PLAN GENERATION IN A MODIFIED BLOCKS WORLD DOMAIN

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree

of

INTEGRATED DUAL DEGREE

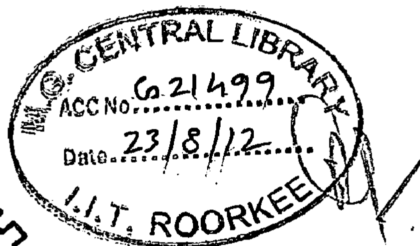
in

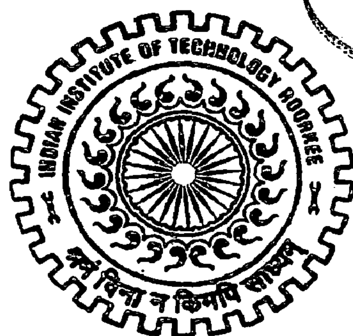
COMPUTER SCIENCE AND ENGINEERING

(With Specialization in Information Technology)

By

SWETHA JAIN KOTHARI





DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE-247 667 (INDIA)

MAY, 2012

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "**Plan Generation in a Modified Blocks World Domain**" towards the partial fulfillment of the requirement for the award of the degree of **Integrated Dual Degree** in **Computer Science** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India is an authentic record of my own work carried out during the period from July 2011 to May 2012, under the joint guidance of **Dr. Rajdeep Niyogi**, Assistant Professor, Department of Electronics and Computer Engineering, IIT Roorkee.

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other Institute.

Date: 22-5-2012
Place: Roorkee

Swetha
(SWETHA JAIN KOTHARI)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 22-5-2012
Place: Roorkee

(Dr. RAJDEEP NIYOGI)
Assistant Professor,
Department of Electronics and Computer Engineering,
IIT Roorkee

ACKNOWLEDGEMENT

First of all, I would like to extend my sincere gratitude to my guide and mentor **Dr. Rajdeep Niyogi**, Assistant Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his valuable advices, guidance and suggestions. I would like to thank him for his support and encouragement throughout the dissertation. It was a learning process with an opportunity to discover new things thereby increasing my knowledge.

I am greatly indebted to all my friends, who have helped me with moral support and valuable suggestions.

Last but not the least; I would like to thank my parents and my family for supporting me in every endeavor. I extend out this thanks note to everyone else who have knowingly or unknowingly helped me carry out this work.

Swetha

SWETHA JAIN KOTHARI

ABSTRACT

Planning is a sub discipline of artificial intelligence that aims at generating plans so that its execution leads to a desirable state from some initial state. When uncertainty is involved in a planning domain, the planning problem is no longer limited to finding a sequence of actions, but to find a plan, also composed of actions, that might include conditional, iterative and/or recursive constructs, i.e. the planning problem now also includes program synthesis. Generating programs automatically is a challenging task. Even for simple planning domains, like the blocks world domain involving recursion and if-then-else, the problem of generating programs needs human intervention. In the dissertation, a modified blocks world domain is considered and planners are developed for the domain. Also, some results for the plans generated by the planners are presented.

Try actions are the trials of physical actions. As another part of the dissertation, the concept of try actions has been extended to joint trials, trials involving more than one agent. Other concepts relating to trials, repeated trials and learning about the capabilities of agents have also been introduced.

TABLE OF CONTENTS

ABSTRACT iii
LIST OF TABLES AND FIGURES vi
CHAPTER 1. INTRODUCTION 1
1.1. Background 1
1.2. Motivation 2
1.3. Problem Statement 2
1.4. Organization of the Report 3
CHAPTER 2. LITERATURE REVIEW 4
2.1. Planning in the presence of sensing 4
2.2. Iterative planning 6
2.3. Reactive planning systems 7
2.4. Try actions 7
2.4.1. Modified BW domain 8
2.4.2. PDL_try 8
2.4.3. Plans in PDL_try for the MBW domain 9
CHAPTER 3. PLANNING IN A MODIFIED BLOCKS WORLD DOMAIN 10
3.1. Model of the Modified Blocks World Domain 10
3.2. Logic for specification of the plans 12
3.3. Plan Generation 13
3.4. Plans for default Initial State 14
3.4.1. Information Gathering 14
3.4.2. Planner for Information Gathering 16
3.4.3. Tower Construction 17
3.4.4. Planner for Tower Construction 19
3.5. Plans from any Initial State 20
3.5.1. Domain of initial states 20
3.5.2. Single Tower construction 22
3.5.3. Planner for single tower construction 22
3.5.4. Example Plans 23
CHAPTER 4. RESULTS 28
4.1. Screenshots 28

4.2. Analysis of the plans 29
CHAPTER 5. INTRODUCTION TO JOINT TRIALS 33
5.1. Multi-agent Planning 33
5.1.1. Planning with multiple simultaneous actions 33
5.1.2. Planning with multiple agents: cooperation and coordination 34
5.2. Joint trials 35
5.3. Other concepts of trials 38
CHAPTER 6. CONCLUSION AND FUTURE WORK 40
REFERENCES 41
PUBLICATIONS 43

LIST OF TABLES AND FIGURES

Table 4.1: Computed parameter values for different number of blocks 30

Figure 3.1: Plan graphs when (a) a deterministic action b is chosen from a state s, (b) a try action Try_a is chosen from a state s... 13

Figure 3.2: Plan graph for information gathering from the default initial state 15

Figure 3.3: Plan graph for stacking of heavy blocks alongside information gathering 18

Figure 3.4: (a) Heavy tower; (b) Unknown tower; (c) Light tower 21

Figure 3.5: Sample configurations of initial state of MBW domain where there are unequal size towers (a) there are heavy towers with unequal number of heavy blocks; (b) there are no heavy and unknown towers but more than one light towers of unequal size 24

Figure 3.6: Sample configurations of initial state of MBW domain where a block is in robot arm: (a) there are more than one non-light towers; (b) there is only one non-light tower 24

Figure 3.7: Sample configurations of initial state of MBW domain where there are light blocks in heavy towers: (a) there are more than one heavy towers; (b) there is only one heavy tower and no unknown towers; (c) there is one heavy tower with unknown towers. 25

Figure 3.8: Sample configuration of initial state of MBW domain where there are unknown towers and no heavy towers.. 26

Figure 4.1: Screenshot for and while running the planner for information gathering from default initial state with input number of blocks as three... 28

Figure 4.2: Screenshot for and while running the planner for tower construction from default initial state with input number of blocks as three... 29

Figure 4.3: Average number of actions vs. Number of blocks 31

Figure 4.4: Height of the plan graph vs. Number of blocks 31

Figure 4.5: Number of nodes vs. Number of blocks 32

Figure 4.6: Average plan generating time vs. Number of blocks 32

CHAPTER 1. INTRODUCTION

1.1. Background

Artificial intelligence is a branch of computer science that aims at making machines better at things that humans are presently better at [1]. Automated planning is a sub discipline of artificial intelligence that aims at generating plans so that its execution leads to a desirable state from some initial state. Plans are made up of actions available to the executors of plans. For planning, explicit propositional or relational representations of states (initial, goal) and actions are required. States are represented as sets of sentences and actions are represented by logical descriptions of preconditions and effects [2]. The representations are studied often, for e.g. [3], in the planning literature in the context of operator representations that ensure trade-off between the range of planning tasks that can be represented and the efficiency with which the planning task can be solved.

STRIPS is a formal language used for representation of inputs (initial state, goal, and the actions) in the STRIPS planner developed in 1971. STRIPS language cannot represent partially known initial states. This language is the base for most of the languages used in present-day planners. The extensions of STRIPS language incorporate representation of partially known initial states too. ADL (Action Description Language) is one of the extensions of STRIPS which also allows conditional effects of operators (or actions). The PDDL (Planning Domain Definition Language) is an attempt to standardize AI planning languages.

The simplest planning domains are those where there is no incomplete or uncertain knowledge, where actions cause known effects and where there is none but yourself (the executor of plans) to change the state of the domain. These are called the classical planning domains. In these domains, plans can always be represented as a sequence of actions. Optimal planning (in these domains) has costs associated with actions, or with goals. The goal of planning is to find a plan with minimum overall cost. Classical planners cannot handle scenarios involving uncertainty. Many a times, there are domains that involve uncertainty. The uncertainty may be about the state of the world and/or about the effects of actions.

