Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Learning Planning Domains for Office Buildings using IoT data

Priyanga Raghavan

Course of Study:	Information Technology
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Dr. Ilche Georgievski
Commenced:	June 22, 2020
Completed:	November 30, 2020

Abstract

Commercial buildings contribute to more energy consumption worldwide. Buildings are equipped with devices like sensors and actuators to know the current state of the building. The intelligence lies in making use of that collected data to make appropriate decisions that reduce energy consumption without compromising the comfort of occupants. For that we require an approach that can effectively plan and execute the operations. Energy consumption in commercial buildings by automatic control of HVAC and lighting systems using Artificial Intelligence (AI) planning ensures energy savings without any compromise in the comfort of its occupants. But AI planning requires an accurate definition of planning tasks. The generation of a planning domain model requires time, planning expertise and knowledge of the domain to be modelled. Alternately, planning domain can be generated automatically, which can save time and compensate for the domain expert's incomplete knowledge. In this work, we use inductive rule learning technique to learn planning domains from Internet of Things (IoT) data. The planning domain for office buildings to control the operation of heating, ventilation and lighting system is defined. As the planning domain is learnt from the sensor data, an approach that can handle continuous numerical information is necessary. The problem of learning planning domain model is modelled as the classification problem and the rules are extracted from the IoT data. The appropriate post-processing techniques are used to convert the learned rules into the Planning Domain Definition Language (PDDL) code. The generated planning domain is tested using the handcrafted domain and the results show that the planning domain learner can learn the continuous numerical information and the relation between them along with propositional fluents.

Contents

List of Figures

List of Tables

Listings

List of Algorithms

1 Introduction

Commercial buildings, such as office buildings have high operational cost and high CO_2 emission level due to high energy demand [7]. Energy saving is necessary for society as the saving of energy will lead to a reduced carbon footprint. Also, any building management needs to reduce operational costs. To ensure greater productivity, the working atmosphere in the offices should be conducive. The measures to reduce energy consumption should be taken without compromising the comfort of the users. Currently, building control and management systems are controlling lighting system, heating, ventilation and air conditioning (HVAC) systems using basic scheduling operations [14]. Let us consider the HVAC system, which is used for maintaining the ambient thermal condition in the building. The set-points of air temperature and humidity should be decided by considering various factors like indoor and outdoor thermal conditions and human-related factors [10]. We can equip buildings with sensors and actuators. From the data collected, the course of actions have to be planned in such a way to improve energy efficiency, the comfort of users and lower the operational costs.

This requires an approach which can effectively plan and execute the operations in the building. In this context, Artificial Intelligence (AI) planning offers a powerful technique to compute effective plans. AI planning is the process of finding a procedural course of action to achieve a given objective in the form of goals while improving the overall performance [17]. To plan the operations, it requires knowledge about the domain. A planning domain model describes the actions which have preconditions that have to be satisfied at any given state, for that particular action to be executed. The effects (postconditions) give the state of the environment after the application of a specified action. The generation of such a planning domain model requires specific expertise which only a domain expert can furnish. The efficiency and success of the planning depend on the skills of the domain expert. The domain expert should have detailed knowledge about the domain and also the planning modelling language in which domain is specified. Specifying accurate action models for real-world problems is a tedious and time-consuming process. In extreme cases, like planning for the control of the Mars Rover or an underwater vehicle, the knowledge cannot be specified accurately, as it is not known completely [21]. So, the success of the planning fully depends on the skill of the domain expert who specifies the action models. Alternately, we can generate the planning domain model automatically using a machine learning technique.

Machine learning is seen as a potentially effective means of making an independent entity like robot or system to perceive the world and make appropriate decisions, which can compensate for the expert's incomplete knowledge, thereby eliminating the possibility of human-induced error. [20]. The planning action models can be automatically learned using a machine learning technique and the domain knowledge is encoded in a formal specification.

1.1 Problem Statement

Most of the existing works, such as [30], [4], [1] are focussed on learning action models only with propositions and [27] could learn propositions and fixed numerical information. In this work, our domain of interest is commercial buildings like office buildings. To improve the energy efficiency, the device operations of HVAC and lighting systems should be planned based on the current state of the environment. The current state of the environment is analysed using the information obtained from sensors. The sensor data has numerical information like temperature of room, luminance of light, air-quality in room and so on. The planning domain we construct should be capable of dealing with these continuous numerical information.

1.2 Contribution

The aim of this thesis is to develop a learning model that can learn planning action model with continuous numerical information from Internet of things (IoT) data.

- The proposed approach can learn logical fluents, fixed and continuous numerical information and the relation between the fluents.
- The learning model can learn planning domain directly from the sensor data, with the knowledge of fluents that can appear in actions as input and without the need for plan traces or example plans.

1.3 Document Structure

The remaining part of this document is organized as follows. Chapter 2 describes the background knowledge of concepts like Artificial intelligence planning, Planning Domain Definition Language (PDDL), NewSLaVe (NSLV) algorithm, DNF rule model and Genetic algorithm, as it is required to understand our work. Chapter 3 deals with the state-of-art technologies and existing work relevant to our topic. In Chapter 4, we describe the Buildings planning domain defined in our work with its PDDL coding. Chapter 5 explains the methodology and approach we followed to learn the planning action models. Chapter 6 gives the details on how the learning model is implemented. In Chapter 7, the learned action model is evaluated and the obtained evaluation results are discussed. Lastly, chapter 8 gives the conclusion of this work and our vision of future work of this research.

2 Background

2.1 Artificial Intelligence (AI) Planning

Artificial intelligence planning is the task of finding a course of action to be executed and is performed by agents like robots or computer programs to achieve a specific goal [17]. AI planning is the branch of Artificial intelligence which uses autonomous techniques to solve planning and scheduling problems. It involves the representation of actions and environment, reasoning the effect of actions and efficiently searching for the possible plans in the search space. A planner is the search engine that solves planning problems. The plan (sequence of actions) is produced by the planner as a possible solution to the planning problem. The planning problem consists of an initial state, a goal state and a set of actions. The initial state describes how the world is before executing the course of action (in other words before invoking the planner). The goal state describes how the world should look like after executing the set of actions. The world in which planning takes place is called the planning domain [20]. In our case, the planning domain is office buildings. The plans computed will automatically organise or schedule the device actions to improve user comfort and energy efficiency. To compute effective plans we require the knowledge about the domain of interest. The planning models are defined using Planning Domain Definition Language (PDDL) which is discussed next.

2.2 Planning Domain Definition Language (PDDL)

In 1998, Planning Domain Definition Language (PDDL) was released by Drew McDermott, which has since become the standard representation language for describing planning domains and planning problems [16]. The syntax is inspired by Lisp programming language. The world consists of finite number of objects, the things in the world in which we are interested in. The property of the objects is described using the statements called predicates, which can be true or false [12]. Actions describe how the state of the world changes. An action consists of parameters, preconditions and effects. Preconditions contain predicates which must be satisfied for the effect to take place. Effects contain predicates that simulate action occurrence.

In PDDL, the description of actions that characterize the behaviour of domain is separated from the description of specific objects, initial state and goal state. Thus, a planning problem is formed by combining the problem description and domain description. For the same domain, the domain description is combined with different problem description to create different planning problems. [8]. Domain description can also be extended with other constructs, such as types of parameters, constants that have the same meaning for the planning problems in the given domain.

Initial versions of PDDL does not support the use of numbers in a domain. PDDL 2.1 is the extended version of 1.2, which includes the features like durative-actions and functions which are referred to as numeric fluents [8]. Numerical fluent is like a predicate but its value is a number instead of being either true or false. The list of a declaration of numerical fluent is provided using the function field in domain description. Conditions on numeric expressions are the comparison between pairs of numeric expressions.The numerical fluent in the effects of an action can be updated using assignment operations. These include direct assignment (such as assign) and relative assignments (such as increase and decrease) [8].

Types are strings describing the types of objects being referred to. It is used to classify various entities that share the same characteristics. Requirements in domain file, declare the features that are used [13].

Some of the requirements are:

`:negative-preconditions` allow not in goal description

`:fluents` allow function definitions, arithmetic preconditions and effects with assignment operators.

