Investigation into the Theoretical Properties of, and the Relationship Between, AI Planning Domain Models.

By
Salihin Farag Shoeeb

A report submitted as a requirement to transfer to third year

School of Computing and Engineering
Dept. of Informatics
University of Huddersfield

TABLE OF CONTENTS

TABLE OF FIGURES

Report outline

The outline of the report will include the following sections.

- Introduction
- Literature review
- Measurement system
- Experiments and empirical analysis
- Future work

Section 1 (Introduction): a brief introduction to Artificial Intelligence Planning. AI applications and techniques which are related to this research area such as machine learning, automated planning and other related techniques.

Section 2 (Literature review): this section summarizes the related previous work. Through this section a set of planning domain model and tools will be presented supported by examples.

Section 3 (Metric algorithms): this section describes in details the domain model metric that is being developed in terms of components, input, and output.

Section 4 (Experiments and empirical analysis): this section illustrates the performance of our domain model metrics. The domains are mainly from the International Planning Competitions (IPC). This section also include a comparative evaluation based on IPC database resource (handcrafted and systematic domain models) to shows the complexities, differences and similarities of the domain models at hand in terms of number of states, number of operators/actions, A degree of arity of operators, A degree of arity of predicates and average number of preconditions and postconditions in each operator.

Section 5 (future work): this section includes the future direction of the domain model metric tool. Finally, a brief description of the paper we aim to publish in PlanSig 2011 – University of Huddersfield will be attached to this report as an appendix.

1 Section one: introduction

1.1 Abstract

Over the past decade, there have been substantial advances in knowledge representation (KR) techniques for AI planning. Typically, planners search a space of solution to find a suitable and most accurate sequence of actions to achieve a specific task from a set of initial and goal states. However, the progress in this field still cannot cope with the ever increasing of the complexities of modern systems, which makes knowledge representation an expensive and error prone process. Planning is considered as one of artificial intelligence fields where knowledge representation (KR) is extremely critical. However, a little work has been aimed at "measuring" domain models; the aim of this research is to develop a set of criteria and metrics to assess the accruing and complexity of a particular classical planning problem domain model. To reach that point the system has to have enough knowledge and knowledge has to be well represented for the problem at hand. In this report, we have outlined the prototype the system and design planning domain model metric tools.

1.2 Introduction

The prevalent view in the Automated Planning community is that the logical separation of planning engine and domain model representing the application and problem at hand is advantageous in that algorithmic and representational concerns can be dealt with relatively independently. The separation between planning engine (planner) and domain model in much of the research in AI planning brings into focus the role and nature of the domain model. Whereas properties of, comparisons between, and analysis of AI planning engines is well developed, similar science of domain model analysis is underdeveloped.

Domain models, in particular those used for the AI planning competition (IPC), have been predominantly hand crafted, and this can be seen as a limiting factor on the use of planning engines. As far as we are aware, all the domain models used in the International Planning Competitions (1996 - present) have been hand crafted. The importance of this knowledge formulation process to the success of applications, and to make planning engines more accessible and open to community use, has led to the establishment of the international competition on knowledge engineering for AI planning (ICKEPS).

It has long been acknowledged that there can be a range of different encodings for the same domain, but this has not been seen to be a problem, as it is a particular encoding that is used as a benchmark, not the domain itself. These concerns are independent of coding language used, as whatever is used there are always choices to be made in modelling.

The need to investigate the properties of a planning domain model has become more accurate given the advances made in domain model acquisition. In this case, quality of the learned domain model has to be measured in terms of the reliability. On other hand, pair of domain models for the same planning problem have to be measured in terms of their specification. Generally, a comparative technique is used as a main component of measuring the domain models of the problem at hand. From the point of view of modelling planning domain ,comparative technique means how close to one model M is another M'.

Differences and similarity of particular domain models can be easily mapped from specification such as, states, actions, predicates, and effects. Some other specification, such as semantics features can be measured to provide overview of the complexity of one domain model from the other. It is expected to find many similarities and differences between the two planning domain models for the same problem, especially when using different methods in describing a particular planning problem (i.e. hand crafted GIPO[1] and autonomic systems LOCM[2]).

1.3 Basic definition

The aim of this research is to address a set of metrics to evaluating planning domain models in terms of their specifications. The first step towards this goal is to make some intuitive definitions to be used during the research.

Definition: ***A*** model of some reality ***R*** is a mathematical or logical abstraction ***A*** of that reality, where $\boldsymbol{I}: \boldsymbol{A} \leftarrow \boldsymbol{R}$ is the interpretation function mapping the abstraction to the reality.

Let us consider changes to the reality - call them plans, made up of sequences of actions. Let assume we have an operator in the model which maps through the interpretation function I to an action in the reality. We say that a state S' in the model is consistent with a real state S if the assertions made in $I(S')$ are true is S. For example, if the model is

$$ontop(a,b) and\ ontable(b)$$

then the (obvious) interpretation of this is consistent with a reality where a is on top of b, and b is on a table. Note this notion of consistency can never be formally "proved", because the reality is not itself a formal system! We use the symbol "=" : so $I(S') = S$ means $I(S')$ is consistent with S. In the area of formal modelling, this leads to a notion of model validation.

As well as states, we can use the interpretation function to map operator applications in a model to real actions, in an obvious way.

Definition: a model is valid if for any plan in the model X', where X is the interpretation of that plan in the real world, then $IoX = X'oI$.

In short, this means that given any state (S) in the model, if we interpret it and observe a plan being carried out to produce a real state of the world (W), then applying Cm to S and taking an interpretation of the resulting state will produce a state of the world not inconsistent with W. Of course, something could change the operation of the plan in the real world that is not modelled in the abstraction.

Definition: A model is adequate, if for any real problem (initial state S, goal state G, solution plan X) that is required in the reality, there are corresponding structures S', X' in the model such that $I(S') = S$, and $I(X') = X$, and the formal operation of X' on S' results in a state G' such that $I(G') = G$.

Hence, validity is saying that the model where it applies is consistent with reality, and adequacy is saying that the model capture all the foreseen requirements of the problem area. These are informal notions, however, as we are matching formal behaviour against the informal world of requirements. In the same way, the validity of any piece of software is not open to complete formal analysis.

1.4 Planning Problem

Plan is denoted as 3-tuple of P= (O, s_0, g) where O, s_0, g refer to a set of operators, initial state, and goal state respectively. Plans are produced by searching a space of actions until a sequence of feasible actions are reached that can carry out the given tasks. In other word, an planning problem comprises a world description: initial, goal states and a domain theory. A domain theory defines a transition state, the way in which the applicable actions change a state of the domain to a new state and their relations with resource. Based on these inputs, planners produce a sequence of executable actions that can reach the goal state from the initial state.

Mainly, one of the keys design philosophy of planning system is domain-independence. In principle, a domain-independent planner works with any planning domain. However, developing domain independent algorithms in many types of problem domains is not reasonable. As a solution of this, some restrictive assumptions were made [3]:

- System has a finite set of state.
- Problem domain fully observable (complete knowledge)
- The outputs of actions are deterministic.
- The system is static (no dynamics events that can change the problem domain)
- Goals are restricted (planner handles only specified goals)
- A solution plan is a linearly ordered finite sequence of actions.
- Actions have no duration
- Change is not allowed during the plan time (off-line planning)

Such planning techniques that accept these assumptions are formally recognized as classical planning. Unluckily, these attempts to improve the domain independence have decreased the usability of planning systems because they brought many restrictions that are infeasible for real world applications.

Classical planners are based on semantic descriptions (i.e., preconditions of actions) provided by a domain model. Recently, an additional expert knowledge are required by planning problems which is may not be fully accessible due to the limitation and complexities of the

domains for the experts to provide such knowledge. Hence, it is difficult to develop learning systems in order to learn knowledge as human contributions are restricted[4].

A typical example of a classical planning problem is a Dock Worker Robot (DWR). This problem involves a number of cranes, locations, robots, containers, and piles. Robot starts moving out from one of the locations. The goal is to transport each container to its final destination in a desired order. Consider, for example, an instance of a simple DWR problem with only one robot, two container, two crane and two locations 1 and 2 as given in Figure 1.

