Available online at www.sciencedirect.com

ScienceDirect

Artificial Intelligence 00 (2018) 1–7

Artificial Intelligence

Learning STRIPS action models with classical planning

Diego Aineto[a], Sergio Jiménez Celorrio[a], Eva Onaindia[a]

[a]*Department of Computer Systems and Computation, Universitat Politcnica de Valncia. Spain*

Abstract

This paper presents a novel approach for learning STRIPS action models from examples of plan executions that compiles this learning task into classical planning. The compilation approach is flexible to various amount and forms of available input knowledge; the learning examples can range from a set of plans (with their corresponding initial and final states) or sequences of state observations to just a set of initial and final states (where no intermediate action or state is given). The compilation accepts also partially specified action models and can be used to validate whether the observation of a plan execution follows a given STRIPS action model, even if this model is not fully specified. What is more, the compilation is extensible to assess how well a STRIPS action model matches a given set of observations. Last but not least, the paper evaluates the performance of the compilation approach by learning action models for a wide range of classical planning domains from the International Planning Competition (IPC) and assessing the learned models with respect to (1), test sets of observations and (2), the true models.

Keywords: Classical planning, Planning and learning, Learning action models, Generalized planning

1. Introduction

Besides *plan synthesis* [1], planning action models are also useful for *plan/goal recognition* [2]. At both planning tasks, an automated planner is required to reason about action models that correctly and completely capture the possible world transitions [3]. Unfortunately, building planning action models is complex, even for planning experts, and this knowledge acquisition task is a bottleneck that limits the potential of AI planning [4].

On the other hand, Machine Learning (ML) has shown to be able to compute a wide range of different kinds of models from examples [5]. The application of inductive ML to the learning of STRIPS action models, the vanilla action model for planning [6], is not straightforward though:

- The *input* to ML algorithms (the learning/training data) usually are finite vectors encoding the value of fixed features in a given set of objects. The input for learning planning action models are observations of plan executions (where each plan possibly has a different length).
- The *output* of ML algorithms usually is a scalar value (an integer, in the case of *classification* tasks, or a real value, in the case of *regression* tasks). When learning STRIPS action models the output is, for each action, the sets of preconditions, negative and positive effects, that define the possible state transitions.

Motivated by recent advances on the synthesis of different kinds of generative models with classical planning [7, 8, 9], this paper introduces an innovative approach for learning STRIPS action models that can be defined as a classical planning compilation. A solution to the classical planning task that results from our compilation is a sequence of actions that determines the learned action model, i.e. the preconditions and effects of the target STRIPS operator schemas.

The compilation approach is appealing by itself, because leverages off-the-shelf planners and opens up a way towards *bootstrapping* planning action models, enabling a planner to gradually learn/update its action model. The practicality of the compilation approach allow us to report learning results over fifteen IPC planning domains. In addition,the compilation approach presents the following features:

1. Is flexible to various amounts and forms of available input knowledge. Learning examples can range from a set of plans (with their corresponding initial and final states) or state observations to just a set of initial and final states where no intermediate state or action is observed. Learning from state observations is a relevant advancement as, in many applications, the actual actions executed by the observed agent are not available but, instead, the resulting states can be observed. Learning action models from state observations broadens the range of application to external observers and facilitates the representation of imperfect observability, as shown in plan recognition [10], as well as learning from unstructured data, like state images [11]).
2. Accepts previous knowledge about the structure of the actions in the form of partially specified action models. In the extreme, the compilation can validate whether an observed plan execution is valid for a given STRIPS action model, even if this model is not fully specified. For the training samples, we adopt a middle ground between unstructured inputs and plan traces, wherein only state observations are required.
3. Assess how well a STRIPS action model matches a given set of observations. Our compilation is extensible to accept a learned model as input besides the state observations. This extension allows us to transform the input model into a new model that induces the observations whilst assessing the amount of edition required by the input model to induce the given observations. The empirical evaluation of our learning approach is two-fold: First the learned STRIPS action models are tested with a set of state observation sequences and second, the learned models are compared to the corresponding reference model.

