ASCoL: a Tool for Improving Automatic Planning Domain Model Acquisition

Rabia Jilani, Andrew Crampton, Diane Kitchin, and Mauro Vallati

School of Computing and Engineering
University of Huddersfield, United Kingdom
{rabia.jilani, a.crampton, d.kitchin, m.vallati}@hud.ac.uk

Abstract. AI Planning requires domain models. Synthesising operator descriptions and domain specific constraints by hand for AI planning domain models is time intense, error-prone and challenging. To alleviate this, automatic domain model acquisition techniques have been introduced. Amongst others, the LOCM and LOCM2 systems require as input some plan traces only, and are effectively able to automatically encode a large part of the domain knowledge. In particular, LOCM effectively determines the dynamic part of the domain model. On the other hand, the static part of the domain – i.e., the underlying structure of the domain that can not be dynamically changed, but that affects the way in which actions can be performed – is usually missed, since it can hardly be derived by observing transitions only.
In this paper we introduce ASCoL, a tool that exploits graph analysis for automatically identifying static relations, in order to enhance planning domain models. ASCoL has been evaluated on domain models generated by LOCM for international planning competition domains, and has been shown to be effective.

Keywords: Automated Planning, Knowledge Engineering, Domain Model Acquisition.

1 Introduction

AI Planning is the process of finding a plan, i.e. a sequence of actions that applied in an initial state reach the desired goals. Planning is a pivotal task that has to be performed by every autonomous system. Planning techniques require domain models as input. Domain models encode the knowledge of the domains in terms of actions that can be executed and relevant properties. Creating such models is a difficult task, that is usually done manually; it requires planning experts and it is time-consuming and error-prone.

The domain model acquisition problem has mainly been tackled by exploiting two approaches. On the one hand, knowledge engineering tools for planning have been introduced over time, for supporting human experts in modelling the knowledge. Two particular examples are itSIMPLE [16] and PDDL studio [14]. A review of the state of the art is provided by Shah et al. [15]. Recently, also crowdsourcing has been exploited for acquiring planning domain models [18]. On the other hand, a number of techniques are currently available for automatic domain model acquisition; they rely on

example data for deriving domain models. Significant difference can be found in terms of quantity and quality of required inputs. For a detailed overview, the interested reader is referred to [11].

The LOCM systems [3, 4] perform automated generation of the dynamic aspects of a planning domain model, i.e. changes in the state of the world occurring due to action execution, by considering a set of plan traces, only. No additional knowledge about initial, goal or intermediate states is needed. In comparison with other systems, LOCM approaches require a minimal amount of information; other systems also require at least partial state information. The drawback of the LOCM process is it can induce only a partial domain model which represents the dynamic aspects of objects and not the static aspects. This is problematic since most domains require static predicates to both restrict the number of possible actions and correctly encode real-world constraints. This is the case for the connections of roads in Driverlog, the level of floors in Miconic domain or the fixed stacking relationships between specific cards in Freecell.

Usually any missing static relations are manually introduced into the domain models provided by the LOCM systems. Recently, the LOP approach has been proposed [8]. In order to extract static relations for extending LOCM-generated domain models, LOP exploits optimal goal-oriented plan traces. Specifically, they compare the optimal input plans with the optimal plans found by using the extended domain models; if the latter are shorter, some static relations are missing. This approach has drawbacks, as LOP strongly depends on the availability of optimal plans: they are usually hard to obtain for non-trivially solvable problems. It should be noted that trivial problems usually lead to extremely short plans, which tend to be not very informative for extracting knowledge. Moreover, LOP identifies the need for a static predicate between a set of action parameters, but does not provide information about the type of relationship that connects the involved predicates.