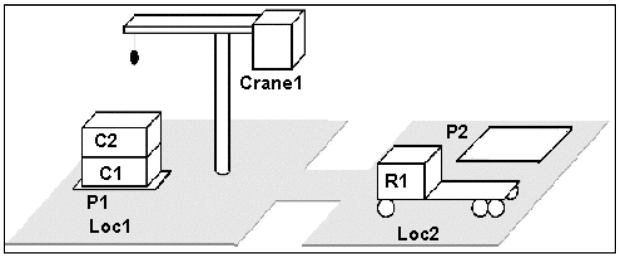

Figure 1.1 Dock Worker Robot Domain

In this particular example the robot at location 2 and the containers at location 1 in the initial state and the goal is to have the containers to location2. The actions in the DWR domain describe driving the robot from location 2 to location1,take a container off of the location1 by the crane1, loading a container into the robot at that location1, driving the robot from location1 to location2, unloading a container from the robot by crane2, and put a container on location2. These actions are given in their standard planning domain description language (PDDL) [5] in Figure 1.1.

```
;; moves a robot between two adjacent locations
  (:action move
     :parameters (?r - robot ?from ?to - location)
     :precondition (and (adjacent ?from ?to)
        (at ?r ?from) (not (occupied ?to)))
     :effect (and (at ?r ?to) (not (occupied ?from))
        (occupied ?to) (not (at ?r ?from)) ))
;; loads an empty robot with a container held by a nearby cr
  (:action load
     :parameters (?k - crane ?c - container ?r - robot)
     :vars (?l - location)
     :precondition (and (at ?r ?l) (belong ?k ?l)
        (holding ?k ?c) (unloaded ?r))
     :effect (and  (loaded ?r ?c) (not (unloaded ?r))
        (empty ?k) (not (holding ?k ?c))))
;; unloads a robot holding a container with a nearby crane
  (:action unload
     :parameters (?k - crane ?c - container ?r - robot)
     :vars (?l - location)
     :precondition (and (belong ?k ?l) (at ?r ?l)
                  (loaded ?r ?c) (empty ?k))
     :effect (and (unloaded ?r) (holding ?k ?c)
                (not (loaded ?r ?c))(not (empty ?k))))
;; takes a container from a pile with a crane
  (:action take
     :parameters (?k - crane ?c - container ?p - pile)
     :vars (?l - location ?else - container)
     :precondition (and (belong ?k ?l)(attached ?p ?l)
                        (empty ?k) (in ?c ?p)
                        (top ?c ?p) (on ?c ?else))
     :effect (and (holding ?k ?c) (top ?else ?p)
                  (not (in ?c ?p)) (not (top ?c ?p))
                  (not (on ?c ?else)) (not (empty ?k))))
;; puts a container held by a crane on a nearby pile
  (:action put
     :parameters (?k - crane ?c - container ?p - pile)
     :vars (?else - container ?l - location)
     :precondition (and (belong ?k ?l) (attached ?p ?l)
                        (holding ?k ?c) (top ?else ?p))
     :effect (and (in ?c ?p) (top ?c ?p) (on ?c ?else)
                  (not (top ?else ?p)) (not (holding ?k ?c))
                  (empty ?k))))
```

Figure 1.2: DWR Domain model in PDDL

The assumptions of classical planning, and the usual mechanisms for solving it, are rather restrictive, and most real problems are neoclassical. The main differences between classical

and neoclassical planning techniques are: in classical planning, for any problem domain consists of group of nodes (search space), every node mapped a partial plan, whereas in neoclassical planning node considered as a set of several partial plan[3]

Many well-known approaches to relax classical planning assumptions have been made, including HTN planning[4, 6-7], MDPs(Markov decision processes) [8], temporal planning [9], and so on. Nonetheless, classical planning algorithms are still restricted to limited categories of planning domains as most of practical planning problems do not satisfy the early mention assumptions of classical planning [3].

1.5 Structure of planning domain model

Generally, a domain model is denoted by $D = \langle P, O \rangle$.

- P: a set of predicate includes variables, atoms, predicates, and constants.
- O: a set of applicable operators. Operator is composed of action preconditions and effects.

Each of which enclosed set of features: a degree of arity of each operator, number of ground and ungrounded terms over the entire domain. This report concentrates on investigation of such features. In PDDL, a domain model can be described in two terms, problem definition and domain definition, see figure 1.1. Problem part consists of a problem name, a domain name, objects, and initial and goal state. Problem can be defined either separate or combined together with the domain definition. Domain includes a domain name, predicates and actions list.

```
define (problem tower10)
   (:domain blocksworld)
   (:objects a b c d e f g h i j )
   (: init (on-table a) (on-table b) (on-table c) (on-table d) (on-table e)
          (on-table f) (on-table g) (on-table h) (on-table i) (On-table j)
          (clear a)  (clear b) (clear c) (clear d) (clear e) (clear j)
          (clear f)  (clear g) (clear h) (clear i) (arm-empty))
   (:goal (and (on a b) (on b c) (on c d) (on d e) (on e f) (on f g)
               (on g h) (on h i) (on i j) )))
```

```
(define (domain blocksworld)
(:predicates (clear ?x)
             (on-table ?x)
             (arm-empty)
             (holding ?x)
             (on ?x ?y))
 (: action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
               (not (arm-empty))))
(:action putdown
  :parameters  (?ob)
  :precondition (holding ?ob)
  :effect (and (clear ?ob) (arm-empty) (on-table ?ob)
               (not (holding ?ob))))
(:action stack
  :parameters  (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob))
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
               (not (clear ?underob)) (not (holding ?ob))))
(:action unstack
  :parameters  (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
  :effect (and (holding ?ob) (clear ?underob)
               (not (on ?ob ?underob)) (not (clear ?ob)) (not (arm-empty)))))
```

Figure 1.3 Planning problem in PDDL

1.6 Objective

Given the great interest in developing the automated planners, domain model interest is still very modest. This research focuses on developing a set of metrics to measure the planning domain models that are cratered in PDDL format. For this purpose, the domain model structure and specification represent the fulcrum in this research. The aim of the research is to develop measurement tools to measure one planning domain model for a particular problem to another. Complexity of one planning domain models can be determined by the similarity and the differences to another, for the same problem.

Figure 1.1 shows a very simple classical hand crafted domain model (Block World) where features such as states, predicates, actions and others can be easily identified and calculated.

The idea of developing planning domain model metrics comes from the usage of deferent technique to create and design a domain model for a particular planning problem. Techniques mention in section 2 part 2.3 are the most used ones to create the domain models. By running a comparative study on some domain model produced by different techniques for the same problem, extra details are found with some of them, see figure 1.4. the complete domain models in appendix A.

```
(:action
   do_up
   :parameters
   (?Nuts1 - nuts ?Hub2 - hub ?Wrench4 - wrench ?Jack3 - jack)
   :precondition
   (and
      (zero_state0)
      (nuts_state0 ?Nuts1)
      (hub_state0 ?Hub2 ?Jack3)
      (wrench_state1 ?Wrench4)
      (jack_state0 ?Jack3 ?Hub2))

   :effect
   (and
      (nuts_state1 ?Nuts1 ?Hub2)
      (not (nuts_state0 ?Nuts1))
      (hub_state2 ?Hub2 ?Jack3)
      (not (hub_state0 ?Hub2 ?Jack3))
      (jack_state2 ?Jack3 ?Hub2)
      (not (jack_state0 ?Jack3 ?Hub2)))
)
```

Produced by LOCM

```
(:action do_up
       :parameters ( ?W - wrench ?H - hub ?J - jack ?N - nuts)
       :precondition (and
            (have_wrench ?W)
            (unfastened ?H)
            (jacked_up ?H ?J)
            (have_nuts ?N)
       )
       :effect (and
            (not (unfastened ?H))
            (not (have_nuts ?N))
            (fastened ?H)
            (loose ?N ?H)
        )
    )
```

Produced by GIPO

Figure 1.4 Parts of tyre domain produce by different techniques

2 Section Two: Literature Review

2.1 Planning modelling language

2.1.1 Planning Domain Definition Language - PDDL

Planning Domain Definition Language (PDDL) [10] has become a de-facto standard for specifying STRIPS-like planning domains and problems with various extensions.

The creation of a planning domain models require a formal method to define all related features of the domain within the desired class of problems. Presently, the most used language for describing planning domains is PDDL (Planning Domain Definition Language). PDDL supports defining parameterized planning operators, objects in each instance of the planning problem, the initial and the goal state of the world, and associated conditions.

Since 1971, the PDDL developed rapidly with the passage of time. In the standard version of PDDL, the state of the problem is defined by a list of facts; properties are hold either true or false at the time.

Operators compose of preconditions and effects (postcondition) lists that have to be true or false for an action to be executed STRIPS-like[11]. Effects are responsible for which propositions that will be added and removed at the time of the action execution. Another version of PDDL provides an optional support for ADL[12] domain preconditions (conjunctions, alternatives and quantified statements). Later version has been updated to meet probabilistic, temporal or metric features expression of the problem worlds. The latest version of this language is PDDL3.1.

2.1.2 Action Description Language (ADL)

This part, briefly describes an important planning description language, the Action Description Language (ADL). The ADL [5] relaxed some of the restrictions assumptions in the STRIPS language and provided some flexibility to encode more realistic world domains. The Problem Domain Description Language [3] was introduced as a standardized syntax for representing ADL, STRIPS, and other languages

2.1.3 Object-Centred Language (OCL)

OCL has been developed by (Donghong Liu and T.L.McCluskey) at the University of Huddersfield. [13] Described OCL as a tool-supported language for domain developers. OCL aims to provide a language that representing the domain models in the classical tradition of AI Planning considering the structure and dynamics of domains.

2.1.4 New Domain Description Language (NDDL)

NDDL is a robust planning language devolved by NASA Ames [14] in support of the spacecraft (Vehicles and Habitats). NDDL aims to simplify knowledge representation of planning problems, such as power management or automated rendezvous in future manned spacecraft.

2.2 Types of planning domain models

Based on a set of planning problem specification and assumptions, domain models can be subdivided into several different types. Varieties of criteria are identified by the International Planning Competition (IPC). Firstly, whether the output of a particular action fully predictable (Classical domain) by the planning algorithm or not (non-predictable-probabilistic domains)? In the case of probabilistic domains, planning algorithms have either prepare alternative plans (contingency plans), if an action should fail, or they have to be ready to re-plan (plan repair).