`:equality` support = as the built-in predicate

Below is the basic structure of domain file [13]:

(define (domain <domain name>

<PDDL code for requirements>

<PDDL code for types>

<PDDL code for functions>

<PDDL code for predicates>

<PDDL code for action schemas>

)

Below is the basic structure of problem file [13]:

(define (problem <problem name>

(:domain <domain name>

<PDDL code for objects>

<PDDL code for the initial state>

<PDDL code for the goal description>

)

2.3 NewSLaVe (NSLV) algorithm

Inductive learning algorithm (ILA) is an iterative and inductive machine learning algorithm which is used for generating a set of classification rule for the training examples [31]. Inductive rule learning solves classification via induction of a ruleset. It is based on the Sequential covering algorithm [18], learning one rule at a time and successively removing the covered examples .

NewSLaVe (NSLV) is a classification algorithm based on inductive rule learning. It is the extension of the genetic iterative approach of the SLAVE algorithm. It is mostly similar to the SLAVE algorithm but it learns a complete rule in each iteration by genetic iterative scheme [18]. The algorithm is explained using Figure 2.1. It takes two inputs, a training set, which has the examples, and the basic structure with the domain variables defined [18]. The loop in the algorithm will learn the subset of rules needed to describe a certain class. Unlike the SLAVE algorithm, NSLV learns the complete rule instead of learning only the antecedent of the rule. For evaluating the rule, the examples that are not covered by the previous rule should be considered. The penalization module in the algorithm, penalize the previously learned rules to obtain new and different rules [19]. Unlike the SLAVE algorithm, it marks the examples that are covered by the learned rules. The iterative process will be terminated when the addition of a new rule does not improve the completeness degree of the fuzzy rule set [18].

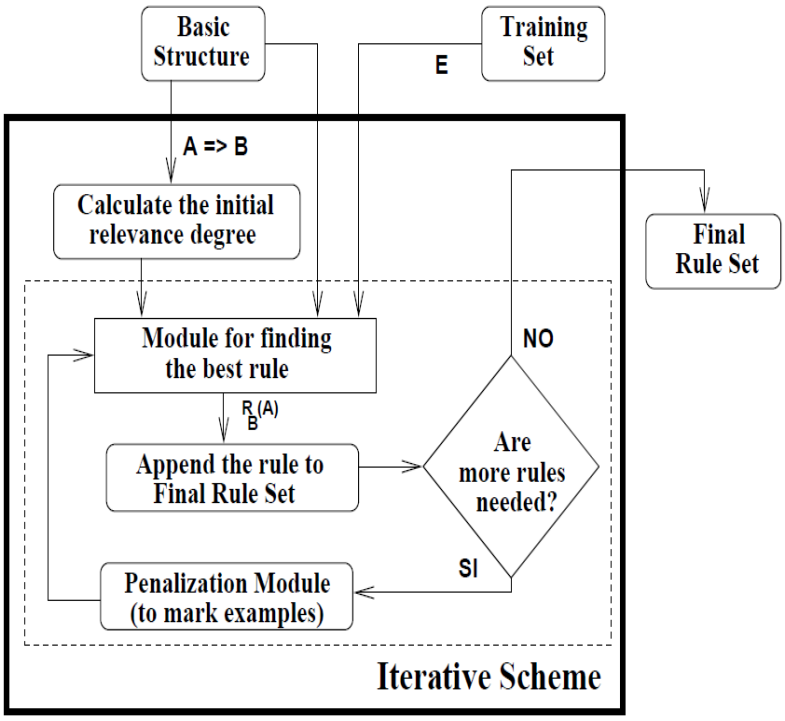


Figure 2.1: NSLV algorithm [18]

NSLV uses a Disjunctive Normal Form (DNF) fuzzy rule set with weight on each rule. The weight will take the value in the interval [0,1]. It denotes the accuracy of the rule created. In other words, it is the rate between the examples correctly covered and the total number of examples covered by the rule [18].

In this work, NSLV algorithm is used to learn the rules that model the action's states, as its learning time is fast and the accuracy is high compared to other algorithms like SLAVE. Also, NSLV can learn both numerical and logic fluent together and the relationship between different type of attributes.

2.4 DNF rule model

The DNF rule model is used in our work as it is more compatible and easily interpretable. In the DNF rule model, the input variables can take the possible values from the set of values and the members are joined using the disjunctive operator. Each input variable will have the associated fuzzy domain. If the dataset has n variables, each variable is associated with the fuzzy domain D_i with m_i components. The antecedent of the rule is encoded as a vector of $(m_1 + m_2 + \ldots + m_n)$. It is binary coded, so these components can take values one or zero. The value of the component is one when it corresponds to the value of that particular variable [11].

Suppose the dataset has 3 variables X_1, X_2 and X_3. Each variable is associated with the following fuzzy domains, respectively.

$$D_1 = \{A_{11}, A_{12}, A_{13}\}$$
$$D_2 = \{A_{21}, A_{22}\}$$
$$D_3 = \{A_{31}, A_{32}, A_{33}, A_{34}\}$$

The variables can take the values from their domain. Suppose if the vector is 110111010, then the antecedent of the rule is

$$X_1 \text{ is } \{A_{11}, A_{12}\}, X_2 \text{ is } \{A_{21}, A_{22}\} \text{ and } X_3 \text{ is } \{A_{31}, A_{33}\}.$$

If the variable takes all possible values from domain then it can be removed from the antecedent of the rule [11]. In our example, X_2 takes all possible values of its domain, so the antecedent of the rule is

$$X_1 \text{ is } \{A_{11}, A_{12}\} \text{ and } X_3 \text{ is } \{A_{31}, A_{33}\}$$

This can be interpreted as

$$\text{IF } X_1 \text{ is } \{A_{11} \text{ or } A_{12}\} \text{ and } X_3 \text{ is } \{A_{31} \text{ or } A_{33}\} \text{ THEN } Y \text{ is } B$$

The values of the variable are joined using the disjunctive operator and the variables are joined by the conjunctive operator.

2.5 Genetic algorithm

Genetic algorithm is the random based classical evolutionary algorithm [9]. It uses a search heuristics that has been developed to imitate the process of natural selection and natural genetics [22]. It is used as a problem solving strategy to find the optimal solution.

Figure 2.2 shows the key terms used in a genetic algorithm.

- Individual - any possible solution to the problem
- Population – Group of all individuals
- Search Space – All possible solutions to the problem
- Chromosome – set of parameters (variables) that defines the individual
- Genome – Collection of all chromosomes for an individual

Initially, the population generally consists of a set of randomly generated individuals. Given a fitness function, each individual is quantitatively evaluated with it. Fitness function is used to select the best individual [23]. In other words, the individual with the maximum fitness value will be selected.

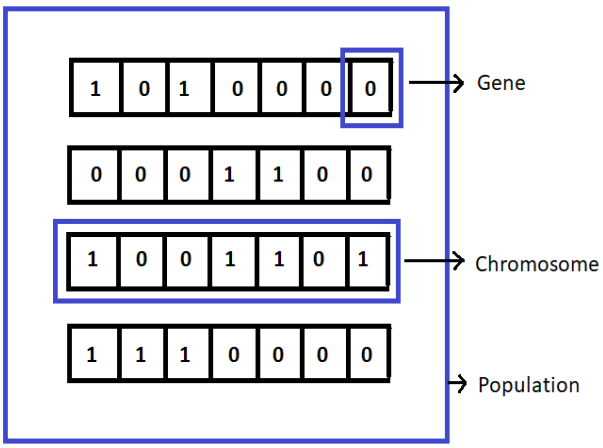


Figure 2.2: Population, Chromosome, Genes

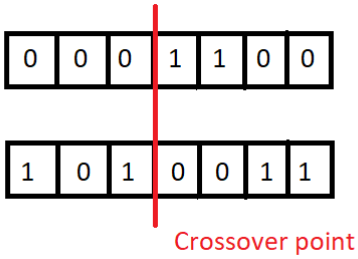
Basic functionality of genetic algorithm:

- Selection - The idea of selection is to select the best individuals to pass their genes on to the next population. Elitist selection is one of the selection techniques in which the members with the maximum fitness value in each generation are guaranteed to be selected [29]. While in the Fitness-proportionate selection technique, the most fit individuals are likely to be selected but not certain [29]. Other selection techniques are Roulette wheel selection, Tournament selection and Scaling selection. The individuals selected are called parents [29].