In this paper we present ASCoL, (Automated Static Constraint Learner), a tool that can effectively identify static relations between predicates by considering any type of plan traces: they can be either optimal / suboptimal goal-oriented or random walks. The only requirement on plans is that they have been generated by problems sharing the same objects; this follows the hypothesis of LOCM, in which different plan traces are generated by observing the behaviour of an agent in its environment. By exploiting a directed graph representation of operator arguments relations, ASCoL is also able to identify the type of relation that pairs of predicates show. Relations between pairs can be extended for deriving more complex n-arity static predicates. A preliminary version of ASCoL has been presented in [10]; this version was able to identify inequality constraints only. In a large experimental analysis, we demonstrate the ability of ASCoL in finding static relations for enhancing domain models automatically acquired by LOCM. Remarkably results have been achieved in complex domains, with regards to the number of static relations, like TPP and Freecell. It should be noted that ASCoL can be also used without LOCM. In particular, we observed that it can be also useful for debugging domain models, in order to identify missing static relations, or to further constrain the search space by pruning useless actions.

The remainder of this paper is organised as follows. Section 2 provides the necessary background on AI Planning and describes the addressed learning problem. Section

3 gives a detailed description of ASCoL. Section 4 presents the results of the empirical evaluation. Finally, Section 5 concludes the paper and gives directions for future research.

2 Background

In this Section we firstly provide the necessary background on AI Planning, and then introduce the learning problem addressed by ASCoL.

2.1 AI Planning

In this work we consider classical planning [6] domain models. Classical planning deals with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic and fully observable environment from some initial state to a desired goal state. In the classical representation *atoms* are predicates. *States* are defined as sets of ground predicates. A *planning operator* $op = (\textit{name}(o), \textit{pre}(o), \textit{eff}^-(o), \textit{eff}^+(o))$ is specified such that $\textit{name}(o) = \textit{op\_name}(x_1, \ldots, x_k)$ (*op_name* is a unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator), $\textit{pre}(o)$ is a set of predicates representing the operator's preconditions, $\textit{eff}^-(o)$ and $\textit{eff}^+(o)$ are sets of predicates representing the operator's negative and positive effects. *Actions* are ground instances of planning operators. An action $A = (\textit{pre}(A), \textit{eff}^-(A), \textit{eff}^+(A))$ is *applicable* in a state s if and only if $\textit{pre}(A) \subseteq s$. Application of A in s (if possible) results in a state $(s \setminus \textit{eff}^-(A)) \cup \textit{eff}^+(A)$.

A *planning domain model* is specified via sets of predicates and planning operators. A *planning problem* is specified via a planning domain, initial state and set of goal atoms. A *solution plan* is a sequence of actions such that a consecutive application of the actions in the plan (starting in the initial state) results in a state that satisfies the goal.

Specifically, the *planning domain model* (hereinafter called domain description, action model or operator schema) has two major elements:

1. Dynamic Knowledge: a set of parametrised action schema representing generic actions and resulting effects in the domain under study.
2. Static Knowledge: relationships/constraints that are implicit in the set of operators and are not directly expressed in the plans. These can be seen as predicates that appear in the preconditions of operators only, and not in the effects. Therefore, static facts never change in the world, but are essential for modelling the correct action execution. According to Wickler [17], let $Op = \{op_1, op_2, \ldots, op_n\}$ be a set of operators and let $Pr = \{Pr_1, Pr_2, \ldots, Pr_n\}$ be a set of all the predicate symbols that occur in these operators. A predicate $Pr_i \in Pr$ is fluent *iff* there is an operator $op_j \in Op$ that has an effect that changes the truth of the predicate Pr_i. Otherwise the predicate is static.

For instance, in the drive operator of the TPP domain model used in the International Planning Competitions (IPC) [2], the *connected* predicate is a static relation between two places. There is no operator in the TPP domain model that can change that predicate, but it is fundamental for modelling the domain.

```
(:action drive-ipc
 :parameters
 (?t - truck ?from ?to - place)
 :precondition
 (and
     (at ?t ?from)
     (connected ?from ?to))
 :effect
 (and
     (not (at ?t ?from))
     (at ?t ?to))
)
```

ASCoL accepts input plans (plan traces) in the same text-based format supported by LOCM. i.e. a training sequence of N actions in order of occurrence, which all have the form:

$$A_i(O_{i1}, ..., O_{ij}) \quad for \quad i = 1, ..., N$$

Where *A* is the action name and *O* is the action's object name. Each action (*A*) in the plan is stated as a name and a list of arguments. In each action (A_i), there are *j* arguments where each argument is an object (*O*) of the problem.