Some more criterions are depended on the actions' features' complexity such as temporal feature (e.g., actions have durations or are continuous processes, which can be executed in parallel). The simple domain models use propositional or first-order logic to encode and describe a planning problem at hand (classical domain).

In this research particularly focus on the classical domain models that are defined in PDDL format.

2.3 Domain Model Acquisition

Within the AI Planning area, we can categorize methods of producing domain models containing actions, ready for use in a planning engine, into three areas:

- **Hand crafted:** perhaps with the use of knowledge acquisition tools such as GIPO [15] or ItSimple [16]. Virtually all domain models used in applications or research are hand crafted. The hand crafted method mainly based on the abilities of experts to define and describe their experiences formally and completely, which is difficult sometimes. Additionally, creating a domain model using this method is time and effort consumed.

- **Translation:** produced using a translator from another formal model [17-19] as was the subject of ICKEPS-09

- **ML methods:** learned from examples [2, 20-22] : In respect of the reasons mention at the beginning of this section, researchers have started to develop and explore a new tools using ML notations to create domain models with less time, effort and in complete form by learning from a set of example or plan traces (observable environment) such as(i.e [21, 23]), some others, such as LOCM[2] ,LAMP[24] have presented to learn with no knowledge provided(non-observable environment).

Given the problems of dynamically changing worlds, domain model maintenance, and the aims of autonomy, learning from examples is appealing, but up to now has seen relatively little research.

There has been a series of systems which learn domain models from examples [20-30]. Characteristic of all this work is that (a) the output of learning is action representations that can be input to planning engines (such as PDDL) and used immediately as a domain theory for planning (b) input to the process is a set of plan scripts and some planning oriented knowledge, such as state information, predicate descriptions, plan goals and initial state, etc.

The minimal input information required by learning algorithms is a set of plan scripts carried out in a domain.

Some recent work on learning action theories has concentrated on learning with little or no supplied domain knowledge. The ARMS system [25] can form STRIPS-type domain theories from example plan scripts and associated initial and goal states only.

The ARMS system inputs object types, predicate specifications, and action headings, and from plan scripts taken from planning solutions, it learns a domain model. The domain model is synthesised using a constraint solver, inputting two sets of constraints: one set is based on assumed physical, consistency and teleological constraints - for example, every action in the example plan script adds at least one precondition for a future action, actions must have non-empty effects, and so on. The other set of constraints is generated using a type of associative classification algorithm which uses each plan script as an itemset, and extracts frequent itemsets to make up constraints.

While ARMS [25] and LAMP [24] are aimed squarely at helping knowledge engineers create a new domain model, and requires types, predicates states and scripts as input, LOCM is an algorithm learns from plan scripts only. As with ARMS, it outputs a planning domain theory in a PDDL format but it inputs only plan scripts - it does not require representations of initial and goal states, or any descriptions of predicates, object classes, states etc. Rather, it assumes objects referenced in the plan scripts maybe acted on by actions - actions either change the object's state, or leave it where it is. Objects are assumed to be instances of sorts, where sorts behave the same when acted on by actions. Using the example above we illustrate LOCM's assumptions as follows:

- ***Different instances with the same action name induce classes of objects***: so for example c1 and c2 are in the same class, as they appear in the same position (1st) after the same action name (open).

- ***Consecutive actions acting on the same object help form behaviour machines for all objects in a sort****:* for example, it can be assumed that the output state of c2 after action "open" is the same as the input state of action "putaway wrench"

- *Where consecutive actions act on the same set of (2,3 or more) objects, LOCM induces relations between the object's classes*: for example, if in all examples of actions put away jack(x,y); .. ;fetch jack(x,z); ..., where x is not referred to between the two actions, have y = z, then LOCM induces that a relational predicate between the sort of x and the sort of y is true at the state of x after put away jack.

- ***Where subsets of a sort's objects appear in certain slots of actions, then a "static relation" is added to the state on the object.***

3 Section Three: Measurement System

3.1 Generic Measurements

In 1790, the French Academy of Sciences has devolved an invariable standard for all the measures and the weights system. Originally, the system is based on metre for length and the kilogram for mass. Most of other metric units were derived from this notion, for example: weight is measured by gram and capacity measured by litre. In the 20th century, the metric system became the basis for the International System of Units, which is now used officially almost worldwide. In like manner, the notion of measurement has been extended to cover most of the areas affecting the human being life: industry, education, health care, production and others. For example, the area of a rectangle is measured by multiplying its breadth by length in metres. For more complex shapes, the area may be determined by calculating the areas of a collection of rectangles that form a covering of the shape.

In the physical sciences and engineering, units of measurement provided a valuable assistance to assess the quality of physical systems. Additionally, they provide an error checking tools comparable to static type checking commonly found with programming languages. It said that units of measurement can provide similar advantages in the computer systems (specification and design of software).

3.2 Software measurement

Goodman [31] defines software metrics as: *"The continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products"* . Figure 3.1, illustrates of this definition to emphasize that software metrics provide the information needed by engineers for technical decisions. If the metric is to provide useful information, everyone involved in designing, implementing, collecting data for and utilizing a software metrics must understand its definition and purpose.

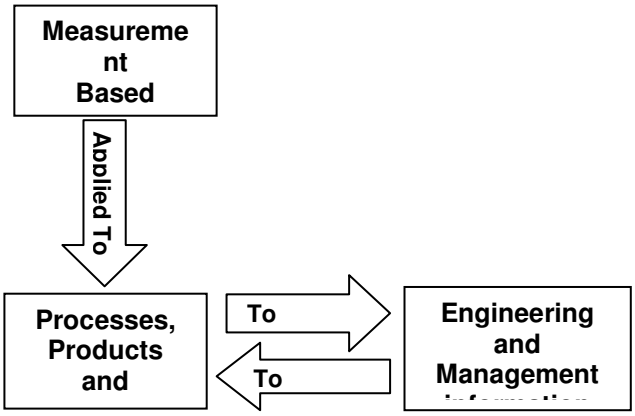

Figure 3.1What are software metrics?

Goodman [31] introduced seven steps to clarify the meaning of designing software metrics.

1. **Objective Statement**

   Metrics aid developer to understand more about:

   - Software products, processes and services.
   - Evaluate our software products, processes and services against established standards and goals.
   - Provide the necessary information to control resources and processes used to produce software.
   - Predict attributes of software entities in the future.

2. **Clear Definitions**

   This step in designing a metric is to agree to a standard definition for the entities and their attributes being measured. Using terms like size, state, and even planning can be interpreted differently based on the context area. These interpretation differences increase when more ambiguous terms like quality, maintainability and user-friendliness are used.

3. **Define the Model**

   Derive a model for the metric. Simply, the model defines a way in which metric calculated. Measuring metrics is depended on their complexity. Simple metrics (called metric primitives) are measured directly and their model typically consists of a single variable. Other more complex metrics are modeled using mathematical combinations of metrics primitives or other complex metrics.

4. **Establish Counting Criteria**

   This defines the measurement mapping system of each metric primitive. Decompose the model into lowest level metric primitives and define the counting criteria used to measure each primitive.

5. **Decide What's Good**

   From the previous steps, questions such as what to measure and how to measure must be answered; through this step a key question like what to do with the results need to be answered. For example, is 10 too few or 100 too many? Should the trend be up or down? What do the metrics say about whether or not the product is ready to ship?

6. **Metrics Reporting**

   This step is to decide how to report the metric, includes:

   - **Report format**: what does the report format, is the metric included in a table with other metrics values for the period?
   - **Data extraction and reporting cycle**: how often the data snap-shot(s) are required and available for use to calculating the metric. The reporting cycle mostly answers two questions: how often the report generated? When is it due for distribution?

7. **Additional Qualifiers:**

   Determining the additional metric qualifiers, a good metric is a generic one. In other word, metric is valid for an entire hierarchy of additional extraction qualifiers, for example period of unplanned outages of the entire product line, a specific product or a release of that product. One could look at outages by customer, business segment or look at them by type or cause.

3.3 Metrics Domain model

Measurement for planning knowledge (domain model) has not yet reached the maturity of other automated planning areas for example, benchmark domain models (i.e block world) are considered as a best way to evaluate planning algorithms (Planners).

At this stage that has emerged in which the need for intelligent systems which are used knowledge-base system has increased the need for a standard to measure quality of this Knowledge. Basically, the need for planning domain measurements comes for the following reasons

- Determine the quality of the current planning domain model or process
- Predict qualities of a domain model
- Improve quality of a domain model

Briefly, a set of metrics integrated into one framework in such a way that each specification of the domain model can be measured independently. Measurement involves comparing two separate domain models' specifications in order to confirm that they describe the same planning problem. To do this assumes that the specifications can be given a common semantics and syntaxes, so that the meaning of the different specifications can be usefully compared.

3.3.1 Metrics Domain Model Motivation

The motivation behind developing domain model metric tools goes back to a paper of S. N. Cresswell and T. L. McCluskey and M. M [2]. In their empirical study they take a closer look at Tyre-World domain models that induced by LOCM and constructed in GIPO. The induced domain is shown extra states in some sorts (i.e ***jack sort***).