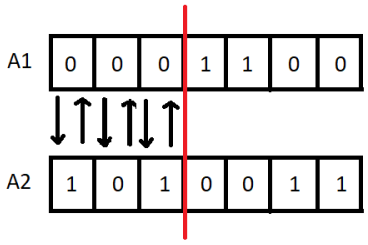
- Crossover - To find the best solution (in genetic algorithms), new individuals should be created from the old ones. Once the individuals (parents) are selected, off-springs (new chromosomes) are produced using a technique called Crossover. Off-spring produced could be a better solution than parents. For each pair of parents to be mated, the crossover point (shown in Figure 2.3a) is chosen at random from within the genes [25]. Single point crossover is most commonly used. The genes after the crossover is interchanged between individuals to produce off-spring as shown in Figure 2.3b. New off-springs, shown in Figure 2.3c are added to the population. However, crossover does not occur always. In that case, the parents are copied to the new population.
- Mutation - After selection and crossover, the population will contain the selected individuals and newly produced off-spring. In order to avoid the chance of same individual being existing in the population, mutation technique is used [29]. In this technique, some of the bits in the bit string are flipped at certain random probability as shown in Figure 2.4 [29]. It is to ensure genetic diversity within the population and to prevent premature convergence [29].

  This process is terminated in one of the following cases [23]:

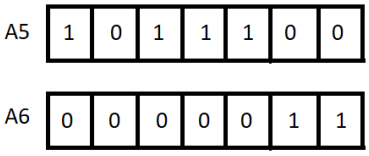
  – if a solution is attained,
  – maximum number of generations is reached,
  – given number of generations, without fitness improvement is performed.



(a) Crossover point



(b) Exchanging genes among parents



(c) New offspring

Figure 2.3: Crossover to get new offsprings

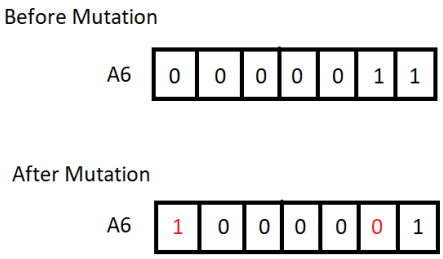


Figure 2.4: Mutation

3 Related Work

The problem of learning action models has been studied for a long time. The first well-known algorithm in learning action models is the Action relation modelling system (ARMS) [30]. ARMS learn action models from a set of successful example plans. It can deal with partial or null observations of intermediate states. It uncovers a number of constraints from plan traces in training data and these constraints are used to build and solve a weighted propositional satisfiability problem with a MAX-SAT solver. The other algorithms like Simultaneous learning and filtering (SLAF) [4] and Learning object-centred models (LOCM) [6] can also learn action models from partial or null observability of intermediate states.They also require the knowledge of the sequence of actions of learning examples.

The other recent approaches like FAMA [1], [2] and [3] learn action models from observations of plan executions. It uses a classical planning compilation approach for learning action models. This approach can handle flexible amount and type of available input knowledge i.e. it can learn from labelled plans (initial state, final state and the full sequence of actions) or even only with minimum observability (fully observed initial state and a partially observed goal state). Compared to other algorithms, this approach can learn from small sets of learning examples.

The approaches mentioned so far can deal only with propositional variables but cannot handle any type of numerical information. This can cause an issue in implementing planning domain learners for some real-world applications. The approaches mentioned above are tested in benchmark domains obtained from International planning competition (IPC) domains. In our work, the domain of interest is commercial buildings like office buildings. The action models associated with controlling of heating and lighting system in buildings are learned. The IoT data that we obtain from sensors like indoor temperature, humidity and air-quality is numerical information. This requires an algorithm that could deal with numerical information.

PlanMiner [28], PlanMiner-O2 [26] and PlanMiner-O3 [27] are the classification algorithms that can learn action models from noisy and incomplete data. These algorithms can learn fixed numerical information and the logical relation between them. PlanMiner uses a series of execution traces of solved plans as input. The plan trace will include initial state, goal state, intermediate states and action. The prestate and poststate of the action could be directly extracted from the plan trace.

Our work is similar to PlanMiner-O3 algorithm, but the input is the IoT data with only the state information. The prestate and poststate of the action is identified from the state information with the knowledge of fluents that can appear in the action. Also our model can learn continuous numerical information in the form of arithmetic operations that should be satisfied for the action to take place.

4 Description of Buildings domain

4.1 Description of domain

In our work, we are trying to learn a planning domain of Office buildings. Modern living environments like office buildings, tend to be equipped with variety of devices. The group of devices includes different sensors and actuators, where, both, the sensors and actuators, provide information about the environment. The important challenge here is to effortlessly emerge from an environment with a variety of embedded devices to an intelligent and computationally capable environment.

The objective is to build an environment that is intelligent, meets the comfort of occupants and achieves energy efficiency. Here, we are using AI planning to efficiently plan the device operation based on the information obtained from the sensors and actuators. The actions described in this planning domain represent the device operations, the functionalities that the specific devices can perform. A device operation changes the state of the associated devices.

The planning domain we have created has the entities like heater, window and light in a room. The actions considered in our work are switch-on and switch-off operations of heater, open and close of window and the switch-on and switch-off operations of light. For example, let us take the actions open and close of window to maintain proper ventilation in buildings. Ventilation is important to maintain good indoor air-quality, to avoid risks of affecting health of occupants. In commercial buildings, where a high number of occupants work in same place, the air-quality should be monitored continuously and when it is above a certain comfort level, ventilators should be used to improve the air-quality. The upper and lower bounds of comfort level are set as threshold values in the initial state in the PDDL problem file. The current state of the environment is taken from the sensor and actuator readings and compared with the comfort level. The goal is to maintain the environment that is conducive for the occupants.

The next section gives the PDDL coding of the domain considered in our work.

4.2 PDDL code of Buildings domain

```
(define(domain OFFICE)
        (:requirements :strips :typing :fluents :negative-preconditions)
        (:types room heater window light occupancy -object)
        (:predicates
            (on-heater ?h - heater)
            (open ?w -window)
            (on-light ?l - light)
            (presence ?p - occupancy)
         )
```

```
10
11          (:functions
12             (temp ?r -room)
13             (air-quality ?r -room)
14             (luminance ?r     room)
15             (temp-threshold-low)
16             (temp-threshold-high)
17             (air-quality-threshold-low)
18             (air-quality -threshold-high)
19             (lum-threshold)
20
21           )
22
23
24     (:action switch_off_heater
25          :parameters (?h - heater ?r - room)
26          :precondition (and (on_heater ?h) (>(temp ?r) (temp-threshold-high)))
27          :effect (and(not(on_heater ?h))(decrease (temp ?r) (- (temp ?r) (temp-threshold-high) )))
28     )
29
30     (:action switch_on_heater
31              :parameters(?h - heater ?r - room)
32              :precondition(and (not(on_heater ?h)) (<(temp ?r) (temp-threshold-low)))
33              :effect(and(on_heater ?h) (increase (temp ?r) (-(temp-threshold-low) (temp ?r))) )
34     )
35
36     (:action open_window
37              :parameters(?w - window ?r - room)
38              :precondition(and (not(open ?w)) (>(air-quality ?r)(air-quality-threshold-high)))
39              :effect (and (open ?w) (decrease (air-quality ?r) (- (air-quality ?r)
                     (air-quality-threshold-high))) )
40     )
41
42     (:action close_window
43              :parameters(?w - window ?r - room)
44              :precondition(and (open ?w) (<(air-quality ?r)(air-quality-threshold-low)))
45              :effect(and (not(open ?w)) (increase (air-quality ?r) (- (air-quality-threshold-low)
                     (air-quality ?r) )))
46     )
47
48     (:action switch_on_light
49              :parameters(?l - light ?r - room ?p - occupancy)
50              :precondition(and (not(on-light ?l)) (presence ?p) (<(luminance ?r)
                     (luminance-threshold-low)))
51              :effect (and (on-light ?l) (increase (luminance ?r)(-(luminance-threshold)(luminance
                     ?r)))
52     )
53
54     (:action switch_off_light
55              :parameters(?l - light  ?p - occupancy)
56              :precondition(and (on-light ?l) (not(presence ?p)))
57              :effect (and (not(on-light ?l)))
58     )
59  )
```

Listing 4.1: Domain file of Buildings domain

```
(:init
     (on-heater h)
     (on-light l)

     ;Set threshold values
     (= (temp-threshold-low) 20)
     (= (temp-threshold-high) 24)
     (= (air-quality-threshold-low) 450)
     (= (air-quality-threshold-high) 800)
     (= (lum-threshold) 400)

)

(:goal
    (and
          (>(temp r) 20)
          (<(temp r) 24)
          (> (air-quality r) 450)
          (< (air-quality r) 800)
          (>(luminance r) 400)
     )
)
```

Listing 4.2: Initial state and goal state of Buildings domain

5 Learning Planning Action Model

This chapter describes the core of our work. We start by explaining the learning task. The input to the learning task with an example is also shown. After that, we describe the overall methodology of our work. Then in the following sections, the approach we followed to learn planning action model is explained in detail.