Section 2 reviews related work on learning planning action models. Section 3 formalizes the classical planning model with *conditional effects* (a requirement of the proposed compilation) and the STRIPS action model (the output of the addressed learning task). Section 4 formalizes the learning of STRIPS action models with regard to different amounts of available input knowledge. Sections describe our compilation approach for addressing the formalized learning tasks. Finally, the last sections report the data collected in a empirical evaluation, discuss the strengths and weaknesses of the compilation approach and propose several opportunities for future research.

2. Related work

Back in the 90's various systems aimed learning operators mostly via interaction with the environment. LIVE captured and formulated observable features of objects and used them to acquire and refine operators [12]. OBSERVER updated preconditions and effects by removing and adding facts, respectively, accordingly to observations [13]. These early works were based on lifting the observed states supported by exploratory plans or external teachers, but none provided a theoretical justification for this second source of knowledge.

More recent work on learning planning action models [14] shows that although learning STRIPS operators from pure interaction with the environment requires an exponential number of samples, access to an external teacher can provide solution traces on demand.

Whilst the aforementioned works deal with full state observability,action model learning has also been studied in domains where there is partial or missing state observability. ARMS works when no partial intermediate state is given. It defines a set of weighted constraints that must hold for the plans to be correct, and solves the weighted propositional satisfiability problem with a MAX-SAT solver [15]. In order to efficiently solve the large MAX-SAT representations, ARMS implements a hill-climbing method that models the actions approximately. SLAF also deals with partial observability [16]. Given a formula representing the initial belief state, a sequence of executed actions and the corresponding partially observed states, it builds a complete explanation of observations by models of actions through a CNF formula. The learning algorithm updates the formula of the belief state with every action and observation in the sequence. This update makes sure that the new formula represents all the transition relations consistent with the actions and observations. The formula returned at the end includes all consistent models, which can then be retrieved with additional processing.

Unlike the previous approaches, the one described in [17] deals with both missing and noisy predicates in the observations. An action model is first learnt by constructing a set of kernel classifiers which tolerate noise and partial observability and then STRIPSrules are derived from the classifiers' parameters.

LOCM only requires the example plans as input without need for providing information about predicates or states [18]. This makes LOCM be most likely the learning approach that works with the least information possible. The lack of available information is addressed by LOCM by exploiting assumptions about the kind of domain model it has to generate. Particularly, it assumes a domain consists of a collection of objects (sorts) whose defined set of states can be captured by a parameterized Finite State Machine (FSM). The intuitive assumptions of LOCM , like the continuity of object transitions or the association of parameters between consecutive actions in the training sequence, yield a learning model heavily reliant on the kind of domain structure. The inability of LOCM to properly derive domain theories where the state of a sort is subject to different FSMs is later overcome by LOCM2 by forming separate FSMs, each containing a subset of the full transition set for the sort [19]. LOP (LOCM with Optimized Plans [20]), the last contribution of the LOCM family, addresses the problem of inducing static predicates. Because LOCM approaches induce similar models for domains with similar structures, they face problems at generating models for domains that are only distinguished by whether or not they contain static relations (e.g. *blocksworld* and *freecell*). In order to mitigate this drawback, LOP applies a post-processing step after the LOCM analysis which requires additional information about the plans, namely a set of optimal plans to be used in the learning phase.

Compiling the learning of action models into classical planning is a general and flexible approach that allows to accommodate various amounts and kinds of input knowledge and opens up a path for addressing further learning and validation tasks. For instance, the example plans in Π could be replaced or complemented by a set $\mathcal{O}$ of sequences of observations (i.e., fully or partial state observations with noisy or missing fluents [10]), so learning tasks $\Lambda = \langle \Psi, \Sigma, \mathcal{O}, \Xi_0 \rangle$ could also be addressed. Furthermore, our approach seems extensible to learning other types of generative models (e.g. hierarchical models like HTN or behaviour trees), that can be more appealing than STRIPS models, since using them to compute planning solutions requires less search effort.

3. Background

This section defines the planning model used on this work and the output of the learning tasks addressed in the paper.

