ASCoL: Automated Acquisition of Domain Specific Static Constraints from Plan Traces

Rabia Jilani and **Andrew Crampton** and **Diane Kitchin** and **Mauro Vallati**
School of Computing and Engineering
University of Huddersfield
United Kingdom
{U1270695, a.crampton, d.kitchin, m.vallati}@hud.ac.uk

Abstract

Domain-independent planning systems require that domain constraints and invariants are specified as part of the input domain model. In AI Planning, the generated plan is correct provided the constraints of the world in which the agent is operating are satisfied. Specifying operator descriptions by hand for planning domain models that also require domain specific constraints is time consuming, error prone and still a challenge for the AI planning community.

The LOCM (Cresswell, McCluskey, and West 2013) system carries out automated generation of the dynamic aspects of a planning domain model from a set of example training plans. We enhance the output domain model of the LOCM system to capture static domain constraints from the same set of input training plans as used by LOCM to learn dynamic aspects of the world.

In this paper we propose a new framework ASCoL (Automated Static Constraint Learner), to make constraint acquisition more efficient, by observing a set of training plan traces. Most systems that learn constraints automatically do so by analysing the operators of the planning world. Out proposed system will discover static constraints by analysing plan traces for correlations in the data. To do this an algorithm is in the process of development for graph discovery from the collection of ground action instances used in the input plan traces. The proposed algorithm will analyse the complete set of plan traces, based on a predefined set of constraints, and deduces facts from it. We then augment components of the LOCM generated domain with enriched constraints.

Introduction

In AI planning, designing the knowledge base and gathering application knowledge is crucial for the efficiency of planning systems, and for the correctness of resulting plans. It requires substantial effort as well as skillful experts to encode and maintain an error free and correct domain model and action constraints.

A constraint is an entity that restricts the values of variables in domain modelling (Nareyek et al. 2005). These can also be seen as static relationships between variables that help planning systems in quick and efficient pruning of the search tree. Existing domain-independent planning systems require domain constraints or invariants to be explicitly provided as part of the input domain model. In the operator schema, static constraints are often hand-coded in the form of predicates which restrict the values of variables in the preconditions of actions and define certain relationships that never change in the definition of a complete action model. Our work is aimed at automating the acquisition of such constraints by using sequences of plans as input training data. Each plan is a sequence of actions, where each action in the plan is stated as a name and a list of objects that are affected or are needed during the action's execution.

Learning domain knowledge from plan traces as input training data has attracted much interest in research from early work in (Benson 1996) to work in the recent past. (Jilani et al. 2014) present a brief overview of knowledge engineering tools in planning and compare these systems on a set of criteria.

This paper proposes ASCoL, a framework for learning static constraints. ASCoL is still in development, an initial part is already implemented and we are still working to extend it. It will be distinct from other systems in that it will learn domain model constraints from examples plans. It requires only plan traces and the type information from the domain model (the one that needs enhancement), without requiring any other information about partial domain model, initial, goal or intermediate states. Beside plan traces, we require the knowledge about the types of objects in input plan traces, and to get this type information we also input LOCM generated domain model. It works on the assumption that plan traces implicitly contain tacit knowledge about constraints acquisition/validation/satisfaction and, based on that assumption, we can draw correlations in the data by pattern discovery in the training input. Other systems require more input assistance. ARMS (Yang, Wu, and Jiang 2007), for example, is a system that learns the domain model in addition to domain constraints and invariants. It makes use of background information as input e.g. predicate definitions of initial and goal states. Similarly SLAF (Shahaf and Amir 2006) learns action schema but also requires definitions of fluents and a partial observation of intermediate states as input.

The main difference to earlier work is that the technique performs pattern discovery in the plan traces given by the user as input, with no additional knowledge about initial, goal or intermediate states. The developed part of the system learns the expected constraints when compared with bench-

mark domains.

To the best of our knowledge, our system will be the only system that can learn domain-independent general constraints as well as domain-specific constraints from the simplest inputs available.