5.1 The Learning Task

In AI planning the world is represented as a conjunction of fluents. The fluent can be a literal fluent whose value could be true or false or a functional fluent whose value is numerical [15]. Each fluent has a list of arguments, where the number of arguments is called arity. Given a set of objects Ω, the set of fluents is induced by assigning the objects to the arguments in fluents. To interpret the incompleteness of state s, two interpretations can be used, Open-world assumption (OWA) or Closed-world assumption (CWA). CWA considers the missing fluents as false. OWA is adopted in our work, in which the fluents which are unobserved in a state are considered as unknown (neither true nor false).

The world state is built upon the set of propositions X and the set of numerical variables X', $F = XUX'$. Thus, a state s is a pair of $< X(s), X'(s) >$ where the value of $X(s)$ is true or false and $X'(s)$ is a numerical value over the set of real numbers, $\mathbb{R}$. Let A be the set of actions in planning domain. An action $a \in A$ can be represented as a tuple $< name, par, pre, eff >$ where

- $name$ is the name of the action
- par are the parameters of the action
- pre are the preconditions that must be true for the execution of the action. It consists of
  - numeric part (pre_{num}), involving arithmetic operations like ($<, \leq, =, \geq, >$), for example ($exp < exp'$)
  - propositional part pre_{prop}, the propositions defined over X
- eff is the effects that take place after the action is executed. It consists of
  - numeric part (eff_{num}) in the form of x, op, exp where $x \in X$ is the numeric fluent affected by the operation op ($+=, -=, =$)
  - (eff^+) and (eff^-) list, which contains respectively the propositions added and deleted after the execution of an action.

Here exp and exp' are the arithmetic expressions involving variables from X'. An action a is applicable in a state s only when both propositional and numerical preconditions are satisfied in s. When an action a is applied in state s, it results in new state s'. The predicates present in $eff^+(a)$ are added to $F(s')$ and the predicates present in $eff^-(a)$ are removed from $F(s')$, and the numerical fluent x of numeric operation $\{x, op, exp\}$ is modified according to the exp and op specified.

Let us consider the action `switch_off_heater` from the buildings domain described in Chapter 4. The parameters are `heater` and `room`. The action `switch_off_heater` will switch-off the heater, when the room temperature is greater than a high-temperature threshold. For the action to be executed, the precondition `(on-heater ?h)` which is a pre_{prop} and the expression `((temp ?r -room) >(temp-threshold-high))` should be satisfied. While effect (eff^-) of the action is `(not(on-heater ?h))` and (eff^num) is `(decrease (temp ?r) (- (temp ?r) (temp-threshold-high) ))`

An action model is defined as an action whose par, pre and eff is not instantiated with the objects in the world. This work focuses on learning the preconditions and effects of an action with the propositional and numerical variable.

A learning task L is a tuple <M, O>, where

- M is an initial partial model of all actions with action headers and parameters of the action. For the domain model described in Chapter 4, the initial action model is shown in Listing 5.1
- O is the observations of the states.

The learning task is to find the preconditions and effects of the action such that we know the fluents that can appear in preconditions and effects of the action. It involves determining which fluents will shape the precondition and effect of an action.

```
(:action switch_on_heater
            :parameters(?h- heater ?r- room)
            :precondition( )
            :effect( )
)
(:action switch_off_heater
            :parameters(?h- heater ?r- room)
            :precondition( )
            :effect( )
)
(:action open_window
            :parameters(?h- heater ?r- room)
            :precondition( )
            :effect( )
)
(:action close_window
            :parameters(?h- heater ?r- room)
            :precondition( )
            :effect( )
)
(:action switch_on_light
            :parameters(?h- heater ?r- room ?p - occupancy)
            :precondition( )
            :effect( )
)
```

```
26
27  (:action switch_off_light
28              :parameters(?h- heater ?r- room ?p - occupancy)
29              :precondition( )
30              :effect( )
31  )
```

Listing 5.1: Initial action model of all actions

5.2 Methodology

Figure 5.1 gives an overview of the approach. The first 3 steps are related to dataset creation for each action. The inner loop shows how the rules are learnt for each class and how the covered examples are penalized to learn new rules from other examples. The outer loop is to repeat the steps to cover all the actions of the domain.

5.2.1 Creation of Dataset from Observations

The set of observations (Obs) is taken as input. Let (ξ) be a partial action model of an action. $I_{\xi,F}$ contains the fluents that can appear in $pre(\xi)$, $add(\xi)$ and $del(\xi)$ of the action model. For example, $I_{heater-on,F} = \{on - heater, temp\}$ are the fluents that can appear in action model `switch_on_heater`. As we know the fluents that can appear in each action model, a table can be created for each action model by taking only that particular fluents.

The actual space of the schemata can be bounded by the following constraints [2]:

- $del(\xi) \subseteq pre(\xi)$.
- $del(\xi) \cap add(\xi) = \varnothing$.
- $pre(\xi) \cap add(\xi) = \varnothing$.

F_j, where $1 \leq j \leq n$, are the fluents which make up the states and Value is the value associated with the fluent. The value of a fluent will be true/ false or numerical value depending on whether it is propositional fluent or numerical fluent respectively. If a fluent does not appear in an example, then its value is taken according to the world assumption [28]. As in this work, the open-world assumption is considered, the value will be filled as Not available (NA) in the table.

From the observations, the data is extracted for each action. Depending on the value change in the fluents, the action that has been taken place is identified. This is possible as the fluents of each action are known. The first observation is considered as prestate (Class_1) and the next observation as the poststate (Class_2) as shown in Table 5.1. Similarly, a dataset is created for each action. This dataset will contain the prestate and poststate for every action that has been extracted from observations [28].

