Austria

Abstract

We present a novel approach for decreasing state uncertainty in planning prior to solving the planning problem. This is done by making predictions about the state based on currently known information, using machine learning techniques. For domains where uncertainty is high, we define an active learning process for identifying which information, once sensed, will best improve the accuracy of predictions.

We demonstrate that an agent is able to solve problems with uncertainties in the state with less planning effort compared to standard planning techniques. Moreover, agents can solve problems for which they could not find valid plans without using predictions. Experimental results also demonstrate that using our active learning process for identifying information to be sensed leads to gathering information that improves the prediction process.

1. Introduction

Consider a service robot which operates in a dynamic and complex household environment to help people in daily tasks such as serving lunch or tidying a room. Environments such as a household contain hundreds of objects with various properties. The robot is equipped with many different manipulation actions to interact with these objects, i.e. grasping, pulling, pushing, placing, or dropping, as well as sensing actions to detect an object's properties. It is difficult for a robot to have a complete and up-to-date model of the environment due to the environment's complexity and dynamic nature. Humans interact with the environment, constantly changing the state of it, for example by introducing new objects or changing

their positions. In this example, sensing and exploration are costly, and the robot cannot sense every type of unknown information. Additionally, not all knowledge can be modelled or inserted by a human operator, so the robot must make some assumptions about the state to complete its task. However, some actions have irreversible effects on the objects, a dropped and broken mug cannot be repaired. For this reason, the assumptions must be as accurate as possible.

In robotics and control, partial-knowledge is often used to estimate the state of a dynamic system or a dynamic model (Barfoot, 2017; Krivic & Piater, 2018). Similarly, instead of sensing all unknown information, a robot could exploit the similarities between objects in the environment based on the information it already possesses, and predict the missing knowledge. For example, suppose the robot knows some properties for a new object. In that case, it can predict whether that object can be dropped without breaking by exploiting similarities with other objects. We utilize this idea and embed it to a standard planning procedure.

Automated planning uses a model of the agents and their environment to reason over complex choices of action, and thereby achieve goals. However, planning in real-world dynamic environments often means planning with incomplete and uncertain information. This is often the case in robotic domains, as described in the motivating example. Domains such as this are complex due to a large number of objects and areas that are unknown to a robot as well as a large number of object properties. Exploration and observation can be costly, and a scenario like this may contain some information that cannot even be sensed. In real-world dynamic environments, planning often has to be completed quickly, and this uncertainty has severe consequences: the planning problem becomes more complex, taking longer to solve, and exacerbating scalability issues.

Uncertainty in planning problems is handled with a variety of different approaches. For example, approaches such as conformant planning (Smith & Weld, 1998; Palacios & Geffner, 2006, 2009), contingency planning (Bonet & Geffner, 2000; Hoffmann & Brafman, 2005), probabilistic planning (Camacho, Muise, & McIlraith, 2016), or replanning techniques (Brafman & Shani, 2012) take some uncertainties into account. However, handling uncertainty in the planning process with these approaches comes with complexity and robustness costs. Moreover, if the lack of information is severe, then the problem can be rendered unsolvable.

We introduce a new approach based on a state prediction and optional active-learning step between sensing and planning in a standard planning and execution loop, as shown in Figure 1. The input to the prediction step is a single incomplete state. The output of the prediction step is an estimated state. The outcome of the active learning component is a set of observation goals that can be added to the problem description. Any domain in the Planning Domain Description Language (PDDL) (McDermott et al., 1998; Fox & Long, 2003) family of languages can be used with our proposed approach.

In this way, the uncertainty is reduced by making predictions about the current state before each planning or replanning process. We translate the incomplete state to a partially-known multigraph in order to apply machine learning techniques. This enables us to capture the underlying structure of the state. Unknown edges in the multigraph are predicted by exploiting the similarities between known properties. Each predicted edge is labelled with

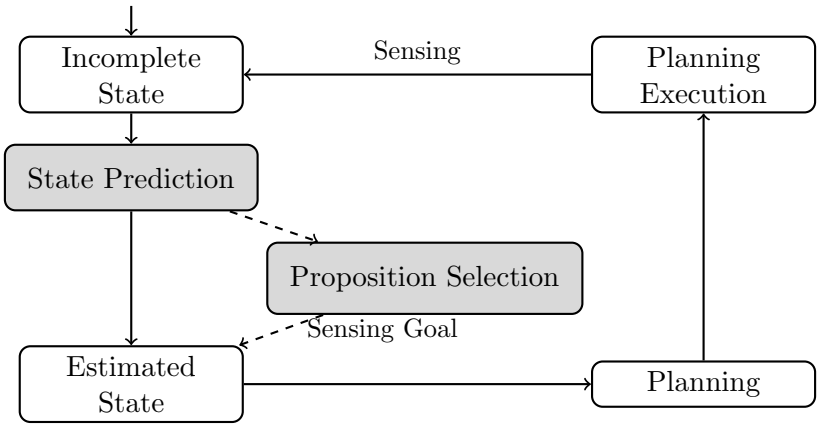


Figure 1: The proposed approach for decreasing state uncertainty in planning using predictions. When confidences for predictions are low, the agent is guided to acquire the necessary information to improve predictions.

a prediction confidence measure. Edges with high prediction confidence are translated back to the state, decreasing uncertainty.

With new information from predictions in the state, the cost of problem-solving is smaller, and generated plans require fewer sensing actions. Although incorrect predictions can increase the chance of plan failure during execution, we show in our evaluation over a range of planning domains that the accuracy of the predictions is very high. Failures in predictions are rare, and they appear when there is a small amount of initial knowledge about the state.

The confidence and accuracy of the prediction depend on the available information about the state. For problems with very high uncertainty in the initial state, confidences for predictions can be very low. To be able to solve a planning problem in such cases, an agent might have to sense some missing information and replan until the completion of a task is possible or generate a contingent plan that branches upon the outcome of sensing actions. We pose this problem in an active learning setting. We formulate a process to select unknown propositions to be sensed. These propositions are those for which the least is known, and whose sensing will best improve the overall prediction accuracy. This active learning strategy enables state exploration, and it generates new goals for the planner to sense the missing information.

In addition to this, given the complexity of the domain and the planning task, it is possible that some information cannot be sensed by the actions available to the agent. In this case, we use the active learning procedure for generating new goals to observe information that will best improve the prediction of the unavailable information. For example, in robotics scenarios, the robot can have a number of available sensing actions, such as tactile sensing, localization, object classification, and so on. However, much information cannot be sensed and is not directly available to the robot. In scenarios like this, the robot can select

which information should be sensed to improve the prediction confidence over the useful information.

We implemented an integrated prediction, planning, and execution system. The system uses an adapted version of *Maximum Margin Multi-Valued Regression* (M^3VR) (Krivic et al., 2015) for learning the model of the state, integrated with ROSPlan (Cashmore et al., 2015), a framework for planning with robotics. The contingent Closed-Loop Greedy Planner (CLG) (Albore & Geffner, 2009) and POPF (Coles et al., 2010) are used to solve the resulting planning problems.

Experimental results show positive impact in terms of reducing uncertainty in the planning problem and improving abilities to solve them compared to standard planning approaches. More precisely, we demonstrate that prediction increases scalability when contingency planning with CLG. We also show that using the active learning approach enables a higher prediction accuracy with fewer sensing actions taken. Finally, using a deterministic planner, we investigate the impact of prediction on the robustness of plans, evaluating the negative impact of incorrect predictions against different baselines. The benchmark set for our experimental evaluation was generated by removing literals from problems with complete information. The literals considered for removal were those that could not be predicted through domain axioms alone, and so the benchmarks could sensibly be used as problems in planning with incomplete information.

In this paper, we present a significantly more detailed description of the prediction step that was introduced in our previous work on initial state prediction (Krivic et al., 2017). Additionally, we present the state prediction as part of a replanning approach shown in Figure 1, extend the prediction process to n-ary predicates, define the confidence measure for a multigraph representation of the planning state, present a procedure for actively selecting sensing goals, and perform a more-in-depth evaluation and discussion. The main contributions of the paper are:

- posing the problem of missing information in the state as the problem of partially-known multigraph completion;
- a novel approach which utilizes machine learning techniques to predict missing information in a partially-known state;
- devising active learning procedures for improving predictions of the state by generating new sensing subgoals.

The paper is structured as follows. First we give some background in the problem of planning under uncertainty. In Section 3, we present the problem of state prediction using machine learning. Section 4 describes the active learning process for identifying the new sensing actions needed for the improvement of predictions. In Section 5 we present the experimental results. We discuss the proposed approach and compare it with the related work in Section 6 and we conclude in Section 7.

2. Background

This section describes basic concepts in automated planning, and planning under uncertainty.

2.1 Automated Planning Problems with Incomplete Initial States

Definition 1 (Planning Instance with Incomplete Initial State) *A planning instance with incomplete initial state* Π *is a pair* $\langle D, P \rangle$*, where domain* $D = \langle Pred, Func, A, arity \rangle$ *is a tuple consisting of a finite set of predicate symbols* $Pred$*, function symbols* $Func$*, a finite set of actions* A*, and a function arity mapping all symbols in* $Pred$ *and* $Func$ *to their respective arities. The triple describing the problem with incomplete initial state* $P = \langle O, s', G \rangle$ *consists of a finite set of domain objects* O*, the incomplete initial state* s'*, and the goal specification* G*.*

The atoms of the planning instance, also referred to as propositions, are the (finitely many) expressions formed by grounding, i.e., applying the predicate symbols $Pred$ to the objects in O (respecting arities). The resultant expressions are the set of propositions $Prop$. The numeric-valued fluents of the planning instance are formed in a similar fashion by grounding the function symbols. The goal specification G is a formula over the propositions and numeric fluents. PDDL (McDermott et al., 1998; Fox & Long, 2003) is used to describe both the planning domain D and problem P. Actions are not specified since the proposed approach is flexible with respect to the action type. However, the approach will need some adaptation to work with PDDL actions which involve time or numeric quantities. A simple example is shown in Figure 2 and Figure 3.

```
;; PDDL domain description of the simply tidy task
(define (domain tidy-room-simple)
(:requirements :strips :typing :durative-actions)
(:types object area edge)
(:predicates (untidy ?o - object) (tidy ?o - object)
             (explored ?r - area) (at ?r - area) (visited ?r - area)
(:durative-action push
    :parameters (?o - object)
    :duration ( = ?duration 10)
    :effect (at end (and (tidy ?o) (not (untidy ?o)))))
(:durative-action explore
    :parameters (?r - area)
    :duration ( = ?duration 120)
    :effect (at end (explored ?r)))
(:durative-action visit
    :parameters (?r1 ?r2 - area)
    :condition (and (at start (at ?r1)))
    :duration ( = ?duration 100)
    :effect (and (at start (not (at ?r1)))
            (at end (visited ?r2)) (at end (at ?r2)))
)
```

Figure 2: The simplified planning domain for a tidying task encoded in PDDL.

```
;; PDDL problem description for the simple tidy domain
(define (problem sample-problem)
(:domain tidy-room-simple)
(:objects ob0 ob1 - object room0 - area)
(:init (untidy ob0) (untidy ob1))
(:goal (and (tidy ob0) (tidy ob1))))
```

Figure 3: A sample problem for the simple tidying task encoded in PDDL.

```
;; PDDL problem description with partial observability
(define (problem sample-travel)
(:domain tidy-room-simple)
(:objects door table sofa - area)
(:init (and (edge door table) (edge table door)
        (edge table sofa) (edge sofa table)
        (edge sofa door) (edge door sofa)
        (oneof (at door) (at table) (at sofa))))
(:goal (and (visited door) (visited table) (visited sofa)))
)
```

Figure 4: A conformant planning task in which the initial position of a robot is uncertain.

The set of propositions for the planning instance is obtained by grounding the predicates as described above to obtain the set:

$$(tidy\, ob0)\ (tidy\, ob1)$$
$$(untidy\, ob0)\ (untidy\, ob1)$$
$$(explored\, room0)$$

A literal l_p represents a valuation of one proposition $p \in Prop$, i.e. l_p and $\neg l_p$. A single state s is described by a set of literals formed from the propositions in $Prop$, $\{l_p, \neg l_p, \forall p \in Prop\}$. A state is *well-formed* if it does not include conflicting literals, i.e. if $l_p \in s$ then $\neg l_p \notin s$. If every proposition from $Prop$ is represented by a literal in the state, then we say that s is a *complete state*. Otherwise, we say that s is an *incomplete state*.

PDDL makes a closed-world assumption about the state. If the literal l_p is not in the state s, then it is assumed that $\neg l_p \in s$ (every state is a complete state). In contrast, we assume that if $\neg l_p \notin s$ and $l_p \notin s$ then instead the value of proposition is unknown. Contingent and Conformant planning reason over *incomplete information* and also make no closed-world assumption.

The approach presented in this paper applies to problems with a well-formed, incomplete initial state.

2.2 Contingent and Conformant Planning

Contingent planning tackles the problem of planning with *incomplete information*. In a problem with incomplete information the initial state is a set of *clauses* over the set of

propositions defining the initial situation, as defined by Albore and Geffner (2009). This is different from the notion of incomplete state in that the clauses can define relationships between propositions, such as *oneof* constraints. An example of a conformant planning problem is shown in Figure 4. This problem describes an uncertain tidy room problem, in which the initial position of the robot is unknown. The initial position of the robot is represented by the disjunction:

$$(oneof\ (at\ door)\ (at\ table)\ (at\ sofa))$$

The notion of a *incomplete state* in this paper does not include this information provided by these constraints. For example, in Figure 4, the three *at* propositions are unknown, and given the *oneof* constraint, only three possible complete initial states can be true. In contrast, by representing the initial state as an incomplete state in which these three propositions are unknown, there are $2^3 = 8$ possible initial states.

An incomplete planning state s can be represented by a state with incomplete information in the following way: for each proposition p not represented by a literal in s, the state with incomplete information can include the clause $(oneof\ l_p\ \neg l_p)$. A state with incomplete information can be represented by an incomplete state by including only those propositions whose values are known to be true or false. Note that this loses information.