The Constraint Acquisition Problem

The overall working of automated planning engines can be logically described around three components.

(a) **The domain model** (referred to as a domain description or action model in this paper) is the specification of the objects, states (predicates), and dynamics of the domain of planning. The main language used for the description of domain models is PDDL (McDermott D. et al. 1998). A domain model has two major components:

1. Dynamic Knowledge: a set of parametrised operator schema representing generic actions and resulting change in the domain under study.
2. Static Knowledge: relationships/constraints that are implicit in the set of operators and are not directly expressed but essentially are present while defining a domain model. These can be seen as predicates that appear in the preconditions of operators only and not in the effects. In simple words static facts never change in the world. According to Wickler (Wickler 2011), let $O = \{O_1, O_2, \ldots, O_n\}$ be a set of operators and let $P = \{P_1, P_2, \ldots, P_n\}$ be a set of all the predicate symbols that occur in these operators. A predicate $P_i \in P$ is fluent *iff* there is an operator $O_j \in O$ that has an effect that changes the truth of the predicate P_i. Otherwise the predicate is static.

(b) **The problem**: to solve a specific problem, automated planners takes problem specifications with domain knowledge as inputs. The problem file indicates the initial conditions and the required goal conditions for the output solution plan. The planning engine then reasons with the knowledge in (a) to solve (b).

(c) **The plan**: given a description of the possible initial state of the world, a description of the desired goal in (b), and a description of a set of possible actions in (a), the planning problem is to find a plan that is guaranteed to have a sequence of actions that leads to one of the goal conditions.

The constraint acquisition problem that this paper addresses is: given the knowledge of object types from (a.1)(generated by LOCM) and knowledge of (c)(input of LOCM), can we design a framework to automatically learn knowledge of (a.2)(to enhance output of LOCM)? We base our methodology on the assumption that plan traces contain tacit knowledge about constraints validation/acquisition.

Providing domain constraints to the planning engine may help the planning system in the quick and efficient pruning of the search tree.

ASCoL - The Proposed System

This paper proposes ASCoL, a framework for learning static constraints and embedding them into the output domain generated by LOCM. LOCM is a knowledge acquisition tool

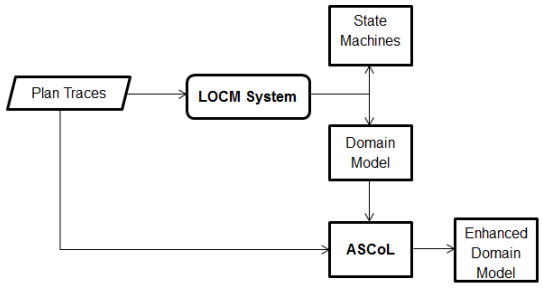

Figure 1: Input Output structure of ASCol

that carries out the automated generation of a planning domain model from example training plans. The uniqueness of LOCM is that it can learn action schema without requiring any information about predicates or initial, goal or intermediate state descriptions for the example action sequences.

The LOCM process can induce a representation for the dynamic aspects of objects but not the static aspects. As LOCM point out that in many domains, there are static facts or constraints, such as the layout of roads in driverlog, the level of floors in miconic domain or the fixed relationships between specific cards in freecell, that never change with the execution of actions. This information is not explicitly captured in the plan traces and is a big challenge to learn such static constraints from them.

LOCM does not generate static preconditions in fully automatic way. It has an option for users to declare manually which arguments of which perticular actions need to be made static constraints. LOCM manually declares this information in the following form:

```
Static(not-equal(L1, L2), sail(L1, L2)).
```

The above mentioned Prolog code line is added manually to the input training data file to make it a part of the output domain model manually. The fact in the first argument of static is added as a precondition of the action in the second operator argument of static, where shared variable names provide the binding between the action and its precondition.

