Chapter 2
Automated Domain Model Learning Tools for Planning

Rabia Jilani

Abstract Intelligent agents solving problems in the real world require domain models containing widespread knowledge of the world. Domain models can be encoded by human experts or automatically learned through the observation of some existing plans (behaviours). Encoding a domain model manually from experience and intuition is a very complex and time-consuming task, even for domain experts. This chapter investigates various classical and state-of-the-art methods proposed by the researchers to attain the ability of automatic learning of domain models from training data. This concerns with the learning and representation of knowledge about the operator schema, discrete or continuous resources, processes and events involved in the planning domain model. The taxonomy and order of these methods we followed are based on their standing and frequency of usage in the past research. Our intended contribution in this chapter is to provide a broader perspective on the range of techniques in the domain model learning area which underpin the developmental decisions of the learning tools.

1 Introduction

Automated planning is one of the most prominent AI challenges. It is the process of finding a procedural course of action through explicit deliberation process to reach a pre-stated objective in the form of goals while optimizing overall performance. Planning is a pivotal task that has to be performed by autonomous agents. The planning community is uplifting planning systems from small problems to capture more complex domains that closely reflect real life applications (e.g., planning space missions, fire extinction management and operation of underwater vehicles)—a way to satisfy the aims of autonomic systems.

R. Jilani (✉)
School of Computing and Engineering, University of Huddersfield, Huddersfield, UK
e-mail: R.Jilani@hud.ac.uk

M. Vallati, D. Kitchin (eds.), *Knowledge Engineering Tools and Techniques for AI Planning*, https://doi.org/10.1007/978-3-030-38561-3_2

In order to perform automated reasoning, planning techniques require formal specification of the application knowledge to be encoded in the form of domain models. In the action-centred view of problem representation, a domain model encodes the domain knowledge in the form of actions that can be executed together with relevant action properties and features. A domain model typically includes the action description, the objects involved in actions, a set of logical state space axioms along with the rules of inference and heuristics to accurately define the operators' description of some real-world domain. It includes both a dynamic and a static object-type hierarchy, constant objects (if any) and the declaration of predicates and functions (for hierarchical domains). In short, it is a declarative depiction of domain world functionalities.

In a centralised approach, this domain model is represented as a knowledge base and automated logical reasoning could be used to determine acts in plans. To generate plans, planning engines search the action descriptions in the provided domain model to achieve the goals. Figure 2.1 shows a typical view of plan generation in AI planning. One of the key factors for the correctness of the planner outcome is the quality of the domain knowledge that otherwise can prove catastrophic.

In the complex domain scenarios, planners also use manually encoded or automatically learned domain-specific control knowledge (in addition to domain model) to guide planner search and cater to scalability issues. Most planners define control knowledge separately from domain model to support different representations. This chapter only focuses on the domain model learning aspect.

Synthesising domain models for planning from scratch by hand is time intense, error-prone and challenging. Knowledge engineering for planning domain models using machine learning (ML) techniques is considered as a paramount for empowering the autonomous learning systems with the capacity to fill implicit human knowledge gaps and errors, requiring least human intervention in domain model development. It not only involves acquisition of a new knowledge from the environment but also the refinement of the already available knowledge of the domain under consideration.

As a result of a planning process the successfully generated plans can be used as the solutions to the desired problems in self-learning systems to enable autonomic properties. The area of ML application to domain model learning systems has received active research attention in recent years but did not make as much stride as the learning of control knowledge.

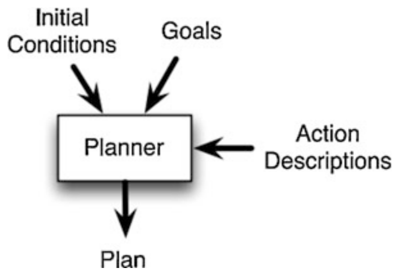

Fig. 2.1 Automated planning as an independent component

1.1 Knowledge Representation for Knowledge Engineering of Domain Models

Knowledge representation (KR) is to encoding human knowledge in the form of symbols which can be processed by a computer to obtain intelligent behaviour. Automated reasoning and KR stay in close association with each other as the core aim of the explicit KR is to enable reasoning and inference process. To represent complex problems, KR uses declarative programming for expressing the logic of the computation and behavioural description as compared to describing the control flow as in procedural programming.

In order to effectively engage in the intelligent behaviours, the key attribute of the theories of autonomous agent relies on the agent's internal representation of intelligence. This must be an implicit representation of a domain model to conduct reasoning process. From the point of view of knowledge engineering of this intelligent behaviour, the question to ask is, what KR formalism an agent needs to know to behave intelligently and which computational mechanisms are needed for manipulation of its knowledge, i.e., description of notions, facts, and rules of the world. The knowledge engineering of a domain model is the engineering of a set of sentences/axioms. These sentences are expressed in some KR language which the agent uses to do inference.

For automated reasoning, expressibility and practicability are generally the two main considerations for the KR language of the domain. Riddle et al. [59] empirically proved that a used representation mechanism makes an extensive difference to the planner's ability to solve a problem by exploring six different representations of the blocks world domain. In Brachman and Levesque [7] the authors argue that the expressive power of the representation language is directly proportional to the computational complexity of reasoning with it. In other words, more expressive language makes the reasoning process difficult. The authors demonstrated this by analysing the frame language which later led to an extensive study of the argument put forward by the author in order to search the optimal trade-off.

In automated planning, the key purpose of an explicit KR language for a domain model formulation is for a planner to be able to reason with it and infer new knowledge from it in the form of plans by predicting action outcomes to display some rational behaviour in an explicit way. The essential part of a domain model in the process of reasoning is the representation of the set of actions that a planner can reason with and the elements that include dynamics of the environment that support the specification of actions. It has long been recognised that there can be a variety of encodings and exploitable languages for a domain model formulation. However, the open question is which of these is the best? The choice of encoding language for KR partially depends on the requirements of the planning application itself.

There is no one KR approach just like the reasoning approach that has combined properties for all types and level of deliberation problems. Similarly, there is no single highly specialized KR mechanism to cover a specialized area of learning domain model.

A well-chosen representation language should explicitly model every action effect the system might confront. In addition to that, a domain modelling representation language should have some salient attributes. It should have supporting tools to check its operation and have logically strong inference mechanism to carry out the reasoning. It should be sufficiently expressive to explicitly model complex scenario of the real world. Moreover, it should be customizable and structured to capture every action effect the system might confront in operator definitions. In addition, it should have clear syntax and semantics to support operational aspects of the model.

2 Domain Model Learning Techniques and Tools

Both knowledge acquisition and learning for AI planning systems are essential to improve their effectiveness and to expand the application focus in practice. Most of the literature on learning for AI planning is based upon classical planning and concentrates on the learning of search control rules. For producing a domain model, a general process includes the study of planning application requirements, creating a model that explains the domain and testing it with suitable planning engines. Domain models can be encoded by the human experts or automatically learned through the observation of some existing plans (behaviours). Encoding a domain model manually from experience and intuition is a very complex and time-consuming task, even for domain experts.

Regarding the significance of automatic domain model learning system, the question that arises is why do we need a learning mechanism to learn from data when we can write a program to fulfil the purpose. The significance of learning mechanism becomes apparent when the same program parameters do not fit the new data or when the learning requirements or assumptions of the same data change slightly. The same hard-coded program needs extensive changes to align with the new requirements.

Machine learning is a broad area with a wide variety of sub-fields. It includes various methods starting from sub-symbolic methods like neural networks to high-level symbolic methods like inductive logic programming. Various approaches and techniques have been used by the researchers for the domain model learning task. Inductive learning is the most common technique to expedite learning solutions that are used in the field of supervised learning. Another less common technique outside the scope of supervised learning is the model-based reinforcement learning algorithms that learn model parameters such as probabilities and rewards, but no algorithm yet can produce states and models from observations and action sequences.

For the sake of succinctness, this section describes the commonly used domain model learning techniques in the literature. These techniques are complete in their own capacity and differ from each other in multiple perspectives including the amount and nature of input required, the extent of learning that takes place in the output, environmental characteristics these can work in, etc. Some of these characteristics are discussed in Sect. 4.