A *conformant* solution to a problem with incomplete information, produced by planners such as Conformant-FF (Brafman & Hoffmann, 2004) and POND (Bryce, Kambhampati, & Smith, 2006), is a sequence of actions that will achieve the goals for every possible complete initial state. No sensing actions can be taken. When sensory actions are expensive or dangerous, as is often the case in robotics planning, then conformant planning may be the best approach (Smith & Weld, 1998). However, in many real-world robotics domains with partial observability, a conformant plan does not exist. As an example, sensing which elevator has arrived in an office building, and branching execution to enter the right door is possible, producing a solution that can be executed without relying on sensory information is more difficult.

A *contingent* solution to a problem with incomplete information uses sensing actions to observe missing information and its execution branches upon the result. Contingent planners, such as Contingent-FF (Hoffmann & Brafman, 2005) and CLG (Albore & Geffner, 2009), are useful in such domains, where it cannot be assumed that a conformant solution exists. A contingent solution assumes that uncertainty is monotonically decreasing, i.e., once the value of a proposition has been observed by a sensing action, then it cannot become unknown again. A contingent solution also assumes that there is no known probability distribution over the possible results of the sensing action.

The domains we consider in this paper are those in which a conformant solution is difficult, or not feasible, and sensing actions are expensive, such as robotics domains, e.g. the Mars rover domain (Francis et al., 2017).

2.3 Probabilistic Planning

If the probability distribution over a non-deterministic effect of an action is known, then the problem could be encoded as a probabilistic PDDL (PPDDL) instance (Younes & Littman, 2004). Probabilistic planning with PPDDL extends PDDL with probabilistic effects, which

```
;; PPDDL domain description of the simply tidy task
(define (domain tidy-room-simple-probabilistic)
(:requirements :conditional-effects :probabilistic-effects)
(:types object room)
(:predicates (untidy ?o - object) (tidy ?o - object))
(:durative-action push
    :parameters (?o - object)
    :effect (at end (probabilistic 0.75 (and (tidy ?o) (not (untidy ?o))))))
)
;; PPDDL problem
(define (problem probabilistic-sample)
(:domain tidy-room-simple-probabilistic)
(:requirements :negative-preconditions)
(:objects ob0 ob1 - object)
(:init
  (probabilistic 0.75 (untidy ob0))
  (probabilistic 0.85 (untidy ob1)))
(:goal (and (tidy ob0) (tidy ob1))))
```

Figure 5: PPDDL encoding of the probabilistic effect example.

allows the specification of Markov decision processes (MDPs). An example PPDDL domain and the problem is shown in Figure 5. In this problem, there are two objects, both of which have some probability of being untidy. The objects can be tidied by pushing them away, but there is a probability that this action fails. PPDDL allows specifying action cost and planning horizons, and the objective is generally defined in the form of minimizing the expected cost.

Planning with non-deterministic effects, including observations encoded as sensing actions with probabilistic effects, differs from contingent planning. Contingent sensing actions for partial observability assume that once the fact is known, it does not change by repeated sensing. We make the same assumption in our domains.

3. State Prediction

In this section, we describe in detail our approach to the problem of state prediction in the case of partial observability.

3.1 Problem Description and Proposed Approach

The problem is, given an incomplete initial state s' as defined in Definition 1, to add at least one new literal to the state so that the state remains well-formed. We call this function *extending a incomplete state* as defined below.

Definition 2 (Extending an Incomplete State) *Let s' be an incomplete state of a planning problem Π. Extending the state s' is a function $Extend(\Pi, s') : s' \rightarrow s''$ where $s' \subset s''$ and $s'' \subseteq s$, where s is a well-formed complete state.*

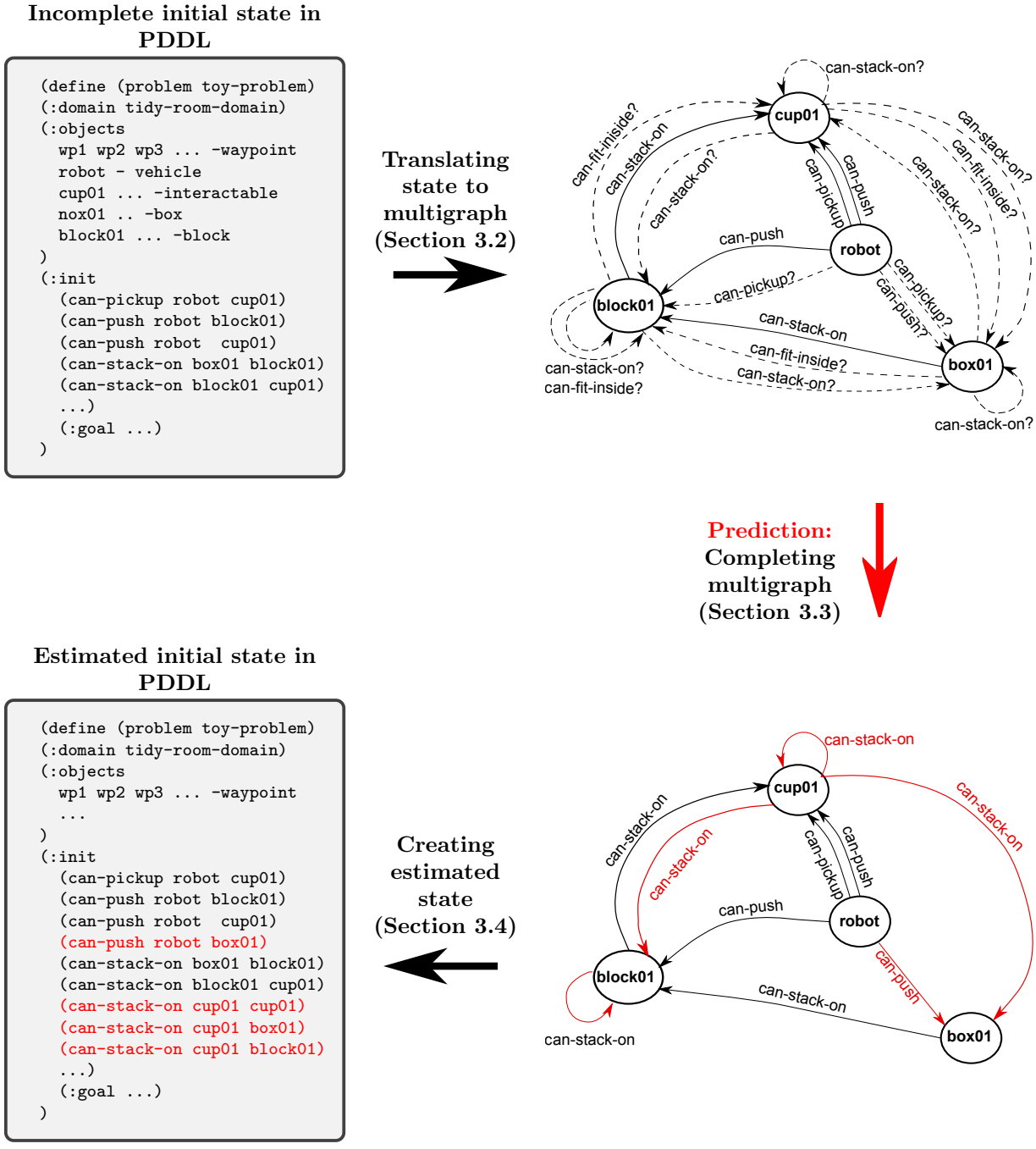


Figure 6: A fragment of an example problem from the *tidy-room* domain where a prediction step is applied. First, the state is translated to the multigraph. Known edges are denoted by solid lines and unknown ones by dashed lines. Then, using prediction, a multigraph is completed. In the last step, the estimated edges are translated back to literals and are added to the state.

The procedure of extending an incomplete state by adding a prediction step is summarised in Figure 6. To be able to apply a machine learning technique to extend an partially-known state, we first translate it to a multigraph. Thus, the function $Extend(\Pi, s')$ is implemented as follows:

1. The incomplete state s' is converted into a multigraph (Section 3.2).
2. Learn a predictive model for prediction of missing edges and extend the multigraph (Section 3.3).
3. The new edges are added as literals to the incomplete state (Section 3.4).

3.2 Constructing the Multigraph

We use E to denote a set of edge values in a complete multigraph M and E' to denote the set of edge values in a partially-known multigraph M'. We represent an incomplete state s' as a partially-known multigraph M', in which vertices represent objects and edges represent propositions.

Definition 3 (Partially-Known Multigraph) *A partially-known multigraph M' is a pair $\langle V, E' \rangle$, where V is a set of vertices, and E' a set of directed edge values.*

An incomplete state s' is described as a partially-known multigraph with vertices corresponding directly to the PDDL objects. That is,

$$V \equiv O.$$

We use the function

$$L_{pred} : \nu \rightarrow \{0, 1, ?\} \tag{1}$$

to describe the existence of an edge in a partially known multigraph, where $\nu \subseteq V$ is the set of objects used to ground the predicate. The set $\{0, 1, ?\}$ corresponds to $\{$*not-existing*, *existing*, *unknown*$\}$. For example, consider a proposition p, formed by grounding the predicate *pred* with objects b and u. Existences of edges are determined as follows.

$$\begin{gathered} \neg l_p \in s' \leftrightarrow L_{pred}(b, u) = 0 \\ l_p \in s' \leftrightarrow L_{pred}(b, u) = 1 \\ (l_p \notin s' \wedge \neg l_p \notin s') \leftrightarrow L_{pred}(b, u) =? \end{gathered}$$

For each predicate and set of objects used to ground the predicate, a directed edge exists in the multigraph of s' if and only if $L_{pred}(\nu) \neq 0$. Therefore, an edge exists in the partially-known multigraph if the corresponding proposition exists in the state or if the proposition is unknown. The order of objects in a proposition's parameters is important, and therefore edges in the multigraph are directed in the order the object symbols appear in the proposition.

Since there can be several edges between two vertices, we form an *edge-vector* for each pair of origin and destination vertices. For an origin object b and a destination object u, we define the *edge-vector* as the vector:

$$\mathbf{e}_{bu} = \left[e_{bu}^{pred}, \forall pred \in Pred \right] = [L_{pred}(b, u), \forall pred \in Pred] \tag{2}$$

This vector describes the existence of all edges directed from b to u. For example, with objects b and u and predicates $Pred = \{pred1, pred2, pred3, pred4\}$, an edge vector $\mathbf{e}_{bu}$ could look like:

$$\mathbf{e}_{bu} = \begin{bmatrix} e_{bu}^{pred1} \\ e_{bu}^{pred2} \\ e_{bu}^{pred3} \\ e_{bu}^{pred4} \end{bmatrix} = \begin{bmatrix} L_{pred1}(b,u) \\ L_{pred2}(b,u) \\ L_{pred3}(b,u) \\ L_{pred4}(b,u) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ ? \\ 0 \end{bmatrix}$$

3.2.1 Example Multigraph

Observe the problem where a robot is able to move between waypoints and make simple operations with objects such as: pick up, push, stack, and drop in box. The predicates `can-pickup`, `can-push`, `can-stack-on`, and `can-fit-inside` describe if these actions are possible, as shown in Figure 6.

In this problem we restrict our attention to four objects: `robot`, `cup01`, `box01`, and `block01`. In PDDL literals that are absent from the state are assumed to be false. However, as described in Section 2.2 we assume those literals to be unknown.

A graph M is generated, and its vertices are

$$O := \{robot, cup01, box01, block01\}$$

As an example, an edge-vector between the vertices *robot* and *block01* in this multigraph is given with:

$$\mathbf{e}_{robot,block01} = \begin{bmatrix} e_{robot,block01}^{can-fit-inside} \\ e_{robot,block01}^{can-stack-on} \\ e_{robot,block01}^{can-push} \\ e_{robot,block01}^{can-pickup} \end{bmatrix} = \begin{bmatrix} L_{can-fit-inside}(robot, block01) \\ L_{can-stack-on}(robot, block01) \\ L_{can-push}(robot, block01) \\ L_{can-pickup}(robot, block01) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ ? \end{bmatrix} \quad (3)$$

In the example state shown in Figure 6, it is known that the propositions `(can-fit-inside robot block01)` and `(can-stack-on robot block01)` are false. Therefore, the edges that correspond to them do not exist in the multigraph. The state also contains information about the proposition `(can-push robot block01)`, which is known to be true. Finally, the proposition `(can-pickup robot block01)` is unknown.

3.3 Completing a Partially-Known Multigraph

Once a multigraph representing the incomplete state is created, a machine learning method can be used for completing a multigraph where edges are partially known. With the assumption that the planning state has an underlying structure, and it is not describing a system without any order, laws or structure, the multigraph can be completed via a supervised learning problem.

As described in Section 3.2, we differentiate between origin and destination vertices for each edge. B denotes the set of origin vertices and U the set of destination vertices, where $B, U \subseteq V$. Edges between the vertices describe a mapping between the sets B and U (Figure 7).

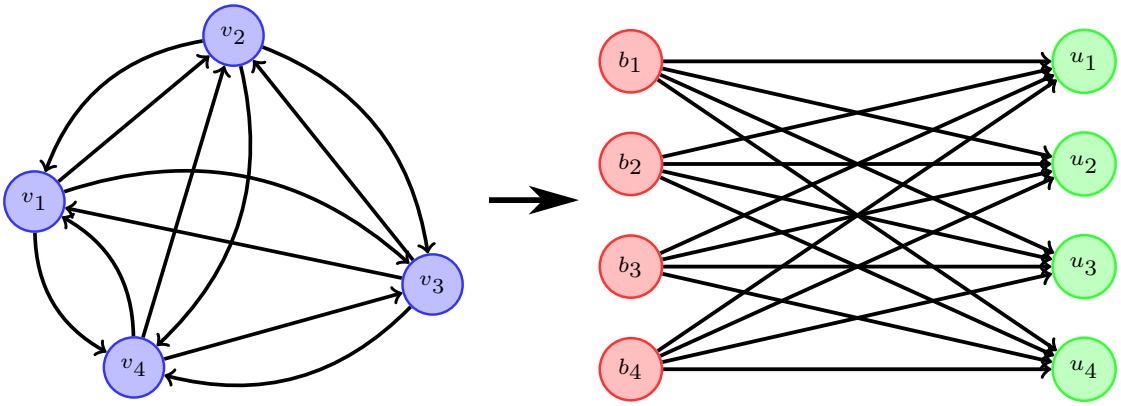


Figure 7: A multigraph representation with directed edges for vertices $v \in V$(left). Another representation of the multigraph where edges connect origin vertices $b \in B$ with destination vertices $u \in U$ where $B, U \subseteq V$(right).