ASCoL works as a separate unit from LOCM in that LOCM first produces a domain model using a set of plan traces as input. The same LOCM generated domain model, along with the same set of input plan traces, will then be fed to ASCoL to first anticipate the required set of constraints, analyse plan traces and then learn constraints. Finally, it embeds these constraints into the correct operators in the LOCM-learnt output to enrich the domain with this additional static knowledge. Figure 1 shows the general structure of ASCoL in terms of its inputs and outputs.

We aim to capture two major kinds of constraints: domain specific and domain independent constraints. By domain specific we mean the static knowledge that is strictly associated with a domain and is not found as a general example e.g. the fixed relationships between specific cards in freecell. Domain independent constraints, by this we mean the static knowledge that is generally associated with almost all domains in one way or the other e.g. non-equality constraints and mutual exclusion constraints. We are also ex-

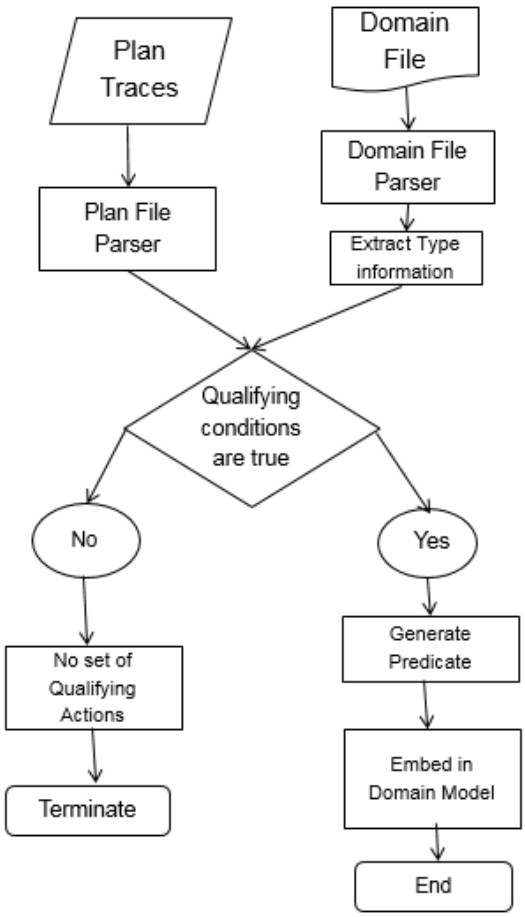

Figure 2: Internal Architecture of ASCoL

ploring more constraints for adding in our system to extend it further.

The steps shown below indicate how the new system will work. First we develop potential constraints, that could include:

1. Non-equality constraints: between object instances from the same object types e.g. ferry domain;
2. Cardinality constraints: relationships between objects of different actions can be one-to-one, one-to-many, many-to-one, or many-to-many;
3. Constraints between object instances from different object types e.g. mutual exclusion;
4. Domain specific constraints: partially ordered, linearly ordered or unordered graph detection e.g. The layout of roads in the driverlog domain, the fixed relationships between specific cards in the freecell domain.

After developing the potential constraints, we then parse the input plan traces and input domain file generated by LOCM, into different components; which can then be automatically processed. As a next step, the system then analyses the complete set of plans to discover whether the potential constraint is being satisfied or not. After that, it checks to see if Qualifying Conditions (QCs) are satisfied. QCs are different for each type of constraint and used in the algorithm of that particular type of constraint acquisition. For example, for domain independent, non-equality constraints, where constraint acquisition only requires the equality check between action parameters that belong to the same type, requires the following two QCs:

i. The arity of the action must be greater than one.

ii. The Action must contain a pair or more instances of same typed parameters.

As a final step the system generates static predicates/constraints and names them (but only for domain specific constraints) and adds the static predicates into the appropriate operators in the parsed LOCM generated domain file

Data Set for Experimentation

In this section we discuss the test data set and the experimentation that we did with the developed part of the system. We demonstrate how it can form an effective solution to the knowledge acquisition problem introduced in the last section. We will use two domain models from past IPCs as illustrative examples and use their plan traces throughout the rest of the paper. The first test candidate is the ferry domain model, used in the AIPS-2000 competition, where a number of cars have to be moved from their start to their goal-locations, using a ferry. Each location is accessible from each other location, cars can be debarked or boarded, and the ferry can only carry one car at any time. The static constraint in this benchmark ferry domain is the fact that for the sail action, the two locations between which the ferry has to sail must be unique.