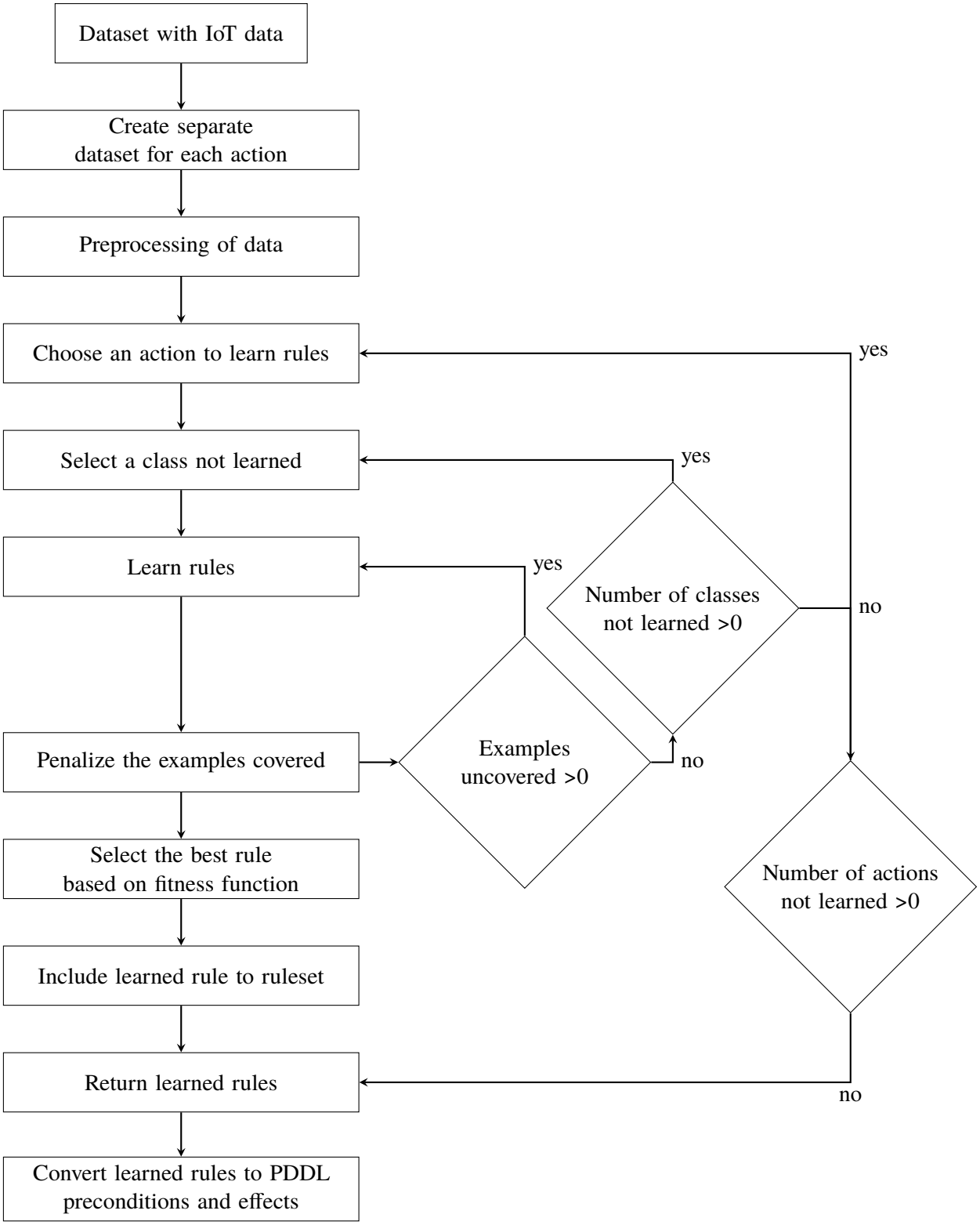


Figure 5.1: Overview of the approach

Table 5.1: Dataset extracted from observations

F_1	F_2	...	F_m	Class
$Value_1$	$Value_1$	...	$Value_1$	$Class_1$
$Value_2$	$Value_2$	...	$Value_2$	$Class_2$

5.2.2 Data preprocessing

Preprocessing of data is done after the dataset has been created for each action. Following are the preprocessing steps done before learning rules. This guides the model to learn more accurate rules:

- The numerical attributes are discretized
- The relation between the numerical attributes in the action, if any, is obtained
- The relation between the prestate and poststate values in each numerical attribute is realized.

Discretization is the process of converting continuous values to discrete values. The numerical fluents in the buildings domain have continuous values. Considering all the values is not practical and significantly affects the computation time. So, the numerical fluents can be discretized in such a way it does not affect the accuracy of the learned model. Each action in our domain has mostly one numerical attribute. Even in case if an action has two numerical attributes, they may not be related to each other. So we need a discretization technique that is suitable for discretizing one-dimensional data. For example, the action `switch-on-heater` has only one numerical fluent which is `temp`. In another case, the action `open-window` and `close-window` have two numerical fluents `temp` and `air quality`. But they are not dependent on one another. In the domain we consider, we may or may not have dependent numerical attributes for an action. So, the discretization technique we have chosen is Fischer-Jenks algorithm. In Fischer-Jenks algorithm the natural breaks in data values are identified using iterative process [24]. This algorithm finds the right way to split the range of data based on how the values are spread.

The second preprocessing step is to check if there is a relation between the numerical attributes of each action. The relation between the numerical attributes in an action can be analysed using different comparison operators like (<, >, <=, >=, =) [27]. Instead of checking with all the operators, the relation between the numerical attributes are analysed using only the operators (<, >=, =). This is because there will not be a significant change in the accuracy of our model for considering only these operators. Also, it reduces the computation time. The number of rows that are satisfied by the comparison operator is calculated and if it is above the fixed threshold value, then the relation between the numerical attributes holds. Otherwise, those numerical attributes are considered to be independent of each other.

Next, the relation between the numerical values present in prestate and poststate is calculated [27]. In the training examples, each pair of prestate and poststate is taken and the difference between the value in prestate and the value in poststate is calculated. This step is required during the conversion of extracted DNF rules to PDDL code.

5.2.3 Learn rules

Once the dataset is created with poststate and prestate information for every action, the rules are learned using the NSLV algorithm to create planning action models [28]. The rules created by NSLV are of the form

IF C_1 and C_2 and ... C_m **THEN** Class is B

where C_i is a condition, where C_i is of the sentence form X_h is A, with A a fuzzy label of the domain of the variable X_n. X_n is an element of X, the set of antecedent variables of the rule [18]. The domain of the variables that correspond to logical fluents will have the value 'true' or 'false', while the numerical variable's domains contain a label for each different value in the examples of the corresponding variable. B is the value that represents a class of a particular problem.

Algorithm 5.1 is based on the Sequential covering strategy of NSLV [18]. In the algorithm, E is the collection of examples and f is the fitness function. Initially, the ruleset is empty. At each iteration, a new rule is extracted and added to the ruleset. The examples that are covered by the extracted rules are penalized. Those examples will not be considered in the next iteration, to learn new rules. The algorithm terminates when Performance is zero or less. In other words, if the newly extracted rule does not increase the degree of completeness (number of new examples the new rule can explain), the algorithm terminates[18].

Algorithm 5.1 Sequential covering strategy to learn rules

Input: Set of examples E
Output: A learned rule set L
Procedure:

```
Initialize L as an empty ruleset
for each class C do
    Learn a rule R
    While Performance (R, E) > 0, do
        Add rule R to the ruleset L
        Penalize the examples covered by the learned rule R
        Learn a rule R
    end while
end for
Return ruleset L
```

The rule extracted should be the best representation of the set of examples. The best rule is selected using the genetic algorithm. As described in Section 2.5, the genetic algorithm uses a fitness function to select the best rule. The fitness function will differ based on the application of the genetic algorithm. In our work, the genetic algorithm is used to select the rule with high accuracy.

Fitness function here measures the consistency and completeness of the rule and the product of them will return the accuracy of the rule [28].

Completeness indicates that each example from the class will be explained by at-least one rule of that class. The completeness degree of a rule [28] is measured as:

$$C(R) = \frac{n^+(R)}{n_{class}} \tag{5.1}$$

where,

- R is the rule
- $n^+(R)$ is the number of examples covered by the rule of a certain class
- n_{class} is the number of elements of the examples set of this class.

Consistency indicates that if an example is explained by the description of one class, then it cannot be the member of another class. The consistency of a rule [28] is measured as :

$$Co(R) = \begin{cases} 1, & if \mathrm{n}^-(R) = 0 \\ \frac{n^+(R) - n^-(R)}{n^+(R)}, & if n^-(R) < n^+(R) \end{cases} \tag{5.2}$$

where,

- $n^+(R)$ is the number of examples covered by the rule
- $n^-(R)$ is the number of examples uncovered by the rule

Given a rule R and set of training examples E, the fitness function of the genetic algorithm is:

$$fitness(R, E) = C(R, E) * Co(R, E) \tag{5.3}$$

The fitness function ensures that the rule with both completeness and consistency at highest degree is selected [28].

5.2.4 Conversion

The DNF rules that are extracted should be converted to PDDL actions. Rule's antecedent can be directly converted to the fluents, which can be used to create PDDL preconditions and effects.

- Actions' preconditions are taken directly from the antecedent of the prestate [18].
- Actions' effects are obtained by deriving the difference between the prestate and poststate. In other words, the changes that should be made in the prestate to obtain the poststate [18]. If the effect of an action has numerical fluent, it is updated using assignment operation like $assign$ or relational assignments like $increase$, $decrease$ [18]. The difference between the prestate and poststate value that is calculated in preprocessing step is considered here. If the difference is less than zero, then the numerical fluent is updated using decrease assignment. If the difference is greater than zero, then the numerical fluent is updated using increase assignment.

6 Implementation

In the previous chapter, we explained the methods and algorithms required in learning the planning domain. The implementation is done using Python. This chapter shows in detail the intermediate steps and the how the extracted rules are converted to PDDL code.

6.1 Dataset creation

In our work, we are trying to learn the action model of 5 actions `switch-on-heater`, `switch-off-heater`, `open-window`, `close-window`, `light-on` and `light-off`. We have used the dataset from UCI machine learning repository [32], [33] and CASAS dataset [5]. We have taken only the attributes that are required to train our model. The training data of our model has the measurements of temperature in $^{\circ}C$, $air-quality$ in ppm and $luminance$ in Lux of a room in a building, presence of occupant and the information about whether the devices are currently on or off. As explained earlier, the dataset will have only the observed fluents and there will be no information about the action that has taken place between the observed fluents. By observing the changes in fluent values in the observations, the action taken place is identified. Although, `off-heater` predicate is equivalent to `not(on-heater)`, in our work, two different fluents `on-heater` and `off-heater` are used to differentiate the switch-on and switch-off actions of the heater. Similarly, the predicates `open` and `close` are used for the object window, `on-light` and `off-light` for the object light.