Another motivating example comes from a paper presented by Hoffmann [32], in the organization of IPC-4. A set of real-world applications of planning (airport ground traffic control, oil derivative transportation in pipeline networks, model-checking safety properties, power supply restoration, and UMTS call setup) are exploited to assess the performance of planners. Typically, planners' performance is measured by testing them agonist benchmark example instances of the planning problem. It said that a planner is high quality performance if at any time, solves these examples most perfectly. Hoffmann defined the AI planning as a hard problem and computational no system can work properly in all problem examples. Therefore, the kinds of chosen example are crucial for planners testing.

Benchmarking should answer questions such as, how do the planning systems and performance comparing to the standard instances? What are the main reasons for the differences? What could be learned from others that would improve planning domain and performance?

As mention early that there is no specific scientific definition for deciding whether a set of benchmarks are useful or not. However, Hoffmann [32] presented some accepted criteria of how the benchmarks should be:

1. Oriented at applications (an application of the technology should reflected by the benchmark).
2. Diverse in structure (different kinds of structure should be covered by a set of benchmarks, not only restate similar tasks).
3. Accessible for a large number of competing systems (formulated in a language that is understood by the systems).

In ICAPS 2011, Wickler [33] presented four planning domain model features in terms of its characterization: *domain types, relation fluency, inconsistent effects and reversible actions*. These features can be exploited as additional criteria to measure the performance of the planning domain models.

In this work, Wickler aimed to support the domain model designers during formulation process by inferring these features' value from the operator schemes. For example, typing feature (allows declaring the related objects, constants and variables in predicate and operators as well) can be used to limit the number of possible values for variables, or a ground backward searcher may use this information to similar effect in case the planner needs to propositionalize the planning domain.

3.4 Terms and vocabularies

Practically, this report concentrate on the representation of the planning domain models generated in PDDL formalism by both hand and automated planning techniques in terms of their features. Therefore, necessary semantics and vocabularies must be identified. Therefore, We have extended the automated planning (AP) dictionary to include the terms and vocabulary of the planning domain model metrics, including states, operators, predicates, pre and post-conditions, objects, and variables that define states. Table 3.1 introduces such terms and vocabulary to be used during the research.

D	D *is a domain*
S	s *is a set of states*
s_0	s_0 *is an initial state*
g	g *is a goal state*
a	a *is an action*
obj	obj *is a physical object*
o	o *is an operator*
p	p *is a predicate*
N_a	N_a *is a number of action*
N_p	N_p *is a number of predicate*
N_s	N_s *is a number of predicate*

Figure 3.2 Domain Metrics, Terms and Vocabulary

3.4.1 Definition 1: planning problem

A planning problem is donated by $= \langle obj, s_0, g \rangle$. It is typified by being closed world problem, composed of set of physical objects, each of which does different task. A particular object may exist in different states.

3.4.2 Definition 2: domain model – $\mathbf{D} = \langle \mathbf{P}, \mathbf{O} \rangle$

Generally, a domain model can be described as a formal representation to model a particular class of tasks in reality (e.g., logistic tasks, fire fighting). The domain model composes of a set of predicates P, and a set of planning operators O. The planning problem is encoded in a specific domain description language.

Let $D = \{d_1, \ldots, d_n\}$ be a set of $n(D)$ domains associated with a feature $n(O)$ where O a set of operators donated by $O = \{o_1, \ldots, o_n\}$. Each domain model consists of set of operator named action schema. Operator can be considered as 3- tuple, denoted as $O\langle a, +c, -c\rangle$ where +c and -c referred to pre and post conditions respectively.

3.4.3 Definition 3: Action

A planning operator associated with set of values of its parameters. This means that all given values of preconditions and post-conditions have variables grounded. From the set the planning operators *O*, a set of actions *A* in a planning problem *P* is calculated by applying all possible variable instantiations. All applicable actions in a particular state of the world s applicable are denoted $\mathrm{A(s)} = \{\mathrm{a} \in \mathrm{A}: \forall_{\mathrm{p} \in \mathrm{precond(a)}} \mathrm{s} \geq \mathrm{p}\}$

Practically, this research aims to develop a set of metrics to evaluate the performance of the planning domain models. Various types of planning domain model description language such as OCL, ADL, PDDL, and others are being used to create and design planning domain models. The metrics we intend to develop are to evaluate the planning domain models that designed in PDDL formulism. Normally, the PDDL domain description can be paired with number of problem descriptions to create different plans.

In the domain description, actions are described at an abstract level. In addition to preconditions and effects, actions also have parameters which are assigned values when the actions are applied. The preconditions and effects are logical propositions, objects and logical connectives. In this part, basic definitions of the PDDL terminologies are highlighted:

- **Definition 1: Predicate.**
  A boolean-variable function whose values may or may not assigned. Formerly, the predicate is called grounded; in the latter, ungrounded. A predicate may represented as: symbol, consisting of a single atom, or complex, consisting of one or more less complex predicates and logical operators (e.g., negation, conjunction, disjunction, quantifiers).
- **Definition 2: Atom.**
  The simplest predicate does not contain any negations, logical connectives or quantifiers.
- **Definition 3: Precondition of Planning Operator.**

All predicates have to be fully grounded, and their logical value in the current state of the world has to be true for the operator to be applied. In other words, they have to be true facts in the state of the world.

- **Definition 4: Post-Condition of Planning Operator.**
  A logical statement describes a transition manner of mapping a current state into a new state of the domain.

PDDL is considered as a general planning language. Consequently, different planners support different parts of it. In a domain description, from requirements parts, planners can easily recognize whether they can handle it or not. Below are the most commonly-used requirements:

:strips
Domain consists of STRIPS syntax only.
:typing
Domain uses types, to declare types of objects and their parameters.
:adl
parts or the entire domain are defined in ADL syntax, e.g. actions have quantified and conditional effects, disjunctions and quantifiers in preconditions and goals.
:equality
Domain uses "=" predicate as equality

3.5 Domain Model Characterization

From the definition of the domain model, we emphasize a set of general features in which the domain model can be measured in. The Following attributes can be considered as most important generic characteristics:

- Number of objects
- Number of states in domain
- Number of operators/actions in domain
  - A degree of arity of operators
  - A degree of arity of predicates
- Average number of preconditions and postconditions in each operator

There are many problem Challenging planning domain model engineers: a domain model must be abstractions of reality. Therefore, it is natural that a domain model consists of ignorance and uncertainty. The standard method of assigning a unique probability distribution

over possible outcomes is poor in the presence of abstraction because many unmodelled variables are not governed by random chance.

3.6 Modelling Metrics

For the purposes of explaining this part (modelling Metrics), a set of basic specifications to provide the domain model structure in terms of measurement.

3.6.1 Metric 1: No_of_Objects

Let ***D*** a domain model consist of ***n*** object. ***obj*** a set of objects. This metric is responsible for the quantity of objects involved in a particular domain model.

3.6.2 Metric 2: No_of_states

D is a domain model and *S* a set of states involved. Metric 2, is the one to calculate the number of states.

3.6.3 Metric 3: No_of_opretors/No_of_action

Let *D* a domain model and *O/A* is a set of operators/actions.

3.6.4 Metric 4: Degree of operators

Calculating a degree of operators (*opr*), actions or predicates is depended on number of arguments or operands that function takes. Let *O* an operator and *arg* a set of argument involved.

3.7 Algorithm Description

Figure 3.2 is a prototype implementation diagram of the domain model metrics, illustrates and explains the mechanism of our ongoing tool. The empirical work is divided into three stages:

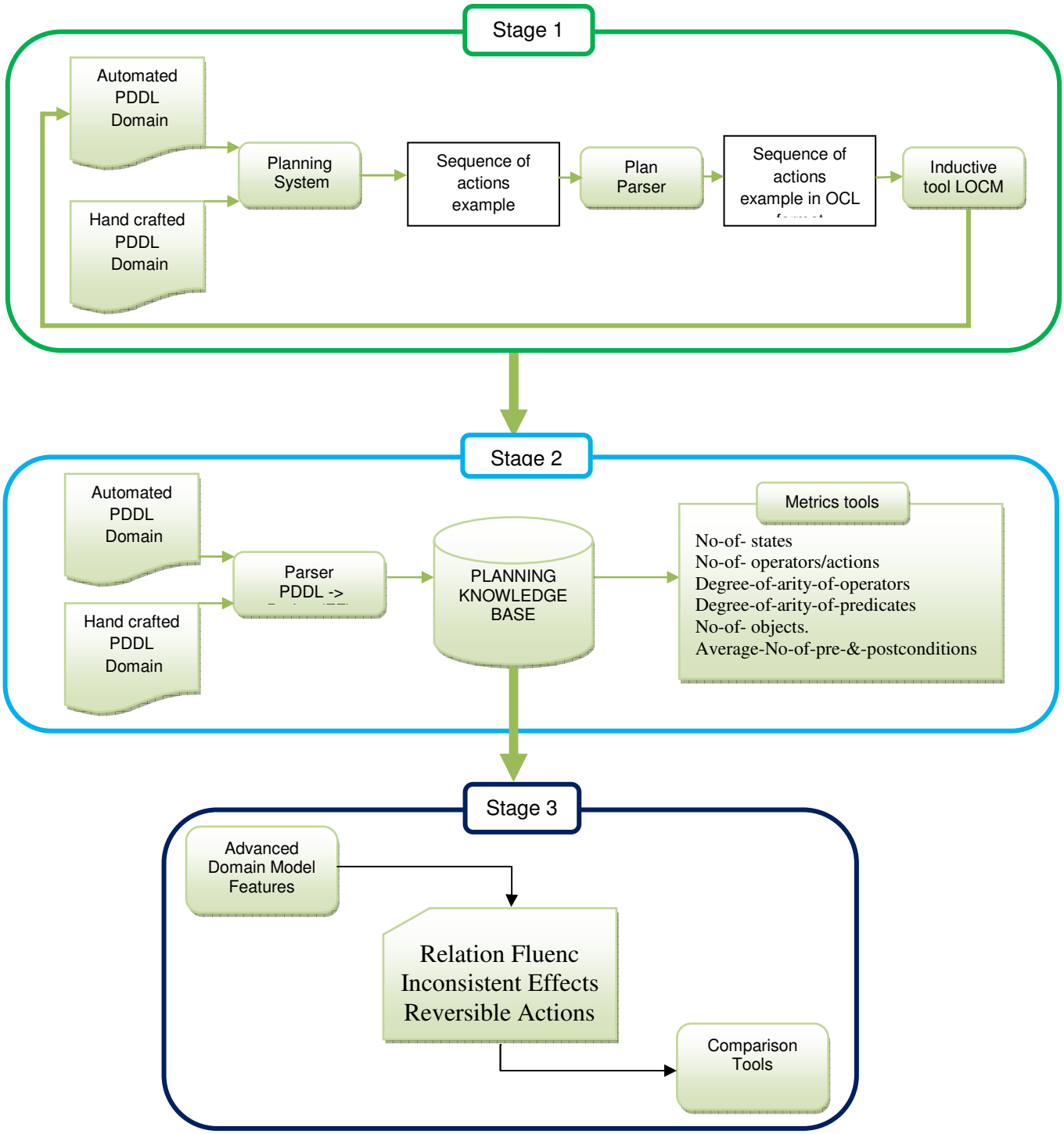

Figure 3.3 Algorithm Architecture

3.7.1 Stage 1:

This stage can be logically described as having three components:

Planner: FF planner(FAST-FORWARD) [34], independent-domain planner works with any planning domain.

Inductive tool: LOCM [2], an ML tool that automatically generate a planning domain model from example training plans. In this experimental, the required examples are provided by FF planner.

Parser: to collect, reformat and translate literal from the output of the planner (FF planner) into OCL notation. Figure 4.1 describe the operation of the parser component.

```
Run FF
        Problem file
        Domain file
        For every domain
          Print:  domain (domain name).
          Print:  sequence_task (#,[
        End
Read x
        If x is an action & not the last action then
          Print:  name of action ( parmeter1, par2,…,parn ),
        Else
          Print :  ],
                    _,_).
        End if
Save (domain name.pl).
Run LOCM : filename.pl
```

Figure 3.4 Parser Psedocode

The following diagram illustrates the mechanism of creating sample plan in OCL [13] format by using a planner (i.e. FF) planning inductive tool such as LOCM [2].

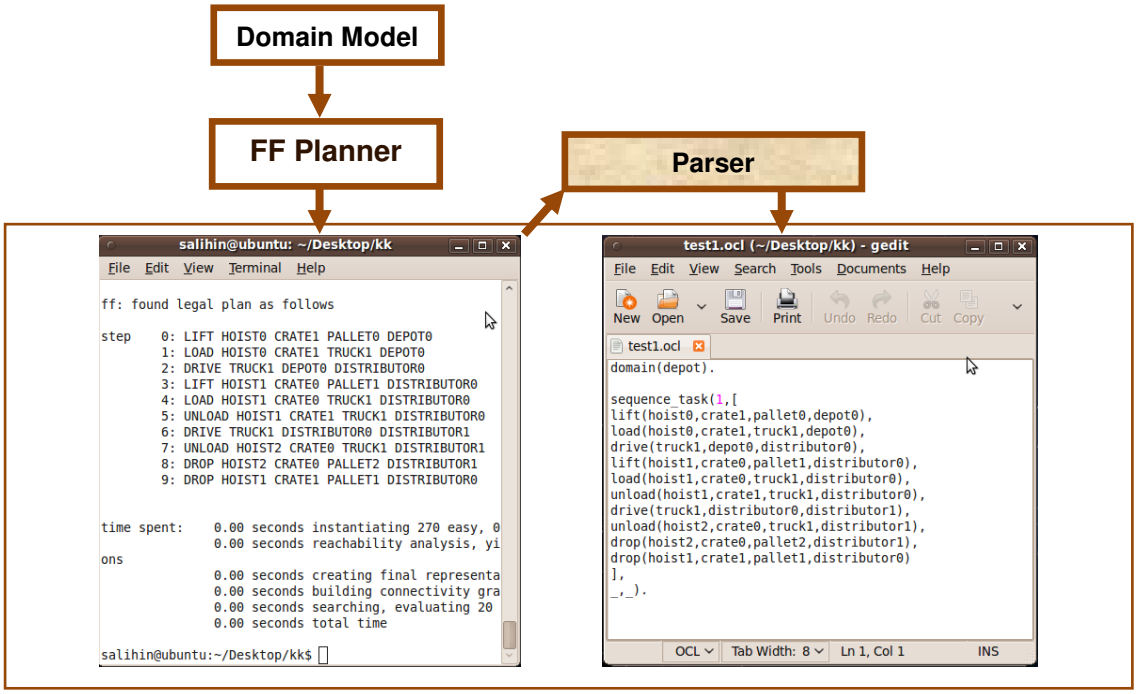

Figure 3.5 Operation of converting PDDL into OCL structure

This stage inputs a planning domain model in PDDL notation (left side of the diagram). The output is a plan in OCL notation (right side of the diagram). The result of this stage is used as input for LOCM to create new domain model (inductive domain model). The research focuses on developing a domain model metrics to find out the complexity in terms of differences and similarity between two models for the same problem.

3.7.2 Stage 2:

A parser is used to reform a PDDL domain into prolog structure. The output of the parser consists of extra details. Therefore, the output is passed to the domain model metric tool to refine and reform it in terms of ignoring some extra details considering the way in which prolog work in. The final result of the algorithms will is considered as a standard database of the metric tool.

3.7.3 Stage 3:

Develop new algorithm using the domain model database, the algorithm should be able to evaluate the domain model in terms of its:

- Number of states in domain
- Number of operators/actions in domain
- A degree of arity of operators
- A degree of arity of predicates
- Average number of preconditions and postconditions in each operator

4 Section Four: Experiments and Empirical Analysis

We have run three empirical experiments through this report:

1. **GIPO-III to design planning domain model**[15]. GIPO supports the planning domain model engineers to describe a particular planning problem formally. GIPO is considered as a handcrafted planning tool.
2. **Formulation a plan example in OCL notation.**

   Tools: planner (FF), planning domain model inductive tools (LOCM), and converter tool.

   Input: handcrafted domain model, plan example in OCL format.

   Output: plan example in OCL format and inductive domain model. This experiment starts by first collecting the set sequence of actions for a particular problem by passing a planning problem domain model to FF planner. Secondly, reforming the set of actions into OCL format to be acceptable as input by LOCM. Finally, the output of LOCM is considered as inductive planning domain model.

Let assign some basic symbols to inputs and outputs of the experiment: D_h denotes a handcrafted domain model, A_{ocl} refers to a set of planning actions in OCL, and D_i Indicates a inductive planning domain model. The following diagram shows the mapping system of this experiment. See figure 3.5, section 3.

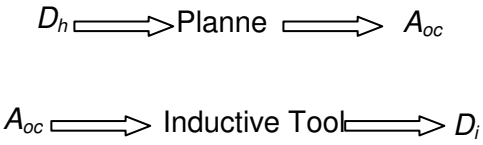

Figure 4.1 Input/output mapping system of the experiment

3. **Creating planning knowledge base.**

Tools: parser (PDDL to Prolog format)

Input: PDDL domain model, see figure 1.1

Output: planning knowledge base composes of planning operator's schema (set of actions, preconditions, and effects)

Example of the result of this experiment:

Figure 4.2 shows Gripper's domain model. The output of the parser is not fully reformed in the structure of the used programming language (prolog). The final result of parser is included some extra symbols, for example "?o" means "o" a variable in PDDL where it is represented by "O" in prolog. Consequently, we wrote a program to maintain and reform the parser's output to be acceptable and manipulated by prolog program see figure 4.2. In this report, the maintained output is considered as a planning domain knowledge base, figure 4.3. The planning domain knowledge base is aimed to be used with the domain model metrics to measure their features.

```
domain(gripper, [strips], _G3058, _G3059, [room(?r), ball(?b), gripper(?g),at-
robby(?r), at(?b, ?r), free(?g), carry(?o, ?g)], _G3061, _G3059, [action(move,
[?from, ?to], [room(?from), room(?to), at-robby(?from)], [at-robby(?to)], [at-
robby(?from)], []), action(pick, [?obj, ?room, ?gripper], [ball(?obj),
room(?room), gripper(?gripper), at(?obj, ?room),at-robby(?room),free(?gripper)],
[carry(?obj, ?gripper)], [at(?obj, ?room), free(?gripper)], []), action(drop,
[?obj, ?room, ?gripper], [ball(?obj), room(?room), gripper(?gripper), carry(?obj,
?gripper),at-robby(?room)], [at(?obj, ?room), free(?gripper)], [carry(?obj,
?gripper)], [])])
```