When the uncertainty is about the effects of actions, the reasoning about actions can be based on qualitative models or quantitative models [4]. In qualitative models, all possible alternatives are equally considered (nondeterministic uncertainty), while in quantitative models, there is a probability distribution on the set of possible alternatives.

In planning, there are three different types of actions known – the physical actions, the sensing actions and the try actions. Physical actions are actions that change the state of the world. Sensing actions help to obtain some information about the state of the world. Try actions are the most recently introduced actions [5] that are trials of physical actions. When it is not known whether some precondition of a physical action is satisfied or not, the physical action can be tried. The success (change the world just as the physical action would) or failure (the physical action not being possible in that state) of a trial results in learning.

Multi-agent planning is more complex than when there is a single agent in the domain due to various effects the other agents may cause to the world. In cooperative multi-agent systems, it is assumed that each agent makes its own plan and shares it with other agents [6]. When the agents are not cooperative, protocols may be used so that the agents can all reach their goals [7], [8]. Different methods have been proposed, for example in [9], so as to reduce the cost of communication in multi-agent planning.

1.2. Motivation

Real world manifests uncertainty in some form or the other. When there is uncertainty involved in a planning domain, the plans are not sequence of actions but they include if-then-else, looping and/or recursive constructs. Program synthesis techniques have been adopted for planning problems involving uncertainty. But, program synthesis is a challenging task. The problem of generating programs (plans), even for the simple blocks world domain, involving recursion and if-then-else requires human intervention. In this dissertation, plans are generated for a modified blocks world domain that has uncertainty in the form of incomplete information in the initial state.

1.3. Problem Statement

The aim of the dissertation is to see the generation of plans in an uncertain domain that uses try actions for learning information. The domain considered is an adaptation of the popular blocks world domain which is called as "modified blocks world" domain.

A small part of the dissertation also aims to looking at how the concept of try actions can be extended.

1.4. Organization of the Report

In chapter 2, few works done with respect to planning in domains where uncertainty is involved have been outlined. Chapter 3 contains the major part of the dissertation, where algorithms for planning in the modified blocks world domain are discussed. Chapter 4 has some screenshots when the planners are run and it also contains analysis of the plans generated by the planner. Chapter 5 contains another minor part of the dissertation where the concept of joint trials, a trial action done by two agents together, is introduced. Chapter 6 concludes the report and mentions what can be done as future work.

CHAPTER 2. LITERATURE REVIEW

Classical plans can be written as a sequence of actions. However, the plans for domains involving uncertainty cannot always be written as a sequence of actions. The uncertainty in a planning domain may be due to incomplete initial state information, non-deterministic and/or probabilistic effects [4] of actions and/or presence of exogenous actions. Following is a review of some of the works on planning under uncertainty.

2.1. Planning in the presence of sensing

In [10], a specification within the situation calculus of conditional and iterative plans over domains that include binary sensing actions has been developed. Sensing actions affect the knowledge of the robot carrying out the sensing action. All the following discussion in this section is from [10]. The plans are programs written in a simple robot program language. The robot programming language includes both ordinary and sensing actions and is executable by an agent that understands the language. Sensing may be required in the world because of incomplete knowledge of the initial state, exogenous actions, uncertain effects of actions. Some examples of problems are handled by the specification in [10] are:

1. The Airport Example: The local airport has only two boarding gates, Gate A and Gate B. Every plane will be parked at either of the two gates. In the initial state, the robot is at home. The robot can go from home to airport and from airport to either gate. At the airport it can also check the departures screen to find out if the flight is using Gate A or not (assume that the flight looked for is present at either of the two gates and none of the gates is not possible). At a gate, the robot can board the plane that is parked there. The goal for the robot is to be on the plane for Flight123.
2. The Omelet Example: There is a supply of eggs in the initial state. Some of them may be bad, but at least 3 of them are good. There is a bowl and saucer, and they can be emptied at any time. An egg always be broken into the bowl, while can be broken into the saucer only if it is empty. The robot can smell a container and tell if it contains a bad egg. The contents of the saucer can be transferred to the bowl. The goal is for the robot to get 3 good eggs and no bad ones in the bowl.
3. The Odd Good Eggs Example: There is an additional sensing action, compared to that in the Omelet example, which tells the robot when there are no more eggs left. The goal is to

have a single good egg in the bowl if the supply contains an odd number of good eggs and otherwise there should be no eggs in the bowl.

4. The More Good Eggs Example: The domain is the same as that in the above example. The goal is to have a single good egg in the bowl if the supply contains more good eggs than bad and none otherwise.

The robot programming language is defined as:

1. *nil* and *exit* are programs.
2. If a is an ordinary action and r is a program, then *seq*(a, r) is a program.
3. If a is a binary sensing action and r_1 and r_2 are programs, then *branch*(a, r_1, r_2) is a program
4. If r_1 and r_2 are programs, then *loop*(r_1, r_2) is a program.

The execution of these programs by an agent (who understands the language) means follows respectively:

1. *nil*: The agent does nothing; *exit*: can be executed only if it is executing a *loop* and what it does is given in the execution of *loop*.
2. *seq*(a, r) executes primitive action a and then r.
3. *branch*(a, r_1, r_2) executes a which is supposed to tell whether some condition φ_a holds or not. If yes, it executes r_1, else it executes r_2.
4. *loop*(r_1, r_2) executes r_1 and if ends with *nil*, repeats r_1, and continues in the similar way until it ends with *exit*, and then executes r_2.

The following are some example robot programs using the above language:

1. The Airport Example:

   seq(go(airport), *branch*(check_departures(Flight123), *seq*(go(gateA), *seq*(board_plane, *nil*)), *seq*(go(gateB), *seq*(board_plane, *nil*)))

2. The Omelet Example:

   loop(*body*, *seq*(transfer(saucer, bowl), *loop*(*body*, *seq*(transfer(saucer, bowl), *loop*(*body*, *seq*(transfer(saucer, bowl), *nil*)))))), where *body* stands for the program *seq*(break_new_egg(saucer), *branch*(smell(saucer), *seq*(dump(saucer), *nil*), *exit*))

The representation in the specification can handle problems involving only binary sensing actions, though it can be extended to handle sensing actions that return a small set of values and those that return a large or infinite set of values which can be ordered in a natural way.

But if the sensing involves reading from a noisy sensor, it cannot be handled by the specification.

A transition based approach for formalizing sensing actions is given in [11]. In [11], the authors distinguish between the state of the world, and the state of the agent's knowledge about the world that arise due to incomplete information about the world. Sensing actions change the state of knowledge about the world and not the state of the world. The authors develop a high-level description language that allows specification of sensing actions and their effects in its domain description. They also provide translations of domain description in this language to axioms in first-order logic.

2.2. Iterative planning

Most of the works on iterative planning have been based on theorem-proving. In [12], an approach, to solve planning problems where some unknown quantity must be dealt with, where generating plans is decoupled from verifying them has been proposed; and an implementation of an iterative planner, the KPLANNER which is written in Prolog and based on the situation calculus, has been described.

The plans in KPLANNER [12] are generated as robot programs with little variation in the programming language from that in [10]. The programs and their execution are defined as:

1. *nil* is a robot program doing nothing.
2. for any primitive action A and robot program P, *seq*(A, P) is a robot program executed by performing A and then P
3. for any primitive action A with possible sensing results R_1 to R_k, and for any robot programs P_1 to P_k, *case*(A, [*if*(R_1, P_1), …, *if*(R_k, P_k)]) is a robot performed by executing A and then on obtaining the sensing result R_i, executing P_i
4. if P and Q are robot programs and B is the result of replacing in P some of the occurrences of *nil* by *exit* and the remaining by *next*, then *loop*(B, Q) is a robot program, executed by repeatedly executing B until the execution terminates with *exit* (rather than *next*), and then executing Q.