3.1. Classical planning with conditional effects

We use F to denote the set of *fluents* (propositional variables) describing a state. A *literal* l is a valuation of a fluent $f \in F$, i.e. either $l = f$ or $l = \neg f$. A set of literals L represents a partial assignment of values to fluents (without loss of generality, we will assume that L does not assign conflicting values to any fluent). We use $\mathcal{L}(F)$ to denote the set of all literal sets on F, i.e. all partial assignments of values to fluents.

A *state* s is a full assignment of values to fluents, i.e. $|s| = |F|$, so the size of the state space is $2^{|F|}$. Explicitly including negative literals $\neg f$ in states simplifies subsequent definitions but often, we will abuse notation by defining a state s only in terms of the fluents that are true in s, as is common in STRIPS planning.

A *classical planning frame* is a tuple $\Phi = \langle F, A \rangle$, where F is a set of fluents and A is a set of actions. An action $a \in A$ is defined with *preconditions*, $\mathsf{pre}(a) \subseteq \mathcal{L}(F)$, *positive effects*, $\mathsf{eff}^+(a) \subseteq \mathcal{L}(F)$, and *negative effects* $\mathsf{eff}^-(a) \subseteq \mathcal{L}(F)$. We say that an action $a \in A$ is *applicable* in a state s iff $\mathsf{pre}(a) \subseteq s$. The result of applying a in s is the *successor state* denoted by $\theta(s, a) = \{s \setminus \mathsf{eff}^-(a)) \cup \mathsf{eff}^+(a)\}$.

An action $a \in A$ with conditional effects is defined as a set of *preconditions* $\mathsf{pre}(a) \in \mathcal{L}(F)$ and a set of *conditional effects* $\mathsf{cond}(a)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a)$ is composed of two sets of literals $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*. An action $a \in A$ is *applicable* in a state s if and only if $\mathsf{pre}(a) \subseteq s$, and the *triggered effects* resulting from the action application are the effects whose conditions hold in s:

$$triggered(s, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a), C \subseteq s} E,$$

The result of applying action a in state s is the *successor* state $\theta(s, a) = \{s \setminus \mathsf{eff}_c^-(s, a)) \cup \mathsf{eff}_c^+(s, a)\}$ where $\mathsf{eff}_c^-(s, a) \subseteq triggered(s, a)$ and $\mathsf{eff}_c^+(s, a) \subseteq triggered(s, a)$ are, respectively, the triggered *negative* and *positive* effects.

```
(:action stack
 :parameters (?v1 ?v2 - object)
 :precondition (and (holding ?v1) (clear ?v2))
 :effect (and (not (holding ?v1))
              (not (clear ?v2))
              (handempty) (clear ?v1)
              (on ?v1 ?v2)))
```

Figure 1: STRIPS operator schema coding, in PDDL, the *stack* action from a four-operator *blocksworld*.

A *classical planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where I is an initial state and $G \subseteq \mathcal{L}(F)$ is a goal condition. A *plan* for P is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$ that induces the *state trajectory* $\langle s_0, s_1, \ldots, s_n\rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, a_i is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$. The *plan length* is denoted with $|\pi| = n$. A plan π *solves* P iff $G \subseteq s_n$, i.e. if the goal condition is satisfied at the last state reached after following the application of the plan π in the initial state I.

3.2. STRIPS *action schemas*

This work addresses the learning of PDDL action schemas that follow the STRIPS requirement [21, 22]. Figure 1 shows the *stack* action schema, coded in PDDL, from a four-operator *blocksworld* [23].

To formalize the output of the learning task, we assume that fluents F are instantiated from a set of *predicates* Ψ, as in PDDL. Each predicate $p \in \Psi$ has an argument list of arity $ar(p)$. Given a set of *objects* Ω, the set of fluents F is induced by assigning objects in Ω to the arguments of predicates in Ψ, i.e. $F = \{p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}\}$ s.t. Ω^k is the k-th Cartesian power of Ω.