2.1 Inductive Learning

Inductive methods produce general rules by searching statistical correlations and consistencies in the large set of input training data. The main theory and method behind supervised learning is the inductive learning (IL—Fig. 2.2). Inductive learning can be defined as learning by inferring generalised rules from the training data given in the form of input–output example *pairs* $\mathbf{P(x_i,\ y_i)}$. By the input–output pairs, we mean the input samples or examples ($\mathbf{x_i}$) and the relevant output observations or the external feedback to the learning system $(\mathbf{y_i} = \mathbf{f(x_i)})$. External feedback or output observations are the function of the input samples. The input–output example pairs generally establish the intended relation of input and output values. In simple words, IL is also referred to as learning from examples for function induction. The input–output example pairs can be generated by another system, produced by an instructor or a human expert or could be the traces of expert's behaviour. These do not necessarily need to be numbers and can contain either continuous or discrete values in the form of logical sentences. In the pairs, states are represented by the set of features, i.e., by factored representation.

The *main IL problem* is to generalise the input-to-output mapping *candidate function* or *hypothesis* (h) that satisfies input data so it can estimate the *target function* f [85]. *Hypothesis space* $H = (h_1, h_2, h_3, \ldots h_n)$ is a set of all possible approximations of target function that can exist. A further subset of hypothesis space (H) which is consistent with the given input data is called version space. Extending a hypothesis with every example or in other words generalising a hypothesis which should be in close approximation with the target function f is not an easy task especially when hypothesis space is complex.

A hypothesis (h) is tested for its consistency and correctness of generalised results by using extensive example test set—a set distinct from the example training set $\mathbf{E_T}$. Formally, using inductive learning algorithm **IL**, the problem is to find a hypothesis h which is consistent with the set of example pairs $\mathbf{P(x_i,\ y_i)}$, such that:

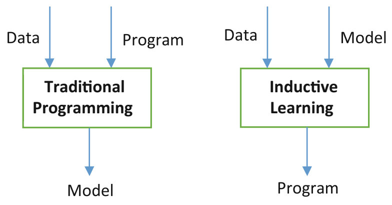

Fig. 2.2 Traditional programming vs. inductive learning

$$\mathbf{IL}\ \boldsymbol{\Lambda}\ \mathbf{P}\,(\mathbf{x_i}, \mathbf{y_i}) \succ \mathbf{h} \qquad \text{(Inductive Learning)}$$

where $\mathbf{y_i} = h(\mathbf{x_i})$ for all P and $\succ$ represents inductive inference. For h to generalise and pass the consistency test, it needs to produce the results like the true target function f by agreeing to the sufficiently large example test set. On getting multiple consistent h in version space, the preference is always given to the simplest h that requires the least speculation and agrees with the data (Ockham's razor principle).

A good example of IL is a classification decision tree algorithm for building prediction models to predict categories from the training dataset. The algorithm has to find a most appropriate tree structure (h) which is consistent with the input data, out of all the available trees with variable attributes in its hypothesis space (H). A decision tree structures like a flow chart where the learning takes place by labelling the variables at nodes and branches of the tree from the training data. Models in which state variables represented in the form of trees can take discrete values are called classification trees. Similarly, trees where state variables can take continuous values are called regression trees. Inducing domain model and its features such as action duration for temporal domains have been well-studied using predictive modelling approaches of a relational decision and regression trees. Inducing regression trees is itself a well-known method for building models for numeric variables.

Artificial neural networks (ANN), reinforcement learning (RL), Bayesian learning (BL) and inductive logic programming (ILP) are some of the most common inductive learning techniques. More common early contributions of these IL techniques since their inception are to learn control knowledge to speed up the classical planning process. An extensive five-dimensional survey of learning-in-planning early work is provided in Zimmerman and Kambhampati [85].

From the domain learning perspective, each of the IL technique produces different types of hypothesis and models based on its expressivity, e.g., BL is used to learn models that allow probabilistic predictions while RL can learn in dynamic and stochastic environmental conditions with no requirement of prior knowledge of transition probabilities. More expressive representation of the learned knowledge which is in close approximation to the target function requires more training examples to narrow down the hypothesis space. Similarly, every technique has its own strengths and weaknesses, e.g., decision trees, ANN and ILP are robust to noisy inputs while RL learns optimal policies even from non-optimal input data.

A number of *properties* that need consideration for developing IL problem include:

- The probability that the training is going to be successful in learning the model. It mainly depends on the quality of training data and also on the inductive bias (discussed ahead in KBIL). Noisy set of training data leads to poor inconsistent models if not given alternate guidance.
- How much input training data should be provided for consistent h to converge? According to computational learning theory, only a few algorithms exhibit this

knowledge through the learning curve with the increase in input examples (PAC learning algorithms).
- Criteria for selection of training data and what presentation medium should be used to learn from available data, i.e., instructions, images, sensory information, environmental perceptions, etc.
- What output examples need to be provided to learn the target function *f* and how it affects the extent and quality of the learnt model in the outcome.
- Estimation of the overall complexity of hypothesis space and the complexity of hypothesis in the space. Complex hypotheses are more susceptible to overfitting.
- What should be the acceptable approximation and estimation error between a consistent hypothesis *h* and a target function *f*.
- What is the size and nature (deterministic or non-deterministic) of the hypothesis space?
- Depending on the nature of the problem, which learning algorithm (e.g., inductive logic programming, Bayesian learning, etc.) and approach to learning will be used (online or batch).
- Approach to use for building hypothesis towards the target function *f*. It could be directly computing the required information to learn the target function, searching for hypothesis from the hypothesis space or the gradual incremental construction of the hypothesis.
- Representation mechanism used to represent learned knowledge. This can include support vector machines, decision trees, graphical models, Bayes networks, finite state machines, logic statements, etc.
- On what time scale learning occurs: eager learners are more common and perform the task up front while lazy learners only learn when needed and are rarely used in machine learning.

Some of the *prominent issues* about the IL that researchers are trying to answer for over a decade include:

- What measures the good hypothesis space and the correctness of the hypothesis *h* if the true function *f* is not known?
- What factors can help reach the trade-off between the complexity of finding consistent *h* and the expressiveness of the hypothesis space? Can we even find *h* in a complex hypothesis space?
- What features and factors build the confidence in the correctness of the output model?
- How to find out computationally complex or intractable problems?

2.1.1 When to Use Inductive Learning

Inductive learning can be used to enhance any essential module or element of the system. It can be used in a variety of situations, some of them include:

1. When the underlying knowledge base or domain model fluctuates frequently and the occurring changes are complex enough that cannot be handled by human efforts every single time in changes, e.g., continuous domains like urban road traffic management, stock market, etc.
2. Situations where the learning only happens with experience and it is not possible to induce learning with a set of instructions, e.g., intelligent self-driving cars [53] where the system requires enormous training data in the form of camera visuals and corresponding steering movements to produce the general rule of driving.
3. Another condition arises when there is no human guidance available to create a reliable domain model, the agent should be intelligent enough to induce the domain model through learning. Building the knowledge base of planetary rovers could be a good example where the lack of reliable, explicit and a priori knowledge can be supported by inductive learning capability of the agent.
4. Last but not the least, situations, where each area of experimentation requires a unique underlying domain model, e.g., all the benchmark domains in International Planning Competition (IPC), require unique domain models and relevant problems to test the efficiency and effectiveness of planning systems.

The strength of the exploited learning approach or algorithm is the key factor to regulate the extent of learning by the system, as it may get stuck in local minima or not be able to capture patterns of the target knowledge within a reasonable time and memory requirements [27]. For example, exploiting reinforcement learning method to learn from a reward-based approach can learn better in a stochastic environment as compared to the inductive learning (which is based on drawing from inference). Similarly, learning for conformant or contingent planning task, the suitable learning approach to adopt is by inference or by inductive generalization to find the best fit for the observed facts. The concept of model-lite [30] planning views a planning problem as an MPE (most plausible explanation) problem. These techniques search for solution plans that are most plausible according to the current domain model, specifically for situations where the first bottleneck is getting the domain model at any level of completeness.