Example:

Tree Chop (TC) Example: Goal is to chop down a tree and pitting away the axe. Actions available are *chop*, which hits the tree once with axe if the tree is up and the axe is in hand; *store*, which puts away the axe in hand; and a sensing action *look*, which tells whether the

tree is up or down. Initially the tree is up and the axe is in hand. The plan can now be written in the robot program as:

loop(*case*(look, [*if*(down, *exit*), *if*(up, *seq*(chop, *next*))]), store)

The approach followed by the KPLANNER [12] is as what follows. There is one fluent in the domain whose value is not known and if it was known or bounded at plan time, loops would not have been required. This fluent is called the *planning parameter*. The application domain and planning parameter F with generating bound N_1 and testing bound N_2 ($> N_1$) are given. The plan is generated that is provably correct for $F \leq N_1$ and the plan is tested if it is provably correct for $F \leq N_2$.

The main steps involved in KPLANNER are generating plans without any loops so that they are provably correct for $F \leq N_1$, generating loops from the plan generated, by determining if the plan is the unwinding of a plan with loops, so as to obtain a plan with loops and the testing the plan (with loops) for $F \leq N_2$, with a specification of the problem (the domain, the planning parameter and the bounds) in Prolog as input to the planner.

KPLANNER is practical only for small plans (however difficult they may be) but not for large, however easy, ones.

2.3. *Reactive planning systems*

Reactive planning systems are the systems able to plan and control execution of plans in a partially known and unpredictable environment. In [13], a theory for reactive planning systems is developed. The theory takes into account the facts that (1) actions may fail, since they are complex programs controlling sensors and actuators working in an unpredictable environment, (2) actions need to acquire information from the real world by activating sensors and actuators, (3) actions need to generate and execute plans of actions, since the planner needs to activate different special purpose planners and to execute the resulting plans.

2.4. *Try actions*

In [5], planning problems in uncertain environments but with no sensing actions have been addressed by introducing the concept of try actions. A logic PDL_try, an extension to PDL has also been developed and planning with the logic in an example domain has been illustrated. Hereafter, all the discussion in the section is from [5], unless specified otherwise

The example domain considered is adapted from the Blocks World (BW) domain. The fluents in BW domain are *OnTable(x)*, *Clear(x)*, *ArmEmpty*, *On(x, y)*. The operators are *PickUp(x)*, *PutDown(x)*, *Stack(x, y)*, *UnStack(x, y)*. (An action is a ground instance of an operator.)

2.4.1. Modified BW domain

There are a number of blocks, all of same size, and an infinite long table. Therefore, there can be any number of blocks on the table, while every block can have only one block exactly on top of it. The blocks are of two different weights, light or heavy. The robot arm can hold a light block, but not a heavy block, alternately a heavy block cannot be picked up or unstacked by the operators *PickUp(x)* and *UnStack(x, y)* respectively, and also neither do *PutDown(x)*, *Stack(x, y)* operators work on heavy blocks. Instead there are new operators *ApplyLever(x, y)* to move from block x on block y. [14] includes another operator *RevLever(x, y)* that moves block x from top of block y to table. The robot may not know the weights of one or more blocks, initially, and it can find whether block x is light or heavy by trying to pick up the block by using the operator *Try_PickUp(x)* (because there are no sensing actions in the domain that can weigh the block and tell whether it is light or heavy). Also, when there are towers of blocks, in every tower, there is no heavy block on a light block.

2.4.2. PDL_try

It is the enhancement of PDL to include try actions and a knowledge operator. The logic consists of two sets of symbols: P – the set of atomic propositions and A – the set of primitive actions (try actions included). For some $a \in A$, $\sigma(try_a)$ rand $\tau(try_a)$ are included in P and they indicate that the try action has succeeded or failed respectively. Success of a try action means that the postcondition of the action tried has been achieved, while failure means that the postcondition has not been achieved.

Syntax:

φ – set of propositions; π – set of plans are defined as follows:

$\varphi = p \in P \mid \text{not } \varphi \mid \varphi \vee \varphi \mid [\pi]\varphi \mid \mathrm{K}\varphi$

$\pi = a, try_a \in A \mid \pi; \pi \mid \pi \text{ union } \pi \mid \pi^* \mid \varphi?$

The semantics of the logic are not discussed here in the report as the aim is to just show the sample plans written in PDL_try logic, which are quite readable themselves.

2.4.3. Plans in PDL_try for the MBW domain

Let A, B, C be three blocks in the modified blocks world (MBW) domain. In the initial state, all the blocks are on table and weights of none of the blocks are known. The goal is to form a tower out of these blocks.

The plan P (in PDL_try) when the blocks' weights are found in the order A, B and C is:

```
P = Try_PickUp(A);
    If σ(Try_PickUp(A)) then P1
    Else { Try_PickUp(B);
      If σ(Try_PickUp(B)) then P3
      Else { Try_PickUp(C);
             If σ(Try_PickUp(C)) then P4
             Else { ApplyLever(C, B); ApplyLever(A, B) } }
P1 = PutDown(A);
     Try_PickUp(B);
     If σ(Try_PickUp(B)) then Q1
     Else { Try_PickUp(C);
       If σ(Try_PickUp(C)) then Q2
       Else Q3 }
P3 = PutDown(B);
     Try_PickUp(C);
     If σ(Try_PickUp(C)) then Q4
     Else Q5
P4 = PutDown(C); ApplyLever(A, B); PickUp(C); Stack(C, A)
Q1 = Stack(B, C); PickUp(A); Stack(A, B)
Q2 = Stack(C, B); PickUp(A); Stack(A, C)
Q3 = ApplyLever(C, B); PickUp(A); Stack(A, B)
Q4 = Stack(C, A); PickUp(B); Stack(B, C)
Q5 = ApplyLever(A, C); PickUp(B); Stack(B, A)
```

In [14], a complete axiomatization of the valid formulas of an epistemic logic for try actions has been proposed and the completeness of the axiomatization and the decidability and PSpace-completeness of the satisfiability problem for the logic have been proven.

CHAPTER 3. PLANNING IN A MODIFIED BLOCKS WORLD DOMAIN

The modified blocks world domain has been discussed in section 2.4.1. This chapter first specifies a model for the domain, then moves on to specify the logic that has been used in the dissertation for the specification of the plans in the domain, and last but not least presents the generation of plans in the domain.

3.1. Model of the Modified Blocks World Domain

The world state [11] of a modified blocks world domain at any instant is specified by the weights of the blocks and their respective positions at that instant. On the other hand, the knowledge state [11] of the domain at that instant is specified by the positions and knowledge about the weights of the blocks at the instant. In the dissertation, a state of the domain refers to its knowledge state, unless explicitly mentioned. Following are the fluents, specifying the state of the domain, and operators (along with preconditions and effects) available to the robot.

Fluents (x, y are distinct blocks) –

1. Relational:
    - ArmEmpty: true if robot arm is empty, false otherwise.
    - OnTable(x): True if x is on table, false otherwise.
    - On(x, y): True if x is on y, false otherwise.
    - Clear(x): True if there's no block on x, false otherwise.
    - Holding(x): True if the robot arm has x, false otherwise.
2. Knowledge:
    - K(Light(x)): True if known that x is light, false otherwise.
    - K(Heavy(x)): True if known that x is heavy, false otherwise.

Note: The above list of fluents is redundant and can be made more compact.

Operators (x, y are distinct blocks) –

- PickUp(x)
    - Preconditions: ArmEmpty $\wedge$ OnTable(x) $\wedge$ Clear(x) $\wedge$ K(Light(x))
    - Effects:
        - Delete: ArmEmpty $\wedge$ OnTable(x)

- Add: Holding(x)

- PutDown(x)
  - Preconditions: Holding(x) ∧ K(Light(x))
  - Effects:
    - Delete: Holding(x)
    - Add: ArmEmpty ∧ OnTable(x)
- Stack(x, y)
  - Preconditions: Clear(y) ∧ Holding(x) ∧ K(Light(x))
  - Effects:
    - Delete: Clear(y) ∧ Holding(x)
    - Add: ArmEmpty ∧ On(x, y)
- UnStack(x, y)
  - Preconditions: ArmEmpty ∧ On(x, y) ∧ Clear(x) ∧ K(Light(x))
  - Effects:
    - Delete: ArmEmpty ∧ On(x, y)
    - Add: Clear(y) ∧ Holding(x)
- Try_PickUp(x)
  - Preconditions: ArmEmpty ∧ OnTable(x) ∧ Clear(x) ∧ not K(Light(x)) ∧ not K(Heavy(x))
  - Effects: i ∨ ii

    i. (Success of trial)
       - Delete: ArmEmpty ∧ OnTable(x)
       - Add: Holding(x) ∧ K(Light(x))

    ii. (Failure of trial)
       - Delete: None
       - Add: K(Heavy(x))
- ApplyLever(x, y)
  - Preconditions: ArmEmpty ∧ Clear(x) ∧ Clear(y) ∧ K(Heavy(x)) ∧ K(Heavy(y))
  - Effects:
    - Delete: Clear(y)
    - Add: On(x, y)
- RevLever(x, y)

- Preconditions: ArmEmpty ∧ On(x, y) ∧ Clear(x) ∧ K(Heavy(x)) ∧ K(Heavy(y))
- Effects:
  - Delete: On(x, y)
  - Add: OnTable(x) ∧ Clear(y)

Note:

i. In the operators, PutDown(x) and Stack(x, y), the precondition K(Light(x)) is redundant, because K(Light(x)) is implied by Holding(x).

ii. Similarly, in RevLever(x, y), the precondition K(Heavy(y)) is redundant, because K(Heavy(y)) is implied by On(x, y) ∧ K(Heavy(x))

3.2. Logic for specification of the plans

A try action *Try_a* gets its name from the fact that it is trial of action *a*. This makes try actions non-deterministic, provided the preconditions for *a* are not known to be either satisfied or not satisfied, because action *a* may succeed or fail when it is tried. In the MBW domain, there is only one non-deterministic operator, the try operator *Try_PickUp(x)*, while all the other operators are deterministic. This section explains the logic that is used in the dissertation for the specification of the plans for the MBW domain.

If *s* be a goal state, then the plan from state *s* is written in this logic as:

Nil

The execution of this plan refers to doing nothing.

If (1) *s* is a state from which a deterministic action *b* is possible, (2) a planner decides the next action from state *s* as *b*, (3) *s'* is the state reached by executing *b* from state *s*, and (4) ψ is the plan from state *s'* to reach goal, then the plan from state *s* is written by the planner, when using this logic, as:

<b> ψ

The execution of this plan is read as execute action *b* and then execute plan ψ. The corresponding plan graph is shown in Figure 3.1(a). (In Figure 3.1, a solid rectangular box represents a state, a solid edge represents an action that causes transition from one state to another, a dashed rectangular box represents a set of states, and a dashed edge represents a plan.)

Now, let s be a state from which *Try_a* is possible (a is not possible). If a planner decides the next action from state s as *Try_a*, and s', where it is known that α is true (and β is false), is the state reached on success of the trial from state s, and s'', where it is known that β is true (and α is false), is the state reached on failure of the trial from state s, and θ, φ are the plans from state s', s'' respectively to reach goal, then the plan from state s, in this logic, is written as:
<Try_a> ((K(α) => θ) | (K(β) => φ))

The execution of this plan is read as execute action *Try_a*, now if it is known that α is true, execute plan θ, otherwise, if it is known that β is true, execute plan φ. The corresponding plan graph is shown in Figure 3.1(b).

Also, if π is a plan in this logic then, "π Nil" and "Nil π" can both be written as "π".

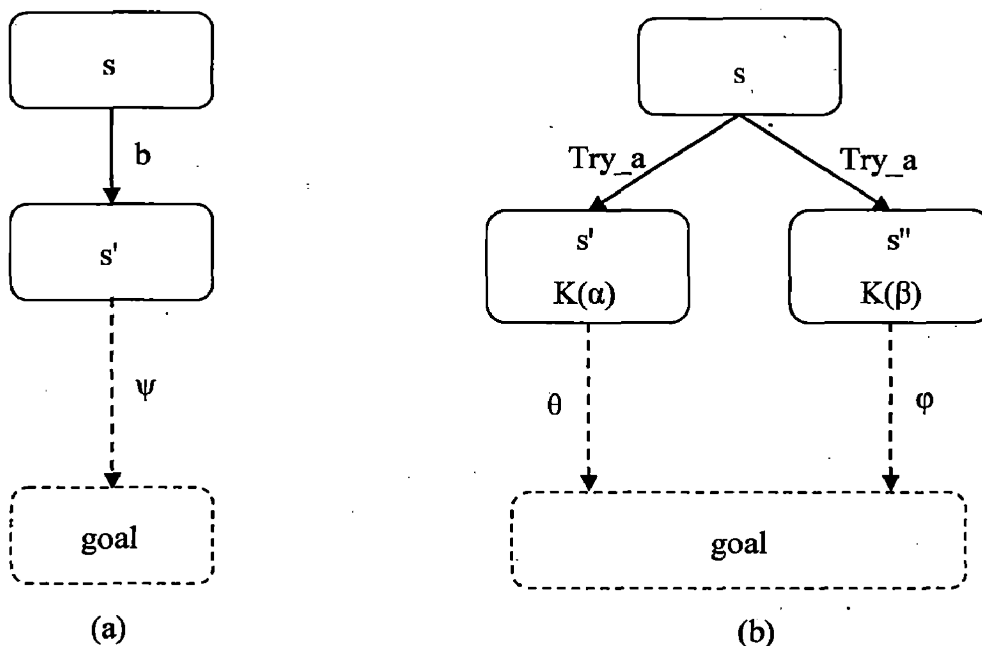


Figure 3.1: Plan graphs when (a) a deterministic action b is chosen from a state s, (b) a try action Try_a is chosen from a state s.

3.3. *Plan Generation*

Unlike classical plans, plans in non-classical domains cannot always be written as a sequence of actions. They have one or more of if-then-else, looping and recursive constructs. Automatic generation of programs (program synthesis) is a non-trivial job. Thus the difficulty of generating plans in non-classical domains. In this chapter, generation of plans in the MBW domain, a non-classical domain due to incomplete state information, is discussed. The plans

generated are in the logic discussed in section 3.2. Initially, (optimal) plans for much simpler problems are generated and then the difficulty of the problems, with respect to the number of problem instances, which the planner can solve, has been increased.

3.4. Plans for default Initial State

The state in which all the blocks are on table and weights of none of the blocks are known is considered as the default initial state of the MBW domain. The problem now is to generate plans for constructing a tower of all the blocks from this default initial state.

3.4.1. Information Gathering

As per the operators available, no block can be moved without obtaining its weight information. Therefore, only a block that is not required to be moved can be of unknown weight. Since the final goal to achieve is a single tower of the blocks and since all the blocks are initially on the table, it implies that all except one of the *N* blocks are required to be moved. Therefore, the weights of at least *N-1* blocks should be obtained before being able to construct the tower.

Also, if there is a block whose weight is not known, only light blocks can be stacked on top of it as per the operators available (also because the block can be light which requires all the blocks above it to be light). Therefore, it befits that if there are blocks that are known to be heavy then one the blocks known to be heavy has to be the base of the tower of all the blocks, which implies that all the other blocks are to be moved on the heavy block that's decided to be as the base, and therefore, the weights of all the other blocks should be known, which results in knowing the weights of all the blocks in the domain.

In short, to construct a tower of blocks, the minimum information that requires to be gathered about the weights of the blocks is to either known all except one of the blocks to be light or know the weights of all the blocks. Once the information required is gathered, the problem reduces to a classical planning problem of constructing a tower of blocks from an initial configuration of blocks (i.e., the tower can be constructed without any more requirement of the non-deterministic operator *Try_PickUp(x)*). In the rest of the dissertation, information gathering means to gather the minimum required information, if not mentioned how much information to gather, for the goal of tower construction, if it is not mentioned

otherwise. After information gathering for a particular goal, the required goal can be achieved without any more use of try actions (trials) i.e. as a simple sequence of actions.

To gather information about the weight of a block, a try pick up operator is to be used on the block. If the block gets picked up, then it is known that the block is light, otherwise it is known that the block is heavy.

To gather the minimum required information, a block is selected from the set of blocks whose weight is not known and it is tried to be picked up. If the minimum required information is gathered, the process is stopped; otherwise the same process is repeated, after putting down the block in the robot arm if the trial succeeds and doing nothing if the trial fails.

Example 3.1:

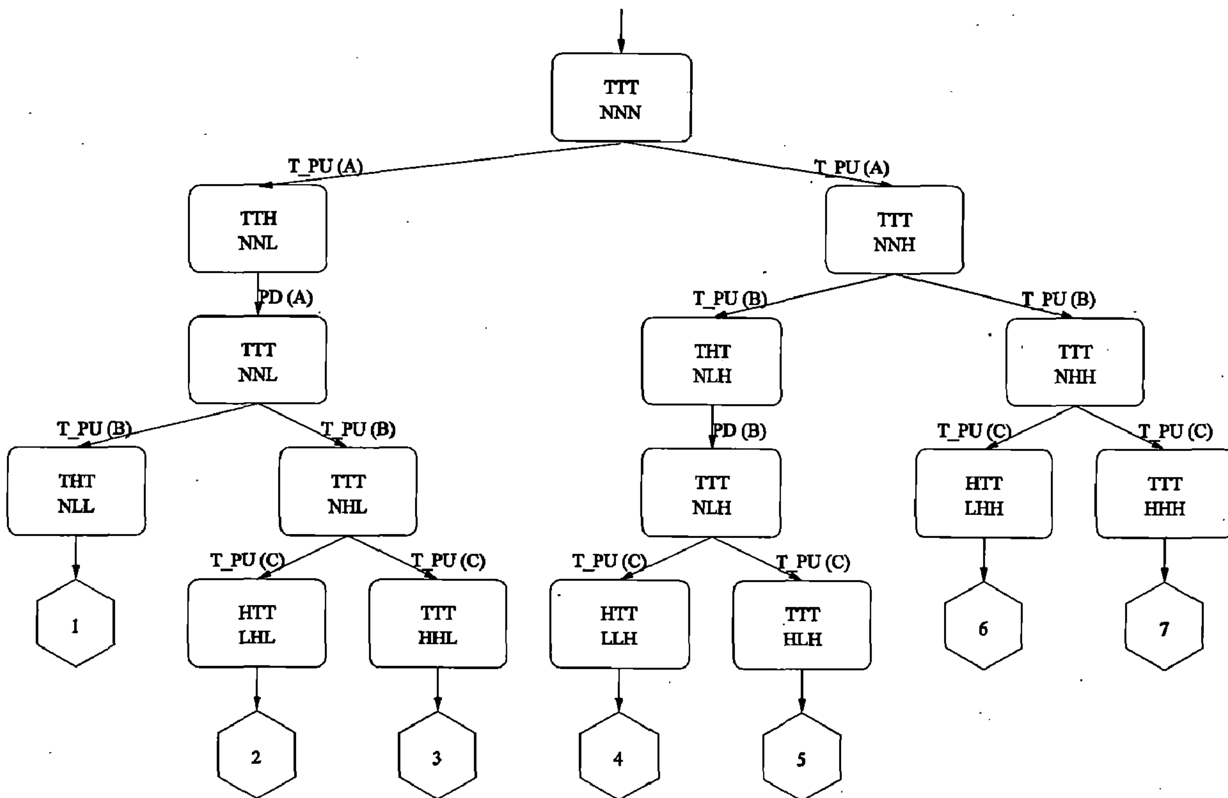


Figure 3.2: Plan graph for information gathering from the default initial state

Consider the 3-blocks MBW domain with A, B and C as the blocks. The plan for gathering minimum required information for tower construction from default initial state is:

<Try_PickUp(A)> ((K(Light(A)) => P1) | (K(Heavy(A)) => P6))

P1 = <PutDown(A)> <Try_PickUp(B)> (K(Light(B)) => P2) | (K(Heavy(B)) => P3))

P2 = Nil

P3 = <Try_PickUp(C)> (K(Light(C)) => P4) | (K(Heavy(C) => P5))

P4 = Nil

P5 = Nil

P6 = <Try_PickUp(B)> (K(Light(B)) => P7) | (K(Heavy(B)) => P10))

P7 = <PutDown(B)> <Try_PickUp(C)> (K(Light(C)) => P8) | (K(Heavy(C) => P9))

P8 = Nil

P9 = Nil

P10 = <Try_PickUp(C)> (K(Light(C)) => P11) | (K(Heavy(C) => P12))

P11 = Nil

P12 = Nil

The plan graph corresponding to the above plan is shown in Figure 3.2. In the figure, the states are named as PQR-XYZ. P, Q and R represent the positions of C, B and A respectively, i.e. whether they lie on table (T) or are in the arm (H) of the robot, or they lie on a particular block and X, Y and Z represent the knowledge about the weights of the blocks C, B and A respectively, i.e. whether the blocks are known to be light (L) or heavy (H) or it is not known (N) whether they are light or heavy. The edges are labeled with actions. T_PU(x) represents the Try_PickUp(x) operator, and PD(x) represents the PutDown(x) operator. The leaf nodes correspond to the valid set of goal states from the state achieved so far.

- Leaf 1: (TAC-NLL) | (TAC-LLL) | (TAC – HLL) | (TCB-NLL) | (TCB-LLL) | (TCB-HLL)
- Leaf 2: (ATB-LHL) | (BTC-LHL)
- Leaf 3: (BTC-HHL) | (TCB-HHL)
- Leaf 4: (BAT-LLH) | (ACT-LLH)
- Leaf 5: (ACT-HLH) | (TAC-HLH)
- Leaf 6: (BAT-LHH) | (ATB-LHH)
- Leaf 7: (BAT-HHH) | (ACT-HHH) | (ATB-HHH) | (BTC-HHH) | (TCB-HHH) | (TAC-HHH)

3.4.2. Planner for Information Gathering

Let *0, 1, 2... N-1* be the *N* blocks. Then procedure *planDIG* returns the plan for gathering the minimum information required for constructing a tower of the blocks from the default initial state.

Procedure *planDIG* {

Plan ← *plannerDIG*(0, 0, false)

Return *Plan* }

Procedure *plannerDIG*(*weightKnownBlocksCount*, *lightBlocksCount*, *isArmEmpty*) {

If *lightBlocksCount* = *N*-1 or *weightKnownBlocksCount* = *N* {

Return Nil }

Plan ← *empty string*

If not *isArmEmpty* {

i ← *weightKnownBlocksCount*-1

Append <PutDown(*i*)> to the end of *Plan* }

j ← *weightKnownBlocksCount*

Append <Try_PickUp(*j*)> ((K(Light(*j*)) => *plannerDIG*(*j*+1, *lightBlocksCount*+1, false)) | (K(Heavy(*j*)) => *plannerDIG*(*j*+1, *lightBlocksCount*, true))) to the end of *Plan*

Return *Plan* }

3.4.3. Tower Construction

After gathering the required information, construction of tower is trivial. All the heavy blocks can be stacked into a tower, followed by the light blocks. If there is a block whose weight is not known, then all the light blocks can be stacked on the former block. (Remember, after the information gathering, if there remains a block whose weight is not known, then all the other blocks are known to be light.)

Starting to stack the blocks only after gathering all the required information is not the most optimum way for constructing tower from the default initial state. If there remains a (light) block in robot arm at the end of information gathering, the light block is required to be put down on the table if the heavy blocks are not already stacked. Alternatively, if the heavy blocks are already stacked, the block in arm can be directly placed on top of the heavy blocks, and thus, operation of additional *PutDown(x)* and *PickUp(x)* are avoided.

Example 3.2:

Consider Figure 3.2 (Example 3.1).

At state HTT-LHH (the node before leaf 6), the plan to achieve the goal of leaf 6 is:
<PutDown(C)> <ApplyLever(B, A)> <PickUp(C)> <Stack(C, B)>

Instead, if during information gathering, the plan graph is as in Figure 3.3 (AL(x, y) represents the operator ApplyLever (x, y), the plan to achieve the goal at 6 from the state HAT-LHH is:

<Stack(C, B)>

thus avoiding one PutDown(C) and one PickUp(C) in this path of the plan.

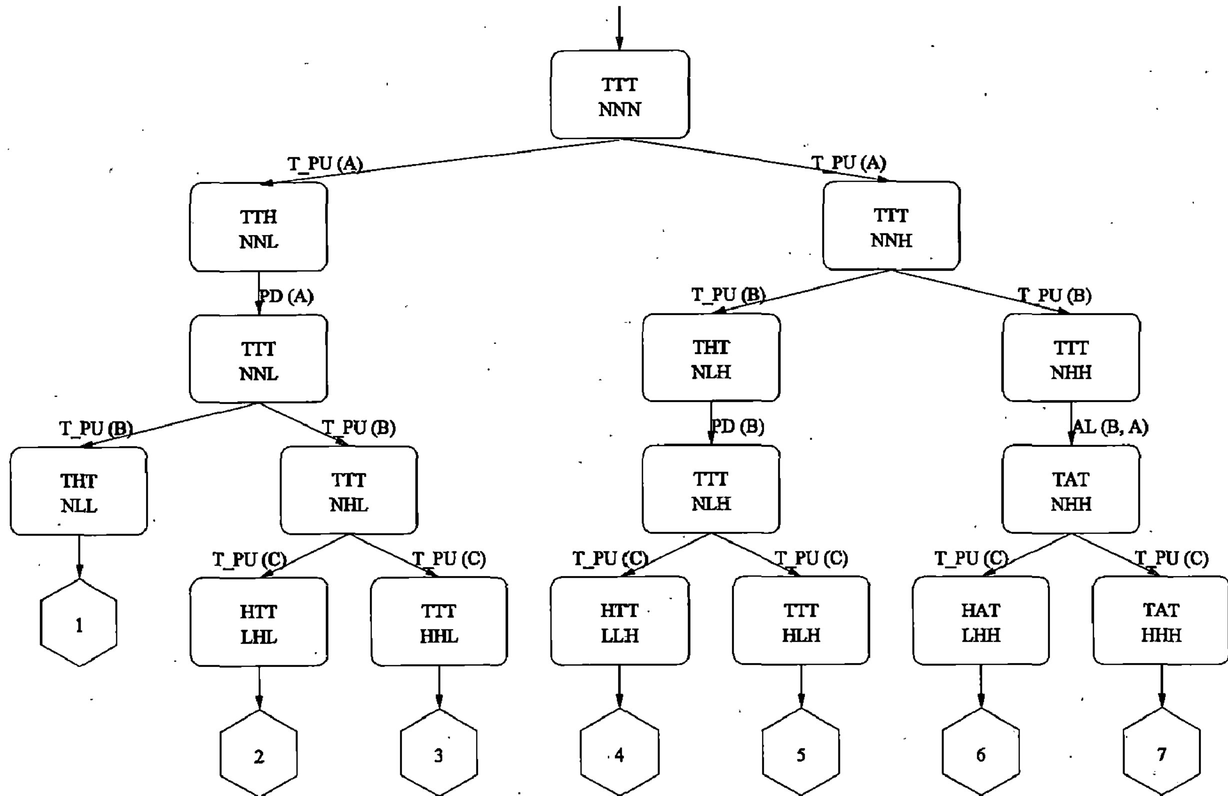


Figure 3.3: Plan graph for stacking of heavy blocks alongside information gathering

The (optimum) plan for tower construction in this 3-blocks MBW domain is:

<Try_PickUp(A)> ((K(Light(A)) => P1) | (K(Heavy(A)) => P6))

P1 = <PutDown(A)> <Try_PickUp(B)> (K(Light(B)) => P2) | (K(Heavy(B)) => P3))

P2 = <Stack(B, C)> <PickUp(A)> <Stack(A, B)>

P3 = <Try_PickUp(C)> (K(Light(C)) => P4) | (K(Heavy(C) => P5))

P4 = <Stack(C, B)> <PickUp(A)> <Stack(A, C)>

P5 = <ApplyLever(C, B)> <PickUp(A)> <Stack(A, C)>

P6 = <Try_PickUp(B)> (K(Light(B)) => P7) | (K(Heavy(B)) => P10))

P7 = <PutDown(B)> <Try_PickUp(C)> (K(Light(C)) => P8) | (K(Heavy(C) => P9))

P8 = <Stack(C, A)> <PickUp(B)> <Stack(B, C)>

P9 = <ApplyLever(C, A)> <PickUp(B)> <Stack(B, C)>

P10 = <ApplyLever(B, A)> <Try_PickUp(C)> (K(Light(C)) => P11) | (K(Heavy(C) => P12))

P11 = <Stack(C, B)>

P12 = <ApplyLever(C, B)>

3.4.4. Planner for Tower Construction

Let *0, 1, 2*... *N-1* be the *N* blocks. Let *lightBlocksArray*[1...*N*] be the array that contain the blocks that are known to be light. Then procedure *planDTC* returns the plan for constructing a tower of the blocks from the default initial state.

Procedure *planDTC* {

  Plan ← *plannerDTC*(0, 0, false, -1)

  If *Plan* is *empty string* {

    Return Nil }

  Return *Plan* }

Procedure *plannerDTC*(*weightKnownBlocksCount*, *lightBlocksCount*, *isArmEmpty*, *top*) {

  Plan ← *empty string*

  i ← *weightKnownBlocksCount*-1

  If *isArmEmpty* {

    If *top* ≠ -1 {

      Append <ApplyLever(*i*, *top*) to the end of *Plan* }

    top ← *i* }

  If *weightKnownBlocksCount* = *N* {

    Append *stackLightBlocks*(*lightBlocksCount*, *isArmEmpty*, *top*) to the end of *Plan*

    Return *Plan* }

  If *lightBlocksCount* = *N*-1 {

    Append *stackLightBlocks*(*lightBlocksCount*, *isArmEmpty*, *lightBlocksCount*) to the end of *Plan*

    Return *Plan* }

  If not *isArmEmpty* {

    Append <PutDown(*i*)> to the end of *Plan*

    lightBlocksArray[*lightBlocksCount*] ← *i* }

```
j ← weightKnownBlocksCount
Append <Try_PickUp(j)> ((K(Light(j)) => plannerDTC(j+1, lightBlocksCount+1, false, top)) | (K(Heavy(j)) => plannerDTC(j+1, lightBlocksCount, true, top))) to the end of Plan
Return Plan }

Procedure stackLightBlocks(count, isArmEmpty, top) {
    Plan ← empty string
    If not isArmEmpty {
        If top = N-1 {
            Append <Stack(N-2, top)> to the end of Plan
            top ← N-2 }
        Else {
            Append <Stack(N-1, top)> to the end of Plan
            top ← N-1 }
        count ← count-1 }
    While count ≠ 0 {
        i ← lightBlocksArray[count]
        Append <PickUp(i)> to the end of Plan
        Append <Stack(i, top)> to the end of Plan
        top ← i
        count ← count-1
    Return Plan }
```

3.5. *Plans from any Initial State*

So far, the initial state was fixed to the default initial state of all the blocks being on the table and weights of none of the blocks known. Now, the concepts used for default initial state would be generalized to having any initial state. In this section the concepts are generalized for constructing a tower from any initial state of the blocks.

3.5.1. Domain of initial states

Having any initial state means that there may be towers of blocks in the initial state and that the weights of some or all of the blocks may be known. The domain is considered to consist of only those states where every block in the domain can be moved. When the weights of all

the blocks are known, the problem (or when all except one of the blocks are known to be light) is reduced to a classical planning problem.

From the operators available, an unknown weight block can be moved (or tried to be moved, and consequently know its weight) only from the table. Since there are no operators to move or even gain information about an unknown weight block if the block is anywhere but on the table, all the blocks whose weights are not known must lay on the table. In an MBW domain, no light block can lie on op a heavy block. Therefore, in a tower, on the top of a block that is known to be light, there can be only light blocks and a block known to be heavy, cannot be on a block that is known to be light or whose weight is not known.

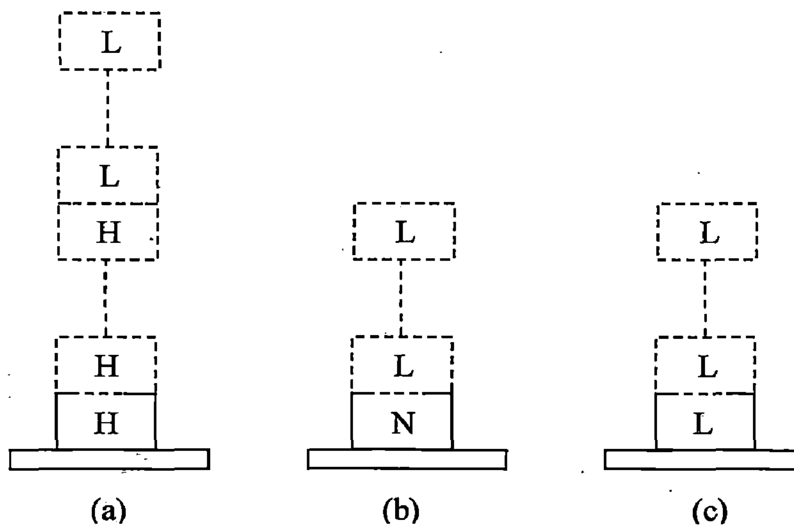


Figure 3.4: (a) Heavy tower; (b) Unknown tower; (c) Light tower

Therefore, there are three possibilities for towers.

i. Heavy Tower: There is at least one block known to be heavy in the bottom part of the tower and zero or more blocks known to be light in the top part of the tower.

ii. Unknown Tower: There is a block whose weight is not known at the bottom of the tower and a stack of zero or more towers, which are known to be light, on the unknown weight block.

iii. Light Tower: The tower contains blocks that are all known to be light.

The three above possibilities of towers are shown in Figures 3.4(a), 3.4(b), 3.4(c) respectively. The knowledge about the weights is presented in each block (L is for light, H is for heavy, and N is for unknown). The dotted blocks in the figure represent that that they are optional.

The initial state in the MBW domain can thus be a set of zero or more towers of each of the above kind with an empty robot arm or with the robot arm holding a light block.

3.5.2. Single Tower construction

To construct a single tower from any initial state of the MBW domain, the minimum information to be known about the blocks is same as discussed in section 3.4.1.

i. If any block in the domain is known to be heavy, then the information regarding the weights of all the blocks must be known or gathered.

ii. The weight of at most one block may not be known if all the other blocks are known to be light.

3.5.3. Planner for single tower construction

This section though does not contain the complete algorithm mentions some points that are useful in developing the planner that generates plans for constructing a tower from a given initial state. The next section illustrates the points with examples. The following are the points that are been talked about.

1. To cause the movement of minimum number of blocks in the construction of a single tower, one of those existing towers is chosen as the destination block where maximum number of blocks are in their right place, and stack all the other blocks (the blocks in other towers and those misplaced in the chosen tower) on the correctly placed blocks in the chosen tower. Thus, the destination tower from the existing towers is chosen in one of the following ways.
   i. If there are one or more heavy towers in the initial state, the heavy tower with maximum number of heavy blocks in it is chosen as the destination tower.
   ii. If there are only light towers in the initial state, then the tower with maximum number of blocks is chosen as the destination tower.
   iii. If there are unknown towers but no heavy towers in the initial state, then one of the unknown towers is chosen as the destination tower after gathering the required information about the weights of unknown blocks in the unknown towers.

2. A block in hand should be placed on table and not on the destination tower only if is known that there are no misplaced heavy blocks. This condition requires consideration of two cases to decide whether a block should be placed on the destination tower or not, one

when there are more than one towers that are not light towers in the current state, and other when this is not true.

3. Light blocks should be moved from above heavy blocks only if there are misplaced heavy blocks. This condition requires two cases to be considered while deciding whether the light blocks from heavy towers are to be move from above them, one when there is only one tower and other when there is more than one heavy tower. In case when there is only one heavy tower, the light blocks from the tower are required to be moved only when an unknown weight block is found to be heavy.
4. Light blocks should be moved from the unknown towers only when the weights of the unknown weight blocks are to be found. The weight of an unknown weight block is to be found only if there are heavy towers or there are other unknown weight blocks. So to allow the movement of only minimum number of blocks, when there are no heavy towers, the information about weight of that unknown block which has maximum number of blocks over it is chosen to be found last. This helps when there are no unknown weight blocks that are newly found to be heavy. This results to consider cases when there are heavy towers in the initial state, when there are no heavy towers in the initial state but one of the blocks whose information is newly gained has been found to be heavy, and the last when there are no heavy towers in the initial state but there is only one unknown weight block in the current state.
5. Light blocks are moved to the destination tower only when there are no heavy towers and unknown towers except the destination tower in the current state. To be able to remove all the heavy towers and the unknown towers with minimum possible moves, the light blocks from above the heavy towers and unknown towers should be moved to table or another light tower in the current state.

3.5.4. Example Plans

In all the figures in this sub-section, inside each block its name (denoted by a number) and the knowledge about their weights (L, H, or N) is written. A block in air represents that the block in robot arm.

Example 3.3:

This example illustrates point 1 in section 3.5.3.

The plan for the initial configuration as in Figure 3.5(a) is <ApplyLever(2,1)>. Note that block 2 is moved to block 1 and not blocks 1 and 0 or the tower with block 2.

Similarly, the The plan for the initial configuration as in Figure 3.5(b) is <PickUp(1)> <Stack(1,0)>.

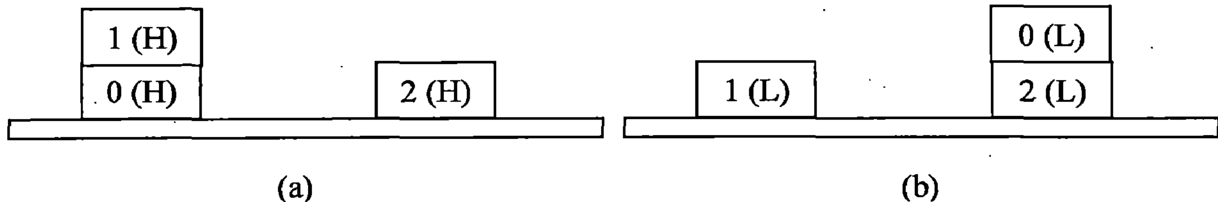


Figure 3.5: Sample configurations of initial state of MBW domain where there are unequal size towers (a) there are heavy towers with unequal number of heavy blocks; (b) there are no heavy and unknown towers but more than one light towers of unequal size

Example 3.4:

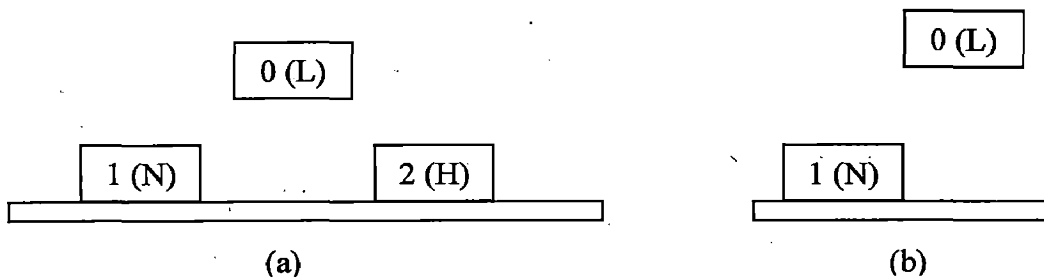


Figure 3.6: Sample configurations of initial state of MBW domain where a block is in robot arm: (a) there are more than one non-light towers; (b) there is only one non-light tower

This example illustrates point 2 in section 3.5.3.

The plan for the initial configuration as in Figure 3.6(a) is:
<PutDown(0)> <Try_PickUp(1)> ((K(light(1)) => P1) | (K(heavy(1)) => P2))
P1 = <Stack(1,2)> <PickUp(0)> <Stack(0,1)>
P2 = <ApplyLever(1,2)> <PickUp(0)> <Stack(0,1)>
The first line in the above plan shows that the block in robot arm is kept down on the table.