Let $\Omega_v = \{v_i\}_{i=1}^{\max_{a\in A} ar(a)}$ be a new set of objects ($\Omega \cap \Omega_v = \emptyset$), denoted as *variable names*, and that is bound by the maximum arity of an action in a given planning frame. For instance, in a three-block *blocksworld* $\Omega = \{block_1, block_2, block_3\}$ while $\Omega_v = \{v_1, v_2\}$ because the operators with the maximum arity, `stack` and `unstack`, have arity two. We define F_v, a new set of fluents s.t. $F \cap F_v = \emptyset$, that results from instantiating Ψ using only the objects in Ω_v and defines the elements that can appear in an action schema. For the *blocksworld*, F_v=\{`handempty`, `holding(`v_1`)`, `holding(`v_2`)`, `clear(`v_1`)`, `clear(`v_2`)`, `ontable(`v_1`)`, `ontable(`v_2`)`, `on(`v_1,v_1`)`, `on(`v_1,v_2`)`, `on(`v_2,v_1`)`, `on(`v_2,v_2`)`\}.

We assume also that actions $a \in A$ are instantiated from STRIPS operator schemas $\xi = \langle head(\xi), pre(\xi), add(\xi), del(\xi)\rangle$ where:

- $head(\xi) = \langle name(\xi), pars(\xi)\rangle$, is the operator *header* defined by its name and the corresponding *variable names*, $pars(\xi) = \{v_i\}_{i=1}^{ar(\xi)}$. The headers of a four-operator *blocksworld* are `pickup(`v_1`)`, `putdown(`v_1`)`, `stack(`v_1,v_2`)` and `unstack(`v_1,v_2`)`.
- The preconditions $pre(\xi) \subseteq F_v$, the negative effects $del(\xi) \subseteq F_v$, and the positive effects $add(\xi) \subseteq F_v$ such that, $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$.

Finally, given the set of predicates Ψ and the header of a STRIPS operator schema ξ, we define $F_v(\xi) \subseteq F_v$ as the subset of elements that can appear in the action schema ξ and that confine its space of possible action models. For instance, for the *stack* action schema F_v(`stack`) $= F_v$ while F_v(`pickup`)=\{`handempty`, `holding(`v_1`)`, `clear(`v_1`)`, `ontable(`v_1`)`, `on(`v_1,v_1`)`\} excludes the fluents from F_v that involve v_2 because the action header `pickup(`v_1`)` contains the single parameter v_1.

4. Learning STRIPS action models

Learning STRIPS action models from fully available input knowledge, i.e. from plans where the *pre-* and *post-states* of every action in a plan are available, is straightforward. When any intermediate state is available, STRIPS operator schemes are derived lifting the literals that change between the pre and post-state of the corresponding action executions. Preconditions are derived lifting the minimal set of literals that appears in all the pre-states of the corresponding actions [24]. This section formalizes a set of more challenging action model learning tasks, where less input knowledge is available.

4.1. *Learning from state observations*

This learning task corresponds to observing an agent acting in the world but watching only the results of its plan executions, the actual executed actions are unobserved. This learning task is formalized as $\Lambda = \langle \mathcal{M}, \Psi, \sigma \rangle$, when no intermediate information is given:

- $\mathcal{M}$ is the set of *empty* operator schemas, wherein each $\xi \in \mathcal{M}$ is only composed of *head*(ξ). In some cases, we may not require to start learning from scratch that is, the operator schemas in $\mathcal{M}$ may be not *empty* but partially specified operator schemes where some preconditions and effects are a priori known.
- Ψ is the set of predicates that define the abstract state space of a given planning domain.
- $\sigma = (s_0, s_n)$ is a $(initial, final)$ state pair, that we call *label*. the *final* state s_n resulting from executing an unknown plan $\pi_t = \langle a_1, \ldots, a_n \rangle$ starting from the *initial* state s_0. When the intermediate states are available, the learning task is defined as $\Lambda = \langle \mathcal{M}, \Psi, O \rangle$, where $O = \langle s_0, s_1, \ldots, s_n \rangle$ is a sequence of *state observations* obtained observing the execution of an *unobserved* plan $\pi = \langle a_1, \ldots, a_n \rangle$.