2.2 Knowledge-Based Inductive Learning (KBIL)

Humans learn knowledge with a sequence of experiences and also by reflection on past experiences to facilitate current learning. The challenge for autonomous learning agents is the lack of training examples to reflect on and build the hypothesis. In many learning systems developed since the 1980s, the issue of lacking training data has been covered by the notion of inductive bias (IB) [41]. Mitchell defines inductive bias as the constraint on the hypothesis space (H) of a learning system in addition to the requirement of consistency with the training examples. It is when a learning system prefers one hypothesis over others in hypothesis space. Inductive bias significantly assists the optimal convergence of target function particularly in case of scarce or incorrect training data. This works especially where the new knowledge that needs learning happens with the same set of examples which learned the knowledge earlier.

One kind of inductive bias is the use of background knowledge (B_K) [5] of the domain theory that explains the input training data. In other words, the agent should already know something about the domain it is going to formally induce in the form of accumulated information. Learners use this background knowledge to distinguish useful features from training examples. Learning the background knowledge is itself learning that an agent has to do and is known as a cumulative or incremental learning process. The background knowledge in some cases can be acquired by the relevant domain experts.

KBIL is one of the good examples of inductive learning (IL) which supports the notion of cumulative learning (Fig. 2.3). To reduce the hypothesis space, KBIL induces hypothesis (**h**) with the inductive bias in the form of background knowledge (**B_K**) and the training examples (**E_T**), such that:

$$\mathbf{IL} \wedge \mathbf{B_K} \wedge \mathbf{P}\,(\mathbf{x_i}, \mathbf{y_i}) \succ \mathbf{h} \qquad \text{(KBIL)}$$

where IL is the inductive learning algorithm. The resulting hypothesis h should explain both the background knowledge and training examples. In KBIL, **B_K**, **h** and **E_T** are represented as a set of clauses or as a logical program with predicates (first-order literals) representing the attributes in them. New knowledge learnt for the incremental construction of hypothesis is exploited to improve the background domain knowledge as well. This process is referred to in the literature as the constructive inductive learning [38].

In most of the domain model learning systems, the fundamental motivation models have to solve is model-based planning tasks. One of the prominent inductive

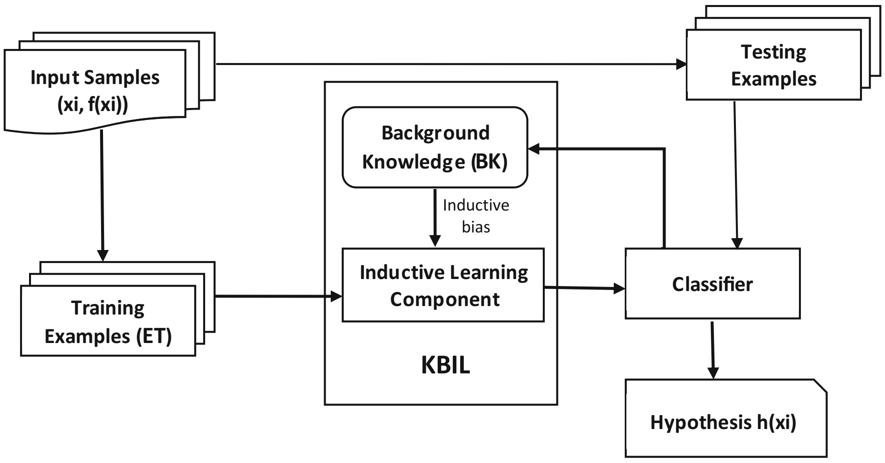

Fig. 2.3 A knowledge-based inductive learning system structure

learning systems is the *LOCM* system [12, 13]. LOCM uses an object-centred representation and performs automated induction of the dynamic aspects of a domain model on FSM representation of object sorts. Each object sort contains objects of the same type that behave in the same way. LOCM requires only a set of fully observable plan traces as the training data with no requirement of background knowledge about the domain. The main assumption which the LOCM relies on is that all objects in the domain go through transitions. This assumption is too strong for some scenarios especially when the domain contains static aspects too (as static constraints are not reflected in the plan traces). Based on this, a drawback of the LOCM process is that it can only induce a 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 LOCM drawback is overcome by the *ASCoL* system [26], an inductive system that exploits graph analysis method to automatically identifying static relations, in order to enhance planning domain models. It uses the same set of input as used by LOCM. *LOP* [21] addresses the same problem of missing static facts in the learned domain model by using optimal goal-oriented plans. LOP compares the optimal input plans with the optimal plans found by using the extended domain model. If the latter is shorter, then some static relations are deemed to be missing.

LOCM2 [86] is an extension of LOCM with the provision of multiple parameterised FSMs to represent each object's separate behavioural aspects. LOCM2 extends the coverage of domains and the captured domain semantics. *NLOCM* [22] extends LOCM to generate fixed action cost numeric domains. It exploits the constraint programming approach to add numeric weights to the states and transitions of the FSMs produced by LOCM. The LOCM family of algorithms is distinct in that it induces the fluents without any additional input support alongside plan traces, i.e., a sequence of actions with no initial, intermediate and goal states mentioned.

Some other inductive leaning systems include *ARMS* [72], *SLAF* [63], *Opmaker* [58], *Opmaker2* [36], *RIMS* [84] and *LSO-NIO* [48]. Zhuo et al. [78] provide two extensions of their work ARMS [72], in the form of a new system *LAMP* which learns domain models written in PDDL, or in other words in terms of quantifiers and logical implications. The other extended system of ARMS learns domain models for hierarchical task networks (HTN), called *HTN-Learner*. *LAMMAS* (learning action models for multi-agent systems) [80] extends ARMS for a multi-agent environment using the same underlying method. *HTNLearn* [81] is another system that induces HTN methods and action models by using as input a collection of plans with partially annotated intermediate state information and a set of annotated tasks.

Hoffmann et al. [23] induce the business process models (BPM) using logs of actions recorded from real life business activity execution. The model then turns into the workflow. The main aim of the developed systems is to use the process mining technique to exploit the sequence of events. The process mining algorithm induces a model in the form of graphs such as Petri Nets.

Framer [34] induces domain models from natural language action description. It uses an estimate of functional similarity, so sentences that describe similar behaviours are represented by the same planning operator. After obtaining planning operators structure, Framer induces formal domain model by using LOCM. Martínez et al. [35] induce a probabilistic relational model including action and exogenous effects. It uses a set of completely observable state transitions as input to learning multiple candidate models using LFIT [25] system. It then uses an optimiser to select the best model out of all the candidates. *LFIT* is a KBIL system that induces a set of propositional rules by understanding the given input transitions in the form of interpretations. In order to learn transition rules of cellular automata, LFIT exploits rules as the background knowledge and conditions on rules as inductive bias.

FAMA [3] induces STRIPS domain model from the observations of plan executions. It demonstrates the ability to learn from partial or totally unobservable actions in plan executions which makes FAMA appropriate to learn from sensory inputs. The least amount of input FAMA requires is the initial and final states of the plan execution along with partial domain model. FAMA also presents two novel model-semantics evaluation metrics that build upon two recognised metrics, precision and recall [14] to evaluate the learned action models with respect to observations of plan executions.

PlanMiner-O2 [62] is an algorithm that uses a classification algorithm, based on inductive rule learning techniques, to learn action models with discrete numerical values (represented as action costs) from incomplete and noisy data. In accepts plan traces with intermediate partially observable states affected by noise as an input.

There exist many more inductive domain model learning systems (with varying level of quality and quantity of input they require) proposed in the last decade. Due to the generality of the topic discussed in this chapter we only include a few here.

2.3 Analytical Learning

KBIL differs from analytical learnings (AL) in the utilisation of training examples. AL mainly uses prior knowledge and exploits a training example just to analyse and discern the relevant features from it. AL uses deductive reasoning method while empirical learning uses inductive reasoning method.