Figure 4.2 Parser PDDL formulation to Prolog

```
action(move,[From,To],[room(From),room(To),at-robby(From)],[at-robby(To)],[at-
robby(From)],[]),

action(pick,[Obj,Room,Gripper],[ball(Obj),room(Room),gripper(Gripper),at(Obj,Room)
,at
robby(Room),free(Gripper)],[carry(Obj,Gripper)],[at(Obj,Room),free(Gripper)],[]),

action(drop,[Obj,Room,Gripper],[ball(Obj),room(Room),gripper(Gripper),carry(Obj,Gr
ipper),at- robby(Room)],[at(Obj,Room),free(Gripper)],[carry(Obj,Gripper)],[])])
```

Figure 4.3 Planning domain knowledge base

5 Section Five: Conclusion and Future Work:

5.1 Conclusion

This report introduces a novel technique of measuring domain model for AI planning problem. This technique has a set of metrics as a method of AI planning for measuring the domain model specification based on PDDL description. These metrics give a comparative report for pair domain model for the same problem, included statistical value of the domain model features such as number of states, actions and others. In our ongoing project we have shown that domain metric is possible and can be a great benefit to the planning community. Moreover, this report introduces a whole new area of research on the domain models' structure and automated planning based modelling problem. The PhD thesis will concentrate on the complete set of domain metrics with a set of structural characteristics and properties for domains with a complete Domain Metric Tool (DMT).

5.2 Future work:

Definition of planning domain model includes number of features which can be used to characterize planning domain models, in PDDL definition: types allow decelerating and specifying the types of variables in predicates and operators. Types are aimed to increase the readability of the domain models. However, it is not compulsory required for most planning system. Another example of features which is useful to measure a domain model is relation fluenc, predicates are usually divided into two categories static/rigid and fluent/dynamic relation. Some more features will be measured such as inconsistent effects and reversible actions. These features can only utilised based on their definition language.

5.2.1 Year Three Plan

Sept to Dec 2011

Metrics implementation and Design:

- Figure 3.2 shows the main components of our prototype implementation. At this point we success to create planning knowledge-base that consists of actions schema parsed into our platform language (prolog). The next step is to design and run prototype for each metric mention in section 3.6
- Extending the work to include investigating other PDDL specification mention in future work, section 5.

- Submit a conference paper on PlanSig 2011conference
- Preparing paper for the school and ICAPS conference
- Test both STRIPS and HTN version of domain model with different existing planner

Jan to Apr 2011

- Prepare final version of planning domain model metrics system
- Test the performance
- Submit a journal paper
- Develop thesis structure

May to August 2011

- Start Writing thesis
- Thesis draft(s) Submission
- Thesis submission

6 Appendix A

6.1 GIPO Tyre Domain

```
(define (domain tyre)
  (:requirements :strips :equality :typing)

  (:types  container nuts hub pump wheel wrench jack)


  (:predicates
    (closed ?container1 - container)
    (open ?container1 - container)
    (tight ?nuts1 - nuts ?hub1 - hub)
    (loose ?nuts1 - nuts ?hub1 - hub)
    (have_nuts ?nuts1 - nuts)
    (on_ground ?hub1 - hub)
    (fastened ?hub1 - hub)
    (jacked_up ?hub1 - hub ?jack1 - jack)
    (free ?hub1 - hub)
    (unfastened ?hub1 - hub)
    (have_pump ?pump1 - pump)
    (pump_in ?pump1 - pump ?container1 - container)
    (have_wheel ?wheel1 - wheel)
    (wheel_in ?wheel1 - wheel ?container1 - container)
    (wheel_on ?wheel1 - wheel ?hub1 - hub)
    (have_wrench ?wrench1 - wrench)
    (wrench_in ?wrench1 - wrench ?container1 - container)
    (have_jack ?jack1 - jack)
    (jack_in_use ?jack1 - jack ?hub1 - hub)
    (jack_in ?jack1 - jack ?container1 - container)
  )
  (:action open_container
       :parameters ( ?C - container)
       :precondition
            (closed ?C)
       :effect (and
            (not (closed ?C))
            (open ?C)
        )
    )
  (:action close_container
       :parameters ( ?C - container)
       :precondition
            (open ?C)
       :effect (and
            (not (open ?C))
            (closed ?C)
        )
    )
  (:action fetch_jack
       :parameters ( ?C - container ?J - jack)
       :precondition (and
            (open ?C)
            (jack_in ?J ?C)
```

```
        )
        :effect (and
             (not (jack_in ?J ?C))
             (have_jack ?J)
         )
    )
  (:action fetch_wheel
        :parameters ( ?C - container ?W - wheel)
        :precondition (and
             (open ?C)
             (wheel_in ?W ?C)
        )
        :effect (and
             (not (wheel_in ?W ?C))
             (have_wheel ?W)
         )
    )
  (:action fetch_wrench
        :parameters ( ?C - container ?W - wrench)
        :precondition (and
             (open ?C)
             (wrench_in ?W ?C)
        )
        :effect (and
             (not (wrench_in ?W ?C))
             (have_wrench ?W)
         )
    )
  (:action fetch_pump
        :parameters ( ?C - container ?P - pump)
        :precondition (and
             (open ?C)
             (pump_in ?P ?C)
        )
        :effect (and
             (not (pump_in ?P ?C))
             (have_pump ?P)
         )
    )
  (:action putaway_wheel
        :parameters ( ?C - container ?W - wheel)
        :precondition (and
             (open ?C)
             (have_wheel ?W)
        )
        :effect (and
             (not (have_wheel ?W))
             (wheel_in ?W ?C)
         )
    )
  (:action putaway_wrench
        :parameters ( ?C - container ?W - wrench)
        :precondition (and
             (open ?C)
             (have_wrench ?W)
        )
```

```
        :effect (and
             (not (have_wrench ?W))
             (wrench_in ?W ?C)
         )
    )
  (:action putaway_jack
        :parameters ( ?C - container ?J - jack)
        :precondition (and
             (open ?C)
             (have_jack ?J)
        )
        :effect (and
             (not (have_jack ?J))
             (jack_in ?J ?C)
         )
    )
  (:action putaway_pump
        :parameters ( ?C - container ?P - pump)
        :precondition (and
             (open ?C)
             (have_pump ?P)
        )
        :effect (and
             (not (have_pump ?P))
             (pump_in ?P ?C)
         )
    )
  (:action loosen
        :parameters ( ?W - wrench ?H - hub ?N - nuts)
        :precondition (and
             (have_wrench ?W)
             (on_ground ?H)
             (fastened ?H)
             (tight ?N ?H)
        )
        :effect (and
             (not (tight ?N ?H))
             (loose ?N ?H)
         )
    )
  (:action tighten
        :parameters ( ?W - wrench ?H - hub ?N - nuts)
        :precondition (and
             (have_wrench ?W)
             (on_ground ?H)
             (fastened ?H)
             (loose ?N ?H)
        )
        :effect (and
             (not (loose ?N ?H))
             (tight ?N ?H)
         )
    )
  (:action jack_up
        :parameters ( ?H - hub ?J - jack)
        :precondition (and
```

```
            (on_ground ?H)
            (fastened ?H)
            (have_jack ?J)
      )
      :effect (and
            (not (on_ground ?H))
            (not (have_jack ?J))
            (jacked_up ?H ?J)
            (jack_in_use ?J ?H)
       )
  )
(:action jack_down
      :parameters ( ?H - hub ?J - jack)
      :precondition (and
            (jacked_up ?H ?J)
            (fastened ?H)
            (jack_in_use ?J ?H)
      )
      :effect (and
            (not (jacked_up ?H ?J))
            (not (jack_in_use ?J ?H))
            (on_ground ?H)
            (have_jack ?J)
       )
  )
(:action do_up
      :parameters ( ?W - wrench ?H - hub ?J - jack ?N - nuts)
      :precondition (and
            (have_wrench ?W)
            (unfastened ?H)
            (jacked_up ?H ?J)
            (have_nuts ?N)
      )
      :effect (and
            (not (unfastened ?H))
            (not (have_nuts ?N))
            (fastened ?H)
            (loose ?N ?H)
       )
  )
(:action remove_wheel
      :parameters ( ?W - wheel ?H - hub ?J - jack)
      :precondition (and
            (wheel_on ?W ?H)
            (unfastened ?H)
            (jacked_up ?H ?J)
      )
      :effect (and
            (not (wheel_on ?W ?H))
            (have_wheel ?W)
            (free ?H)
       )
  )
(:action put_on_wheel
      :parameters ( ?W - wheel ?H - hub ?J - jack)
      :precondition (and
```

```
            (have_wheel ?W)
            (free ?H)
            (jacked_up ?H ?J)
            (unfastened ?H)
       )
       :effect (and
            (not (have_wheel ?W))
            (not (free ?H))
            (wheel_on ?W ?H)
        )
    )
  (:action undo
       :parameters ( ?W - wrench ?H - hub ?J - jack ?N - nuts)
       :precondition (and
            (have_wrench ?W)
            (jacked_up ?H ?J)
            (fastened ?H)
            (loose ?N ?H)
       )
       :effect (and
            (not (fastened ?H))
            (not (loose ?N ?H))
            (unfastened ?H)
            (have_nuts ?N)
        )
    )
  )
(define (problem task1)
   (:domain tyre)
   (:objects
         boot - container
         nuts_1 - nuts
         hub0 - hub
         pump0 - pump
         wheel1 wheel2 - wheel
         wrench0 - wrench
         jack0 - jack
        )
    (:init
        (wrench_in wrench0 boot)
        (wheel_in wheel1 boot)
        (have_wheel wheel2)
        (pump_in pump0 boot)
        (tight nuts_1 hub0)
        (have_jack jack0)
        (on_ground hub0)
        (fastened hub0)
        (closed boot)
        )
    (:goal
      (and
        (fastened hub0)
        (jacked_up hub0 jack0)
        (wheel_on wheel2 hub0)
       ))
)
```

```
(define (problem task2)
   (:domain tyre)
   (:objects
         boot - container
         nuts_1 - nuts
         hub0 - hub
         pump0 - pump
         wheel1 wheel2 - wheel
         wrench0 - wrench
         jack0 - jack
        )
    (:init
        (have_wrench wrench0)
        (have_wheel wheel1)
        (wheel_on wheel2 hub0)
        (pump_in pump0 boot)
        (loose nuts_1 hub0)
        (jack_in_use jack0 hub0)
        (jacked_up hub0 jack0)
        (fastened hub0)
        (closed boot)
        )
    (:goal
      (and
        (open boot)
        (jacked_up hub0 jack0)
        (loose nuts_1 hub0)
        (wheel_in wheel2 boot)
        (wheel_on wheel1 hub0)
        (wrench_in wrench0 boot)
       ))
)
(define (problem task3)
   (:domain tyre)
   (:objects
         boot - container
         nuts_1 - nuts
         hub0 - hub
         pump0 - pump
         wheel1 wheel2 - wheel
         wrench0 - wrench
         jack0 - jack
        )
    (:init
        (wrench_in wrench0 boot)
        (wheel_on wheel1 hub0)
        (have_wheel wheel2)
        (have_pump pump0)
        (tight nuts_1 hub0)
        (jack_in jack0 boot)
        (on_ground hub0)
        (fastened hub0)
        (open boot)
        )
    (:goal
      (and
```

```
        (open boot)
        (jacked_up hub0 jack0)
        (jack_in_use jack0 hub0)
        (have_nuts nuts_1)
        (have_pump pump0)
        (wrench_in wrench0 boot)
       ))
)
(define (problem task4)
   (:domain tyre)
   (:objects
         boot - container
         nuts_1 - nuts
         hub0 - hub
         pump0 - pump
         wheel1 wheel2 - wheel
         wrench0 - wrench
         jack0 - jack
        )
    (:init
        (wrench_in wrench0 boot)
        (wheel_on wheel1 hub0)
        (have_wheel wheel2)
        (pump_in pump0 boot)
        (have_nuts nuts_1)
        (jack_in_use jack0 hub0)
        (unfastened hub0)
        (jacked_up hub0 jack0)
        (closed boot)
        )
    (:goal
      (and
        (closed boot)
        (have_wheel wheel1)
       ))
)
```