As we have the knowledge of the fluents that can appear in the action model, from the entire dataset, 6 datasets are created, one for each action. Then with the change in the fluent value (which corresponds to a device action like on-heater), the prestate and poststate of the action are identified and that information is added to the dataset.

Then during preprocessing steps, the relation between the numerical attributes in an action identified. The threshold value is set to 80%. If the same relation between the numerical attributes is satisfied for 80% of examples in class, then we consider that two attributes are related to each other. If the threshold value is not met, then it is considered that the two numerical attributes are independent of each other, then the condition is learnt for each numerical attributes separately. Then the numerical attributes are discretized and the cut-points are calculated.

6.2 Learn rules

Once the dataset is created for each actions and preprocessing of data is done, the next step is to learn the rules for each class in an action. Learning the rule here involves selecting the attribute-value pairs that define the antecedent of the rule that best fits the set of examples. An attribute-value pair

is the relation between the attribute and its values, which can be represented in the form of <a rel v>, where a is an attribute, rel denotes a relational operator ($<, >, =$ . . . .) and v is a specific value or set of values of the attribute. The attribute-value pair is selected by the genetic algorithm.

The chromosome label is formed by combining the possible values of the attributes of an action as shown in Figure 6.1. The attribute which is of boolean type takes true and false as label, while the continuous attributes take the cut-points calculated during pre-processing as label. The initial population of chromosomes are generated from the randomly selected examples [19]. For example, the chromosome 1000100000 corresponds to `close-window = True and air-quality = V3`. For each individual generated, the fitness value is calculated using Equation 5.3. Then the new individuals are created using selection and mutation techniques. The tuples <attribute, value> with the highest fitness value is selected as it best fits the examples [19].

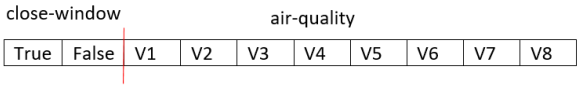


Figure 6.1: Chromosome label

6.2.1 Extracted Ruleset

In each action's dataset, the rule is learned for the classes prestate and poststate. Below are the rules extracted for each action:

For action switch_on_heater:
IF on-heater=FALSE and temp < 18.9 THEN Class is prestate
IF on-heater=TRUE and temp ≥ 22 THEN Class is poststate

For action switch_off_heater:
IF off-heater=FALSE and temp ≥ 23.6 THEN Class is prestate
IF off-heater=TRUE and temp < 21.2 THEN Class is poststate

For action open_window:
IF open=FALSE and air-quality ≥ 753 THEN Class is prestate
IF open=TRUE and air-quality < 454 THEN Class is poststate

For action close_window:
IF close=FALSE and air-quality < 427 THEN Class is prestate
IF close=TRUE and air-quality ≥ 559 THEN Class is poststate

For action switch_on_light:
IF on-light=FALSE and presence=TRUE and luminance < 392 THEN Class is prestate
IF on-light=TRUE and luminance ≥ 570 THEN Class is poststate

For action switch_off_light:
IF off-light=TRUE and presence=FALSE THEN Class is prestate
IF off-light=FALSE THEN Class is poststate

6.3 Conversion

Once the rules are extracted, they are converted to PDDL code. As discussed in Section 5.2.4, the precondition is formed directly from the rule of prestate. Let us conside the prestate and poststate rules of action `switch-off-heater`.

The rule IF off-heater=FALSE and temp ≥ 23.0 THEN Class is prestate converted as, (and(not(off-heater)) (temp ≥ 23.0)).

The effects of the action is the changes that is made in prestate to obtain poststate. The numerical fluents in the effects are updated using an assignment operation or relational assignments as explained in Section 5.2.4. In our extracted rules, the poststate has the numerical fluent with conditional operator. For example, the poststate of action `switch-off-heater` is IF off-heater=TRUE and temp<20.25 THEN Class is poststate. To achieve the poststae from prestate, the predicate `off-heater` will be set as true and the numerical fluent `temp` is updated as `decrease ( (temp) (- (temp) 20.25))`. The `decrease` is obtained from preprocessing step and the value is taken from the poststate rule. The arithmetic operator is decided based on the comparison operators in prestate and poststate.

Initially for the identification of the action taken place between the observed fluents, we have used two different predicates for each object. For example, `on-heater` and `off-heater` for the object `heater`. Generally in PDDL coding, we typically use the positive predicate and negate them. So during the conversion of rule to PDDL, we replace the negative predicates by the negation of the positive predicates. For example, `(off-heater)` is converted as `(not(on-heater)` and `(not(off-heater)))` is converted as `(not(on-heater))`.

Then the converted code is inserted in the empty action model. Listing 6.1 shows the learned action model.

```
1          (:action switch-on-heater
2              :parameters (?h- heater ?r- room)
3              :precondition (and (not(on-heater)) (<(temp ?r)18.9)
4              :effect (and ((on-heater ?h)) (increase (temp ?r) (- (22) (temp ?r))))
5          )
6
7          (:action switch-off-heater
8              :parameters (?h- heater ?r- room)
9              :precondition (and (on-heater ?h)  (>(temp ?r) 23.6))
10             :effect (and (not(on-heater ?h)) (decrease (temp ?r) (- (temp ?r) (21.2) )))
11         )
12
13         (:action open-window
14             :parameters (?w- window ?r- room)
15             :precondition (and (not(open ?w)) (>(air-quality ?r) 753))
16             :effect (and(open ?w) (decrease (air-quality ?r) (- (air-quality ?r) (454))))
17         )
18
```

```
19          (:action close-window
20              :parameters  (?w- window ?r- room)
21              :precondition (and (open ?w)  (<(air-quality ?r) 427))
22              :effect (and (not(open ?w)) (increase (air-quality ?r) (- (559) (air-quality ?r) ))))
23          )
24
25          (:action switch-on-light
26              :parameters(?l- light ?r- room ?p - occupancy)
27              :precondition(and (not(on-light ?l)) (presence ?p) ((<(luminance ?r) 392 ))
28              :effect (and (on-light ?l) (increase (luminance ?r) (- (570) (luminance ?r) )))
29          )
30
31          (:action switch-off-light
32              :parameters(?l- light ?r- room ?p - occupancy)
33              :precondition(and (on-light ?l) (not(presence ?p)))
34              :effect (and not((on-light ?l)))
35          )
```

Listing 6.1: Learned action model

7 Evaluation

This chapter discusses the evaluation study that we perform to assess the learned domain model. The learned planning domain is evaluated using three different criteria. We first evaluate the learned model using domain error rate, then the accuracy of numerical information in the learned model is assessed. Finally the ability of learned model to achieve goal state (objective) is evaluated.

Noise is included in the observations, that are used for training the model, by randomly modifying the values of the fluent according to the percentage of noise [27]. This is to check the robustness of the learned model against the noise. Similarly, to test the model's capacity to learn from incomplete information is tested by having missing values of fluents in observations [27].

7.1 Evaluation using domain error

The domain error is measured by comparing the learned planning domain model with the reference domain model. The domain error is calculated by the number of extra or missing fluents in the learned domain's preconditions and effect divided by the number of total fluents that can appear in the preconditions and effects [27]. Domain error for different percentage of noise in observations is shown in Figure 7.1.

$$Domain\_error = \frac{\sum_a error(a)}{|Actions|}$$

where,

$error(a)$ is the number of extra or missing fluents in an action a, where $a \in Actions$

$|Actions|$ is the number of actions

7.2 Accuracy of numerical information

As our domain has mainly numerical information, it is important to check the accuracy of the numerical information in preconditions and effects of the learned domain. It is calculated as how much percentage, the learned value has deviated from the expected value. The expected value is the value of the fluent which is taken from the reference domain.

$$d = \frac{|\ expected - learned\ |}{expected}$$

where,

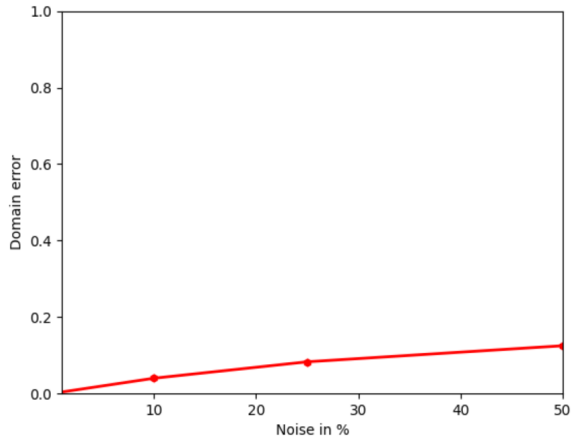


Figure 7.1: Domain error rate with respect to different percentages of noise

d is the deviation

$expected$ is the value that is expected to be learnt

$learned$ is the value that is learned by the domain model

$$Total\_deviation = \frac{\sum_{(a)} d}{n}$$

where,

n is the total number of actions in the domain

The deviation is calculated for preconditions and effects of the planning domain model separately. The average value shows how much the value of the numerical fluents is deviated in the domain. The results are shown in Table 7.1.