Explanation-based learning (EBL) [15] is the most common type of AL. In EBL the generalised rule logically follows only the prior or background knowledge and does not learn any new facts from examples, such that for EBL the following expression should be valid:

$$\mathbf{B_K} \succ \mathbf{h} \qquad \text{(EBL)}$$

EBL differentiates from the pure IL in that it looks for only the relevant positive examples to logically justify the background knowledge while IL learns all the true features including relevant to learn from and irrelevant to ignore. This is the reason why IL requires way more training data compared to EBL that can even learn with one relevant example. Mitchell and Thrun [45] quotes a very useful chess example to explain the difference in detail.

Much like the beneficial side of AL, it also suffers from the problem of doggie outcomes if the background knowledge of the domain is incorrect, e.g., in domains with no correct or complete background knowledge available to produce logically justified hypothesis like the stock market. In such situations a reliable source of learning could be the training examples to identify the relevant feature and regularities, i.e., inductive learning to learn statistically justified hypothesis. This leads to a hybrid or mixed inductive-analytical learning approach discussed in the next section.

Some more examples of analytical learning techniques include memorization, static analysis and abstractions learning and case-based reasoning. The analytical learning method is rarely used for domain model learning while the common use of it is in learning of search control knowledge.

One of the most prominent early works that utilise analytical learning is *PRODIGY* [9]. PRODIGY is a planning and learning architecture that integrates a number of learning modules to improve learning and reasoning mechanism. It refines and improves the underlying domain knowledge through experimentation and learns the control rules through experience.

Under a deterministic environment, PRODIGY incrementally learns the domain model actions by a closed-loop integration of observing other agents, learning, planning and executing plans in the environment. It produces operator hypothesis by observing the sequence of changes happening in the environment as the effects a particular action execution. It verifies the correctness of its hypothesised operators during the plan execution stage of planning. In addition to the observations of sequential changes and observed state changes of its learned action, PRODIGY also takes object types and predicate specifications as inputs to learn a domain model.

Among the six main learning modules of the PRODIGY system, the *Experiment* [10] and the *Apprentice* [28] are the two modules used to acquire and improve the underlying domain knowledge through inductive learning.

The remaining four modules of the Prodigy, i.e., *EBL* [39], *Static* [17], *Analogy* [11] *and Alpine* [31], that learn control rules for the PRODIGY planner and assist efficient planning process, use analytical learning.

2.4 Hybrid Learning

Hybrid learning or the multi-strategy learning offers support when either of inductive (statistical) or analytical (logical) knowledge learning cannot individually generalise because of scarce training data or poor background knowledge, respectively. Also, since most machine learning algorithms are custom designed with particular datasets or learning tasks, merging two or more techniques together can improve the overall results and effectiveness of the learner in most of the cases. There are a number of hybrid learning approaches in planning where inductive and analytical learning perform hand-in-hand. These include explanation-based neural network (EBNN), explanation-based learning and inductive logic programming (EBL and ILP) and a combination of explanation-based learning and reinforcement learning (EBL and RL). On literature search, it becomes apparent that the most accepted analytical learning technique in multi-strategy learning is EBL. Among the most common uses of logical learning in EBL and of hybrid approaches is in learning search control rules and heuristics for planning speedup. We only discuss the hybrid techniques which touch the topic of domain model learning.

From the domain model learning perspective, explanation-based neural network (EBNN) blends the two techniques together [44, 45]. It uses NN (backpropagation algorithm) as a form of inductive learning and puts the domain inferences together by updating the NN weights in consistency with training data. For analytical learning, EBL analyses and explains the input training data in terms of the extracted slopes from the prior domain knowledge of already learned NN. The contribution and extent of participation of each technique in EBNN vary depending upon the accuracy of training examples and correctness of the prior knowledge.

PRODIGY, *SOAR* [33] and *THEO* [42] combines both inductive and analytical learning, relying more heavily on the EBL-like deductive methods for acquiring control knowledge.

2.5 Surprise-Based Learning (SBL)

Autonomous agents commonly encounter the unknown events which most of the times are realistic and still not engineered in their knowledge base. For example, AUV (autonomous underwater vehicle) that meets an unexpected underwater creature for which it has no model. Ideally, these events can provide the opportunity to learn by experiencing them. Unlike knowledge-based learning methods, SBL is specially designed for autonomous learning and planning in an unforeseen situation with no background knowledge (of the domain available) to the learning agent. SBL works with the notion of prediction rules. Such prediction rules present the agent with the observational model of the environment for pre-action execution time and the predicted observations for post-action execution.

Most of the times in SBL, the learning occurs based on the notion of goal-driven autonomy (GDA) [71]. It is a conceptual model for creating an autonomous agent. In GDA the learning agent continuously monitors and evaluates the activity/plan execution outcome with the already predicted observations [47]. Wherever the action outcome does not match the observed predictions, the algorithm detects and records the discrepancy. The agent then builds explanation by analysing the discrepancy, its cause and effects and updates its hypothesis model and reformulates goals to align with its primary objective. Agents that perform goal reasoning, explicitly model and reason about the goals they try to achieve [2].

The discrepancies in the observations which present themselves as an opportunity to learn are termed as ‘surprise’ in the title, i.e., situation where the action effects violate its predicted model. SBL is shown to be successful on a modular robot learning and navigating in a small static and the fully observable environment with no prior knowledge available in the knowledge base [55].

FOOLMETWICE [46], the extension of *ARTUE* [47], is a system with a relaxed assumption about domain model completeness. The system implements GDA and presents an algorithm to learn (and apply) environment models of unknown exogenous events. It generalises a new model’s preconditions by learning from the states that cause inconsistency.

Nguyen and Leong [51] present *STAR* (surprise triggered adaptive and reactive) system that dynamically learns models of its opponents’ strategies in response to surprises. Ranasinghe and Shen [55] present an algorithm for an agent to learn and refine its action models based on the SBL. The model can be used to predict the state changes and identify when surprises occur.

The *LIVE* system [64] enhanced the GPS (general problem-solving) system [16] with the ability to learn models. LIVE learns prediction rules by observing changes in the environment. It assimilates action exploration, experimentation, learning and problem-solving. To create STRIPS-like rules, it requires actions and percept from the environment, a process to provide observation from the environment and the state description of the environment.

EXPO [20] is a learning-by-experimentation system for refining incomplete planning operators. It refines operators, which have missing preconditions and effects by failure-driven experimentation with the environment. It does this by observing plan execution and detecting the inconsistency between observations and predictions. To adjust the inconsistency, it produces a set of hypothesis and empirically tests each. EXPO does incremental learning and also learns conditional effects.

Another example of refinement and incremental learning is the *OBSERVER* [69] system. It induces a STRIPS-like initial model and repairs it continuously during operation by monitoring expert agents. It applies version spaces algorithm [40] to the observations. It repairs plans from incorrect and partial domain knowledge. The framework learns planning operators by observing expert agents and subsequent knowledge refinement in a learning-by-doing paradigm. To refine the learnt operators, it solves practice problems with operators, analyses and learns from the execution traces of the resulting solutions. It uses the pure inductive methodology and does not require background knowledge to do learning. The method is implemented inside PRODIGY [67] that includes a general-purpose planner and several learning modules to improve the planning domain knowledge and also the control knowledge to support the planning algorithm.

To efficiently learn models, there needs to be a quantitative bound on a number of input samples or the required amount of interactions with the environment. Walsh and Littman [68] demonstrate that learning STRIPS operators through raw-experience can require an exponential number of samples, but restricting the size of the precondition lists allows for sample-efficient learning. An external teacher is needed to fulfil the demand of required solution observations in order to eliminate the restriction on the size of the precondition lists.

2.6 Transfer Learning

Through transfer learning (TL), the system exploits data from one or more source domains to improve learning performance in a different target domain in the situations with like and limited training data availability. Knowledge engineering through transfer learning especially for planning domain models has recently received active attention and the resulting learning technique provides a good corpus of work for interested researchers.

Many machine learning methods work well only under a common assumption that the training and test data are taken from the same feature space and the same distribution. In the case of altered feature space and distribution, most statistical models need to be rebuilt from scratch using newly collected training data. In many practical applications, it is expensive or impossible to recollect the needed training data and rebuild the models. It would be nice to reduce the need and effort to recollect the training data especially when data is scarcely available or when it easily becomes outdated. In such cases, knowledge transfer or transfer learning between task domains is desirable. Pan and Yang [52] in a survey on TL explain the benefit of using TL to cover the same feature space assumption of machine learning. The survey also addresses the primary issues of what, when and how to transfer.