A solution is a set of operator schema $\mathcal{M}'$ compliant with the headers in $\mathcal{M}$, the predicates Ψ, and the state observation sequence O. In this learning scenario, a solution must not only determine a possible STRIPS action model but also the plan π, that explain the given observations using the learned STRIPS model. Figure 3 shows a $\Lambda = \langle \mathcal{M}, \Psi, O \rangle$ task for learning the *blocksworld* STRIPS action model from the five-state observations sequence that corresponds to inverting a 2-block tower.

```
;;;;;; Headers in M

(pickup v1) (putdown v1)
(stack v1 v2} (unstack v1 v2)


;;;;;; Predicates Ψ

(handempty) (holding ?o  - object)
(clear ?o - object) (ontable ?o - object)
(on ?o1 - object ?o2 - object)


;;;;;; Observations O

;;; observation #0
(clear B) (on B A) (ontable A) (handempty)

;;; observation #1
(holding B) (clear A) (ontable A)

;;; observation #2
(clear A) (ontable A) (clear B) (ontable B) (handempty)

;;; observation #3
(holding A) (clear B) (ontable B)

;;; observation #4
(clear A) (on blockA B) (ontable B) (handempty)
```

Figure 2: Example of a $\Lambda = \langle \mathcal{M}, \Psi, O \rangle$ task for learning a STRIPS action model in the *blocksworld* from a sequence of five state observations.

4.2. *Learning from a labeled plan*

Here we augment the input knowledge with the actions executed by the observed agent and define the learning task $\Lambda = \langle \mathcal{M}, \Psi, \sigma, \pi \rangle$:

```
;;; Predicates in Ψ

(handempty) (holding ?o  - object)
(clear ?o - object) (ontable ?o - object)
(on ?o1 - object ?o2 - object)
```

```
;;; Plan π

0: (unstack A B)
1: (putdown A)
2: (unstack B C)
3: (stack B A)
4: (unstack C D)
5: (stack C B)
6: (pickup D)
7: (stack D C)
```

;;; Label $\sigma = (s_0^1, s_n^1)$

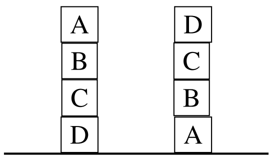

Figure 3: Example of a task for learning a STRIPS action model in the blocksworld from a labeled plan.

- The plan $\pi = \langle a_1, \ldots, a_n \rangle$, is an action sequence that induces the corresponding state sequence $\langle s_0, s_1, \ldots, s_n \rangle$ such that, for each $1 \leq i \leq n$, a_i is applicable in s_{i-1} and generates $s_i = \theta(s_{i-1}, a_i)$.

Figure 3 shows an example of a *blocksworld* learning task $\Lambda = \langle \mathcal{M}, \Psi, \sigma, \pi \rangle$, that corresponds to observing the execution of an eight-action plan for inverting a four-block tower.

4.3. Learning from multiple plans

The previous task define the learning of planning action models from a single plan execution. These definitions can be extended to the more general case where learning from multiple plans:

- $\Lambda = \langle \mathcal{M}, \Psi, \Sigma \rangle$ where $\Sigma = \{\sigma_1, \ldots, \sigma_\tau\}$ is a set of (*initial*, *final*) state pairs, that we call *labels*. Each label $\sigma_t = (s_0^t, s_n^t)$, $1 \leq t \leq \tau$, comprises the *final* state s_n^t resulting from executing an unknown plan $\pi_t = \langle a_1^t, \ldots, a_n^t \rangle$ starting from the *initial* state s_0^t.
- $\Lambda = \langle \mathcal{M}, \Psi, \Sigma, \Pi \rangle$ where $\Pi = \{\pi_1, \ldots, \pi_\tau\}$ is a given set of example plans where each plan $\pi_t = \langle a_1^t, \ldots, a_n^t \rangle$, $1 \leq t \leq \tau$, is an action sequence that induces the corresponding state sequence $\langle s_0^t, s_1^t, \ldots, s_n^t \rangle$ such that, for each $1 \leq i \leq n$, a_i^t is applicable in s_{i-1}^t and generates $s_i^t = \theta(s_{i-1}^t, a_i^t)$.