To predict an unknown edge between two vertices, knowledge about other known edges involving those vertices can be exploited. The completion of the multigraph is the task of building a predictive model:

$$m : B \times U \to E \tag{4}$$

which maps inputs (ordered pairs of vertices (b, u)), to the outputs (edges values $\mathbf{e}_{bu}$ defined in Eq. 2) based on the existing knowledge about edges values in a multigraph. The model is trained based on the pairs $((b, u), L_{pred}(b, u))$ for known values $L_{pred}(b, u)$, 0 or 1, in edges $\mathbf{e}_{bu} \in E'$. The predictive model m aims to capture the underlying structure of a multigraph, and therefore the structure of the planning state.

Various machine learning techniques can be used to learn the predictive model m from the known information such as ML methods for completing a multigraph (Cesa-Bianchi et al., 2013; Gentile et al., 2013; Latouche & Rossi, 2015), recommender systems (Jahrer et al., 2010), or data imputation methods (Lin et al., 2011; Bertsimas et al., 2018). We utilize Maximum Margin Multi-Valued Regression (M^3VR) for predicting edges which is described in Appendix A.

Once the predictive model m is created and learned for the incomplete planning state, it is used to predict unknown $L_{pred}(b, u) =?$ values for that state, and so update the function L_{pred} (Eq. 1) In this way, predictions are performed for all unknown edges and the graph completed.

When the agent acquires new information and discovers the true values of some edges, the model m is updated to capture the current planning state. The method in which the model m is updated with new information depends upon the machine learning algorithm used, particularly whether the algorithm is incremental, and whether the planning problem is probabilistic. Incremental algorithms make updates with new information, without retraining the entire model from scratch. In the case that knowledge about the state is probabilistic, it is necessary to train the model from scratch since there can be changes in

the state. M^{3}VR creates the model m of the current single incomplete state each time State Prediction step is triggered (Figure 1).

3.3.1 Completing a Partially-Known Hypergraph

So far we have discussed only binary relations – predicates with only two object parameters (b, u). Now we extend this concept to predictions for n-ary relations. N-ary relations can be represented by hyperedges, which are edges that can join N number of vertices.

A hypergraph is a tuple $\langle V, E \rangle$, where V is a set of vertices, and $E \subseteq 2^V$ is the set of hyperedges. Thus E is a subset of the power set of the vertices. It is assumed that the subsets corresponding to the hyperedges are not empty, and all subsets of V with size 1, the singletons, are included into E. Every hyperedge representing an N-ary relation corresponds to a subset of vertices with size N. A general hypergraph might contain hyperedges of any size.

The extension can be carried out by transforming the corresponding hypergraph into a factor graph. That graph is a bipartite graph with two sets of vertices: one is formed from the original vertices, and the other contains the hyperedges (factored vertices). An edge is drawn between two sets if a hyperedge covers a vertex in the original graph. The order of the arguments of predicates is preserved in the direction of the hyperedge in a new factored vertex by keeping the same order in the factored vertex subscript. To distinguish the connection between an original vertex and the introduced factored vertex, we use the notation of parentheses $(.,.)$ in edges subscripts to denote factored vertices. An example is given in Figure 8.

From the `tidy-room` domain, consider the predicate:

```
(can-drop-at ?robot ?object ?waypoint)
```

Consider the proposition grounded from this predicate using objects o_1, o_2, and o_3. These will be connected by the hyperedge $e_{1,2,3}$ as shown on the left side of Figure 8 in the original hypergraph. To create a multigraph, a factored vertex $h_{2,3}$ is generated and connected with edge $e_{1,(2,3)}$ as shown on the right side of Figure 8.

Edges and vertices in the factor graph can be treated as the edges and vertices in a graph with binary relations, and the prediction process described above can then be directly applied.

3.3.2 Confidence Measure

In the prediction process, the estimation of unknown values in an edge vector is made by exploiting the knowledge about other edges. Based on the assumption that two edges are similar if they share either the same origin b or the same destination vertex u, we can determine a confidence measure for an unknown value in the edge $\mathbf{e}_{bu}$. Let us define the set of all known edge values which share the same origin vertex b as

$$E'_b = \{e^{pred}_{bv} \in E' | v \in U, pred \in Pred\} \tag{5}$$

and the set of all known edge values which share the same destination vertex u as

$$E'_u = \{e^{pred}_{vu} \in E' | v \in B, pred \in Pred\}. \tag{6}$$

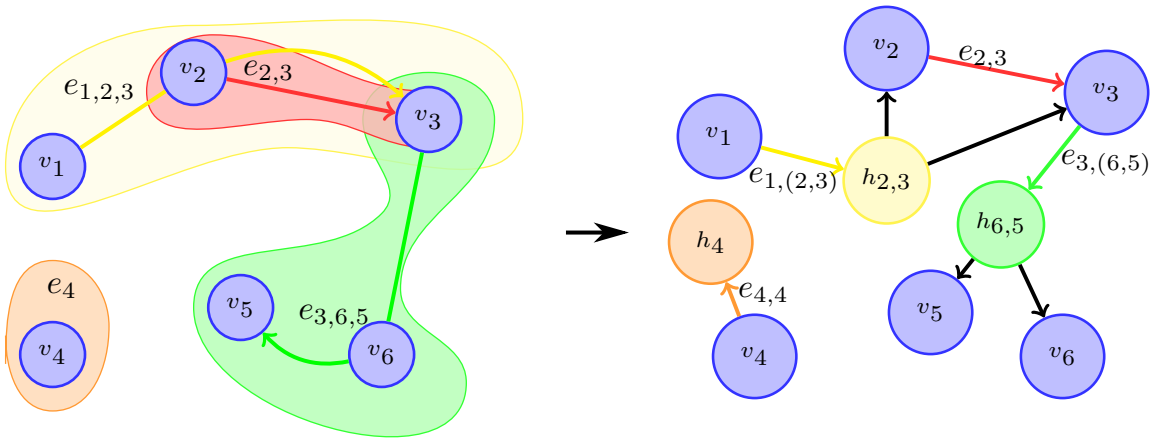


Figure 8: Transforming hypergraph into a factor graph by adding factored vertices h_4, $h_{2,3}$ and $h_{6,5}$. As the edge $e_{2,3}$ connects two vertices, it does not require adding any factored vertices. In the factored graph on the right side, introduced factored vertices are denoted in edges with parentheses, e.g. the edge $e_{1,(2,3)}$ is an edge from vertex v_1 to the vertex $h_{2,3}$. In this way, the order of predicates in the direction of the hyperedge is preserved.

If there is no information about edges with vertices b or u, sets E'_b and E'_u are empty. This also means that there is no knowledge which can be used for prediction of the missing edge $\mathbf{e}_{bu}$. Therefore, we define the confidence measure as

$$\text{conf}\{\mathbf{e}_{bu}\} = \frac{1}{2}\frac{\mathbf{N}_{E'_b}}{N_{E_b}}(1-\sigma^2_{E'_b}) + \frac{1}{2}\frac{\mathbf{N}_{E'_u}}{N_{E_u}}(1-\sigma^2_{E'_u}) \tag{7}$$

where $\mathbf{N}_{E'_b}$ denotes numbers of known edge values with the vertex b as the origin vertex, and $\sigma^2_{E'_b}$ is variance[1] of those values. Similarly $\mathbf{N}_{E'_u}$ and $\sigma^2_{E'_u}$ are defined for edges that share the same destination vertex u. N_{E_b} denotes numbers of all edge values with the vertex b as the origin vertex, and N_{E_u} denotes numbers of all edge values with the vertex u as the destination vertex. In our case $N_{E_b} = N_{E_u} = N_V * N_{Pred}$ since $B = U = V$. In this way, the uncertainty of the prediction is high if the standard deviation of known edge values is high or if the number of unknown edges exploited for the prediction is high. Confidence values are between 0 and 1.

Example Observe the problem presented in Section 3.2.1 and in Figure 6 top-right where the partially known multigraph is presented. In this example we would like to predict missing value $e^{can-pickup}_{robot,block01}$ in edge $\mathbf{e}_{robot,block01}$ (Eq. 3). The set of all edge values which share the same origin vertex $b = robot$ consists of all values e^{pred}_{bv} of edges

1. The variance σ^2 is defined as the average of the squared deviations from the mean (μ). For a set $X = \{x_1, x_2, ..., x_N\}$, the variance is calculated as $\sigma^2 = \sum_{i=1}^{N}(x_i - \mu)^2/N$.

$\mathbf{e}_{robot,robot}, \mathbf{e}_{robot,box01}, \mathbf{e}_{robot,cup01}$ and $\mathbf{e}_{robot,block01}$. From Figure 6 this is:

$$E_{b=robot} = \{0, 0, 0, 0, 0, 0, ?, ?, 0, 0, 1, 1, 0, 0, 1, ?\}.$$

The set of all edge values that share the same destination vertex $u = block01$ is similarly determined:

$$E_{u=block01} = \{0, 0, 1, ?, ?, 1, 0, 0, 0, ?, 0, 0, ?, ?, 0, 0.\}$$

In this case the number of edge values are $N_{E_b} = N_{E_u} = 16$, $N_{E'_b} = 13$, $N_{E'_u} = 11$ where the set of known edge values which share the same origin vertex $b = robot$ is $E'_{b=robot} = \{0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1\}$ and the set of all known edge values that share the same destination vertex $u = block01$ is $E'_{u=block01} = \{0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0\}$. The variance is calculated for these. The confidence value of the missing link is therefore calculated as:

$$\text{conf}\{e^{can-pickup}_{robot,block01}\} = \frac{1}{2} \cdot \frac{13}{16} \cdot (1 - 0.1775) + \frac{1}{2} \cdot \frac{11}{16} \cdot (1 - 0.1488) = 0.6266$$

.

3.4 Extending a Partially-known State with Predictions

Given a completed multigraph and prediction confidences, we extend the partially-known state s'. We add new literals to the state where the confidence exceeds a confidence threshold, i.e.:

$$\begin{gathered} l_p \in s' \leftrightarrow \\ (L^*_{pred}(b, u) = 1) \wedge conf\{e^{pred}_{bu} = 1\} > c_t, \end{gathered} \tag{8}$$

where l_p is the positive literal of the proposition p which is formed by the predicate $pred$ grounding objects b and u, and c_t is the confidence threshold. We discuss the choice of the threshold in Section 6.

For edges predicted not to exist in a graph with confidence higher than a threshold value c_t, we can extend the incomplete state s' similarly:

$$\begin{gathered} \neg l_p \in s' \leftrightarrow \\ (L^*_{pred}(b, u) = 0) \wedge conf\{e^{pred}_{bu} = 0\} > c_t, \end{gathered} \tag{9}$$

In the example given in Figure 6, the partially-known graph contains a dashed edge which represents that the proposition `(can-pickup robot box01)` is unknown. Before prediction,

$$L_{can-pickup}(robot, box01) = ?$$

After the state prediction, $L_{can-pickup}(robot, box01) = 1$, with confidence higher than $c_t = 0$ and the literal `(can-pickup robot box01)` is added to the state.

Note that as we are completing an *incomplete state* the predictions should be valid with respect to the description of the possible initial situations. For example, if using a contingent planner and performing prediction on the initial state given in Figure 4, it would not be correct to add both (*at door*) and (*at table*) to the initial state, even if both are predicted to be true.

```
(:action classify-object
    :parameters (?r - robot ?wp - waypoint ?o - object)
    :precondition(and
        (at_robot ?r ?wp)
        (at_object ?o ?wp))
    :observe (at end (is_class ?o))
    :effect()
)
```

Figure 9: Example of sensing action.

4. Guiding an Agent to Improve Predictions

For cases where the uncertainty in the state is too high, such that the state cannot be predicted with high confidence, we propose to use an active learning approach. This approach is used to identify the best sensing actions for observing the information that will increase the confidence of predictions. For example, in the robot scenario described in Section 3.2.1, there are many objects of a similar type, and the robot has low confidence about whether these objects can be grasped. In this situation, it is desirable to perform sensing actions that will generate the most significant improvement in predictions over the unknown information, such as attempting to grasp one of the many objects. The robot could sense whether it is possible to grasp one such object, and so predict that the similar objects can also be grasped with increased confidence. This approach allows us to reduce the number of sensing actions needed in general to achieve the planning task. An example of a sensing action is given in Figure 9. This sensing action observes the value of the proposition `(is_class ?o)` for some object described by parameter `'?o'`.

During the last decades, active learning has evolved into a popular paradigm to improve the accuracy of learning algorithms (Cohn, Ghahramani, & Jordan, 1996). It is widely used for data collection and annotation in pattern recognition (Li & Sethi, 2006), and natural language processing (NLP) (Tong & Koller, 2002; Chen et al., 2015). It works by selecting the most informative sample from unlabelled data and querying information about it. In many settings, this information is provided by a user.

Following this idea, we guide an agent to acquire information with an active learning approach. This process is denoted as *Proposition Selection* in Figure 1. It is an optional step that can be performed when the uncertainty in the state is high, and thus the process is represented by dashed lines. The main problem for this step is how to choose which information should be sensed. This information is chosen for one of two reasons:

1. **Greedy Task Solving:** to select specific information to help solve the planning task (Section 4.2). When solving a planning problem, the planner selects sensing actions to obtain missing information necessary for completing the task. However, in some situations, that information cannot be sensed or is expensive to sense. For those cases, we present an active learning approach that selects different information to be sensed that will increase the confidence of prediction over the useful information. For example, in the problem given in the Figure 6 the robot cannot sense if it can pick

up the object 'cup01' without simply trying that action, but attempting the action and failing can break the cup. In this case, the robot can improve its prediction using other knowledge in the state that is safe to observe, such as picking up similar objects that are not in danger of breaking.

2. **State Exploration:** to select information used for general exploration of the state (Section 4.3) when there is much information about which prediction confidence is low, and for which performing comprehensive sensing is expensive. Propositions can be selected to improve the overall confidence of predictions without targeting any specific predictions that are important to a single planning problem.

Below we formally define the Active Learner (Section 4.1) and how it is used to select propositions (Sections 4.2 and 4.3) with two different motivations. Then we describe how this selection produces goals that are added to the planning problem (Section 4.4).