Zhuo et al. [76] learn the hypothesis model from plan traces by transferring useful information from other domains whose domain models are already known. The system creates a metric to measure the shared information and transfer this information according to the metric. The larger the metric is, the bigger the information is transferred.

Inspired by the educational psychology of meta-cognitive reflection for better inductive transfer learning practices, a novel L2T framework [73] has been proposed for transfer learning. The system automatically optimizes what and how to transfer between a source and a target domain by leveraging previous transfer learning experiences.

Zhuo et al. [75] present *t-LAMP* (transfer Learning Action Models from Plan traces), which can learn action models in PDDL language with quantifiers. The system exploits plan traces using Markov logic networks to enable knowledge transfer.

Zhuo and Yang [82] proposed *TRAMP* (Transfer learning Action Models for Planning) system, to learn action models with limited training data in the target domain, by transferring as much of the available information from source domains by using web search on top of transfer technique to bridge the transfer gap.

2.7 Policy Learning

A policy is a state-action mapping and policy learning is learning what to do in every possible situation. Based on the assumption that the goals of the learning system are static, instead of learning the domain model of the environment, the system simply learns the reaction or response to common situations that may arise in the environment. The system does not learn the domain model of the environment to take action rather it only learns the policies to respond to the exact situation and this is why this approach is called policy learning. This differs from knowledge-based apprentice systems to learn domain models (discussed in the next section). The planning literature also describes this method as *learning by observation, imitation* or *watching*.

Behavioural cloning [8] also known as learning by imitation is an example of a policy learning approach. In behavioural cloning method, the learning system observes and reproduces the skills of the trainer agent (which is usually human) in carrying out the particular task. It then records the responses of the trainer in every situation along with the cause that gave rise to the response. A sequence of these responses based on the sub-cognitive skills and actions of a trainer is used as input to the learning system. When the learning system itself confronts some situation, it compares this situation with the learnt situation from the trainer and responds to act according to the learnt response of the trainer. It outputs a set of rules which reproduce skilled behaviour. This technique has many advantages including the capability to quickly learn from a few training sessions and to act more reliably than a human trainer that it learnt from Benson [6]. This method is useful for building an automatic control system.

From the perspective of the complex domains, it appears to be a difficult exercise to train the system for all the possible values and situations that can occur inside the environment. However, in cloning the response to take according to a particular situation is usually the same for a big set of possible state space, as the system can then generalise the response and develop a policy to cover more inputs.

ALVINN system [54] is one of the best examples of behavioural cloning. Learning from fully or partially annotated demonstrations by domain expert has been used by several systems for knowledge acquisition in robotics and task modelling [4, 19] but has rarely been used to learn declarative domain model [50] for AI planners. This is partly because even for the moderately complex domains, it is unfeasible for the area expert to specify conjecture for every action, explanations for every inconsistency and all possible effects of the model, as the performance of the learner is affected by the ability of the trainer. In order to cover for trainer's implicit knowledge gap, many systems use reinforcement learning techniques to co-operate with this type of learning such as using Q-learning [70]. Two of the common demonstration approaches include tele-operation and shadowing [4].

2.8 Other Methods of Knowledge Acquisition

This section includes potential knowledge acquisition and learning methods that are either not formally classed as the typical learning methods for planning or are in their infancy yet have produced some effective domain learning systems in the past.

Apprentice Systems Mitchell et al. [43] coined the term apprentice system as an interactive knowledge-based system which induce knowledge by observing the users interaction with the system and analysing the problem-solving steps. These systems capture and infer from the training examples of the user's activity and the context on which decisions involved in activity were taken. It then generalizes rules from the training examples that are comparable to the hand-generated rules. The idea has been implemented in a number of application areas. Based on the idea, the same authors created LEAP—a learning apprentice and advice system for digital circuit design. LEAP integrates new knowledge by its experience with the user approval/rejection of its advice about circuit decomposition by observing and analysing the user's problem-solving decisions and steps.

ARMS (acquiring robotic manufacturing schemata) system [61] represents an important first step towards a learning-apprentice system for manufacturing. It learns from a user interface where the user instructs the simulated robots to perform simple tasks. Michele [49]—a groupware toolkit based on a multi-agent model of communication—induces learning mechanism by its interactions with the user and stores learned knowledge in each user's environment. It induces a decision tree for each query to the user and exploits ID4 [60] learning algorithm. Jourdan et al. [29], Tecuci and Dybala [65] and Abbeel et al. [1] are some more examples of the apprentice systems that use various methods of interaction with the user including passive observations or querying the user to record the reason behind the particular decision made.

Crowdsourcing Recently crowdsourcing [24] has been exploited as a novel approach for acquiring planning domain models. Collecting a large amount of training data is not always feasible in terms of reach and cost, e.g., in a situation like a military operation. Instead of collecting training examples, crowdsourcing methods engage different annotators that could include various sources like domain experts, stakeholders, previous data or experience of the general public about the domain to learn. The outcome from various annotators built as the soft constraints can later be solved using max-sat solver to generate a domain model.

While crowdsourcing is comparatively new in learning for planning, it has been used in several planning applications, e.g., Zhang et al. [74] enable a crowd to effectively and collaboratively resolve global constraints to carry out itinerary planning. Gao et al. [18] propose a technique to handle the discrepancy in crowd inputs by first building a set of human intelligence tasks (HITs) for values collection and then estimate the actual values of variables and feed the values to a planner to solve the problem. Raykar et al. [56] label training data for machine learning by crowdsourcing information from experts and non-experts. The system not only evaluates the different experts but also gives an estimate of the actual hidden labels.

Recently, crowdsourcing has also been exploited for acquiring planning domain models [77]. It is worth noting that the problem of encoding domain models is being analysed not only from the point of view of generating models in a specific description language—such as PDDL—but also for generating different sorts of automatically exploitable models. Konidaris et al. [32] proposed a method for constructing symbolic representations for high-level planning by establishing a close relationship between an agent's actions and the symbols required to plan to use them.

MLN To deal with the probability along with imperfect and uncertain knowledge, Markov logic network (MLN) [57] is a dense language to determine very large Markov networks, and has the ability to flexibly and modularly incorporate a wide range of domain knowledge. Many learning systems exploit the technique of MLN that applies the concept of a Markov network to the first-order logic (FOL) and draws the inference from the evidence. In a Markov graph, the vertices are taken as the atomic FOL formulas, and the edges act as the logical connectives used to construct the formulas. Several systems including LAMP—to learn domain model with quantifiers and logical implications [83] and AMAN—a system for action-model acquisition from noisy plan traces [79], learn domain models based on the idea of Markov network as a major driving approach.

LAMP system uses the MLN technique to select the most likely subsets of candidate formulas from all the generated formulas which are later transformed into learned action models. It learns STRIPS action models (with quantifiers and logical implications) for classical planning from plan traces with partially observed states. It learns a domain model for an observable and deterministic environment from training plans with little or no pre-engineered domain knowledge including object types, predicate specifications and action headers.

AMAN builds a graphical model to capture the relations between actions (in plan traces) and states and then learns the parameters of the graphical model. After that, AMAN generates a set of action models according to the learnt parameters. Specifically, the system first exploits the observed noisy plan traces to predict correct plan traces and the domain model based on the graphical model and then executes the correct plan traces to calculate the reward of the predicted correct plan traces according to a predefined reward function. Then, AMAN updates the predicted plan traces and domain model based on the reward. It iteratively performs the above-mentioned steps until a given number of iterations is reached. Finally, the predicted domain model is provided.

TRAMP [82] system conducts MLNs assisted transfer learning to learn domain models.