The second domain model we used for experimentation purposes is a freecell domain, used in the AIPS-2000 competition, and is a STRIPS encoding of a card game (similar to Solitaire) that comes free with Microsoft Windows. Starting from an initial state, with a random configuration of cards across eight columns, the user can move cards in a specified order onto four home cells, following typical card stacking rules, and using a number of free cells as a resource. The domain specific static constraints of the freecell domain is the allowed sequential arrangement of cards in the free cells, the home cells and among the card columns using actions such as colfromfreecell, sendtohomecell etc. We assume that the plan traces implicitly contain tacit knowledge about constraints acquisition/validation/satisfaction – extracting the needed information from traces obtained from an instructor, a planner, or control system execution require a separate algorithm for each.

In ASCoL, the domain model, generated by LOCM, containing (a.1) knowledge is referred to as the input domain model. The input of the knowledge (c) is referred to as the input plan traces. Therefore, there are two inputs to ASCoL: the domain model and a set of plan traces.

The Input domain model is in standard PDDL format and it consists of:

Type names: this denotes types used in the domain world. There are a number of types (or classes) each containing a

set of objects such that each object belongs to one type only (called a sort in LOCM terminology). For example in the ferry domain:

```
(:types car loc)
```

Dynamic predicate definitions: these are facts representing relationships between objects of different types and all the possible values of an objects state. For ferry domain, the predicate definition induced by LOCM is shown below. Here the first three predicates indicate three different states of a car e.g. car boarded on ferry, car debarked from ferry and car available before boarding the ferry. The last two predicates are two different location states for ferry e.g. ferry location before the sail action and after the sail action.In LOCM, the following PDDL representation of these states include predicates with automatically generated unique labels which actually represent finite state machine (FSM) states, and makes more sense to reader with FSM in front.

```
(:predicates
      (car_state_1_1 ?v1 - car)
      (car_state_1_2 ?v1 - car)
      (car_state_1_3 ?v1 - car)
      (loc_state_1_1 ?v1 - loc)
      (loc_state_1_2 ?v1 - loc)
)
```

Action definitions: each action contains an action name, a list of parameters that will be affected in the action's execution and the number of predicates that collectively describe the changes in an object's state before and after the action executes. In the ferry domain description, sail is an operator where the ferry, after boarding a car, sails from one location to the next unique location before debarking the car. The fact that ensures this functionality is a predicate (not-equal ?L1, ?L2), where L1 and L2 are two different locations. Below is an example of the sail action generated by LOCM for the ferry domain, without static facts in the precondition.

```
   (:action sail
   :parameters (?L1 - loc ?L2 - loc)
   :precondition
      and
      (loc_state_1_2 ?L2)
      (loc_state_1_1 ?L1))
   :effect
      and
      (loc_state_1_1 ?L2)
      (not (loc_state_1_2 ?L2))
      (loc_state_1_2 ?L1)
      (not (loc_state_1_1 ?L1)))
    )
```

At the moment ASCoL completely learns non-equality constraints in all the test domains including ferry and freecell domain. We have already implemented the base structure of the frame work and next we are working on domain specific constraints now. ASCoL learns the set of static constraints and outputs it in standard PDDL format - we refer to it as *static knowledge*. It not only learns the required static constraints but also embeds them into the appropriate operators in the domain model; enhancing it with static facts. Domain constraints are learnt in the form of predicates, for example: for non-equality constraint, two locations cannot be the same in the sail operator of the ferry domain and this is expressed as (not-equal ?L1 ?L2).

To generate static facts for the ferry domain, we identify two object types, *Car* and *Location*. The relevant instances are multiple numbers of locations: L1, L2, L3, ... and multiple numbers of cars: c1, c2, c3, ...

After applying the ASCoL algorithm, the output is an enhanced version of the sail operator for the ferry domain which was previously generated by LOCM. The introduction of the static constraint (not-equal ?L1 ?L2) ensures that a ferry can only sail between unique locations, which is consistent with the AIPS 2000 ferry domain. Below we show the PDDL code for the sail action generated by ASCoL.

```
   sail
   :parameters
   (?L1 - loc ?L2 - loc)
   :precondition
   (and
      (not_equal ?L1 ?L2)
      (loc_state_1_2 ?L2)
      (loc_state_1_1 ?L1))

   :effect
   (and
      (loc_state_1_1 ?L2)
      (not (loc_state_1_2 ?L2))
      (loc_state_1_2 ?L1)
      (not (loc_state_1_1 ?L1)))
)
```

For domain independent constraints, we name them according to their respective function in the operator preconditions. For example, for domain independent, non-equality constraints, where constraint acquisition only requires an equality check between action parameters that belong to the same type, we have hard coded the name of the predicate in the relevant algorithm as (not-equal ?para1 ?para2). In the case where learning the static facts of a domain strictly depends on the specific functioning of the domain (i.e., domain dependent/specific constraints), we leave it for the user to name them appropriately after the learning process. For example in the freecell domain the required static predicate (can-stack ?card ?card1) and (face-val ?card) totally depends upon the objects of actions in the input plan traces.

Conclusion

The ASCoL algorithm is limited by the correctness and completeness of the input information it is given. By correctness we mean that plans should be noise free. Noise is inevitably introduced into plan traces when some sensors are occasionally damaged or with unintentional mistakes in the recording of the action sequence. By completeness we mean that plan traces should cover all the possible actions of the world. We assume that input plan sequences are noise free while the input domain file at least contains type information for all those operators that the algorithm aims to enhance.

Grant, in (Grant 2010), discusses the limitations of using plan traces as the source of input information. ASCoL faces similar difficulties as the only input source to verify constraints are sequences of plans. To overcome some of the known issues related to plan observations, we tested two different sets of plans, namely plans generated by random walks and goal-oriented plans:

i) Random-walk generators can be used to artificially produce large sequences of plan traces from the underlying domain model. Operators are generally chosen in a random but uniform way to ensure a good spread of ground actions for the knowledge acquisition system. However, problems can arise from this approach as it can fail to capture relationships between objects in other actions, this leads to not learning or missing predicates for those particular actions.

ii) Goal-oriented plans can provide useful information on effective actions combination and interaction. However, goal-oriented data can not capture all possible actions permutations, unlike randomly generated data. Hence a random generator can show more action transitions when compared to goal-oriented data.

We see several avenues for future work. We are working to include more algorithms in the ASCoL framework for learning a larger number of domain specific constraints e.g. to learn the complete map of locations from Truck domain plan traces.

References

Benson, S. S. 1996. *Learning action models for reactive autonomous agents*. Ph.D. Dissertation, stanford university.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using locm. *The Knowledge Engineering Review* 28(02):195–213.

Grant, T. 2010. Identifying domain invariants from an object-relationship model. *PlanSIG2010* 57.

Jilani, R.; Crampton, A.; Kitchin, D. E.; and Vallati, M. 2014. Automated knowledge engineering tools in planning: State-of-the-art and future challenges.

McDermott D. et al. 1998. PDDL–the planning domain definition language. Technical report, Available at: `www.cs.yale.edu/homes/dvm`.

Nareyek, A.; Freuder, E. C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H.; Rintanen, J.; and Tate, A. 2005. Constraints and ai planning. *Intelligent Systems, IEEE* 20(2):62–72.

Shahaf, D., and Amir, E. 2006. Learning partially observable action schemas. In *Proceedings of the national conference on artificial intelligence (AAAI)*.

Wickler, G. 2011. Using planning domain features to facilitate knowledge engineering. *KEPS 2011*.

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