4.1 Active Learner

Based on the confidence measure $\text{conf}\{\mathbf{e}_{bu}\}$ defined in Section 3.3.2, we select information important to the predictive model. This information represents an unknown proposition which needs to be observed by the agent. This is realized with an approach similar to uncertainty pool-based sampling (Lewis & Gale, 1994) which is based on selecting the least confident sample from a pool of unknown data.

The assumption is that the most uncertain samples are the ones with the lowest confidence. The confidence measure of an edge captures the information provided by known neighbours. As a consequence, the confidence measures for all edges indirectly depend on each other via the topology of the graph. Therefore selecting an edge with the lowest confidence is not independent of other unknown edges. That underlying interdependence is also a source of information to update the confidences when new information, a measurement on a selected edge, becomes available by recomputing the predictor function F.

An agent decides which sample to retrieve in order to provide more information to the predictor F, and to improve the prediction accuracy and confidences. This selection is determined by the active learner, defined as follows:

Definition 4 (Active learner) *An active learner is a quintuple $\langle F, E, E', S, P\rangle$ where F is a predictor function learned on a set of known values $E' \subset E$. $P \subseteq E \setminus E'$ is the set of unknown values from which information is selected to be sensed. S is a function which selects a set $I \subseteq P$ of informative sample edges.*

We describe a procedure implementing function S.

1. The incomplete state s' is converted into a multigraph.
2. Edges in the multigraph are learned using a predictive model F.
3. Given the confidences of predictions and set of missing values P, a set of unknown edges are selected based on the prediction confidence values:

$$I = \{e_{bu}^{pred} | \, \text{conf}(e_{bu}^{pred}) = \arg\min_{e_{bu}^{pred} \in P} \text{conf}(e_{bu}^{pred})\}$$

The active learner can be used to sample from different sets P based on the domain, the planning task and available information. As previously noted, we present the selection of information for two reasons. These processes differ in the definition of the set P as described below.

4.2 Selecting Information to Solve the Planning Task

When solving a specific planning task, it can happen that a proposition unknown to a planner is also not observable, but is essential in solving the planning problem. For example in the *tidy-room* domain, which contains a large number of different objects, some properties cannot be observed but are significant for decision making, such as whether an object can be pushed of picked up. The two propositions that describe this property can form a disjunctive landmark in completing the goal of tidying the object. Trying to execute some actions in order to gain knowledge can be harmful to the robot or an object, e.g. if the object is too heavy, contains liquids, or is very fragile. Rather than sensing the property directly, a robot can make observations of some other properties of the object or a similar object which will improve its predictions. In a case like this, it is of interest to improve predictions for the information about the unknown literal l_p represented by a specific unknown edge value e_{bu}^{pred}.

To improve prediction of the specific missing information represented by the edge value e_{bu}^{pred} in the multigraph, the function S will select from a set of samples P that are the most relevant for the prediction of this unknown edge value. These are edges which share the same origin b or the same destination u with the edge $\mathbf{e}_{bu}$. The unknown values of these edges define the set P from which informative samples are selected. We call this strategy *Greedy Task Solving*.

4.3 Selecting Information to Explore the Domain

Selecting which information to observe first can also be used in domains where the agent has a very small amount of initial information available. In the example problem of the *tidy-room* domain, this would be the case when the robot enters a child's room for the first time where almost all objects and their properties are unobserved. Immediate planning with the task of tidying up the room can fail due to the lack of information. Instead, a robot can make some observations to gain more knowledge about the state.

An approach to choose which sensing actions an agent should take first could be random or defined by some domain or task-specific exploration strategy. However, in order to minimise sensing actions needed to perform general exploration, we suggest using the active learner by selecting propositions from all unknown data in the state. Thus, the set P contains all unknown values of E. We call this strategy *State Exploration*.

We make a trade-off similar to the exploration vs. exploitation in reinforcement learning strategies: the set I is selected with an ϵ-greedy strategy. The information in the state for which the agent is least certain about is selected with a probability of 1-ϵ, and a random unknown value to be sensed is selected with a probability ϵ. This prevents possible over-fitting or under-fitting of the prediction m in Eq. 4. The ϵ component can play a significant role in domains where the domain is highly complex or in real-world executions where the model is not entirely describing the real-world state. For the case $\epsilon = 0.0$, the strategy

becomes purely greedy. The values of ϵ are usually chosen to be small so that the greedy approach selects the unknown information to be sensed. Therefore, the least certain information based on the confidence measure defined with Eq. 7 is often chosen, while sometimes the algorithm explores the other randomly selected unknown values. This approach ensures that all the space of unknown information about the state is explored.

4.4 Introducing Sensing Goals in the Planning Problem

Given the set of propositions that should be sensed, $p \in Prop$, represented by the edges described by I, goals to sense these propositions are added to the planning problem. To do this, for each such edge value $e_{bu}^{pred} \in I$, a new proposition is added to the planning problem: `(sensed_`p`)`. This new proposition describes whether the proposition p has been observed, and is added as an effect of each sensing action that observes p.

The goal G of the planning problem is then updated to include that these propositions must be observed. These new propositions `(sensed_`p`)` are either added as new goals to the estimated state (see Figure 1), or replace the existing goals. In both cases, any valid plan must involve an action that observes the information important to the predictive model. This information is then used in future planning iterations.

In this way, the plan is forced to include that observation. The planner can also decide on other necessary sensing actions to achieve the current goal. Also, if there can be more than one sensing action to observe the same proposition, then the planner is able to make the choice about which sensing action is used, considering costs or other constraints.

5. Evaluation

We integrated the prediction process into the planning and execution framework ROSPlan (Cashmore et al., 2015) in the Robot Operating System (ROS) (Quigley, Conley, Gerkey, Faust, Foote, Leibs, Wheeler, & Ng, 2009). The M^{3}VR method is integrated as a ROS service enabling its use as an automatic routine. Gaussian kernels, which appeared the best for this type of the problem, are used as the feature functions in M^{3}VR with equal parameters for all domains and experiments. The confidence value for completing the graph with predictions was set to $c_t = 0$ in all experiments.

With this framework we were able to generate randomised problem instances from eight domains: *tidy-room*, inspired by the problem of cleaning a child's room with an autonomous robot presented by Krivic et al. (2015), *course-advisor*, adapted from Guerin et al. (2012) who model an academic advising tool; *mars-rovers*, a multi-robot version of the navigation problem by Cassandra et al. (1996), in which several robots are gathering samples; and *persistent-auv*, a domain of autonomous underwater vehicles for intervention described by Palomeras et al. (2016), The domains *TPP*, *blocksworld*, *openstacks*, and *satellite* were taken from the International planning Competition (Pommerening, Torralba, & Balyo, 2018). The number of predicates in each domain is given in Table 1. The system can be easily used with other domains as well (Krivic, Cashmore, Magazzeni, Ridder, Szedmak, & Piater, 2018).

Domain	*tidy-room*	*mars-rovers*	*course-advisor*	*persistent-auv*
Number of predicates	6	10	7	11
Domain	*blocksworld*	*openstacks*	*satellite*	*TPP*
Number of predicates	12	3	8	5

Table 1: Number of predicates in each test domain

5.1 Accuracy of Prediction

Experimental Setup To evaluate prediction accuracy, we generated examples of states in each domain varying the size of the problem and the percentage of the knowledge in the state. For the first 4 domains, the problem size is varied by increasing the number of objects from 5 to 100 by an increment of 5. For the second 4 (IPC) domains, the problem size is varied by increasing the number of goals from 1 to 8. In each, the number of goals is proportional to the number of objects. Prediction accuracy was calculated over the estimated planning state in comparison with the ground truth.

The initial knowledge is varied by generating complete states and removing literals at random. After prediction, the estimated state was verified against the original complete state, the ground truth. The literals considered for removal are propositions that describe the properties of objects, and cannot be predicted through the propagation of domain-knowledge axioms. For example, while the position of a rover can be derived directly from the knowledge of where it is not, the "can be picked up" property of a cup cannot be derived and must be sensed or predicted. Therefore the partially known states produced can be sensibly represented as states with incomplete information, as defined in Section 2.2.

The known data in the state is the training data for prediction. The percentages of knowledge used as known data of the state in tests are 0.5%, 1%, 2%, 3%, 5%, 8%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, and 80%. To examine the reproducibility of prediction problems, we randomly generated 10 states for each combination of the percentage of knowledge and problem size. Thus, for each domain were generated $20 \times 14 \times 10$ problems in total. The prediction was applied to every problem independently, and results were compared to the ground truth. Accuracy was calculated over the set of predicted unknown literals with respect to the ground truth.

Results The prediction results for the first four domains are shown in Figure 10. The results represent a mean over the 10 problem instances. Brighter colors indicate high accuracy. To examine the lower limit on problem size, we extracted the amount of the known data that is needed to achieve accuracy higher than 90%. This is shown in Figure 11. The number of learned relations is large for each domain. With 20% of the known predicates and with 20 objects, accuracy is higher than 90% for all domains except *mars-rovers*. For a better overview of the results, we illustrate mean values together with the variance of accuracy, precision and recall in the case of 20 and 40 objects in Figure 12. These line charts represent slices of the 3-D plots given in Figure 10. Precision and recall are calculated with respect to predicted positive literals.[2]

2. Precision is calculated over the predicted set of unknown literals as *number of true positive literals/(number of true positive literals + number of false positive literals).*

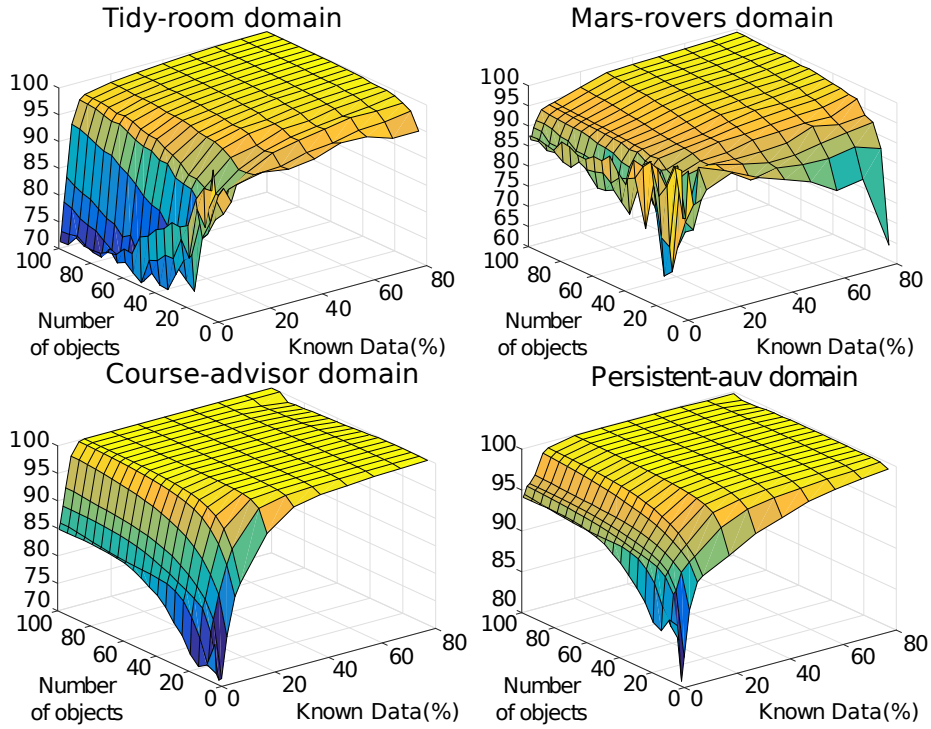


Figure 10: Accuracy results obtained by varying the number of objects and the known data.

The prediction results for the second 4 (IPC) domains are shown in Figure 13. The figure contains a table with information about the number of objects corresponding to the number of goals.

Accuracy increases with the size of the problems, as can be seen in Figures 10 and 13. With 40 objects in each problem domain, only 8% of known data is enough for accuracy over 90% except for *satellite* domain where 20% of known data is necessary. *Blocksworld* and *persistent-auv* domains contain more predicates compared with the other domains. This results in denser multigraphs. Thus, for these domains, the accuracy is better for smaller numbers of objects as well. The high accuracy of predictions in these domains indicates domains with structure and order. Compared to other domains, the *rovers* and *satellite* domains appear to be the least ordered. With a small number of objects and initial knowledge, it is harder to capture the structure of the domain. Thus 60% initial knowledge is required to achieve 90% accuracy in the case of 15 objects for *rovers* domain.

5.2 Evaluating Robustness

In further experiments, we investigate whether the high accuracy results in robust prediction and plans.

Recall is calculated over the predicted set of unknown literals as
number of true positive literals/(number of true positive literals + number of false negative literals).

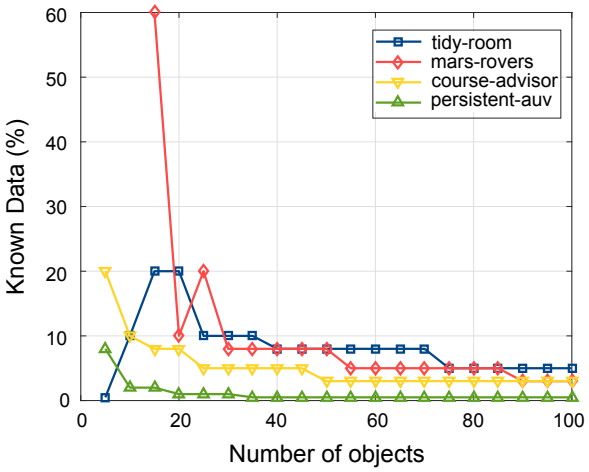


Figure 11: Minimal percentage of known data which gives stable accuracy equal to or higher than 90% for different problem sizes.

Experimental Setup We make the problems deterministic in three ways: (a) by accepting all predictions (i.e., lowering the confidence threshold to 0), (b) by an optimistic baseline of assuming unknown propositions to be true, and (c) by a pessimistic baseline of assuming unknown propositions to be false. This corresponds to solving the problem according to the closed-world assumption of the PDDL semantics. This was repeated 10 times for each different percentage of initial knowledge. The problems were then solved using the planner POPF (Coles et al., 2010). The resulting plans were validated against the ground truth using VAL (Howey, Long, & Fox, 2004).