3 Characteristics of the Domain Model Learning Tools

Using machine learning methods, several tools and techniques have been presented in the recent past to facilitate the transformation process of real planning application requirements into a solver ready PDDL domain model that uses training or observation as inputs. These techniques use various types of knowledge besides plan traces, like general properties and constraints about domain actions, as well as partial knowledge about the kind of domain in which they are operating. To learn expressive domain models, systems tend to require more detailed inputs and substantial a priori knowledge which often include details about initial and goal state information. Some systems also require state information before and after an action execution within each training plan. The main aim of this type of learning is to overcome the knowledge acquisition bottleneck [30], to help planning agents become more autonomous and make them able to adapt and plan for unseen situations and to debug existing domain models.

This section presents a brief overview of the automated tools along with their characteristics that can be exploited to automatically induce planning domain models.

Input Characteristics Input characteristics of the system depend on what the system is trying to learn, the learning method and the extent of learning. Some systems aim to design the complete domain model of a particular world, some refine already built partial domain model and some aim to transfer knowledge from one domain to other. Learning techniques can be supervised or unsurprised learning. It depends if the trainer indicates when something goes wrong in the case of supervised learning. All traditional domain model learning systems accept input in the form of training plans. Based on the learning capability some systems may also have to exploit additional background knowledge $\mathrm{B_K}$ or information about the surrounding world. $\mathrm{B_K}$ can be in any form like observations, constraints, type structure, initial, intermediate and goal states, fluent, etc. Some systems also require a partial domain model (with missing preconditions and effects in the actions) in the input.

Potential plan traces can be gathered from multiple sources and applications, for example, the sequence of workflow in some process execution, logs of commands for installing a piece of software or the moves or steps captured from game playing, etc.

Obtaining the training plans from sensors, sometimes noise inevitably gets introduced into plan traces when some sensors are occasionally damaged, with unintentional mistakes in the recording of the action sequence, or may be due to the presence of other agents in the same environment. To deal with this, several systems learn domain models with noisy inputs.

Output Characteristics For output, systems can be classified based on the extent and capability to learn in varying world dynamics and the state of observability in the environment. For instance, the characteristics of the surrounding environment can be discrete or continuous, static or dynamic. Action effects can be stochastic or non-deterministic (rolling of dice) compared to a fully deterministic environment. In terms of observability, the learning environment can be fully or partially observable. Learning from deterministic, fully observable, discrete and static environmental characteristic offer lesser challenges than continuous and dynamic environmental features. Jiménez et al. [27] provide a thorough review of techniques based on the learning targets of the systems from various planning paradigms, i.e., learning in varying level of world dynamics and state observability.

The extent of learning by a system is the granularity of the output and the amount of details learnt, for example, full or partial domain model in the output or leaning of domain model with quantifiers and logical implications [83].

No standard evaluation and analysis methods exist to verify the output domain model completeness and quality and like the requirements specification, these characteristics cannot be objectively assessed and proven correct. Learning systems and their output are typically evaluated empirically, based on their divergence from the reference model syntax (which itself can be questionable from multiple perspectives by multiple experts). A step forward in defining the quality of domain models and to improve evaluation method semantically, FAMA [3] presents a method to assess the quality and performance of the learning approaches. The evaluation method alleviates the common limitation of syntactic evaluation methods. A usual limitation of syntactic evaluation methods is when the learned model is semantically correct but syntactically differ from the benchmark model. Unable to evaluate correct but redundant preconditions in the model is another downside of syntax-based assessment methods.

McCluskey et al. [37] use the idea of domain model as a formal specification of a domain and consider what it means to measure the quality of such a specification. To build the notion of quality assessment, they used dynamic and static testing of the domain model. Vallati and McCluskey [66] present a quality framework which aims at representing all the aspects that affect the quality of knowledge in domain models. The framework is based on the interaction between seven different sets that underpin the domain quality.

Some learning systems additionally produce heuristics and various graphical views like finite state machines along with the domain model while some just improve the partial domain model by learning the missing preconditions and effects of operators.

Representational Language and Mechanism A well-chosen representation language should explicitly model every action effect the system might confront. In addition to that, a domain modelling representation language should have some salient attributes. It should have supporting tools to check its operation and have logically strong inference mechanism to carry out reasoning. It should be sufficiently expressive to explicitly model a complex scenario of the real world. Moreover, it should be customizable and structured to capture every action effect the system might confront in operator definitions. In addition, it should have clear syntax and semantics to support the operational aspects of the model.

Different learning systems use different languages to express the output domain model including PDDL, STRIPS, OCL, etc. While choosing the representation language for effective system output, a well-known complexity-expressiveness trade-off of representation is not easy to attain. More expressive languages let the systems produce domains that can express input data in a better way while reducing the expressivity enhances the complexity of the consistent hypothesis. To overcome the KE bottleneck, most systems in the area of domain model learning use propositional and first-order logic to represent domain model for the logical agents and output the model in some variant of PDDL in their initial acquisition phase.

Most learning systems use action-centred representation mechanism where applying an action on a state transforms the state of the system into a new state. These models are represented mostly by PDDL and its variants built on first-order logic. Another mechanism is the object-centred representation that captures the dynamic relationship between objects in state parameters. OCL (object centred language) is used to support this mechanism. Figure 2.4 shows a (non-exhaustive) characteristic that a domain model learning system can cover.

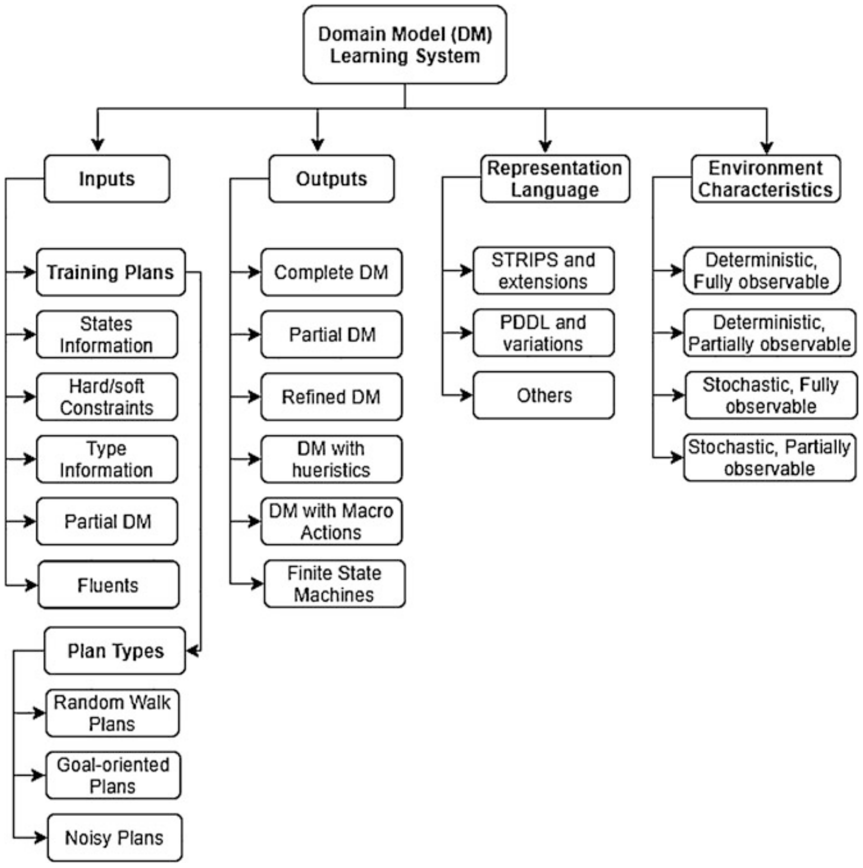

Fig. 2.4 Various characteristics of domain model learning tools

4 Conclusion

Learning is fundamental to autonomic behaviour and it can be defined in many ways. From the point of view of machine learning, it is defined as a change in behaviour through learning to allow improvement in performance. This chapter investigates various classical and state-of-the-art methods to attain such capability for automatically learning domain models from training data. The taxonomy and order of these methods we followed is based on their standing and frequency of usage in the past research. The choice of the learning method and the design of the learning mechanism based on it can be made easy by thinking in terms of the following four factors:

- Firstly, the choice of the learning method depends on the learning goals in a particular domain. For instance, does it need to design the complete domain model of a particular world? Is it trying to transfer knowledge from one domain to other?
- Secondly, it depends upon the availability of the information about the surrounding world. If available, then what is the type and consistency of input assistance or prior knowledge? Is it supervised or unsurprised learning? Is input data consistent or noisy? Sometimes the supervised learning situations become semi-supervised based on the deliberate systematic inconsistencies in the data, e.g., learning the age of people from observing pictures where in the training data some people lied about their age.
- Another factor to consider is what feedback is available. For example, does a trainer indicate when something goes wrong in the case of supervised learning?
- Finally, the characteristics of the world to learn knowledge from also matters, for instance, observability and stochasticity of the environment.