Table 7.1: Error in numerical fluents with respect to observations

Noise	Precondition	Effect	Average
0%	6.03%	19.18%	12.6%
10%	9.88%	17.55%	13.72%
25%	9.13%	16.7%	12.92%
50%	9.05%	18.85%	11.95%

7.3 Evaluation using test problems

The learned domain model's capability of achieving the objective (goal state) is evaluated using test problems. The test problems were created with initial state and goal state. The goal state is the same in every test problem, which is to maintain the ambient temperature, air quality and

luminance. An example of a test problem is shown in the Listing 7.1. The planning problems are solved using MetricFF planner which supports numerical fluents. We had about 150 observations of fluents. About 120 observations were used to train the model and 30 observations are used to test the learned domain model. Each observation is taken as the initial state in the PDDL problem file and tested if a plan can be generated to reach the goal state from the initial state using the learned domain model. First, the test problems were tested against the reference domain to check if the plan exists. Then for those problems which have a plan are tested against the learned domain model. The results are shown in Figure 7.2 and Figure 7.3 for different percentage of noise and missing observations of fluent values respectively.

```
(:init
     (on-heater h)
     (on-light l)

)

(:goal
    (and
          (>(temp r) 18)
          (<(temp r) 24)
          (> (air-quality r) 350)
          (< (air-quality r) 650)
          (>(luminance r) 500)
     )
)
```

Listing 7.1: Test problem

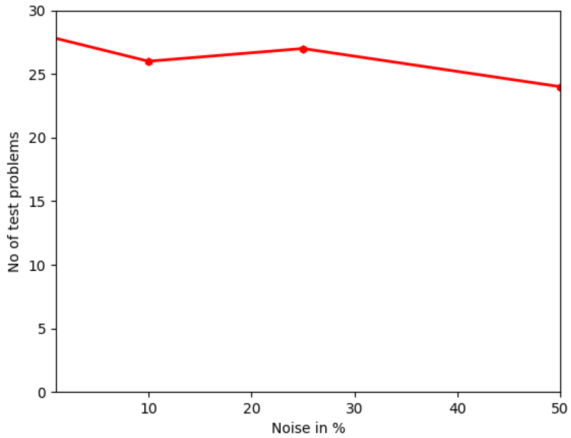


Figure 7.2: Number of test problems solved with respect to different percentages of noise

7.4 Discussion

The results in Figure 7.1 show that the approach could learn planning domain completely when there is no error or missing value in the observation. The error rate gradually increases as the percentage of noise increases. This shows that our learning model could tolerate the noise in observations in a better way even without any special technique to reduce noise.

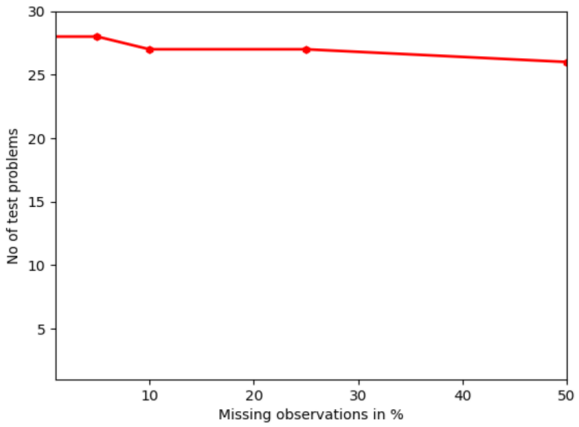


Figure 7.3: Number of test problems solved with respect to different percentages of missing observations

When there is noise in the observations, the numerical fluents will not be missed in the action, but the accuracy of the numerical information will be affected. To measure how much the quality of numerical information is affected due to presence of noise in the observation, we use different criteria. The percentage of deviation of learned numerical information from the expected value is shown in Table 7.1. The results show that, presence of noise increase the deviation in numerical information learnt. Though there is not much change in average deviation, there is significant changes in the precondition and effects of an action. This will affect our model, as the action may not take place at the right time as the precondition is not met. While the deviation in effects of an action may result in not reaching the required state.

When there is a deviation in the numerical values learnt, it affects the capability of planning domain to meet the goal state or in other words to maintain the ambient temperature in the room. This is tested using the test problems. The result shown in Figure 7.2 explains how the presence of noise affects the planning domain´s problem-solving capability. From Figure 7.3, we can see that there is not much variation in the number of test problems solved successfully due to missing values of fluents in observations. This is because the learning model has the knowledge of fluents that can occur in each action. Also, missing of fluent values in observation does not affect the accuracy of numerical information much. One main reason for this is, we discretize the numerical attributes before extracting the rules. When there is missing values in the numerical attributes, with the numerical information present, the cut-points could be generated, which varies only to certain acceptable level.

8 Conclusion and Future works

8.1 Conclusion

In this work, a planning domain for Office buildings to control the operations of heating, ventilation and lighting systems is defined. Then an approach is proposed to learn planning domains from sensor data. As the planning domain is learned from sensor data, the approach should be able to handle the continuous numerical information. In the buildings planning domain defined, numerical information (sensor data) plays an important role in making the appropriate decision at the right time. The proposed approach will meet the requirements as it could learn the propositional fluents, continuous numerical information and the relation between them. The sensor data collected may have noise and missing data values. So our learning model's robustness to noise and missing observation is analysed. When there is no noise and the observations are complete, the planning domain model is learned completely. Then the learning model is tested against a different percentage of noise or missing values in observations. The results show that the learning model has good resistance to the missing values of fluents in the observations, but it could handle the noise only to some extent. The results also show that the learned planning domain model can solve more than 90% of test problems when there is no noise or missing values in the observations.

8.2 Limitations

The concentration of the work happened to be on learning planning domain of buildings. This approach requires the sensor data describing the state of the building at different time, to train the model. For buildings domain, as we have many datasets with sensor information, the proposed approach will be very suitable. But this is not the case for all other application domains. When a dataset with sensor information is unavailable for the planning domain to be learned, the proposed approach will not be suitable.

8.3 Future works

As future work, we could extend the approach to learn disjunctions in preconditions and conditional effects in the effects of an action. This enables the possibilities to learn the planning domain with complex real-world actions. Considering all the possibilities of arithmetic relation between numerical fluents becomes difficult when the number of numerical fluents to be learnt is higher in an action of planning domain. This problem can be solved by including new pre-processing

techniques that can handle the complex arithmetic relation between numerical fluents. Furthermore, it will be interesting to test the learned model in the real-world. Testing in real-world will help us analyse how much the approach is suitable to improve energy efficiency in commercial buildings.

If the sensor data is available for benchmark planning domains, which are used in International Planning Competition (IPC), the proposed approach can be used to learn those planning domains and the results can be compared with other domain learners like ARMS.

Bibliography

[1] D. Aineto, S. Jiménez Celorrio, E. Onaindia. "Learning action models with minimal observability". In: *Artificial Intelligence* 275 (2019), pp. 104–137. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2019.05.003. URL: http://www.sciencedirect.com/science/article/pii/S0004370218304259 (cit. on pp. 16, 25).

[2] D. Aineto, S. Jiménez, E. Onaindia. "Explanation-Based Learning of Action Models". In: Mar. 2020, pp. 3–20. ISBN: 978-3-030-38560-6. DOI: 10.1007/978-3-030-38561-3_1 (cit. on pp. 25, 33).

[3] D. Aineto, S. J. nez, E. Onaindia. "Learning STRIPS Action Models with Classical Planning". In: *CoRR* abs / 1903.01153 (2019). arXiv: 1903.01153. URL: http://arxiv.org/abs/1903.01153 (cit. on p. 25).