Results The results are shown in Figure 14 showing that even for 30% of known data, the prediction approach (a) leads to robust plans. Using the pessimistic baseline (c), without prediction, none of the problems could be solved using a deterministic planner. Even with the knowledge of 80% of the state, the goal was not in the reachable state-space. On the other hand, using the optimistic baseline (b), many plans were generated, but only plans in the *tidy-room* domain were found to be valid. At 40% of initial knowledge, the planner produced 6 valid plans out of 10 problems. This comparison with the optimistic approach demonstrates that higher accuracy in the prediction approach does provide a benefit in terms of robustness. Note that the prediction approach does not provide a guarantee that all plans will be valid, and as shown in Figure 14, produces invalid plans as a result of incorrect predictions when the initial known data is low.

5.3 Impact of Predictions on the State

Experimental Setup To provide an illustrative analysis of the impact of predictions on a state, we generate a relaxed plan-graph (RPG) and count the number of reachable actions for an agent before prediction (where unknown propositions are assumed to be false) and

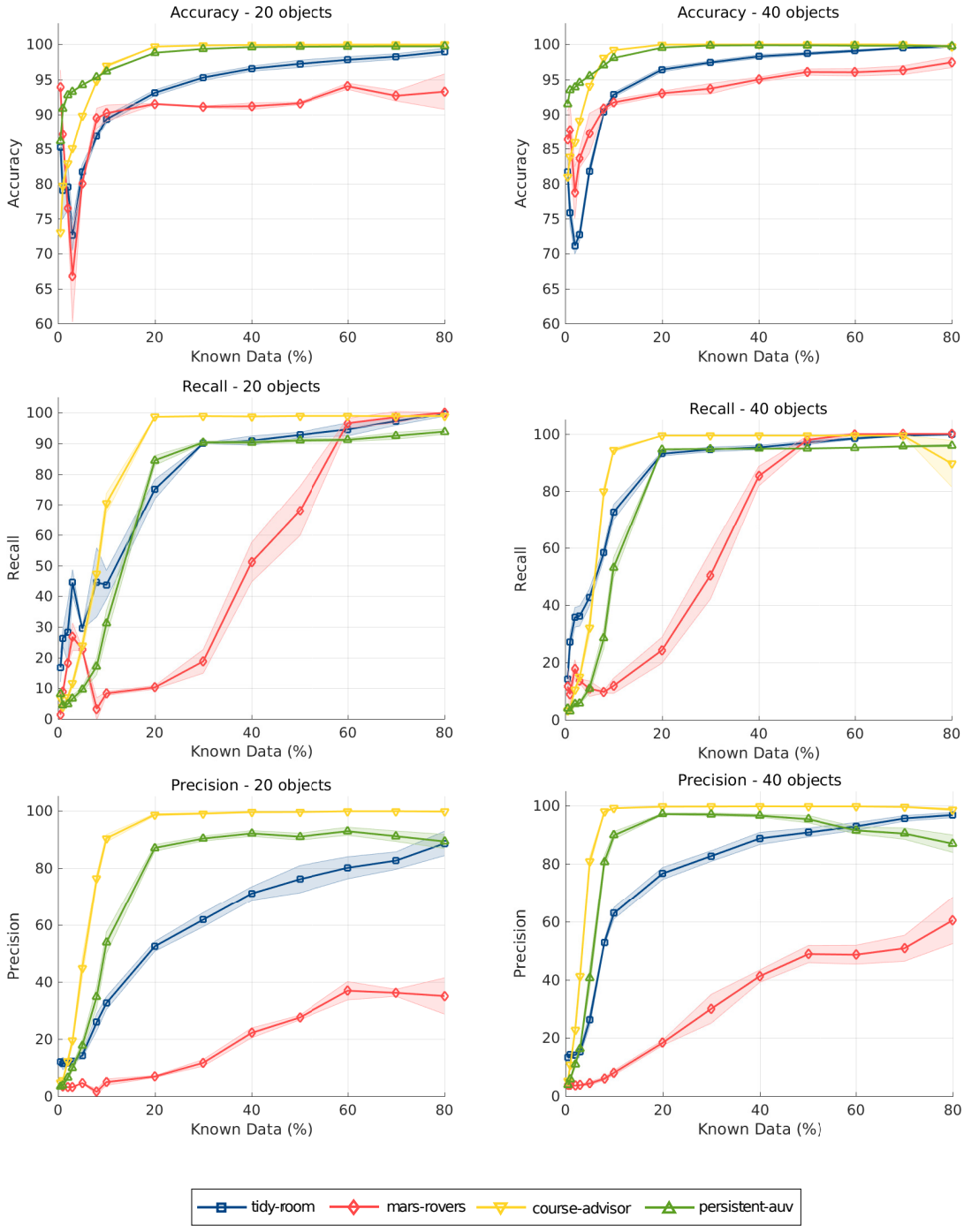
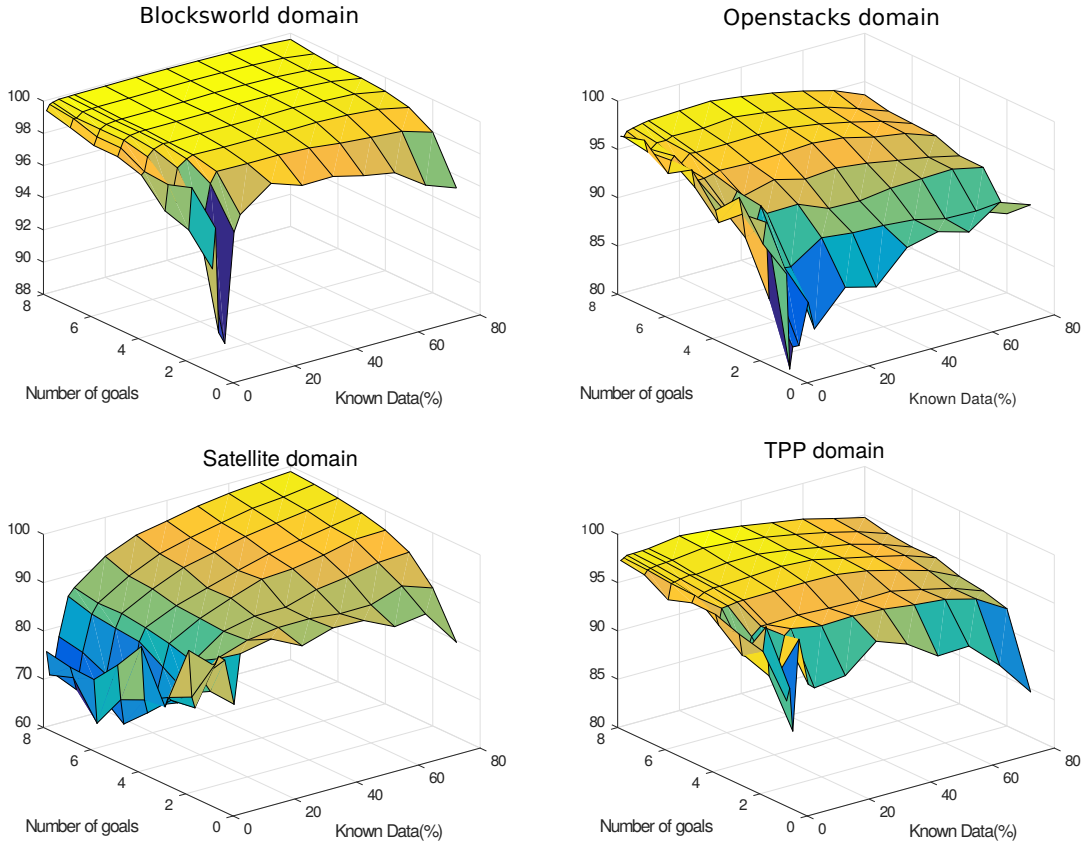


Figure 12: Comparison of accuracy, precision and recall with standard deviation in the case of 20 and 40 objects. Each of the six graphs represents one slice through each of the four graphs of Figure 10, plus standard deviations.

after prediction (using the resulting complete state). This is done for the first four test domains.

Results Figure 15 shows the number of new reachable actions as a percentage of the total number of reachable actions in the ground truth. The increase in reachable actions is very large, especially with a smaller amount of known data. This increase in the number of valid actions enabled by the prediction demonstrates an increase in the size of reachable state space. This is an indication that the prediction does have the potential to improve the performance, especially in deterministic planning, by making reachable a goal that could



	Number of objects in domains							
Number of goals	1	2	3	4	5	6	7	8
blocksbworld	15	25	36	47	58	69	80	91
openstacks	16	26	36	46	56	66	76	86
satellite	12	17	22	27	32	37	42	47
TPP	16	32	48	64	80	96	112	128

Figure 13: Accuracy results obtained by varying the number of goals and the known data.

not be achieved prior to prediction. Note that since the RPG is an over-approximation of the reachable state-space, Figure 15 only shows the potential for improving the planner's performance and no guarantee that a previously unsolvable problem can then be solved after prediction. Instead, what we show is that prediction can bring the potentially reachable state-space from a partially-known state closer to the state-space that would be reachable if the ground truth was known.

5.4 Contingency Planning Effort

We also set up experiments to examine the planning effort in terms of complexity and time needed to complete planning with incomplete information both with and without predictions.

Experimental Setup In this set of experiments, sensing actions were introduced into the *tidy-room* domain, allowing the agent(s) to determine the ground truth of unknown propositions. As described in Section 5.1, the problems for this domain were constructed in such a way that they can be sensibly represented as initial states with incomplete information and solved using a contingent planner. Problems were generated with 10% to 80% of known information in the initial state. All of the problems involve 20 objects. For each percentage of known data, we randomly created 10 instances of problems, and results represent the mean values. The number of goals was varied from 2 to 6. We used the

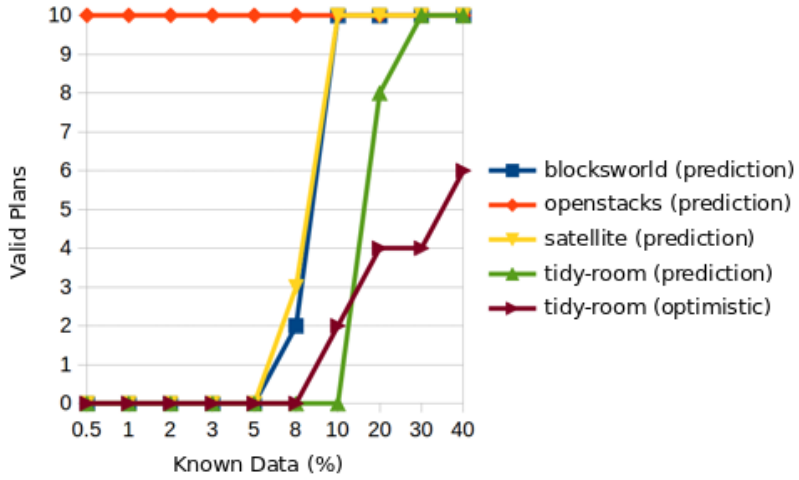


Figure 14: The number of problems for which valid plans were generated, for varying amounts of knowledge. Experiments were done for the problems with 6 goals (*blocksworld, openstacks, satellite*), or 70 objects (*tidy-room*). With the pessimistic baseline, none of the problems was solved. Plans produced with optimistic baseline were invalid except for *tidy-room* domain.

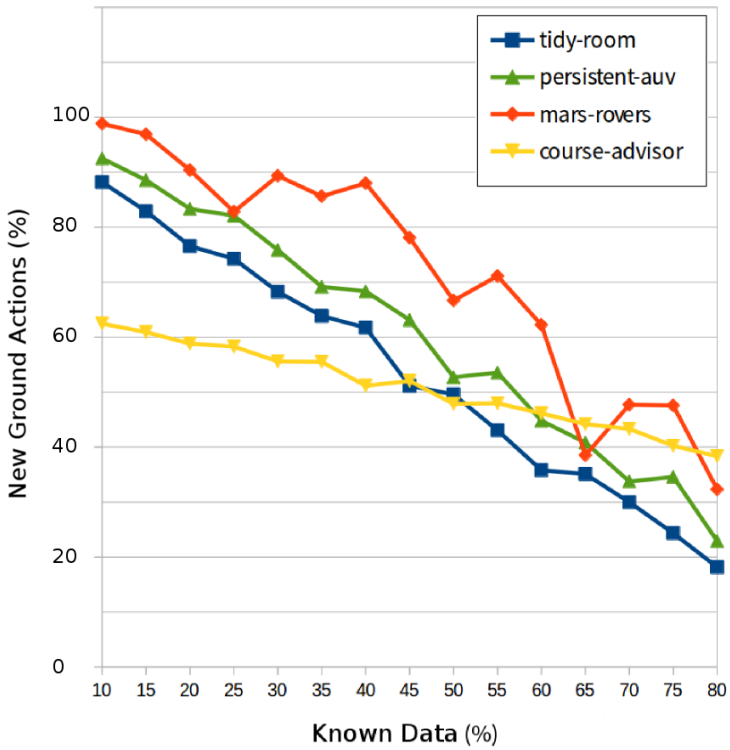


Figure 15: The number of newly reachable actions after prediction, as a percentage of actions in the ground truth. Tests were done for problems with 20 objects and varying amounts of knowledge in all four domains.

planner CLG (Albore & Geffner, 2009) (offline mode) to solve problems in these extended domains, producing branching contingent plans. A time limit of 1800 [s] was given to planning (including any prediction). Experiments were done with and without predictions. The prediction was applied before a single planning attempt.

Results We recorded the time taken to generate a contingent plan. The solution times are shown in Figure 16. For problems with 5 goals and 6 goals, the problem could not be solved by CLG without predictions within the time limit. With prediction these problems were solved with an average of 89 [s]. These results illustrate the benefit of prediction in enhancing the scalability of contingent planning.

5.5 Domain Exploration with Active Learning

In this experiment we examine the impact of active selection of sensing actions on prediction accuracy.