There are several knowledge engineering tools with varying capabilities. These tools support automated planning, not only in the knowledge elicitation process but also for the design, validation and verification of developed models.

There are several relevant aspects that need more focused attention of the planning and learning community in the future. In order to deal with the real-world planning-inherent complexity, learning-augmented planning systems should be able to apprehend the environment, generate corresponding effects and enhance their performance according to the previous experience. This has attracted much research in the recent past but the current state-of-the-art is still a long way from human-level abilities to work in real world. Instead, most of these systems work under classical restrictive environmental assumption with toy domains setup that are more comprehensible and limited proxies of the real-world environment.

Learning systems can be categorized as offline (learns before the planning process starts) and online (learns during the plan search and execution stages). From the domain model learning viewpoint, offline learning is comparatively popular starting from the learning of domain invariants to learning complete domain model from variable sources. Both online and offline domain model learning have pros and cons. Online learning can continuously/incrementally refine the domain model in case an anomaly is detected by improving or adapting to the changes while for offline planning, the planner has to bear with the predefined version till the planning process finishes. Similarly, for online learning of the domain model, the overhead cost incurred for the joint planning-learning process is higher in terms of processing time and efficiency compared to offline learning [85]. This may also explain why online incremental domain model learning has not been very popular in recent years and needs active research attention to effectively reduce the overhead cost.

References

1. Abbeel, P., D. Dolgov, A. Y. Ng and S. Thrun (2008). *Apprenticeship learning for motion planning with application to parking lot navigation*. 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE.
2. Aha, D., M. Klenk, H. Munoz-Avila, A. Ram and D. Shapiro (2010). Goal-driven autonomy: Notes from the AAAI workshop, Menlo Park, CA: AAAI Press.
3. Aineto, D., S. J. Celorrio and E. Onaindia (2019). "Learning action models with minimal observability." *Artificial Intelligence*.
4. Argall, B. D., S. Chernova, M. Veloso and B. Browning (2009). "A survey of robot learning from demonstration." *Robotics and autonomous systems***57**(5): 469–483.
5. Baxter, J. (1995). Learning internal representations. *Proceedings of the eighth annual conference on Computational learning theory*. Santa Cruz, California, USA, ACM**:** 311–320.
6. Benson, S. (1995). *Action model learning and action execution in a reactive agent*. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95).
7. Brachman, R. J. and H. J. Levesque (1984). *The tractability of subsumption in frame-based description languages*. AAAI.
8. Bratko, I. and T. Urbančič (1997). "Transfer of control skill by machine learning." *Engineering Applications of Artificial Intelligence***10**(1): 63–71.
9. Carbonell, J., O. Etzioni, Y. Gil, R. Joseph, C. Knoblock, S. Minton and M. Veloso (1991). "Prodigy: An integrated architecture for planning and learning." *ACM SIGART Bulletin***2**(4): 51–55.
10. Carbonell, J. G. and Y. Gil (1990). Learning by experimentation: The operator refinement method. *Machine learning*, Elsevier**:** 191–213.
11. Carbonell, J. G. and M. Veloso (1988). *Integrating derivational analogy into a general problem solving architecture*. Proceedings of the First Workshop on Case-Based Reasoning.
12. Cresswell, S. (2009). "LOCM: A tool for acquiring planning domain models from action traces." *ICKEPS 2009*.
13. Cresswell, S., T. L. McCluskey and M. M. West (2009). *Acquisition of Object-Centred Domain Models from Planning Examples*. ICAPS.
14. Davis, J. and M. Goadrich (2006). *The relationship between Precision-Recall and ROC curves*. Proceedings of the 23rd international conference on Machine learning, ACM.
15. DeJong, G. and R. Mooney (1986). "Explanation-based learning: An alternative view." *Machine learning***1**(2): 145–176.
16. Ernst, G. W. and A. Newell (1969). *GPS: A case study in generality and problem solving*, Academic Pr.
17. Etzioni, O. (1991). *STATIC: A Problem-Space Compiler for PRODIGY*. AAAI.
18. Gao, J., H. H. Zhuo, S. Kambhampati and L. Li (2015). *Acquiring Planning Knowledge via Crowdsourcing*. Third AAAI Conference on Human Computation and Crowdsourcing.

19. Garland, A. and N. Lesh (2003). "Learning hierarchical task models by demonstration." *Mitsubishi Electric Research Laboratory (MERL), USA–(January 2002).*
20. Gil, Y. (1992). Acquiring domain knowledge for planning by experimentation, DTIC Document.
21. Gregory, P. and S. Cresswell (2015). *Domain Model Acquisition in the Presence of Static Relations in the LOP System*. ICAPS.
22. Gregory, P. and A. Lindsay (2016). *Domain model acquisition in domains with action costs*. Twenty-Sixth International Conference on Automated Planning and Scheduling.
23. Hoffmann, J., I. Weber and F. Kraft (2009). *Planning@ sap: An application in business process management*. 2nd International Scheduling and Planning Applications woRKshop (SPARK'09).
24. Howe, J. (2008). *Crowdsourcing: How the power of the crowd is driving the future of business*, Random House.
25. Inoue, K., T. Ribeiro and C. Sakama (2014). "Learning from interpretation transition." *Machine Learning***94**(1): 51–79.
26. Jilani, R., A. Crampton, D. Kitchin and M. Vallati (2015). *Ascol: A tool for improving automatic planning domain model acquisition*. Congress of the Italian Association for Artificial Intelligence, Springer.
27. Jiménez, S., T. De la Rosa, S. Fernández, F. Fernández and D. Borrajo (2012). "A review of machine learning for automated planning." *The Knowledge Engineering Review***27**(4): 433–467.
28. Joseph, R. L. (1989). "Graphical knowledge acquisition." *In Proceedings of the Fourth Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, Canada.*
29. Jourdan, J., L. Dent, J. McDermott, T. Mitchell and D. Zabowski (1993). Interfaces that learn: A learning apprentice for calendar management. *Machine learning methods for planning*, Elsevier**:** 31–65.
30. Kambhampati, S. (2007). *Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models*. Proceedings of the National Conference on Artificial Intelligence, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.
31. Knoblock, C. A. (1991). Automatically Generating Abstractions for Problem Solving, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.
32. Konidaris, G., L. P. Kaelbling and T. Lozano-Perez (2014). "Constructing symbolic representations for high-level planning." *Proceedings of the 28th AAAI Conference on Artificial Intelligence*.
33. Laird, J. E., A. Newell and P. S. Rosenbloom (1987). "Soar: An architecture for general intelligence." *Artificial intelligence***33**(1): 1–64.
34. Lindsay, A., J. Read, J. F. Ferreira, T. Hayton, J. Porteous and P. Gregory (2017). *Framer: Planning models from natural language action descriptions*. Twenty-Seventh International Conference on Automated Planning and Scheduling.
35. Martínez, D., G. Alenya, C. Torras, T. Ribeiro and K. Inoue (2016). *Learning relational dynamics of stochastic domains for planning*. Twenty-Sixth International Conference on Automated Planning and Scheduling.
36. McCluskey, T., S. Cresswell, N. Richardson, R. Simpson and M. M. West (2008). "An evaluation of Opmaker2." *The 27th Workshop of the UK Planning and Scheduling Special Interest Group, December 11–12th, 2008, Edinburgh.*: 65–72.
37. McCluskey, T. L., T. S. Vaquero and M. Vallati (2017). *Engineering knowledge for automated planning: Towards a notion of quality*. Proceedings of the Knowledge Capture Conference, ACM.
38. Michalski, R. S. (1993). Learning= inferencing+ memorizing. *Foundations of Knowledge Acquisition*, Springer**:** 1–41.
39. Minton, S., J. G. Carbonell, O. Etzioni, C. A. Knoblock and D. R. Kuokka (1987). *Acquiring effective search control rules: Explanation-based learning in the PRODIGY system*. Proceedings of the fourth International workshop on Machine Learning, Elsevier.