6.2 LCOM Tyre Domain

```
(define
   (domain tyre)
   (:requirements :typing)
   (:types boot hub jack nuts wheel wrench zero)
   (:predicates
      (boot_state0 ?v1 - boot)
      (boot_state1 ?v1 - boot)
      (hub_state0 ?v1 - hub ?v2 - jack)
      (hub_state1 ?v1 - hub ?v2 - jack)
      (hub_state2 ?v1 - hub ?v2 - jack)
      (hub_state3 ?v1 - hub)
      (jack_state0 ?v1 - jack ?v2 - hub)
      (jack_state1 ?v1 - jack ?v2 - hub)
      (jack_state2 ?v1 - jack ?v2 - hub)
      (jack_state3 ?v1 - jack)
      (jack_state4 ?v1 - jack ?v2 - boot)
      (nuts_state0 ?v1 - nuts)
      (nuts_state1 ?v1 - nuts ?v2 - hub)
      (nuts_state2 ?v1 - nuts ?v2 - hub)
      (wheel_state0 ?v1 - wheel ?v2 - hub)
      (wheel_state1 ?v1 - wheel)
      (wheel_state2 ?v1 - wheel ?v2 - boot)
      (wrench_state0 ?v1 - wrench ?v2 - boot)
      (wrench_state1 ?v1 - wrench)
      (zero_state0))

   (:action
   close_container
   :parameters
   (?Boot1 - boot)
   :precondition
   (and
      (zero_state0)
      (boot_state0 ?Boot1))

   :effect
   (and
      (boot_state1 ?Boot1)
      (not (boot_state0 ?Boot1)))
)

   (:action
   do_up
   :parameters
   (?Nuts1 - nuts ?Hub2 - hub ?Wrench4 - wrench ?Jack3 - jack)
   :precondition
   (and
      (zero_state0)
      (nuts_state0 ?Nuts1)
      (hub_state0 ?Hub2 ?Jack3)
      (wrench_state1 ?Wrench4)
      (jack_state0 ?Jack3 ?Hub2))
```

```
    :effect
    (and
       (nuts_state1 ?Nuts1 ?Hub2)
       (not (nuts_state0 ?Nuts1))
       (hub_state2 ?Hub2 ?Jack3)
       (not (hub_state0 ?Hub2 ?Jack3))
       (jack_state2 ?Jack3 ?Hub2)
       (not (jack_state0 ?Jack3 ?Hub2)))
)

    (:action
    fetch_jack
    :parameters
    (?Jack1 - jack ?Boot2 - boot)
    :precondition
    (and
       (zero_state0)
       (jack_state4 ?Jack1 ?Boot2)
       (boot_state0 ?Boot2))

    :effect
    (and
       (jack_state3 ?Jack1)
       (not (jack_state4 ?Jack1 ?Boot2)))
)

    (:action
    fetch_wheel
    :parameters
    (?Wheel1 - wheel ?Boot2 - boot)
    :precondition
    (and
       (zero_state0)
       (wheel_state2 ?Wheel1 ?Boot2)
       (boot_state0 ?Boot2))

    :effect
    (and
       (wheel_state1 ?Wheel1)
       (not (wheel_state2 ?Wheel1 ?Boot2)))
)

    (:action
    fetch_wrench
    :parameters
    (?Wrench1 - wrench ?Boot2 - boot)
    :precondition
    (and
       (zero_state0)
       (wrench_state0 ?Wrench1 ?Boot2)
       (boot_state0 ?Boot2))

    :effect
    (and
       (wrench_state1 ?Wrench1)
       (not (wrench_state0 ?Wrench1 ?Boot2)))
```

```
)

    (:action
    jack_down
    :parameters
    (?Hub1 - hub ?Jack2 - jack)
    :precondition
    (and
       (zero_state0)
       (hub_state2 ?Hub1 ?Jack2)
       (jack_state2 ?Jack2 ?Hub1))

    :effect
    (and
       (hub_state3 ?Hub1)
       (not (hub_state2 ?Hub1 ?Jack2))
       (jack_state3 ?Jack2)
       (not (jack_state2 ?Jack2 ?Hub1)))
)

    (:action
    jack_up
    :parameters
    (?Hub1 - hub ?Jack2 - jack)
    :precondition
    (and
       (zero_state0)
       (hub_state3 ?Hub1)
       (jack_state3 ?Jack2))

    :effect
    (and
       (hub_state2 ?Hub1 ?Jack2)
       (not (hub_state3 ?Hub1))
       (jack_state2 ?Jack2 ?Hub1)
       (not (jack_state3 ?Jack2)))
)

    (:action
    loosen
    :parameters
    (?Nuts1 - nuts ?Hub2 - hub ?Wrench3 - wrench)
    :precondition
    (and
       (zero_state0)
       (nuts_state2 ?Nuts1 ?Hub2)
       (hub_state3 ?Hub2)
       (wrench_state1 ?Wrench3))

    :effect
    (and
       (nuts_state1 ?Nuts1 ?Hub2)
       (not (nuts_state2 ?Nuts1 ?Hub2)))
)

    (:action
```

```
    open_container
    :parameters
    (?Boot1 - boot)
    :precondition
    (and
       (zero_state0)
       (boot_state1 ?Boot1))

    :effect
    (and
       (boot_state0 ?Boot1)
       (not (boot_state1 ?Boot1)))
)

    (:action
    put_on_wheel
    :parameters
    (?Wheel1 - wheel ?Hub2 - hub ?Jack3 - jack)
    :precondition
    (and
       (zero_state0)
       (wheel_state1 ?Wheel1)
       (hub_state1 ?Hub2 ?Jack3)
       (jack_state1 ?Jack3 ?Hub2))

    :effect
    (and
       (wheel_state0 ?Wheel1 ?Hub2)
       (not (wheel_state1 ?Wheel1))
       (hub_state0 ?Hub2 ?Jack3)
       (not (hub_state1 ?Hub2 ?Jack3))
       (jack_state0 ?Jack3 ?Hub2)
       (not (jack_state1 ?Jack3 ?Hub2)))
)

    (:action
    putaway_jack
    :parameters
    (?Jack1 - jack ?Boot2 - boot)
    :precondition
    (and
       (zero_state0)
       (jack_state3 ?Jack1)
       (boot_state0 ?Boot2))

    :effect
    (and
       (jack_state4 ?Jack1 ?Boot2)
       (not (jack_state3 ?Jack1)))
)

    (:action
    putaway_wheel
    :parameters
    (?Wheel1 - wheel ?Boot2 - boot)
    :precondition
```

```
    (and
       (zero_state0)
       (wheel_state1 ?Wheel1)
       (boot_state0 ?Boot2))

    :effect
    (and
       (wheel_state2 ?Wheel1 ?Boot2)
       (not (wheel_state1 ?Wheel1)))
)

    (:action
    putaway_wrench
    :parameters
    (?Wrench1 - wrench ?Boot2 - boot)
    :precondition
    (and
       (zero_state0)
       (wrench_state1 ?Wrench1)
       (boot_state0 ?Boot2))

    :effect
    (and
       (wrench_state0 ?Wrench1 ?Boot2)
       (not (wrench_state1 ?Wrench1)))
)

    (:action
    remove_wheel
    :parameters
    (?Wheel1 - wheel ?Hub2 - hub ?Jack3 - jack)
    :precondition
    (and
       (zero_state0)
       (wheel_state0 ?Wheel1 ?Hub2)
       (hub_state0 ?Hub2 ?Jack3)
       (jack_state0 ?Jack3 ?Hub2))

    :effect
    (and
       (wheel_state1 ?Wheel1)
       (not (wheel_state0 ?Wheel1 ?Hub2))
       (hub_state1 ?Hub2 ?Jack3)
       (not (hub_state0 ?Hub2 ?Jack3))
       (jack_state1 ?Jack3 ?Hub2)
       (not (jack_state0 ?Jack3 ?Hub2)))
)

    (:action
    tighten
    :parameters
    (?Nuts1 - nuts ?Hub2 - hub ?Wrench3 - wrench)
    :precondition
    (and
       (zero_state0)
       (nuts_state1 ?Nuts1 ?Hub2)
```

```
      (hub_state3 ?Hub2)
      (wrench_state1 ?Wrench3))

   :effect
   (and
      (nuts_state2 ?Nuts1 ?Hub2)
      (not (nuts_state1 ?Nuts1 ?Hub2)))
)

   (:action
   undo
   :parameters
   (?Nuts1 - nuts ?Hub2 - hub ?Wrench4 - wrench ?Jack3 - jack)
   :precondition
   (and
      (zero_state0)
      (nuts_state1 ?Nuts1 ?Hub2)
      (hub_state2 ?Hub2 ?Jack3)
      (wrench_state1 ?Wrench4)
      (jack_state2 ?Jack3 ?Hub2))

   :effect
   (and
      (nuts_state0 ?Nuts1)
      (not (nuts_state1 ?Nuts1 ?Hub2))
      (hub_state0 ?Hub2 ?Jack3)
      (not (hub_state2 ?Hub2 ?Jack3))
      (jack_state0 ?Jack3 ?Hub2)
      (not (jack_state2 ?Jack3 ?Hub2)))
)
)
```