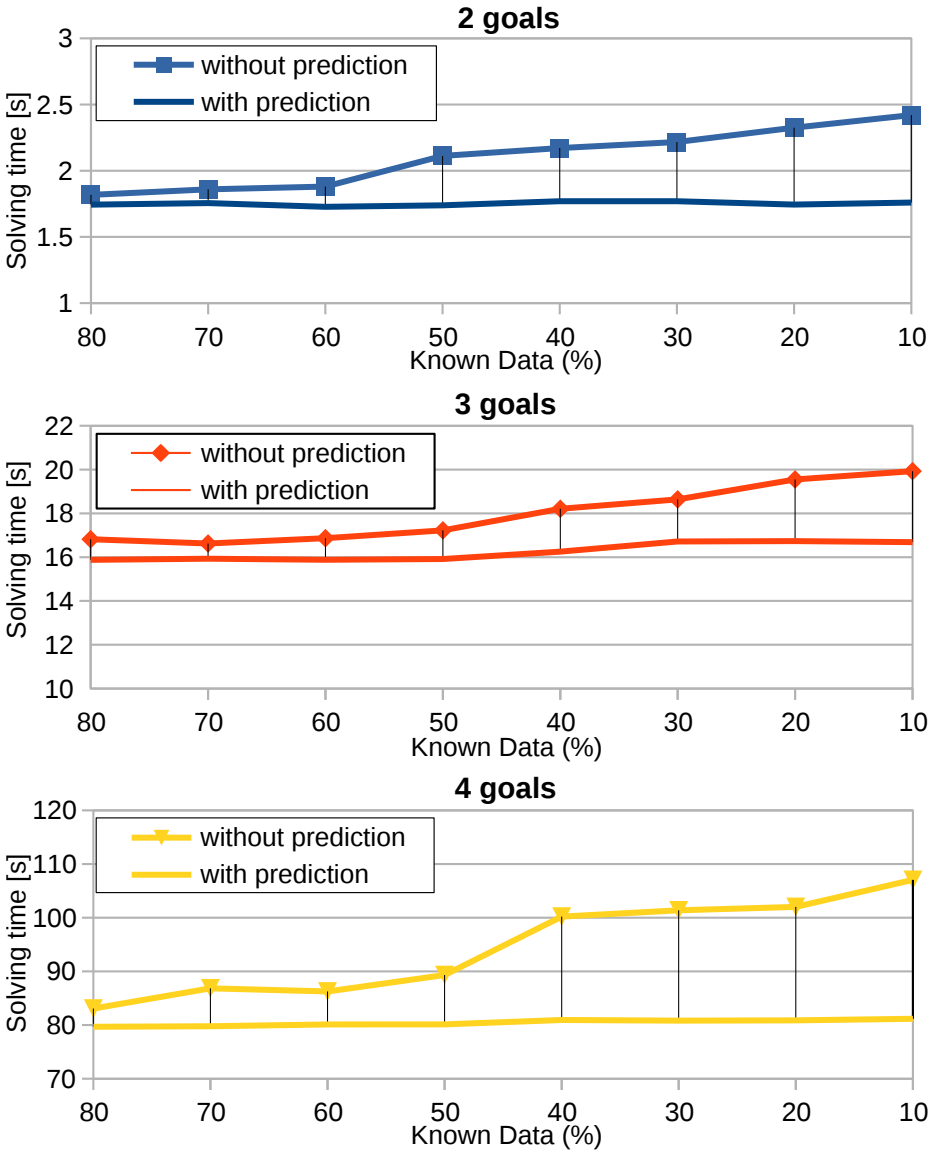


Figure 16: The time taken by CLG to solve problems with 2, 3, and 4 goals, without prediction (upper lines) and with prediction (lower lines). The values also include prediction time.

Experimental Setup To test the proposed ϵ-greedy exploration of the domain, we compare it with passive learning, where new information added to the state is randomly selected, and with the pure greedy strategy, where exploration component is set to 0. In our experiments, an autonomous agent starts planning with 5% of known information about the initial state. We examine the behaviour of the prediction accuracy by adding 1000 missing literals

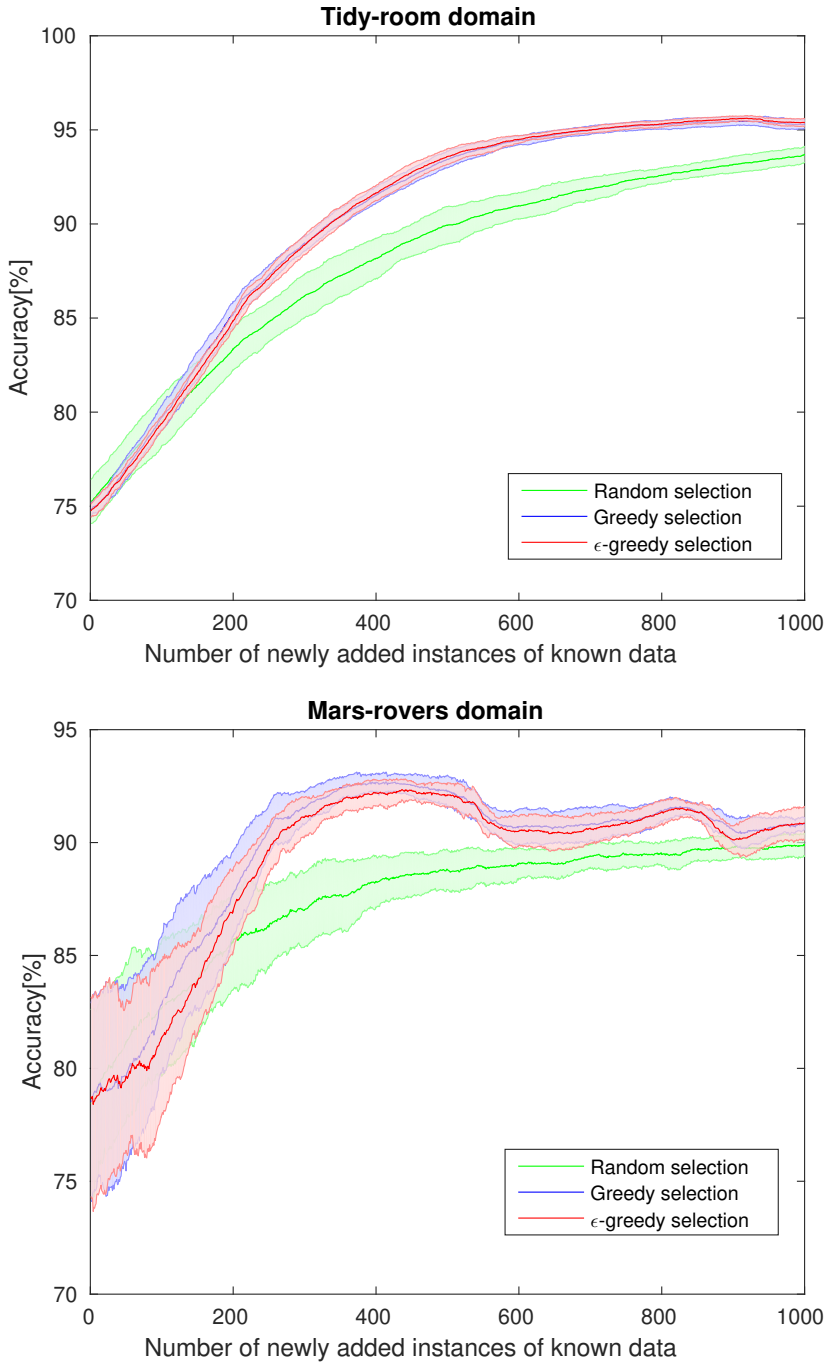


Figure 17: Prediction accuracy following ϵ-greedy active learning compared with random selections of newly added data and pure greedy strategy for *tidy-room* and *mars-rovers* domains. Points represent the mean accuracy of ten experiments, while the shaded regions represent corresponding variances.

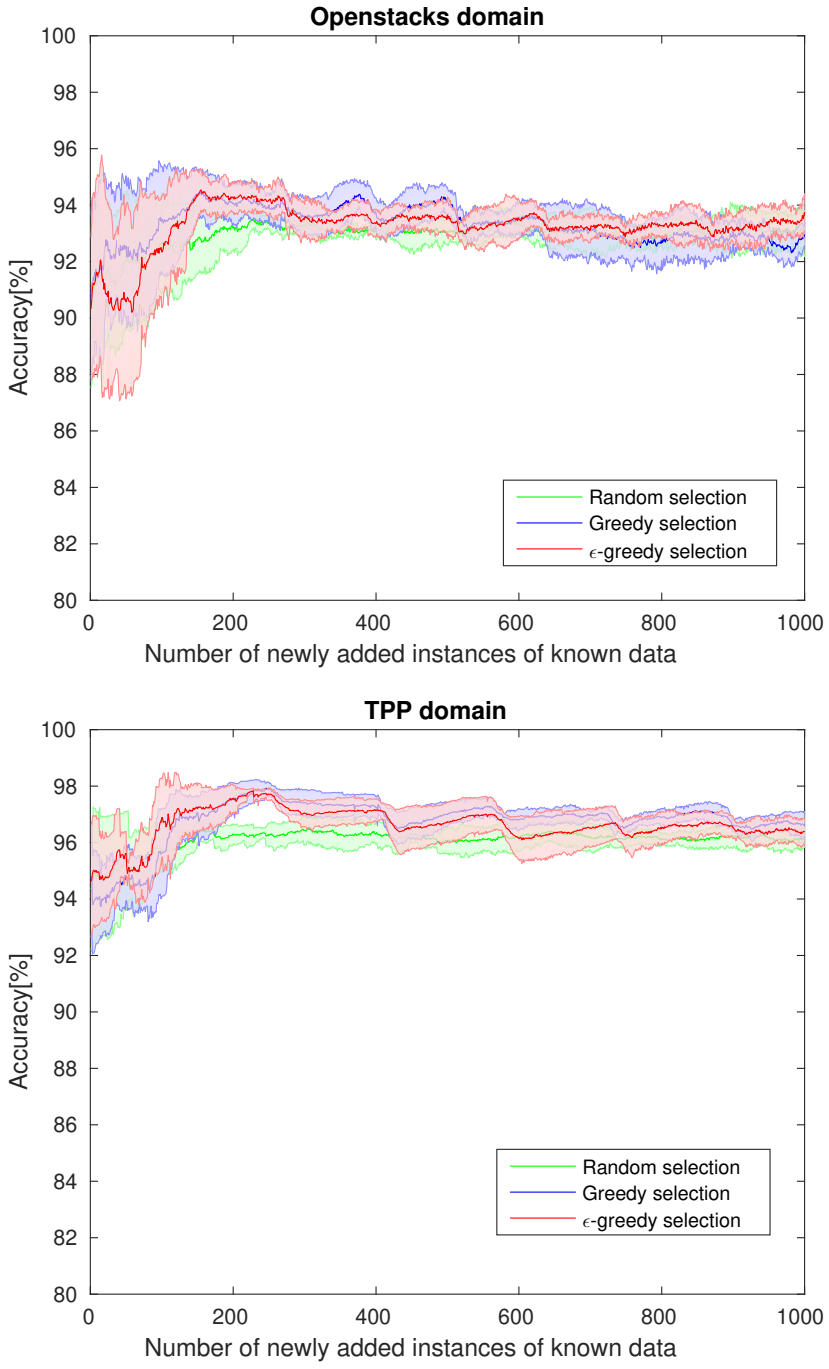


Figure 18: Prediction accuracy following ϵ-greedy active learning compared with random selections of newly added data and pure greedy strategy for *Openstacks* and *TPP* domains. Points represent the mean accuracy of ten experiments, while the shaded regions represent corresponding variances.

one by one, chosen randomly and with active exploration. Prediction is made after each new instance of information added. We repeat this procedure 10 times with different initial states. We performed tests with *tidy-room*, *mars-rovers* with 30 objects, *openstacks* with 3 goals, and *TPP* with 2 goals since they appear to have a more complex structure, as shown in prediction tests (Section 5.1). The number of literals added (1000) represents 12.76% of the complete state in the *mars-rovers* domain, 7.23% in the *tidy-room* domain, 25.72% in *openstacks* domain and 19.5312% in *TPP* domain. The value of $\epsilon = 0.05$ is set for ϵ-greedy exploration for all domains.

Results The mean and variance of learning curves are shown in Figures 17 and 18. In all domains, the accuracy of the predictions is much higher using the active learning approach compared to random selection. This demonstrates that the approach is an effective means of directing the planning agent in its exploration. With sensing actions selected by the active learner, the agent is able to have higher accuracy with less known data. With only 7.82% of known data in the *tidy-room* domain and 7.67% in the *mars-rovers* domain, the accuracy of prediction is higher than 90%. In addition, the low variance shows that the agent is consistent in selecting informative data for improving the prediction accuracy. The impact of the exploration component ϵ is not high since the domains are structured. In the *tidy-room* domain, the variance is slightly smaller in the case of ϵ-greedy domain. The accuracy is higher with the ϵ-greedy strategy in the *openstacks* domain, while in *mars-rovers* and *TPP* domains, the pure greedy strategy has slightly better accuracy results.

6. Discussions

In this section, we discuss contributions, advantages, and disadvantages of the proposed approach, as well as related work.

6.1 General Discussion

Decreasing Uncertainty in the State using Predictions. Predictions can be combined with the planning in the following way. The prediction can be applied with a high confidence threshold, removing some of the uncertainty in the state. Contingent planning (or conformant planning, etc.) can be used to deal with the remaining uncertainty by executing sensing actions to observe remaining unknown propositions. This alleviates the high complexity of contingent planning while reducing errors from inaccurate prediction with lower confidences.

Choice of Confidence Threshold. Confidence values range between 0 and 1. In some domains, e.g. the *course-advisor* domain, all predictions can be added to the state since a wrong prediction will not harm the agent or execution. In such domains, the confidence threshold can be ignored, i.e. $c_t = 0$, and the resulting complete state s is used for planning. However, for many other domains, the confidence threshold plays an important role, e.g. in robotic domains where an incorrect prediction can lead the robot into a state from which it is very expensive to recover. In such cases, it is better to set a positive confidence threshold $c_t > 0$. The confidence threshold can be set as a firm value manually, or it can be obtained via some automatic procedure that depends on the properties of the specific domain. The confidence itself depends on the available data of the state, so the percentage of known data

can be used as one of the parameters for the threshold choice. Strategies for the selection of the confidence threshold with respect to the domain and known data are part of the future work.

Predicates with Categorical Values. It is possible to perform predictions on planning domains with integer variables if they are bounded. These bounded integers can be represented as a categorical value (in which each integer value is a category). To enable prediction in these domains we assign a separate edge to every category for each predicate. The incomplete state s' in such domains is described with a partially-known multigraph:

$$\begin{aligned} V &\equiv O \\ E' &= \{e_{pred_c}(b,u) | (b,u) \in V \times V, \forall c \in C_{pred}\} \end{aligned}$$

where C_{pred} is a set of possible categories of the predicate *pred*. This represents a limited set of planning domains with numeric variables. Extending predictions to unbounded numeric variables and real variables is left for future work.