40. Mitchell, T. M. (1977). *Version spaces: A candidate elimination approach to rule learning*. Proceedings of the 5th international joint conference on Artificial intelligence-Volume 1, Morgan Kaufmann Publishers Inc.
41. Mitchell, T. M. (1980). *The need for biases in learning generalizations*, Department of Computer Science, Laboratory for Computer Science Research ....
42. Mitchell, T. M., J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette and J. C. Schlimmer (1991). "Theo: A framework for self-improving systems." *Architectures for intelligence*: 323–355.
43. Mitchell, T. M., S. Mabadevan and L. I. Steinberg (1990). LEAP: A learning apprentice for VLSI design. *Machine learning*, Elsevier**:** 271–289.
44. Mitchell, T. M. and S. Thrun (2014). *Explanation based learning: A comparison of symbolic and neural network approaches*. Proceedings of the Tenth International Conference on Machine Learning.
45. Mitchell, T. M. and S. B. Thrun (1996). "Learning analytically and inductively." *Mind matters: A tribute to Allen Newell*: 85–110.
46. Molineaux, M. and D. W. Aha (2014). *Learning unknown event models*. Twenty-Eighth AAAI Conference on Artificial Intelligence.
47. Molineaux, M., M. Klenk and D. Aha (2010). *Goal-driven autonomy in a Navy strategy simulation*. Twenty-Fourth AAAI Conference on Artificial Intelligence.
48. Mourao, K., L. S. Zettlemoyer, R. Petrick and M. Steedman (2012). "Learning strips operators from noisy and incomplete observations." *arXiv preprint arXiv:1210.4889*.
49. Nakauchi, Y., T. Okada and Y. Anzai (1991). *Groupware that learns*. [1991] IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings, IEEE.
50. Nejati, N., P. Langley and T. Konik (2006). *Learning hierarchical task networks by observation*. Proceedings of the 23rd international conference on Machine learning, ACM.
51. Nguyen, T.-H. D. and T.-Y. Leong (2009). *A Surprise Triggered Adaptive and Reactive (STAR) Framework for Online Adaptation in Non-stationary Environments*. AIIDE.
52. Pan, S. and Q. Yang (2010). A survey on transfer learning. IEEE Transaction on Knowledge Discovery and Data Engineering, 22 (10), IEEE press.
53. Pomerleau, D. A. (1989). *ALVINN: An autonomous land vehicle in a neural network*. Advances in neural information processing systems.
54. Pomerleau, D. A. (1991). "Efficient training of artificial neural networks for autonomous navigation." *Neural Computation***3**(1): 88–97.
55. Ranasinghe, N. and W.-M. Shen (2008). *Surprise-based learning for developmental robotics*. 2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS), IEEE.
56. Raykar, V. C., S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni and L. Moy (2010). "Learning from crowds." *Journal of Machine Learning Research***11**(Apr): 1297–1322.
57. Richardson, M. and P. Domingos (2006). "Markov logic networks." *Machine learning***62**(1–2): 107–136.
58. Richardson, N. E. (2008). *An operator induction tool supporting knowledge engineering in planning*, University of Huddersfield.
59. Riddle, P. J., R. C. Holte and M. W. Barley (2011). *Does Representation Matter in the Planning Competition?* Ninth Symposium of Abstraction, Reformulation, and Approximation.
60. Schlimmer, J. C. and D. Fisher (1986). *A case study of incremental concept induction*. AAAI.
61. Segre, A. M. (1987). Explanation-Based Learning of Generalized Robot Assembly Plans, ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB.
62. Segura-Muros, J. Á., R. Pérez and J. Fernández-Olivares (2018). "Learning Numerical Action Models from Noisy and Partially Observable States by means of Inductive Rule Learning Techniques." *KEPS 2018*: 46.
63. Shahaf, D. and E. Amir (2006). *Learning partially observable action schemas*. Proceedings of the National Conference on Artificial Intelligence, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

64. Shen, W.-M. and H. A. Simon (1989). *Rule Creation and Rule Learning Through Environmental Exploration*. IJCAI, Citeseer.
65. Tecuci, G. and T. Dybala (1998). *Building Intelligent Agents: An Apprenticeship, Multistrategy Learning Theory, Methodology, Tool and Case Studies*, Morgan Kaufmann.
66. Vallati, M. and T. L. McCluskey (2018). "Towards a Framework for Understanding and Assessing Quality Aspects of Automated Planning Models." *KEPS 2018*: 28.
67. Veloso, M., J. Carbonell, A. Perez, D. Borrajo, E. Fink and J. Blythe (1995). "Integrating planning and learning: The PRODIGY architecture." *Journal of Experimental & Theoretical Artificial Intelligence***7**(1): 81–120.
68. Walsh, T. J. and M. L. Littman (2008). *Efficient learning of action schemas and web-service descriptions*. AAAI.
69. Wang, X. (1995). *Learning by observation and practice: An incremental approach for planning operator acquisition*. ICML.
70. Watkins, C. J. C. H. (1989). *PhD Thesis: Learning from delayed rewards*, University of Cambridge England.
71. Weber, B. G., M. Mateas and A. Jhala (2012). *Learning from demonstration for goal-driven autonomy*. Twenty-Sixth AAAI Conference on Artificial Intelligence.
72. Wu, K., Q. Yang and Y. Jiang (2005). "Arms: Action-relation modelling system for learning action models." *CKE*: 50.
73. Ying, W., Y. Zhang, J. Huang and Q. Yang (2018). *Transfer learning via learning to transfer*. International Conference on Machine Learning.
74. Zhang, H., E. Law, R. Miller, K. Gajos, D. Parkes and E. Horvitz (2012). *Human computation tasks with global constraints*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM.
75. Zhuo, H., Q. Yang, D. H. Hu and L. Li (2008). *Transferring knowledge from another domain for learning action models*. Pacific Rim International Conference on Artificial Intelligence, Springer.
76. Zhuo, H., Q. Yang and L. Li (2009). *Transfer learning action models by measuring the similarity of different domains*. Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer.
77. Zhuo, H. H. (2015). *Crowdsourced action-model acquisition for planning*. Twenty-Ninth AAAI Conference on Artificial Intelligence.
78. Zhuo, H. H., D. H. Hu, Q. Yang, H. Munoz-Avila and C. Hogg (2009). *Learning applicability conditions in AI planning from partial observations*. Workshop on Learning Structural Knowledge From Observations at IJCAI.
79. Zhuo, H. H. and S. Kambhampati (2013). *Action-model acquisition from noisy plan traces*. Twenty-Third International Joint Conference on Artificial Intelligence.
80. Zhuo, H. H., H. Muñoz-Avila and Q. Yang (2011). *Learning action models for multi-agent planning*. The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems.
81. Zhuo, H. H., H. Muñoz-Avila and Q. Yang (2014). "Learning hierarchical task network domains from partially observed plan traces." *Artificial intelligence***212**: 134–157.
82. Zhuo, H. H. and Q. Yang (2014). "Action-model acquisition for planning via transfer learning." *Artificial intelligence***212**: 80–103.
83. Zhuo, H. H., Q. Yang, D. H. Hu and L. Li (2010). "Learning complex action models with quantifiers and logical implications." *Artificial Intelligence***174**(18): 1540–1569.
84. Zhuoa, H. H., T. Nguyenb and S. Kambhampatib (2013). *Refining incomplete planning domain models through plan traces*. Proceedings of IJCAI.
85. Zimmerman, T. and S. Kambhampati (2003). "Learning-assisted automated planning: looking back, taking stock, going forward." *AI Magazine***24**(2): 73–73.
86. Cresswell, S. and P. Gregory (2011). *Generalised domain model acquisition from action traces*. Twenty-First International Conference on Automated Planning and Scheduling.