2.2 The Learning Problem

We define the learning problem that ASCoL addresses as follows. Given the knowledge about object types, operators and predicates, and a set of plan traces, how can we automatically identify the static relation predicates that are needed by operators' preconditions? We base our methodology on the assumption that plan traces contain tacit knowledge about constraints validation/acquisition.

Specifically, a *learning problem description* is a tuple *(P, T)*, where *P* is a set of plan traces and *T* is a set of types of action arguments in *P*. The *output* for a learning problem is a *constraint repository R* that stores all admissible constraints on the arguments of each action *A* in plan traces *P*.

3 The ASCoL Tool

We now present the ASCoL tool that has been developed for identifying useful static relations. ASCoL requires as input a set of plan traces and a partial domain model. The provided output is an extended domain model including static relations. The process can be summarised as follows:

1. Read the partial domain model and the plan traces.
2. Identify, for all operators, all the pairs of arguments involving the same object types.
3. For each of the pairs, generate a directed graph by considering the objects involved in the matching actions from the plan traces.
4. Analyse the directed graphs and extract hidden static relations between arguments.

5. Run inequality check.
6. Return the extended domain model that includes the identified static relations.

It should be noted that ASCoL relies on two assumptions:

- The dynamic part of the domain model provided as input is correct, and includes the type of each operator argument.
- The plan traces have been generated by considering problems that share the same objects, and object names.

With regards to the first assumption, LOCM systems are able to effectively model the dynamic side of domain models. In terms of plan traces, the ASCoL assumption is particularly reasonable when traces come from real-world application observations; in that scenario sensors are identifying and naming objects – they usually exploit standard names. Moreover, it is acceptable to assume that the relevant objects of a real-world scenario will not quickly change. For example, machines in a factory or trucks and depots for logistics companies.

The main information available for ASCoL comes from the input plan traces. As a first control, we remove from the plan traces all the actions that refer to operators that do not satisfy the following conditions:

i. The arity of the operator must be greater than one.
ii. The operator must contain at least two arguments of the same type.

Even though, theoretically, static relations can hold between objects of different types, they mostly arise between same-typed objects. This is the case in transport domains, where static relations define connections between locations. Moreover, considering only same-typed object pairs can reduce the computational time required for identifying relations. It is also worth noting that, in most of the cases where static relations involve objects of different types, this is due to a non-optimal modelling process. For instance, if a factory machine m is statically located in a position p, a more compact and efficient representation should consider m and p as the same object. Furthermore, such relations can be easily identified by naively checking the objects involved in actions; whenever some objects of different type always appear together, they are likely to be statically related.

For the purpose of illustrating the ASCoL process, we consider as a domain model example *Freecell*, used in the IPC3 planning competition [12]. It is a STRIPS [5] 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 are the allowed sequential arrangement of cards in the free cells, the home cells and among the card columns using actions such as `colfromfreecell` and `sendtohomecell`.

The Freecell domain provides a suitable framework to integrate our approach. We choose this domain as an illustrative example because all its ten operators satisfy the above mentioned conditions and it is rich in terms of a challenge to learn static facts, e.g.

Can-Stack and *Successor* constraints. Freecell plan traces only contain three types, i.e. *card*, *suit* and *num*. This domain is made more complex by the fact that one single type *num* is used to represent three things and this in turn removes the boundary between three different behaviours. *num* is used to represent the face value of cards and to count free cells and free columns. Let us consider the `sendtofree-b` operator. The type signature of the operator is as follows:

```
sendtofree-b(?card -card ?cells ?ncells ?cols ?ncols -num)
```

This operator sends the *?card* to a free cell, considering this card as the last one in column. *?cells* and *?ncells* represents the number of available free cells while *?cols* and *?ncols* represents the number of empty columns after sending last card of the column to free cell. Here type *num* is used to represent surrogate of free cell count as well as empty column count.

3.1 Generation of Directed Graphs

Step 2 of ASCoL scans the partial domain model provided. This is done in order to identify all the directed graphs that have to be generated. For each operator satisfying the aforementioned two conditions, a directed graph is generated between each ordered couple of same-typed arguments. At Step 3, each of the previously identified directed graphs *G = (IDs, Conn)* is generated. *G* is generated by considering the objects that appear in actions of plan traces, corresponding to instantiation of the operators' arguments. Specifically, *IDs* is the set of vertices of the graph *G*, which includes all the objects observed from plan traces; *Conn* is the finite set of arcs in *G*. We decide to generate an arc that goes from the object used in the first considered argument, to the second. Therefore, an arc (o_1, o_2) is directed from o_1 to o_2, and it is added to *Conn* if the objects o_1 and o_2 appear, in this order, in the place of the considered arguments of one action of the plan trace. To explain the structure of *G* better, we continue with our example. Considering the `sendtofree-b` action, the following pairs of arguments are identified at Step 2: pair1 (?cell, ?ncells), pair2 (?cell, ?cols), pair3 (?cell, ?ncols), pair4 (?ncells, ?cols), pair5 (?ncells, ?ncols), pair6 (?cols, ?ncols). For each of the identified pair of arguments, a directed graph is generated, by considering the objects used in plan traces. In order to exemplify how directed graphs are generated, let us suppose that from a plan trace we collected the following two instances of the `sendtofree-b` action.

```
sendtofree-b(cardX N1 N0 N3 N4)
sendtofree-b(cardY N2 N1 N2 N3)
```

In order to generate the directed graph for the pair1 arguments, ASCoL considers all the objects used as second and third arguments. Given our example, *IDs*$= \{N1, N0, N2\}$. The *Conn* set includes the following arcs: $\{(N1, N0), (N2, N1)\}$. This is because, in the first instance of `sendtofree-b`, $N1$ is before $N0$, and in the second $N2$ appears before $N1$. Finally, the directed graph *G* of our example has the following structure:

$$N2 \rightarrow N1 \rightarrow N0$$

3.2 Analysis of Directed Graphs

In step 4, the generated directed graphs are investigated. In particular, we are interested in identifying their structure, for determining whether there is a specific ordering among the involved objects, and consequently among the arguments of corresponding operators. The directed graphs can have three different shapes:

Totally ordered graph Given the directed graph G, for each pair of vertices a, b we say that $a > b$ if there exists a directed route that connects a to b. Moreover, we say that $a = b$ only if a and b are the same vertex. This order relation between vertices allows checking for total ordering among them. Total ordering is assessed by checking the trichotomy property for each pair of vertices [1]. The trichotomy property states that given two objects a and b , exactly one relation between $a < b$, $a = b$, $a > b$ holds. In addition to trichotomy law, also the following properties can hold:

1. Transitivity: $a > b$ and $b > c$ implies $a > c$.
2. Antisimmetry: $a > b$ and $b > a$ implies $a = b$.

Cyclic graph In a cyclic graph, there exists at least one route that can return to its starting vertex.

Acyclic graph A directed acyclic graph is a graph with no directed cycles. Starting from any vertex, there is no directed route to return to the starting point.

ASCoL examines the structure of all the directed graphs, in order to identify static relations between the corresponding pairs of operator arguments. If the graph G is proved to be totally ordered, this indicates a strong static relation between the arguments. In particular, the arguments are used as a scale of values, i.e. it is always possible to identify, given two values, an ordering between them. This sort of structure is usually exploited in STRIPS for modelling levels or quantities. For instance, in the well-known Zenotravel domain, fuel levels of aircraft are modelled by using `next` static-predicates, which are used for providing an ordering between the different `flevel`.