[4] E. Amir, A. Chang. "Learning Partially Observable Deterministic Action Models". In: *CoRR* abs / 1401.3437 (2014). arXiv: 1401.3437. URL: http://arxiv.org/abs/1401.3437 (cit. on pp. 16, 25).

[5] *CASAS Data Set*. URL: https://archive.ics.uci.edu/ml/machine-learning-databases/00506/. (accessed: 12.07.2020) (cit. on p. 39).

[6] S. Cresswell, T. Mccluskey, M. West. "Acquiring planning domain models using LOCM". In: *The Knowledge Engineering Review* 28 (June 2013). DOI: 10.1017/S0269888912000422 (cit. on p. 25).

[7] D. D'Agostino, B. Cuniberti, P. Bertoldi. "Energy consumption and efficiency technology measures in European non-residential buildings". In: *Energy and Buildings* 153 (2017), pp. 72–86. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2017.07.062. URL: http://www.sciencedirect.com/science/article/pii/S037877881730676X (cit. on p. 15).

[8] M. Fox, D. Long. "PDDL2.1: An extension to PDDL for expressing temporal planning domains". In: *J. Artif. Intell. Res. (JAIR)* 20 (Dec. 2003), pp. 61–124. DOI: 10.1613/jair.1129 (cit. on pp. 17, 18).

[9] M.-C. Frunza. "Chapter 2D - Genetic Algorithms". In: *Solving Modern Crime in Financial Markets*. Ed. by M.-C. Frunza. Academic Press, 2016, pp. 151–161. ISBN: 978-0-12-804494-0. DOI: https://doi.org/10.1016/B978-0-12-804494-0.00010-3. URL: http://www.sciencedirect.com/science/article/pii/B9780128044940000103 (cit. on p. 21).

[10] G. Gao, J. Li, Y. Wen. "Ënergy-Efficient Thermal Comfort Control in Smart Buildings via Deep Reinforcement learning"". In: *CoRR* abs / 1901.04693 (2019). arXiv: 1901.04693. URL: http://arxiv.org/abs/1901.04693 (cit. on p. 15).

[11] D. Garc'ia, A. González-Muñoz, R. Pérez. "Overview of the SLAVE learning algorithm: A review of its evolution and prospects". In: *International Journal of Computational Intelligence Systems* 7 (Nov. 2014). DOI: 10.1080/18756891.2014.967008 (cit. on p. 20).

[12] I. Georgievski, M. Aiello. “Automated Planning for Ubiquitous Computing”. In: *ACM Comput. Surv.* 49 (Dec. 2016). DOI: 10.1145/3004294 (cit. on p. 17).

[13] I. Georgievski, M. Aiello". *Planning for Pervasive System*. June 2018. URL: http://www.cs.rug.nl/~aiellom/pdf/AielloGeorgievskiCh2a.pdf (cit. on p. 18).

[14] I. Georgievski, T. A. Nguyen, F. Nizamic, B. Setz, A. Lazovik, M. Aiello. “Planning meets activity recognition: Service coordination for intelligent buildings”. In: *Pervasive and Mobile Computing* 38 (2017), pp. 110–139. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2017.02.008. URL: http://www.sciencedirect.com/science/article/pii/S1574119217301050 (cit. on p. 15).

[15] A. Gerevini, A. Saetti, I. Serina. “An approach to efficient planning with numerical fluents and multi-criteria plan quality”. In: *Artificial Intelligence* 172 (May 2008), pp. 899–944. DOI: 10.1016/j.artint.2008.01.002 (cit. on p. 31).

[16] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, D. Weld. “PDDL - The Planning Domain Definition Language”. In: (Aug. 1998) (cit. on p. 17).

[17] M. Ghallab, D. Nau, P. Traverso. *Automated Planning: Theory and Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Amsterdam: Morgan Kaufmann, 2004. ISBN: 978-1-55860-856-6. URL: http://www.sciencedirect.com/science/book/9781558608566 (cit. on pp. 15, 17).

[18] A. González-Muñoz, R. Pérez. “Improving the genetic algorithm of SLAVE”. In: *Mathware Soft Computing* 16 (Jan. 2009), pp. 59–70 (cit. on pp. 19, 36, 37).

[19] A. González-Muñoz, R. Pérez. “SLAVE: A genetic learning system based on an iterative approach”. In: *Fuzzy Systems, IEEE Transactions on* 7 (May 1999), pp. 176–191. DOI: 10.1109/91.755399 (cit. on pp. 19, 40).

[20] R. Jilani. “Automated Domain Model Learning Tools for Planning”. In: Mar. 2020, pp. 21–46. ISBN: 978-3-030-38560-6. DOI: 10.1007/978-3-030-38561-3_2 (cit. on pp. 15, 17).

[21] S. Jiménez, T. De la Rosa, S. Fernandez, F. Fernández, D. Borrajo. “A review of machine learning for automated planning”. In: *The Knowledge Engineering Review* 27 (Dec. 2012). DOI: 10.1017/S026988891200001X (cit. on p. 15).

[22] H. Lingaraj. “A Study on Genetic Algorithm and its Applications”. In: *International Journal of Computer Sciences and Engineering* 4 (Oct. 2016), pp. 139–143 (cit. on p. 21).

[23] S. Mardle, S. Pascoe, M. Tamiz. “An investigation of genetic algorithms for the optimization of multi-objective fisheries bioeconomic models”. In: *International Transactions in Operational Research* 7 (Jan. 2000), pp. 33–49. DOI: 10.1111/j.1475-3995.2000.tb00183.x (cit. on pp. 21, 22).

[24] M. North. “A Method for Implementing a Statistically Significant Number of Data Classes in the Jenks Algorithm”. In: vol. 1. Jan. 2009, pp. 35–38. DOI: 10.1109/FSKD.2009.319 (cit. on p. 35).

[25] C. Reeves. “Genetic Algorithms”. In: vol. 146. Sept. 2010, pp. 109–139. DOI: 10.1007/978-1-4419-1665-5_5 (cit. on p. 22).

[26] J. Segura-Muros, R. Pérez, J. Fdez-Olivares. “Learning Numerical Action Models from noisy and partially observable states by means of Inductive rule learning techniques”. In: 2018, pp. 46–53 (cit. on p. 25).

[27] J. Segura-Muros, R. Pérez, J. Fdez-Olivares. “Learning Planning Action Models with Numerical Information and Logic Relationships Using Classification Techniques: 18th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2018, Granada, Spain, October 23–26, 2018, Proceedings”. In: Jan. 2018, pp. 372–382. ISBN: 978-3-030-00373-9. DOI: 10.1007/978-3-030-00374-6_35 (cit. on pp. 16, 25, 35, 43).

[28] J. Segura-Muros, R. Pérez, J. Fdez-Olivares. “Using Inductive Rule Learning Techniques to Learn Planning Domains”. In: Jan. 2018, pp. 642–656. ISBN: 978-3-319-91478-7. DOI: 10.1007/978-3-319-91479-4_53 (cit. on pp. 25, 33, 36, 37).

[29] A. Thengade, R. Dondal. “Genetic Algorithm – Survey Paper”. In: *IJCA Proc National Conference on Recent Trends in Computing, NCRTC* 5 (Jan. 2012) (cit. on pp. 21, 22).

[30] K. Wu, Q. Yang, Y. Jiang. “ARMS: An automatic knowledge engineering tool for learning action models for AI planning”. In: *Knowledge Eng. Review* 22 (June 2007), pp. 135–152. DOI: 10.1017/S0269888907001087 (cit. on pp. 16, 25).

[31] X. Wu. “Inductive learning: Algorithms and frontiers”. In: *Artificial Intelligence Review* 7 (Apr. 1993), pp. 61–124. DOI: https://doi.org/10.1007/BF008490799 (cit. on p. 19).

[32] F. Zamora-Martínez, P. Romeu, P. Botella-Rocamora, J. Pardo. “On-line learning of indoor temperature forecasting models towards energy efficiency”. In: *Energy and Buildings* 83 (2014), pp. 162–172. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2014.04.034. URL: http://www.sciencedirect.com/science/article/pii/S0378778814003569 (cit. on p. 39).

[33] F. Zamora-Martínez, P. Romeu, P. Botella-Rocamora, J. Pardo. *SML2010 Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/SML2010#. (accessed: 30.06.2020) (cit. on p. 39).

All links were last followed on December 18, 2020

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature