Action Model Acquisition using Sequential Pattern Mining

Ankuj Arora, Humbert Fiorino, Damien Pellier, Sylvie Pesty

Action Model Acquisition using Sequential Pattern Mining

Ankuj Arora, Humbert Fiorino, Damien Pellier and Sylvie Pesty

Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France
firstname.lastname@univ-grenoble-alpes.fr

Abstract. This paper presents an approach to learn the agents' action model (action blueprints orchestrating transitions of the system state) from plan execution sequences. It does so by representing intra-action and inter-action dependencies in the form of a maximum satisfiability problem (MAX-SAT), and solving it with a MAX-SAT solver to reconstruct the underlying action model. Unlike previous MAX-SAT driven approaches, our chosen dependencies exploit the relationship between consecutive actions, rendering more accurately learnt models in the end.

1 Introduction

In the planning community, intelligent agents require an action model to plan and solve real world problems. It is, however, becoming increasingly cumbersome to codify this model, and is more efficient to learn these action blueprints from plan execution sequences. This learning provides an opportunity for the evolution of the model towards a version more consistent and adapted to its environment, augmenting the possibility of success of the plans. Our approach, called SRMLearn (**S**equential **R**ules-based **M**odel **Learn**er), uses alternating state-action representations as input to learn an action model as the output. It proceeds as follows: it represents a set of intra-action and inter-action dependencies in the form of constraints of a weighted maximum satisfiability problem. This problem is then solved with the help of a MAX-SAT solver, the solved constraints being used to reconstruct the underlying action model. This paper is divided into the following sections: we present some related work in section 2, and define our learning problem in section 3. We then detail our approach in section 4, and present our empirical evaluations in section 5. We conclude the paper with some perspectives and future work in section 6.

2 Related Work

Learning action models in the field of Automated Planning (AP) has a considerable history. Some prominently used machine learning (ML) techniques to learn action models include: inductive techniques (e.g. PELA [7]), reinforcement learning techniques (e.g. LOPE [6]) and so on. More specific to our case, various approaches have used the MAX-SAT framework to learn deterministic actions (e.g. ARMS [10]), macro-actions [12], models in Hierarchical Task Networks (HTNs) [11] and so on. In particular, our

approach is on the same lines of ARMS, which also generates intra-action and inter-action constraints (mined with the Apriori algorithm [1]). As compared to ARMS, we hypothesize and experimentally demonstrate that short term dependencies among consecutively executing action pairs are a stronger indicator of correlation between actions, leading to improved learning.

3 Preliminaries and Problem Formulation

We begin by providing some definitions of key concepts. *Predicates* are properties that constitute the world state and actions. Here, each action a∈A where A = $\{a_1, a_2, \ldots, a_n\}$, n being the maximum number of actions in the domain. An *action model* m is the blueprint of all the domain-applicable actions, each action defined as an aggregation of: (i) the action name (with zero or more typed variables as parameters), and (ii) predicates in the form of preconditions (*pre* list i.e. predicates whose satisfaction determines the applicability of the action) and effects (*add* and del list i.e. predicates added and deleted respectively from the current world state by action execution). A *plan* is a sequence of actions $\pi = [a_1, a_2, \ldots, a_n]$ that drives the system from the initial state to a goal state. Each such sequence consisting of (i) the initial state of the world, (ii) alternating action and state representations, and (iii) a desired goal state; constitutes a *trace* of a *trace set* T. A *sequential rule* $a_x \rightarrow a_y$ is a relationship between two actions a_x, $a_y \in A$ such that if a_x occurs in a sequence, then a_y will occur successively in the same sequence. Two measures are defined for sequential rules, these are (i) $support(a_x \rightarrow a_y) = |a_x \rightarrow a_y|/|T|$, and (ii) $confidence(a_x \rightarrow a_y) = |a_x \rightarrow a_y|/|a_x|$. Given the aforementioned information, our learning problem is as follows: given a set of plan traces *T*, the objective is to learn the underlying action model m which best explains the observed plan traces. AP uses a certain number of domains from the International Planning Competition (IPC), out of which we use the *gripper* domain to illustrate SRM-Learn (see Figure 1). In this domain, the task of the robot is to move an object from one room to another, the principal domain actions being *move, pick* and *drop*.

4 Approach

Our approach is divided into three phases (see Figure 2) which are elaborated in the forthcoming subsections.

4.1 Annotation and Generalization

Firstly, each trace is taken one by one, and each action as well as each predicate from the initial, goal and intermediate states is scanned to substitute the instantiated parameters with corresponding variable types, producing generalized actions and predicates. We then associate each action with its relevant predicates, where a predicate is said to be relevant to an action if they share the same variable types. The set of relevant predicates to an action $a_i \in A$ can be denoted as $relPre_{a_i}$. With generalized actions and predicates, a candidate action dictionary is built for the actions, where the key is the name of the action a_i and the value is a list of all relevant predicates to that action $relPre_{a_i}$ (see Figure 1).

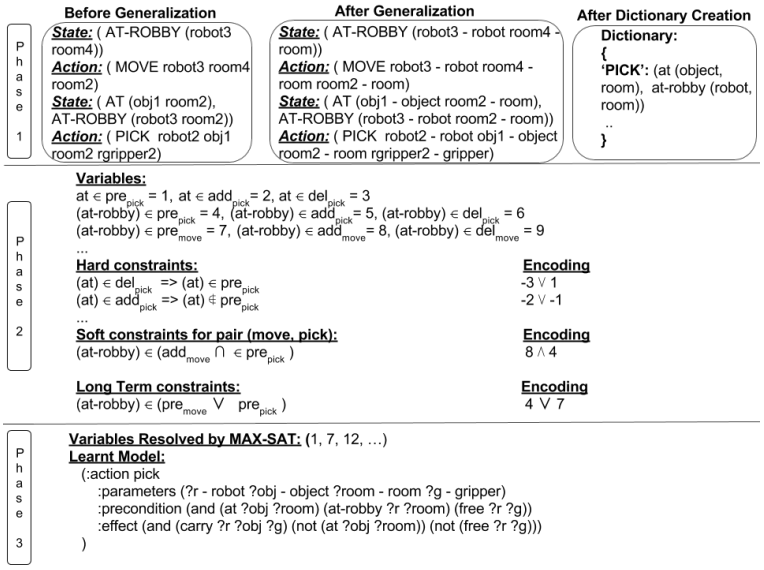

Fig. 1: Illustration of our learning problem. The learnt action model is written in PDDL (Planning Domain Description Language) [9] and conforms to the semantics of STRIPS [3].

4.2 Constraint Generation

In this phase, we account for certain intra-action hard constraints and inter-action soft constraints.

Hard constraints In order to satisfy the the semantics defined in section 3, each action in A must satisfy certain *intra-action constraints*. Thus, for each action $a_i \in [a_1, a_2, \ldots, a_n]$ and each relevant predicate $p \in relPre_{a_i}$: (i) p cannot be in the *add* list and the *del* list for the same action, and (ii) *p* cannot be in the *add* list and the *pre* list for the same action.

Soft Constraints The soft constraints among the actions may be short-term or long-term.

Short-Term Constraints We hypothesize that if a sequential pair of actions appears frequently in the traces, there must be a reason for their frequent co-existence. We thus employ an algorithm called TRuleGrowth [5, 4] used for mining sequential rules common to several sequences that appear in a trace set. Given (1) a trace set *T*, and (2) two user-specified thresholds, namely $support$ and $confidence$ as input, TRuleGrowth outputs all sequential rules having a support and confidence higher than $support$ and $confidence$ respectively. Starting with 20 traces, we consistently double the number of traces till we reach 200. In the process, we identify frequent sequential rules (action

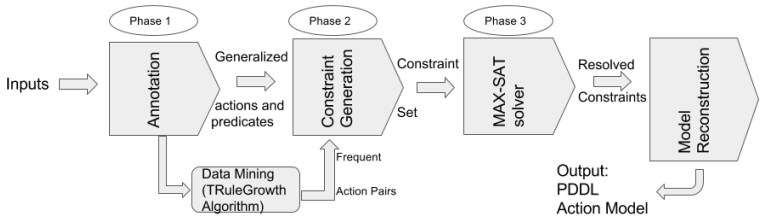

Fig. 2: Approach phases of SRMLearn

pairs) which consistently maintain the *confidence* and *support* over an increasing number of traces. These frequent pairs can be suspected to share a "semantic" relationship among themselves. These relationships are quantified by the ARMS [10] system, and serve as heuristics to explain the frequent co-existence of these actions. These heuristics produce good results in the case of the ARMS system, which serves as incentive for re-using them. More precisely, if there is an action pair $(a_i, a_j), 0 \leq i < j \leq (n-1)$ where n is the total number of actions in the plan; and pre_i, add_i and del_i represent a_i's *pre*, *add* and *del* list, respectively:

- A predicate p such that ($p \in relPre_{a_i}$, $p \in relPre_{a_j}$) added by the first action ($p \in add_i$), which serves as a prerequisite for the second action a_j ($p \in pre_j$), cannot be deleted by the first action a_i.
- A relevant predicate p ($p \in relPre_{a_i}$, $p \in relPre_{a_j}$) added by the first action a_i also appears in the *pre* list of the second action a_j.
- A predicate p that is deleted by the first action a_i is added by the second action a_j.
- The above plan constraints can be combined into one constraint and restated as: $\exists p((p \in (pre_i \cap pre_j) \wedge p \notin (del_i)) \vee (p \in (add_i \cap pre_j)) \vee (p \in (del_i \cap add_j)))$

Long-Term Constraints We introduce this set of constraints to explore the relationships between a chain of actions constituting a plan.

- If a predicate p is observed to be true for the last action a_n of a plan sequence and p is a relevant predicate to $a_1, \ldots, a_i, \ldots, a_n$ where $0 \leq i < n$, then the predicate p must exist in the *add* list of a_i. This can be expressed as $p \in add_{a_1} \vee add_{a_2} \vee \ldots \vee add_{a_{n-1}}$.
- Predicates constituting the initial state of the plan are preconditions of the first executed action in the plan [10].
- If a predicate p is observed to be true in the intermediate states right before an action a_k of a plan sequence, and p is a relevant predicate to $a_{k+1}, \ldots, a_n$, then the predicate p must serve as a precursor to these following actions. This can be expressed as $p \in (pre_{a_{k+1}} \vee pre_{a_{k+2}} \vee \ldots \vee pre_{a_n})$.

5 Evaluation

We evaluate the accuracy of SPMSAT. For this, we construct a CNF formula consisting of a conjunction of the hard and soft constraints generated in Phase 2, each associated

with a specific weight. The weights of the hard constraints and the long term constraints are "hyperparameters" which must be continuously tweaked and fine tuned to obtain the most accurate model. The weight of the short term constraints are equivalent to the support of the rules which are associated with the frequent action pairs obtained with the TRuleGrowth algorithm. The support is chosen because it is an indicator of the frequency of the action pair over the entire trace set. This CNF formula is then fed to a weighted MAX-SAT solver which finds a truth assignment that maximizes the total weight of the satisfied constraints, thus producing as output the constraints which evaluate to true. The true constraints are used to reconstruct the entire model, termed as the empirical model. This model is compared with artificial models which are considered as the ground truth. Let $diffpre_{a_i}$ represent the syntactic difference in *pre* lists of action a_i in the ground truth model and the empirical model. Each time the *pre* list of the ideal model presents a predicate which is not in the *pre* list of the empirical model, the count $diffpre_{a_i}$ is incremented by one. Similarly, each time the *pre* list of the empirical model presents a predicate which is not in the *pre* list of the ideal model, the count $diffpre_{a_i}$ is incremented by one. Similar counts are estimated for the *add* and *del* lists as $diffadd_{a_i}$ and $diffdel_{a_i}$ respectively. This total count is then divided by the number of relevant constraints for that particular action $relCons_{a_i}$ to obtain the cumulative error per action. This error is summed up for every action and averaged over the number of actions of the model to obtain an average error E for the entire model.

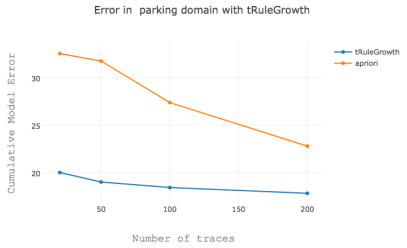

Fig. 3: Error comparison in *parking* domain

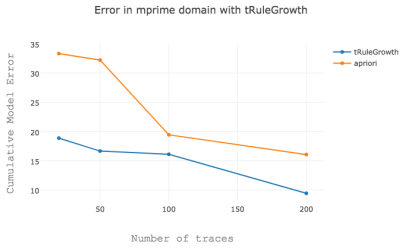

Fig. 4: Error comparison in *mprime* domain

The cumulative error for the model is thus represented by:

$$E = \frac{1}{n}\sum_{i=1}^{n}\frac{diffPre_{a_i} + diffAdd_{a_i} + diffDel_{a_i}}{relCons_{a_i}} \tag{1}$$