In case of a directed cyclic graph, ASCoL tests if G is fully connected or not. In the former case, a static relation of type "connection" is added to the pre-conditions of the operator. This is because, mainly in transport-like domains, these sort of graphs indicate the presence of an underlying map of strongly connected locations.

Finally, if the graph is acyclic, the current version of ASCoL can not derive any information. An acyclic graph can either indicate that no static relation is in place between the considered arguments, or there exists some partial ordering between them. For instance, the presence of a not-strongly connected map of locations; this is particularly true if plan traces are not generated by random walks.

3.3 Inequality Check

For each graph G that has been proven to be cyclic, ASCoL checks the presence of self-loops on vertices. If no loops are observed, then an inequality precondition (i.e., a precondition that forces the two arguments to be different) is added to the operator. This is the case of the `sail` operator from the *Ferry* domain. The two locations shall never be equal. The following shows the enhanced operator:

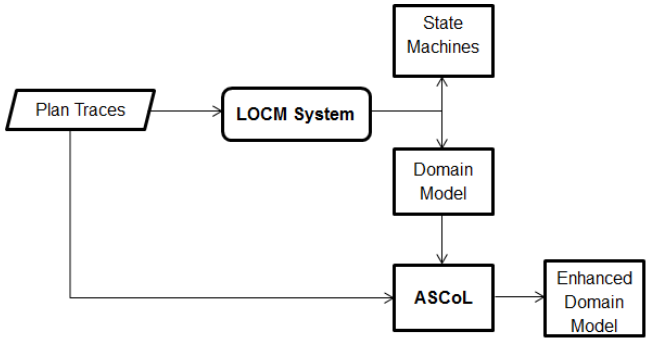

Fig. 1. The testing framework used for evaluating ASCoL.

```
(:action sail
    :parameters (?from - location ?to - location)
    :precondition (and
        (not (= ?from ?to))
        (at-ferry ?from))
    :effect (and
        (at-ferry ?to)
        (not (at-ferry ?from)))
)
```

ASCoL forces inequality by using *negative precondition* and *equality* features of PDDL [13]. Currently, many planners support such features, also due to the use of them in recent IPCs.

4 Experimental Evaluation

The aim of this experimental analysis is to assess the ability of the ASCoL tool in identifying static relationships between arguments. The testing framework used is shown in Figure 1. LOCM system is used for generating the partial domain model, that includes only the dynamic knowledge of the domain. The generated domain models, and the plan traces, are provided as input to ASCoL, which returns an extended version of the domain model.

15 domain models have been considered, taken either from IPCs[1] or from the FF domain collection (FFd)[2]: Barman (IPC7), Driverlog (IPC3), Ferry (FFd), Freecell (IPC3), Gold-miner (IPC6), Gripper (IPC7), Hanoi (FFd), Logistics (IPC2), Miconic (IPC2), Mprime (IPC1), TPP (IPC5), Trucks (IPC5), Spanner (IPC6), Storage (IPC5) and ZenoTravel (IPC3). These domains have been selected because they are encoded

[1] http://ipc.icaps-conference.org/
[2] https://fai.cs.uni-saarland.de/hoffmann/ff-domains.html

using different models and modelling strategies, and their operators include more than one argument per object type. All domains but Gripper, Logistics and Hanoi, exploit static relations. Plans have been generated by using Metric-FF planner [9] on randomly generated problems, sharing the same objects. ASCoL has been implemented in Java, and run on a Core 2 Duo/8GB processor.

Table 1 shows the results of the experimental analysis. Results are shown in terms of the number of identified static relationships (Learnt SR) and number of additional static relations provided (Additional SR) that were not included in the original domain model. We also provide information about the number of plans required for ASCoL to converge – i.e. adding more plans does not change the number of identified static relations –, the average length of plans, and the CPU time required by ASCoL. Interestingly, we observe that ASCoL is usually able to identify all the static relations of the considered domains. Moreover, in some domains it is providing additional static relations, which are not included in the original domain model. Remarkably, such additional relations do not reduce the solvability of problems, but reduce the size of the search space by pruning useless instantiations of operators.