The Effect of Incorrect Predictions. Introducing the prediction step into the planning routine may also introduce incorrect facts into the state, even if the prediction accuracy is high. This can result in plans impossible to realise, and the agent can fail in executing a task. However, as seen from the experimental results, if there are uncertainties in the state, the performance of planning is already low, and introducing predictions improves it significantly. Also, when planning in real-world dynamic and changeable domains, there is always a risk of inconsistency between the state and the real world. Taking confidences into account when introducing new information into the state reduces the possibility of obtaining invalid plans, but does not eliminate it.

Applicability. The impact of incorrect predictions also varies based on the domain and the task. The importance of incorrect predictions is not the same in different domains. In hazardous and sensitive domains such as power plant management it can be very dangerous. On the other hand, in domains such as a robotic household, failure in executing a plan with an incorrect prediction could have no lasting consequence and allow the robot to refine its knowledge for another attempt.

Proposition Selection. In the approach in this article, guiding an agent to acquire new information is based on the prediction confidence. In standard active learning settings, there is an assumption that getting a label for unlabelled instances comes without a cost. In planning domains, this is not the case. Different sensing actions come with an associated cost of execution. Also, as previously noted, sometimes information cannot be sensed or explored by the agent. In the presented approach for selecting information to be sensed in order to improve predictions, sensing action execution costs are not taken into account when selecting which information has to be acquired. However as future work in active learning, sensing costs can be taken into account in two ways. Firstly, a cost function derived from the planning domain can be taken into account by the active learner when selecting which new information should be obtained. Secondly, the active learner can choose sets of propositions which can improve predictions, then the sensing actions which provide the desired information with minimal cost can be chosen by the agent. Finally, these options can

be combined, so that the active module takes costs into account when providing suggestions, and the planner also uses those costs during planning. We will explore this in future work.

Greedy Task Solving vs. State Exploration. Two different strategies for actively sensing information are presented. These strategies can be used in different planning scenarios and for different planning goals and metrics. If the purpose of the planning system is to achieve the goal as rapidly as possible, then the Greedy Task Solving strategy should be used. On the other hand, if the purpose of the agent is exploration, or there is a trade-off between completing the current task and exploration to aid future tasks, the State Exploration strategy can be used.

6.2 Related Work

We discuss the proposed approach with respect to the related work in this subsection.

6.2.1 Planning Under Uncertainty

Planning under uncertainty has been addressed in a wide variety of forms (Boutilier et al., 1996; Blythe, 1999; Jimnez et al., 2012). Blythe (1999) provides an overview of planning under uncertainty. There are different sources of uncertainty in the planning process, such as uncertainty in the initial state, uncertainty caused by non-deterministic effects of action models or the uncertainty caused by noise in sensing actions.

In this article, we focused on the first problem, which is reflected in missing information in the initial state by introducing a new paradigm in AI planning field based on ML. This problem can also be addressed within standard planning approaches at the expense of time needed for planning, such as contingency planning (Bonet & Geffner, 2000; Hoffmann & Brafman, 2005), conformant planning (Smith & Weld, 1998; Palacios & Geffner, 2006), or replanning techniques (Brafman & Shani, 2012). Conformant planners take the uncertainty of the system into account by observing a set of possible initial states $s_0 \in S$ instead of a single initial state. These planners work under the assumption of null run-time observability, meaning that sensing actions cannot be a part of the domain. Contingent planners address the problem of planning in domains with incomplete information in the state and where the current state can be observed with sensing actions. If the lack of information is severe, the problem can become unsolvable by these techniques. Our proposed approach can be combined with these techniques for planning under uncertainty, using state prediction first to reduce the uncertainty and complete information about the state.

The problem of uncertainty in planning due to non-deterministic effects of actions is commonly addressed with probabilistic planners. Mausam and Weld (2006) address the problem of probabilistic temporal planning with uncertain durations. Pineda and Zilberstein (2014) introduce a family of MDP reduced models which improve the performance of planning. Celorrio et al. (2008) show that machine learning algorithms can efficiently assist automated planning to build robust plans in stochastic environments by upgrading the probabilistic action model with knowledge learned from execution. Problems with partial observability due to noisy or lack of sensing actions can be formulated as partially observable Markov decision processes (POMDPs) (Blythe, 1999; Kaelbling et al., 1998; Ong et al., 2010). In this manner Ong et al. (2010) use a factored model to represent separately the

Planning problem	State	Action Models	Sensing actions
Classical planning	complete	deterministic	-
Conformant planning	incomplete	deterministic and stochastic	no
Contingent planning	incomplete	deterministic	yes
MDP	complete	stochastic	yes
POMDP	incomplete	stochastic	yes

Table 2: Comparison of planning problems with respect to uncertainty in state and action models as well as applicability of sensing actions.

fully and partially observable components of a robots state and derive a compact lower-dimensional representation of its belief space. This factored representation allows them to compute approximate POMDP solutions. Generally, planning with POMDPs has a high computational complexity, and therefore the size of the state space compared to classical planners is limited.

We give an overview of planning problems and properties of the planning model in Table 2. The approach that we describe in this paper addresses the problem of incomplete information in a preprocessing step prior to planning. The predictions from this step do not necessarily result in a complete state. As such, it can be used in combination with any of the techniques for planning with incomplete information discussed above.

6.2.2 Machine Learning and Automated Planning

Zimmerman and Kambhampati (2003) and Jimnez et al. (2012) provide extensive coverages of approaches that combine machine learning techniques and automated planning: learning to speed up planning, improving the plan quality, learning and improving domain knowledge, and using reinforcement learning as to learn policies for selecting a suitable action which should be executed in the current state as alternative to standard planning procedures. Our work follows the trends of speeding up the planning process as well as improving the domain knowledge. However, there is not much work on learning-based approaches for planning under uncertainty. Yang et al. (2007) developed an action-relation modelling system for learning action models from a set of successful observed plans. Amir and Chang (2008) also address the problem of uncertainty in action models by learning outcomes of deterministic actions from partial observations over time.

There are some architectures which use standard planning languages such as PDDL with learning techniques. Botea et al. (2005) propose an adaptive technique that learns macro-operators from previous experience. This exploits the domain structure to solve new problems more quickly. Similar to our method, they also exploit underlying structure of the domain. Coles and Smith (2007) propose Marvin, a planner that learns macro-actions. de la Rosa et al. (2011) use decision trees to guide the search of a heuristic planner. Aineto et al. (2018) propose an approach for learning STRIPS action models from examples that compile this inductive learning task into a classical planning task. Celorrio et al. (2008) integrate a relational tree learner for learning probabilistic action models from experience

with classical and decision-making planners. Similarly to ours, their architecture offers the possibility of applying different machine learning and planning techniques.

There are fewer machine learning methods used to acquire knowledge for classical planning. Fuentetaja and Borrajo (2006) present active learning schemas to generate new training problems for machine learning techniques in the context of problem-solving. The proposed active learning methods are independent of the planner and the used machine learning technique, as in our approach.

6.2.3 Sensing Action Selection in Contingent Planning

There are a number of approaches to planning with incomplete information and sensing actions. For example, Weld et al. (1998) extend Graphplan with an explicit model of sensing actions and introducing a *conditioning* threat-resolution method into the backward-chaining solution extraction. The solution extracted by their planner SGP is a contingent plan. In another approach, Albore and Geffner (2009), based on the conformant planning approach of Palacios and Geffner (2007), translate the non-deterministic search problem in belief space into a non-deterministic problem in state space whose literals represent beliefs. Then, planners perform a search in this state space, selecting sensing actions necessary for achieving the goal of the problem. In contrast, Bonet and Geffner (2000) and Hoffmann and Brafman (2005) both perform forward-search in the belief space. As a final step of the heuristic used by Hoffmann and Brafman (2005), observation goals are produced. These represent an approximation of the observations that are required to achieve the goal.

In all of these approaches, the planner is used to select observation actions that are necessary to achieve the goal of the problem. This is related to our strategy for the selection of sensing actions. A crucial difference is that the active-learning approach can select observation goals that are not necessary for achieving the goal, but instead for improving the prediction of the observation that is necessary. This can be used to address expensive, risky, or impossible observations. The active learning approach we propose creates observation goals that the planner is required to achieve. This is external to the planning process. In addition to this, the observation goals can have other purposes, unrelated to the immediate goal of the problem: for general exploration or to aid the prediction step (as described above). As such, the selection of these observation goals cannot be made by the planner.

7. Conclusions

In this paper, we have shown how state uncertainty in planning can be decreased with predictions using machine learning. To enable predictions in the state, we introduced an approach in which a state with uncertainty is represented as a partially-known multigraph. We demonstrated how M^3VR is used to predict edges and how these edges are translated into the state as predicted propositions.

With tests in different domains, we demonstrated that with 20% knowledge about the state, the accuracy of the state prediction is higher than 90%. We also showed that the accuracy of the predictions leads to robust plans increasing the scalability of planners.

The proposed system uses the confidence measure of predictions to select which predicted information can be introduced in the state and which should be verified with sensing actions.

We demonstrated how the proposed approach could be used to inform a planner in directing the agent to perform sensing actions. Selecting the right information to improve predictions significantly reduces overall sensing costs and time, avoiding redundant and unnecessary sensing actions. The importance of the proposition selection step is also reflected in increasing the confidence of predictions for information which the agent cannot sense.

In future work, we plan to explore the similarities between different domain models and how knowledge can be transferred between different domains. Also, we will examine how the active learning process can take into account the cost of sensing actions.

Appendix A. Multigraph Completion with M^{3}VR

In our system, we use the Maximum Margin Multi-Valued Regression (M^{3}VR), also used at the core of a recommender system (Ghazanfar et al., 2012), for an affordance learning problem (Szedmak et al., 2014), and to refine spatial relations for planning to tidy up a child's room (Krivic et al., 2015). These results show that it can deal with sparse, incomplete and noisy information, that predictions are fast despite the algorithm reacquires retraining from scratch each time new information is obtained. We build on the previous work, describing how the approach can be generalised and used for planning.

The main idea of M^{3}VR is to capture this hidden structure in the graph with a model and use it to predict unknown edges. This model is represented by a collection of predictor functions that learn mappings between edges and vertices.

Using M^{3}VR we construct a function which captures knowledge about existing edges. This is done by assigning a predictor function $f_b(u) : U \rightarrow E$ to each origin vertex $b \in B$ that maps all destination vertices to corresponding edges. Thus the number of predictor functions is equal to the number of vertices B.

These predictor functions have to capture the underlying structure of a graph, which can be very complex and non-linear. Thus representing that accurately in a low dimensional Euclidean space has strong limitations. We assume that there is a linear relationship in feature space between the feature vector of all destination vertices and the feature vector of all connected edges. A feature space or a Hilbert space is a vector space $\mathcal{H}$ defined over the field of the real or complex numbers equipped with an inner product $\langle \cdot, \cdot \rangle$ and norm defined by $\|x\| = \sqrt{\langle x, x \rangle}$ which turn $\mathcal{H}$ into a complete metric space. The functions mapping vertices and edges into feature spaces are defined on feature representations of vertices and edges with following conditions.

Condition 1 *There exists a mapping ϕ of all subsets of $B = U$ into a Hilbert space H_ϕ, equipped by a kernel function $\kappa_{vertice}$ defined for all possible pairs B' and U' of all subsets of $B \times U$ such that $\kappa_{vertice}(B', U') = \langle \phi(B'), \phi(U') \rangle$.*

Condition 2 *There exists a mapping ψ of E into a Hilbert space H_ψ equipped by a kernel function κ_{edge} defined for all pairs $(\boldsymbol{e}_i, \boldsymbol{e}_j) \in E \times E$, such that $\kappa_{edge}(\boldsymbol{e}_i, \boldsymbol{e}_j) = \langle \psi(\boldsymbol{e}_i), \psi(\boldsymbol{e}_j) \rangle$.*

Kernel functions are symmetric functions which return the inner product between feature vectors of the data in some feature space. H_ϕ and H_ψ are feature representations of domains of U and E. The vectors $\phi(\cdot)$ and $\psi(\cdot)$ are called *feature vectors*. $\langle ., . \rangle$ is the inner product and it can be used as a measure of similarity between feature vectors.

This allows us to assign prediction functions for each origin vertex b defined on feature vectors, i.e., $F_b : H_\phi \to H_\psi$. The relationship in feature space between the feature vector of all destination vertices H_ϕ and the feature vector of all connected edges H_ψ is represented by linear operator $\mathbf{W}_b$. The non-linearity of the mapping f_b can be expressed by choosing non-linear functions ψ and ϕ. The predictor functions F_b can be created as

$$\mathbf{W}_b\phi(u) \to \psi(f_b(u)) \sim \psi(\mathbf{e}_{bu}), (b,u) \in B \cap U. \tag{10}$$

The similarity between the vectors $\mathbf{W}_b\phi(u)$ and $\psi(\mathbf{e}_{bu})$ is described by the inner product $\langle \psi(\mathbf{e}_{bu}), \mathbf{W}_b\phi(u) \rangle_{H_\psi}$. If the similarity between $\mathbf{W}_b\phi(u)$ and $\psi(\mathbf{e}_{bu})$ is higher, the inner product will have a greater value.

To measure the error of predictor functions f_b we use the following maximum margin Hinge loss function:

$$L_b(\mathbf{e}_{bu}, F_b(u)|B) = \max(0, 1 - \langle \psi(\mathbf{e}_{bu}), \mathbf{W}_b\phi(U) \rangle). \tag{11}$$

To exploit the knowledge about existing edges linking the same destination vertices $u \in U$, predictor functions F_b for origin vertices $b \in B$ are coupled by shared slack variables $\boldsymbol{\xi}_u$ representing the loss to be minimized by the predictor function F_b. While training prediction functions F_b , the value of slack variables $\boldsymbol{\xi}_u$ has to remain the same. With this procedure, knowledge about existing edges is exploited to train prediction functions F_b (Figure 19).