References

1. R. M, S., K. D. E, and T.L. McCLUSKEY, *Planning domain definition using GIPO.* Knowl. Eng. Rev., 2007. 22(2): p. 117-134.
2. Cresswell, S.N.M., T.L. West, Margaret M., *Acquisition of Object-Centred Domain Models from Planning Examples.* Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling, AAAI Press, Menlo Park, California, USA, 2009: p. 4.
3. M. Ghallab, D.N., and P. Traverso, *Automated Planning: Theory and Practice*. 2004: Morgan Kaufmann.
4. C.Hogg, U.K., H.Munoz-Avila, *From Plan Traces to Hierarchical Task Networks Using Reinforcements: A Preliminary Report*, in *AAAI Press. IJCAI-09*, W.o.L.S.I.f.T. (STRUCK-09), Editor. 2009.
5. Committee, A.-P.C., *PDDL - The Planning Domain Definition Language.* Yale Center for Computational Vision and Control., 1998. Tech. rep. CVC TR-98-003/DCS TR-1165.
6. Nau, D.S., et al., *SHOP2: An HTN Planning System.* Journal of Artificial Intelligence Research, 2003. 20: p. 379-404.
7. Zhuo, H.H., et al., *Learning HTN Method Preconditions and Action Models from Partial Observations*, in *International Joint Conference on Artificial Intelligence*. 2009. p. 1804-1810.
8. Kaelbling, L.P., M.L. Littman, and A.R. Cassandra, *Planning and Acting in Partially Observable Stochastic Domains.* Artificial Intelligence, 1998. 101(1-2): p. 99-134.
9. Bacchus, F. and F. Kabanza, *Using temporal logics to express search control knowledge for planning.* Artificial Intelligence, 2000. 116(1-2): p. 123-191.
10. McDermott, D., *PDDL-the planning domain definition language*. 1998.
11. Nilsson, R.E.F.a.N.J., *Strips: A new approach to the application of theorem proving to problem solving.* Artificial Intelligence, 1971. 2: p. 189-208.
12. Pednault, E.P.D., *ADL: exploring the middle ground between STRIPS and the situation calculus*, in *Proceedings of the first international conference on Principles of knowledge representation and reasoning*. 1989, Morgan Kaufmann Publishers Inc. p. 324-332.
13. Liu, D., and McCluskey, T. L. , *The OCL Language Manual, Version 1.2. Technical report.* Department of Computing and Mathematical Sciences, University of Hudderseld., 2000.
14. Frank, J., et al., *Constraint-Based Attribute and Interval Planning.* Constraints, 2003. **8**(4): p. 339-364.
15. McCluskey, T.L.A., D.A. Liu, and R.M.A. Simpson, *GIPO II: HTN planning in a tool-supported knowledge engineering = environment =0A=*. Proceedings of the Thirteenth International Conference on Automated = Planning and Scheduling=0A=, ed. E.A. Giunchiglia, N.A. Muscettola, and D.A. Nau. 2003, Menlo Park, California=0A=: AAAI Press=0A=. 92-101=0A=.
16. Vaquero, T.S., F. Tonidandel, and J.R. Silva, *The itSIMPLE tool for Modeling Planning Domains*, in *Monterey Workshop*. 2005.

17. Zhuo, H., et al., *Transferring Knowledge from Another Domain for Learning Action Models*, in *Proceedings of the 10th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence*. 2008, Springer-Verlag: Hanoi, Vietnam. p. 1110-1115.
18. Smith, D.E. and W. Cushing, *The ANML Language.*
19. Matthew, E.T. and S. Peter, *Transfer Learning for Reinforcement Learning Domains: A Survey.* J. Mach. Learn. Res., 2009. 10: p. 1633-1685.
20. Scott, S.B., *Learning Action Models for Reactive Autonomous Agents*. 1997, Stanford University.
21. Xuemei, W., *Learning by observation and practice: a framework for automatic acquisition of planning operators*, in *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2).* 1994, American Association for Artificial Intelligence: Seattle, Washington, United States.
22. Eyal, A. and C. Allen, *Learning partially observable deterministic action models.* J. Artif. Int. Res., 2008. 33(1): p. 349-402.
23. Qiang, Y., W. Kangheng, and J. Yunfei, *Learning action models from plan examples using weighted MAX-SAT.* Artif. Intell., 2007. 171(2-3): p. 107-143.
24. Zhuo, H.H., et al., *Learning complex action models with quantifiers and logical implications.* Artificial Intelligence, 2010. 174(18): p. 1540-1569.
25. WU, K., Q. YANG, and Y. JIANG, *ARMS: an automatic knowledge engineering tool for learning action models for AI planning.* Knowl. Eng. Rev., 2007. 22(2): p. 135-152.
26. Zhuo, H.H., et al., *Learning Applicability Conditions in AI Planning from Partial Observations*.
27. Sungwook, Y., F. Alan, and G. Robert, *Learning Control Knowledge for Forward Search Planning.* J. Mach. Learn. Res., 2008. 9: p. 683-718.
28. Hankz Hankui, Z., et al., *Learning HTN method preconditions and action models from partial observations*, in *Proceedings of the 21st international jont conference on Artifical intelligence*. 2009, Morgan Kaufmann Publishers Inc.: Pasadena, California, USA.
29. Dafna, S. and A. Eyal, *Learning partially observable action schemas*, in *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*. 2006, AAAI Press: Boston, Massachusetts.
30. Hankui, Z., et al., *Transferring Knowledge from Another Domain for Learning Action Models*, in *Proceedings of the 10th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence*. 2008, Springer-Verlag: Hanoi, Vietnam.
31. Goodman, P., *The Practical Implementation of Software Metrics*. 1993: McGraw-Hill, Inc. . 230.
32. J, et al., *Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4.* J. Artif. Int. Res., 2006. 26(1): p. 453-541.
33. Wickler, G., *Using Planning Domain Features to Facilitate Knowledge Engineering*, in *ICAPS 2011*. 2011: University of Freiburg, Germany. p. 39-46.
34. J, H. rg, and N. Bernhard, *The FF planning system: fast plan generation through heuristic search.* J. Artif. Int. Res., 2001. 14(1): p. 253-302.