With regards to the number of plans required by ASCoL to converge, we observe that in many cases, a few "long" plans are enough. Apparently, there is not a strong relation between the number of required plans and the overall number of static relations. Instead, a larger number of plans is needed when some operators are rarely used, or if the number of arguments of the same type are high, per operator.

Considering a classification terminology, we can divide the relations identified by ASCoL in to four classes: true positive, true negative, false positive and false negative.

True positive These are correctly identified static relations. Relations identified by ASCoL are almost always static relations which are included in the original domain models.

True negative Dynamic relations that are correctly not encoded as static relations. ASCoL did not identify a static relation between arguments that are actually connected by a dynamic relation in any of the considered domains.

False positive In some domains ASCoL infers one or two additional relations facts that are not included in the original domain model. From the Knowledge Engineering point of view, and considering the fact that such additional preconditions do not reduce the solvability of problems, such inferred relations can add value to the original model in terms of effectiveness of plan generation.

False negative In Freecell and Gold-miner domains ASCoL does not identify all the static relations.

The ability of ASCoL to correctly identify static relations that should be included as preconditions of specific operators, depends on the number of times the particular operator appears in the provided plan traces. The higher the number of instances of the operator in the plan, the higher the probability that ASCoL will correctly identify all the static relations. For example, in TPP domain only 2 plans of average length 200 actions each yields the learning of 85% of the static relations. With the addition of another plan of length 200 actions in input, 100% are identified. TPP domain is one of the more complex considered domains, in terms of having more same type arguments (4 same typed) in 3 out of 4 operator headings.

Table 1. Overall results on considered domains. For each original domain, the number of operators (# Operators), and the total number of static relations (# SR) are presented. ASCoL results are shown in terms of the number of identified static relationships (Learnt SR) and number of additional static relations provided (Additional SR) that were not included in the original domain model. Such relations do not compromise the solvability of problems, but prune the search space. The seventh and eighth columns indicate respectively the number of plans provided in input to ASCoL, that allows it to converge, and the average number of actions per plan (A/P). The last column shows the CPU-time in milliseconds

Domain	# Operators	# SR	Learnt SR	Additional SR	# Plans	Avg. A/P	CPU-time (ms)
TPP	4	7	7	0	7	28	171
Zenotravel	5	4	6	2	4	24	109
Miconic	4	2	2	0	1	177	143
Storage	5	5	5	0	24	15	175
Freecell	10	19	13	0	20	60	320
Hanoi	1	0	1	1	1	60	140
Logistics	6	0	1	1	3	12	98
Driverlog	6	2	2	0	3	12	35
Mprime	4	7	7	0	10	30	190
Spanner	3	1	1	0	1	8	144
Gripper	3	0	1	1	1	14	10
Ferry	3	1	2	1	1	18	130
Barman	12	3	3	0	1	150	158
Gold-miner	7	3	1	0	13	20	128
Trucks	4	3	3	0	6	25	158

We now discuss some of the most interesting results.

4.1 Freecell

In this domain there are six cases out of nineteen where ASCoL can not identify static relations. We observe that such relations are somehow peculiar to the domain. They describe stacking rule relations between cards of different suits. In particular, those relations model at the same time the fact that one card $c1$ can be stacked over another card $c2$ if: (i) the colour of the suit of $c1$ is red, and the colour of the suit of $c2$ is black (or vice-versa), and (ii) the nominal value of $c1$ is smaller (higher if stacking at home) then the value of $c2$.

Apart from the aforementioned six cases, which compress two static relations in a single predicate, all the other are correctly identified by ASCoL. Remarkably, even though `sendtofree-b` and `newcolfromfreecell` are operators that appear rarely in plan traces (and are used rarely in the card game as well, they model the fact that one card is sent to a free cell or to a free column), ASCoL correctly identifies all their static preconditions.