We evaluate the performance of SRMLearn with our implementation of ARMS over five domains as follows: for each of the domains, we set the number of traces as (20, 50, 100, 200), generate and solve constraints with the help of two SAT solvers ([2, 8]), and calculate the cumulative error for SRMLearn and ARMS. It should be noted that the key difference between SRMLearn and ARMS lies in SRMLearn's additional

long term constraints and chosen data mining algorithm (TRuleGrowth). The evaluation domains include *depots, parking, mprime, gripper* and *satellite* and are represented in the figures 3, 4, 5, 6 and 7. As can be seen, the error percentages with SRMLearn are lower than with ARMS, indicating that frequent sequential action pairs mined with the TRuleGrowth algorithm demonstrate a stronger correlation than those mined with the Apriori algorithm.

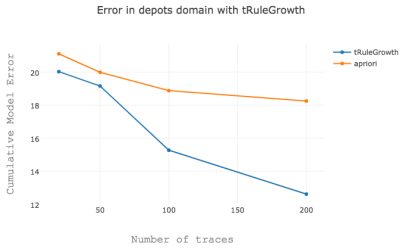

Fig. 5: Error comparison in *depots* domain

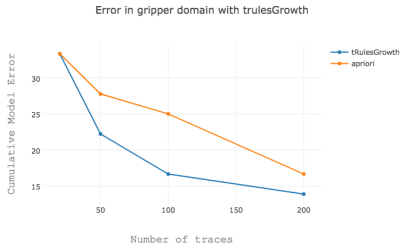

Fig. 6: Error comparison in *gripper* domain

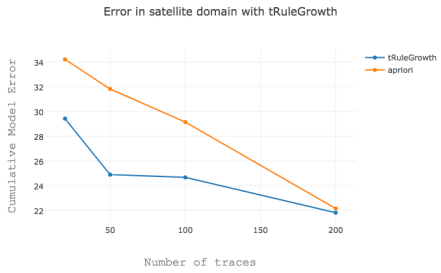

Fig. 7: Error comparison in *satellite* domain

6 Conclusion

This paper presents an approach called SRMLearn which learns the agents' action model by encoding the inter and intra-action dependencies in the form of a maximum satisfiability problem (MAX-SAT) and solves it with a weighted MAX-SAT solver to reconstruct the underlying model. Experimental results reinforce our hypothesis that exploiting relationships between consecutive actions improves the learning accuracy. In future work, we intend to extend SRMLearn to learn temporal action models comprising durative actions.

References

1. Agrawal, R. and Srikant, R., 1994. Fast algorithms for mining association rules. In VLDB, Vol. 1215, pp. 487-499.
2. Borchers, B. and Furman, J., 1998. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. Journal of Combinatorial Optimization, 2(4), pp. 299-306.
3. Fikes, R.E. and Nilsson, N.J., 1971. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2(3-4), pp. 189-208.
4. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W. and Tseng, V.S., 2014. SPMF: a Java open-source pattern mining library. Journal of Machine Learning Research, 15(1), pp. 3389-3393.
5. Fournier-Viger, P., Wu, C.W., Tseng, V.S. and Nkambou, R., 2012. Mining sequential rules common to several sequences with the window size constraint. In Canadian Conference on Artificial Intelligence, pp. 299-304.
6. García-Martínez, R. and Borrajo, D., 2000. An integrated approach of learning, planning, and execution. Journal of Intelligent & Robotic Systems, 29(1), pp.47-78.
7. Jiménez, S., Fernández, F., and Borrajo, D., 2008. The PELA architecture: integrating planning and learning to improve execution. In AAAI.
8. Kautz, H.A., Selman, B. and Jiang, Y., 1996. A general stochastic approach to solving problems with hard and soft constraints. Satisfiability Problem: Theory and Applications, 35, pp. 573-586.
9. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D. and Wilkins, D., 1998. PDDL-the planning domain definition language.
10. Yang, Q., Wu, K. and Jiang, Y., 2007. Learning action models from plan examples using weighted MAX-SAT. Artificial Intelligence, 171(2-3), pp. 107-143.
11. Yoon, S. and Kambhampati, S., 2007. Towards model-lite planning: A proposal for learning & planning with incomplete domain models. In ICAPS Workshop on Artificial Intelligence Planning and Learning.
12. Zhuo, H.H., Nguyen, T.A. and Kambhampati, S., 2013, August. Refining Incomplete Planning Domain Models Through Plan Traces. In IJCAI, pp. 2451-2458.