The training is done via an optimisation process where Hinge loss L_b is minimised. To write down the entire optimisation problem realizing the learning task we need some additional notation. Let the projection of known edges E' into U be given by

$$U_b = \{u | u \in U, \mathbf{e}_{bu} \in E'\}, \forall b \in B.$$

Linear mappings $\mathbf{W}_b$ are determined by optimization routine:

$$\begin{array}{ll} \min & \frac{1}{2}\sum_{b=1}^{N_b} \|\mathbf{W}_b\|^2_{Frobenius} + C\sum_{u=1}^{N_u} \boldsymbol{\xi}_u \\ \text{w.r.t.} & \{\mathbf{W}_b\} : H_\phi \to H_\psi, \text{ linear operators} \\ & \boldsymbol{\xi} \in \mathbb{R}^{N_u}, \text{ error vector} \\ \text{s.t.} & \langle \boldsymbol{\psi}(\mathbf{e}_{bu}), \mathbf{W}_b\boldsymbol{\phi}(u) \rangle_{H_\psi} \geq \mathbf{1} - \boldsymbol{\xi}_u, b \in B, u \in U_b \\ & \boldsymbol{\xi}_u \geq \mathbf{0}, u \in U \end{array} \tag{12}$$

Here, C is a constant, $N_u = N_b$ is the number of vertices and ξ_u is the error vector for all incoming edges for the same destination vertex u.

Once determined, the linear mappings $\mathbf{W}_b$ allow us to make predictions of unknown edges for elements b. In the optimization procedure for determining linear mappings there are as many constraints as the number of known edges in E'. Therefore the complexity of the prediction problem is proportional to $O(|E'|)$, where $|E'|$ stands for the cardinality of the set E'.

A.1 Predictions

The value of the inner product of the edge feature vector $\psi(\mathbf{e}_{bu})$ and learned $\mathbf{W}_b\phi(u)$ should be interpreted as a measure of the edge belonging to a specific class (in this case 0 or 1). Thus, for each prediction $L^*_{pred}(b,u)$, we update the existence of the directed edges:

$$L_{pred}(b,u) = \underset{k \in \{0,1\}}{\arg\max} \langle \psi(\mathbf{e}_{bu} | L^*_{pred}(b,u) = k), \mathbf{W}_b\phi(u) \rangle_{H_\psi}$$

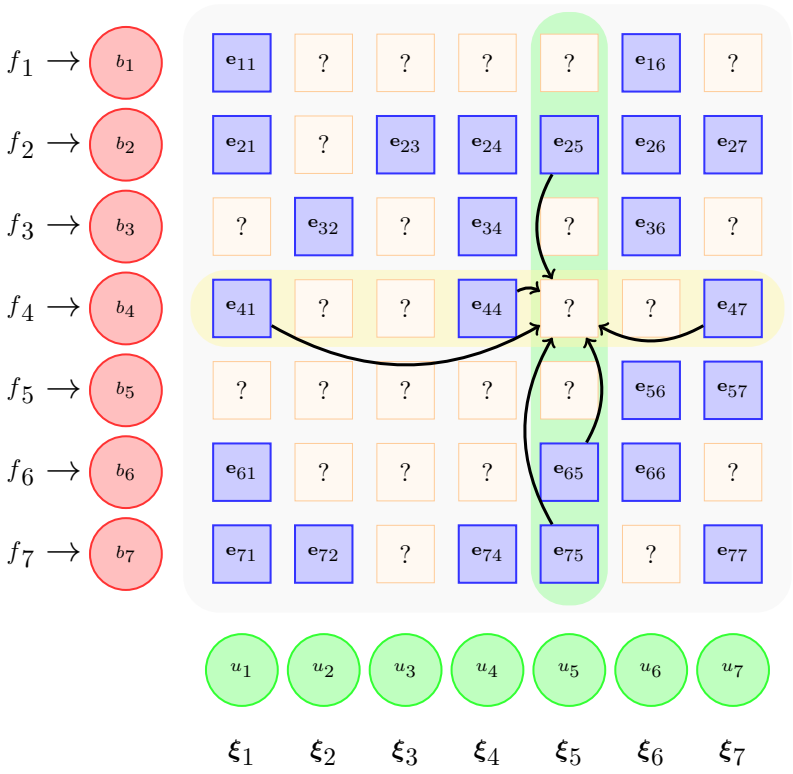


Figure 19: The core mechanism of M^{3}VR to predict an unknown edge based on the available information on the object relation graph. For simplicity, in this example, origin vertices are represented as rows and destination vertices as columns for a single predicate *pred*. The edge which represents proposition $pred(b_4, u_5)$ is missing. To predict the value of $L_{pred}(b_4, u_5)$, those edges that share the same origin or destination vertices as the unknown edge can be exploited.

where $k \in \{0, 1\}$, and $L^*_{pred}(b, u)$ is an unknown value in $\mathbf{e}_{bu} \in E \setminus E'$ of predicate *pred*. In this way predictions are done for all unknown edges and the graph is completed.

A.2 Feature Vector Selection

Kernel functions represent a similarity measure between two objects. They are given by the relation: $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \langle \psi(\mathbf{x}_1), \psi(\mathbf{x}_2) \rangle$ where $\psi(x)$ is a mapping function that embeds a vector $\mathbf{x}$ in the feature space. By replacing inner products of feature representations $\psi(x)$ with kernels, it is possible to obtain flexible representations, which enables the use of the *kernel trick*. We choose Gaussian kernels for $\kappa_{vertice}$ and κ_{edge}. The Gaussian kernel is

defined by

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = e^{\frac{-||\mathbf{x}_1 - \mathbf{x}_2||^2}{2\sigma^2}}$$

where σ determines the width of the Gaussian kernel. With a Gaussian kernel, each point $\mathbf{x}$ is represented by the distance to all other points in the dataset in the feature space.

Gaussian kernels are *universal kernels* since they have the *universal approximating property* regardless of the Gaussian width. This means that they can approximate arbitrary functions uniformly on any compact subset of the input space (Micchelli et al., 2006). Choice of Gaussian kernels results in smooth functions in the feature space.

References

Aineto, D., Jiménez, S., & Onaindia, E. (2018). Learning STRIPS action models with classical planning. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018.*, pp. 399–407.

Albore, A.; Palacios, H., & Geffner, H. (2009). A translation-based approach to contingent planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*.

Amir, E., & Chang, A. (2008). Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, *33*(1), 349–402.

Barfoot, T. D. (2017). *State Estimation for Robotics*. Cambridge University Press.

Bertsimas, D., Pawlowski, C., & Zhuo, Y. D. (2018). From predictive methods to missing data imputation: An optimization approach. *Journal of Machine Learning Research*, *18*(196), 1–39.

Blythe, J. (1999). *An Overview of Planning Under Uncertainty*, chap. An Overview of Planning Under Uncertainty, pp. 85–110. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bonet, B., & Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pp. 52–61.

Botea, A., Enzenberger, M., Müller, M., & Schaeffer, J. (2005). Macro-ff: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, *24*, 581–621.

Boutilier, C., Dean, T., & Hanks, S. (1996). Planning under uncertainty: Structural assumptions and computational leverage. In *In Proceedings of the Second European Workshop on Planning*, pp. 157–171. IOS Press.

Brafman, R., & Hoffmann, J. (2004). Conformant planning via heuristic forward search: A new approach. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pp. 355–364.

Brafman, R. I., & Shani, G. (2012). Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research*, *45*, 565–600.

Bryce, D., Kambhampati, S., & Smith, D. E. (2006). Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, *26*.

Camacho, A., Muise, C., & McIlraith, S. A. (2016). From fond to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *The 26th International Conference on Automated Planning and Scheduling*, pp. 65–69.

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtos, N., & Carreras, M. (2015). Rosplan: Planning in the robot operating system. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*.

Cassandra, A. R., Kaelbling, L. P., & Kurien, J. A. (1996). Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Celorrio, S. J., Fernández, F., & Borrajo, D. (2008). The PELA architecture: Integrating planning and learning to improve execution. In *AAAI*.

Cesa-Bianchi, N., Gentile, C., Vitale, F., & Zappella, G. (2013). Random spanning trees and the prediction of weighted graphs. *Journal of Machine Learning Research*, *14*, 1251–1284.

Chen, Y., Lasko, T. A., Mei, Q., Denny, J. C., & Xu, H. (2015). A study of active learning methods for named entity recognition in clinical text. *Journal of Biomedical Informatics*, *58*, 11–18.

Cohn, D. A., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *Journal Of Artificial Intelligence Research*, *4*(1), 129–145.

Coles, A., & Smith, A. (2007). Marvin: a heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, *28*, 119–156.

Coles, A., Coles, A., Fox, M., & Long, D. (2010). Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pp. 42–49.

de la Rosa, T., Celorrio, S. J., Fuentetaja, R., & Borrajo, D. (2011). Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence research, JAIR, Vol 40(2011)*.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, *20*, 61–124.

Francis, R., Estlin, T., Doran, G., Johnstone, S., Gaines, D., Verma, V., Burl, M., Frydenvang, J., Montao, S., Wiens, R. C., Schaffer, S., Gasnault, O., DeFlores, L., Blaney, D., & Bornstein, B. (2017). Aegis autonomous targeting for chemcam on mars science laboratory: Deployment and results of initial science team use. *Science Robotics*, *2*.

Fuentetaja, R., & Borrajo, D. (2006). Improving control-knowledge acquisition for planning by active learning. In Fürnkranz, J., Scheffer, T., & Spiliopoulou, M. (Eds.), *Machine Learning: ECML 2006*, pp. 138–149. Springer Berlin Heidelberg.

Gentile, C., Herbster, M., & Pasteris, S. (2013). Online similarity prediction of networked data from known and unknown graphs. *Journal of Machine Learning Research*, *30*.

Ghazanfar, M. A., Prügel-Bennett, A., & Szedmak, S. (2012). Kernel-mapping recommender system algorithms. *Information Sciences*, *208*, 81–104.

Guerin, J. T., Hanna, J. P., Ferland, L., Mattei, N., & Goldsmith, J. (2012). The academic advising planning domain. In *Proceedings of the 3rd Workshop on the International Planning Competition at ICAPS*, pp. 1–5.

Hoffmann, J., & Brafman, R. I. (2005). Contingent planning via heuristic forward search witn implicit belief states. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, pp. 71–80.

Howey, R., Long, D., & Fox, M. (2004). VAL: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04)*.

Jahrer, M., Töscher, A., & Legenstein, R. (2010). Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pp. 693–702. ACM.

Jimnez, S., De La Rosa, T., Fernndez, S., Fernndez, F., & Borrajo, D. (2012). A review of machine learning for automated planning. *The Knowledge Engineering Review*, *27*(4), 433467.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1-2), 99–134.

Krivic, S., Cashmore, M., Magazzeni, D., Ridder, B., Szedmak, S., & Piater, J. (2017). Decreasing Uncertainty in Planning with State Prediction. In *International Joint Conference on Artificial Intelligence*, pp. 2032–2038. IJCAI.

Krivic, S., Cashmore, M., Magazzeni, D., Ridder, B., Szedmak, S., & Piater, J. (2018). *State Predictions System together with Domains and Test Scripts*. https://github.com/Senka2112/StatePredictions.

Krivic, S., & Piater, J. (2018). Pushing corridors for delivering unknown objects with a mobile robot. In *Autonomous Robots*, Vol. 43. Springer.

Krivic, S., Szedmak, S., Xiong, H., & Piater, J. (2015). Learning missing edges via kernels in partially-known graphs. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*.

Latouche, P., & Rossi, F. (2015). Graphs in machine learning: an introduction. In *Proceedings of the 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pp. 207–218.

Lewis, D. D., & Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pp. 3–12. Springer-Verlag.

Li, M., & Sethi, I. K. (2006). Confidence-based active learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(8), 1251–1261.

Lin, Z., R.Liu, , & Su, Z. (2011). Linearized alternating direction method with adaptive penalty for low-rank representation. *Advances in Neural Information Processing Systems*, *24*, 612–620.

Mausam, & Weld, D. S. (2006). Probabilistic temporal planning with uncertain durations. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pp. 880–887. AAAI Press.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL – the planning domain definition language. Tech. rep., Yale Center for Computational Vision and Control.

Micchelli, C. A., Xu, Y., & Zhang, H. (2006). Universal kernels. *Journal of Machine Learning Research*, *7*, 2651–2667.

Ong, S. C. W., Png, S. W., Hsu, D., & Lee, W. S. (2010). Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, *29*(8), 1053–1068.

Palacios, H., & Geffner, H. (2006). Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proceedings of the 21st Conference on Artificial Intelligence (AAAI'06)*.

Palacios, H., & Geffner, H. (2007). From conformant into classical planning: Efficient translations that may be complete too. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'07, pp. 264–271. AAAI Press.

Palacios, H., & Geffner, H. (2009). Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, *35*.

Palomeras, N., Carrera, A., Hurts, N., Karras, G. C., Bechlioulis, C. P., Cashmore, M., Magazzeni, D., Long, D., Fox, M., Kyriakopoulos, K. J., Kormushev, P., Salvi, J., & Carreras, M. (2016). Toward persistent autonomous intervention in a subsea panel. *Autonomous Robots*, *40*.

Pineda, L. E., & Zilberstein, S. (2014). Planning under uncertainty using reduced models: Revisiting determinization. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, (ICAPS)*.

Pommerening, F., Torralba, A., & Balyo, T. (2018). *The International Planning Competition*. http://www.icaps-conference.org/index.php/Main/Competition.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: an open-source robot operating system. In *ICRA workshop on open source software*, Vol. 3.

Smith, D. E., & Weld, D. S. (1998). Conformant graphplan. In *Paper presented at the meeting of the AAAI/IAAI (AAAI'98)*, pp. 889–896.

Szedmak, S., Ugur, E., & Piater, J. (2014). Knowledge propagation and relation learning for predicting action effects. In *Intelligent Robots and Systems (IROS'14)*, pp. 623–629.

Tong, S., & Koller, D. (2002). Support vector machine active learning with applications to text classification. *Journal of Artificial Intelligence Research*, *2*, 45–66.

Weld, D. S., Anderson, C. R., & Smith, D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pp. 897–904, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Yang, Q., Wu, K., & Jiang, Y. (2007). Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, *171*(2), 107 – 143.

Younes, H. S., & Littman, M. L. (2004). PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Tech. rep., Carnegie Mellon University.

Zimmerman, T., & Kambhampati, S. (2003). Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, *24*(2), 73–96.