Due to the rare occurrence of the above mentioned two operators, it prevents ASCoL from learning the corresponding relations from just a few plans. A large number of plans are required; at least 20. Remarkably, only eight plan traces are required for correctly identifying the static relations of the remaining operators. On the other hand, the high

number of plans is also due to the fact that num type models different aspects of the game. Empirically, we observed that for reducing the number of plans splitting the num type is useful. num is used for modelling three different aspects of the game, therefore we split it as follows: $freeCol$ to keep a record of free cells available, col to keep record of empty columns and $home$ to record arrangement of cards in home cell. This way ASCoL only requires eight plans to learn all the relations.

4.2 TPP domain

TPP is an example of one of the most complex domains for the ASCoL approach. Out of four operators, three contain seven arguments each, four of them of the same type *level*. Out of the seven, four arguments are of the same type *level*. The following shows the mentioned operators; arguments names indicate the corresponding type.

```
buy (goods truck market level1 level2 level3 level4)
unload (goods truck depot level1 level2 level3 level4)
load (truck goods market level1 level2 level3 level4)
```

The large number of arguments of same type gives rise to six pairs per operator, with the corresponding graphs. Out of six pairs, ASCoL correctly identify the two static binary relations as the benchmark domain also contains only two static facts in each i.e. *(next level2 level1) (next level4 level3)*.

4.3 Miconic domain

Miconic domain has two static predicates: one in *up* and the other in *down* actions. Such static relations are used for indicating the ordering of arguments of type *floor*, i.e. the order between the different floors which are connected by the elevator. In this domain, the predicate `above` is used for modelling the fact that one floor is above the lower one. Such predicate is used both for up and down movements of the elevator, but in a different way. ASCoL successfully detects the presence of a link between floors, but generates two different static relations, one per operator. Although different from the original model, such encoding is an alternative way for modelling the relation between floors.

4.4 Gold-miner

Gold-miner domain model exploits a static relation (namely, *connected*), which indicates that two cells are connected, in three operators: *move*, *detonate-bomb* and *fire-laser*. In our experimental analysis, we generated plans using the problem set provided for IPC6. By considering the corresponding plans, ASCoL learns easily the connection fact used in the *move* operator. Interestingly, ASCoL is not able to correctly identify the same static relationship in the remaining two operators. This is due to the fact that both *detonate-bomb* and *fire-laser* are rarely used, usually once per plan. Moreover, in the considered IPC6 problems, the structure of problems forces the robot to fire the laser (or detonate bombs) within the same row of the grid, but not across different columns.

Changing the structure of problems slightly, so that robot can detonate bombs (or fire laser) also between cells of different columns, allows ASCoL to learn all the static relations of the gold-miner domain model. It should be noted that changing the problems does not modify the overall structure of the domain model. Moreover, the very peculiar shape of the considered problems is possibly due to a bug in the random problem generator.

5 Conclusion and Future Goals

Planning is a fundamental task for autonomous agents. In order to perform planning, it is critical to have a model of the domain in which the agent has to interact and operate. Encoding domain knowledge can be done manually, but it requires both planning and domain experts. Moreover, it is time-consuming and error-prone. To overcome such issues, techniques for automatic domain acquisition have been proposed over time. Most of them require a large amount of knowledge to be provided as input. Currently, LOCM systems can effectively encode the dynamic part of domain models by analysing plan traces, only. Static knowledge, that is represented as preconditions in PDDL operators, can not be identified by LOCM or similar approaches.