5. Evaluation

6. Conclussions

Acknowledgment

This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R. Diego Aineto is partially supported by the *FPU16/03184* and Sergio Jiménez by the *RYC15/18009*, both programs funded by the Spanish government.

References

[1] M. Ghallab, D. Nau, P. Traverso, Automated Planning: theory and practice, Elsevier, 2004.
[2] M. Ramírez, Plan recognition as planning, Ph.D. thesis, Universitat Pompeu Fabra (2012).
[3] H. Geffner, B. Bonet, A concise introduction to models and methods for automated planning (2013).
[4] S. Kambhampati, Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models, in: National Conference on Artificial Intelligence, AAAI-07, 2007.
[5] R. S. Michalski, J. G. Carbonell, T. M. Mitchell, Machine learning: An artificial intelligence approach, Springer Science & Business Media, 2013.
[6] R. E. Fikes, N. J. Nilsson, Strips: A new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (3-4) (1971) 189–208.

[7] B. Bonet, H. Palacios, H. Geffner, Automatic derivation of memoryless policies and finite-state controllers using classical planners., in: International Conference on Automated Planning and Scheduling (ICAPS), 2009.

[8] J. Segovia-Aguas, S. Jiménez, A. Jonsson, Hierarchical finite state controllers for generalized planning, in: International Joint Conference on Artificial Intelligence, IJCAI-16, AAAI Press, 2016, pp. 3235–3241.

[9] J. Segovia-Aguas, S. Jiménez, A. Jonsson, Generating context-free grammars using classical planning, in: International Joint Conference on Artificial Intelligence, ICAPS-17, 2017.

[10] S. Sohrabi, A. V. Riabov, O. Udrea, Plan recognition as planning revisited, in: International Joint Conference on Artificial Intelligence, IJCAI-16, 2016, pp. 3258–3264.

[11] M. Asai, A. Fugunaga, Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary, in: National Conference on Artificial Intelligence, AAAI-18, 2018.

[12] W. Shen, H. A. Simon, Rule creation and rule learning through environmental exploration, in: International Joint Conference on Artificial Intelligence, IJCAI-89, 1989, pp. 675–680.

[13] X. Wang, Learning by observation and practice: An incremental approach for planning operator acquisition, in: International Conference on Machine Learning, 1995, pp. 549–557.

[14] T. J. Walsh, M. L. Littman, Efficient learning of action schemas and web-service descriptions, in: National Conference on Artificial Intelligence, 2008, pp. 714–719.

[15] Q. Yang, K. Wu, Y. Jiang, Learning action models from plan examples using weighted max-sat, Artificial Intelligence 171 (2-3) (2007) 107–143.

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

[17] K. Mourão, L. S. Zettlemoyer, R. P. A. Petrick, M. Steedman, Learning STRIPS operators from noisy and incomplete observations, in: Conference on Uncertainty in Artificial Intelligence (UAI), 2012, pp. 614–623.

[18] S. N. Cresswell, T. L. McCluskey, M. M. West, Acquiring planning domain models using LOCM, The Knowledge Engineering Review 28 (02) (2013) 195–213.

[19] S. Cresswell, P. Gregory, Generalised domain model acquisition from action traces, in: International Conference on Automated Planning and Scheduling, ICAPS-11, 2011.

[20] P. Gregory, S. Cresswell, Domain model acquisition in the presence of static relations in the LOP system., in: International Conference on Automated Planning and Scheduling, ICAPS-15, 2015, pp. 97–105.

[21] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL – The Planning Domain Definition Language (1998).

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

[23] J. Slaney, S. Thiébaux, Blocks world revisited, Artificial Intelligence 125 (1-2) (2001) 119–153.

[24] S. Jiménez, T. De La Rosa, S. Fernández, F. Fernández, D. Borrajo, A review of machine learning for automated planning, The Knowledge Engineering Review 27 (04) (2012) 433–467.