In this paper we introduced ASCoL, an efficient and effective tool for identifying static knowledge missed by domain models automatically acquired. The proposed approach generates a directed graph for each pair of same-type arguments of operators and, by analysing some properties of the graphs, identifies relevant relations between arguments. Remarkably, the contributions of ASCoL, as demonstrated by our large experimental analysis, are: (i) the ability to identify two different types of static relations, by exploiting graph analysis; (ii) ASCoL can work with both optimal and suboptimal plan traces; (iii) considering pairs of same-typed objects allow the identification of all the static relations considered in the benchmark models, and (iv) it can be a useful debugging tool for improving existing models, which can indicate hidden static relations helpful for pruning the search space.

ASCoL can be exploited both by domain and planning experts. Domain experts can use ASCoL for improving their PDDL knowledge, and generating more robust and more optimised domain models. Planning experts can exploit the proposed tool for identifying hidden static relations between different aspects of the domain, thus improving their understanding of the specific application.

We see several avenues for future work. We plan to thoroughly test the differences between using different kind of plan traces, e.g., random walks and goal-oriented generated by different planners. Grant, in [7], 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. We are also interested in extending our approach for considering static relations that involve more than two arguments: although they are not usually exploited in domain models, and can be broken down to a set of static relations between pairs, they can generally be useful. In particular, we aim to extend the approach for merging graphs of different couples of arguments. Finally, we plan to identify heuristics for extracting useful information also from acyclic graphs.

References

1. Apostol, T.M.: Calculus, volume I. John Wiley & Sons (2007)
2. Coles, A., Coles, A., Olaya, A.G., Jiménez, S., López, C.L., Sanner, S., Yoon, S.: A survey of the Seventh International Planning Competition. AI Magazine 33(1), 83–88 (2012)
3. Cresswell, S., McCluskey, T.L., West, M.M.: Acquisition of object-centred domain models from planning examples. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2009)
4. Cresswell, S.N., McCluskey, T.L., West, M.M.: Acquiring planning domain models using LOCM. The Knowledge Engineering Review 28(02), 195–213 (2013)
5. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2(3), 189–208 (1972)
6. Ghallab, M., Nau, D., Traverso, P.: Automated planning: theory & practice (2004)
7. Grant, T.: Identifying Domain Invariants from an Object-Relationship Model. PlanSIG2010 p. 57 (2010)
8. Gregory, P., Cresswell, S.: Domain model acquisition in the presence of static relations in the LOP system. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2015)
9. Hoffmann, J.: The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables 20, 291–341 (2003)
10. Jilani, R., Crampton, A., Kitchin, D.E., Vallati, M.: ASCoL: Automated acquisition of domain specific static constraints from plan traces. In: The UK Planning and Scheduling Special Interest Group (UK PlanSIG) 2014 (2014)
11. Jilani, R., Crampton, A., Kitchin, D.E., Vallati, M.: Automated Knowledge Engineering Tools in Planning: State-of-the-art and Future Challenges. In: The Knowledge Engineering for Planning and Scheduling workshop (KEPS) (2014)
12. Long, D., Fox, M.: The 3rd International Planning Competition: Results and analysis. J. Artif. Intell. Res.(JAIR) 20, 1–59 (2003)
13. Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL - The Planning Domain Definition Language. Tech. rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control (1998)
14. Plch, T., Chomut, M., Brom, C., Barták, R.: Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL studio. System Demonstrations and Exhibits at ICAPS pp. 15–18 (2012)
15. Shah, M., Chrpa, L., Jimoh, F., Kitchin, D., McCluskey, T., Parkinson, S., Vallati, M.: Knowledge engineering tools in planning: State-of-the-art and future challenges. In: Proceedings of the Workshop on Knowledge Engineering for Planning and Scheduling (2013)
16. Vaquero, T.S., Romero, V., Tonidandel, F., Silva, J.R.: itSIMPLE 2.0: An Integrated Tool for Designing Planning Domains. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS). pp. 336–343 (2007)
17. Wickler, G.: Using planning domain features to facilitate knowledge engineering. KEPS 2011 (2011)
18. Zhuo, H.H.: Crowdsourced Action-Model Acquisition for Planning. In: Proceedings of the AAAI Conference on Artificial Intelligence (2015)