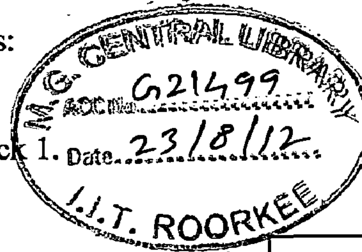
The plan for the initial configuration as in Figure 3.6(b) is <Stack(0,1)>. The block in robot arm is not placed on table but directly placed on the destination block.

Example 3.5:

The example illustrates point 3 in section 3.5.3.

The plan for the initial configuration as in Figure 3.7(a) is:
<UnStack(0,2)> <PutDown(0)> <UnStack(1,3)> <PutDown(1)> <ApplyLever(3,2)> <PickUp(0)> <Stack(0,3)> <PickUp(1)> <Stack(1,0)>
The light blocks 0 and 1 are to be should be moved from above 2 and 3 respectively.

The plan for the initial configuration as in Figure 3.7(b) is:
<PickUp(1)> <Stack(1,0)>
Here the light block 0 need not be moved from above block 1.



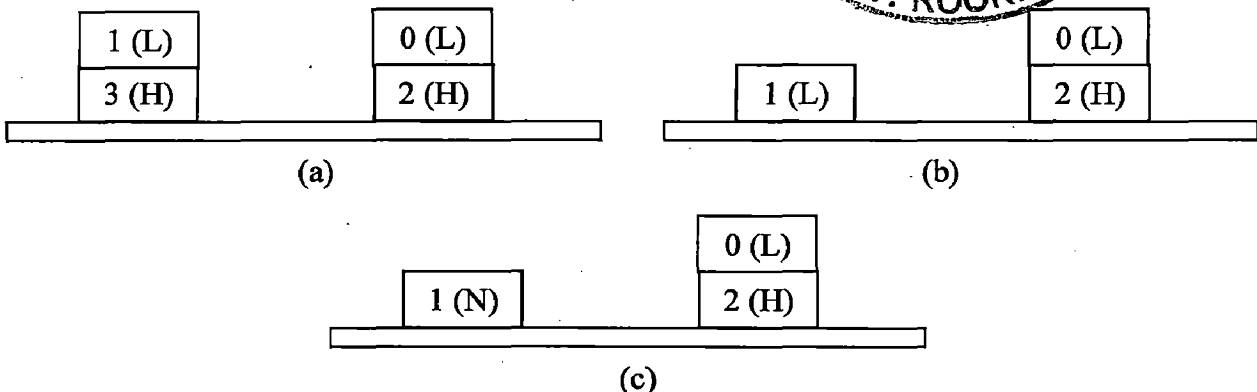


Figure 3.7: Sample configurations of initial state of MBW domain where there are light blocks in heavy towers: (a) there are more than one heavy towers; (b) there is only one heavy tower and no unknown towers; (c) there is one heavy tower with unknown towers.

The plan (line numbers are included for explanation below) for the initial configuration as in Figure 3.7(c) is:

1 <Try_PickUp(1)> ((K(light(1)) => P1) | (K(heavy(1)) => P2))
2 P1 = <Stack(1,0)>
3 P2 = <UnStack(0,2)> <PutDown(0)> <ApplyLever(1,2)> <PickUp(0)> <Stack(0,1)>

In this plan, note that initially the light block 0 is not removed from top of the heavy block 2 (line 1). 0 is removed from top of 2 only when block 1 is found to be heavy (line 3) and not otherwise (line 2).

Example 3.6:

The example illustrates point 4 in section 3.5.3.

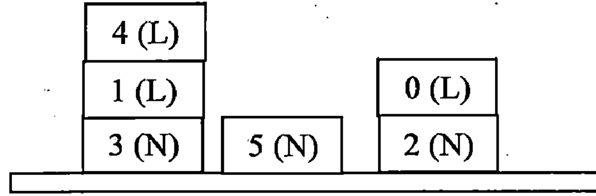


Figure 3.8: Sample configuration of initial state of MBW domain where there are unknown towers and no heavy towers.

The plan for the initial configuration as in Figure 3.8 is:

<Try_PickUp(5)> ((K(light(5)) => P1) | (K(heavy(5)) => P6))

P1 = <PutDown(5)> <UnStack(0,2)> <PutDown(0)> <Try_PickUp(2)> ((K(light(2)) => P2) | (K(heavy(2)) => P3))

P2 = <Stack(2,4)> <PickUp(5)> <Stack(5,2)> <PickUp(0)> <Stack(0,5)>

P3 = <UnStack(4,1)> <PutDown(4)> <UnStack(1,3)> <PutDown(1)> <Try_PickUp(3)> ((K(light(3)) => P4) | (K(heavy(3)) => P5))

P4 = <Stack(3,2)> <PickUp(5)> <Stack(5,3)> <PickUp(0)> <Stack(0,5)> <PickUp(4)> <Stack(4,0)> <PickUp(1)> <Stack(1,4)>

P5 = <ApplyLever(3,2)> <PickUp(5)> <Stack(5,3)> <PickUp(0)> <Stack(0,5)> <PickUp(4)> <Stack(4,0)> <PickUp(1)> <Stack(1,4)>

P6 = <UnStack(0,2)> <PutDown(0)> <Try_PickUp(2)> ((K(light(2)) => P7) | (K(heavy(2)) => P10))

P7 = <PutDown(2)> <UnStack(4,1)> <PutDown(4)> <UnStack(1,3)> <PutDown(1)> <Try_PickUp(3)> ((K(light(3)) => P8) | (K(heavy(3)) => P9))

P8 = <Stack(3,5)> <PickUp(0)> <Stack(0,3)> <PickUp(2)> <Stack(2,0)> <PickUp(4)> <Stack(4,2)> <PickUp(1)> <Stack(1,4)>

P9 = <ApplyLever(3,5)> <PickUp(0)> <Stack(0,3)> <PickUp(2)> <Stack(2,0)> <PickUp(4)> <Stack(4,2)> <PickUp(1)> <Stack(1,4)>

P10 = <ApplyLever(2,5)> <UnStack(4,1)> <PutDown(4)> <UnStack(1,3)> <PutDown(1)> <Try_PickUp(3)> ((K(light(3)) => P11) | (K(heavy(3)) => P12))

P11 = <Stack(3,2)> <PickUp(0)> <Stack(0,3)> <PickUp(4)> <Stack(4,0)> <PickUp(1)> <Stack(1,4)>

P12 = <ApplyLever(3,2)> <PickUp(0)> <Stack(0,3)> <PickUp(4)> <Stack(4,0)> <PickUp(1)> <Stack(1,4)>

Few notes from the plan that explain point 4 from section 3.5.3:

Try_PickUp(3) action is suggested at P3 (5 is known to be light and 2 is known to be heavy), or P7 (5 is known to be heavy and 2 is known to be light) or P10 (both 5 and 2 are known to be heavy).

1. First thing is that it is always the last one whose weight information is tried to be gained.
2. Second, its weight is not tried to be known when both 5 and 2 are known to be light since there are no heavy blocks known in the initial state and after gaining the information about the two other unknown weight blocks as them being light, there remain only one unknown weight block and all other light blocks.
3. Third, when both 5 and 2 are known to be light and there remains only 3 whose weight is not known (and there are no heavy blocks), the blocks 1, 4 from above 3 are not required to be moved unlike in all the other cases.

For illustrating point 5 in section 3.5.3, note that in the plans in all the four examples above, whenever light blocks are moved from heavy or unknown towers they are placed on table. They could also be placed on other light towers, but not on another heavy or unknown tower.

CHAPTER 4. RESULTS

The algorithms for planners for information gathering and tower construction from the default initial state in the modified blocks world domain have been discussed in the previous chapter. These algorithms were implemented as a java program and run using java version 1.6.0_20, OpenJDK Runtime Environment (IcedTea6 1.9.13) (6b20-1.9.13-0ubuntu1~10.04.1), OpenJDK Client VM (build 19.0-b09, mixed mode, sharing) on the terminal window of an Intel® Core™ 2 CPU T5200 @ 1.60GHz. Section 4.1 presents some screenshots of the terminal window when the planners were run on it. Section 4.2 analyzes the plans generated by the planners.

4.1. Screenshots

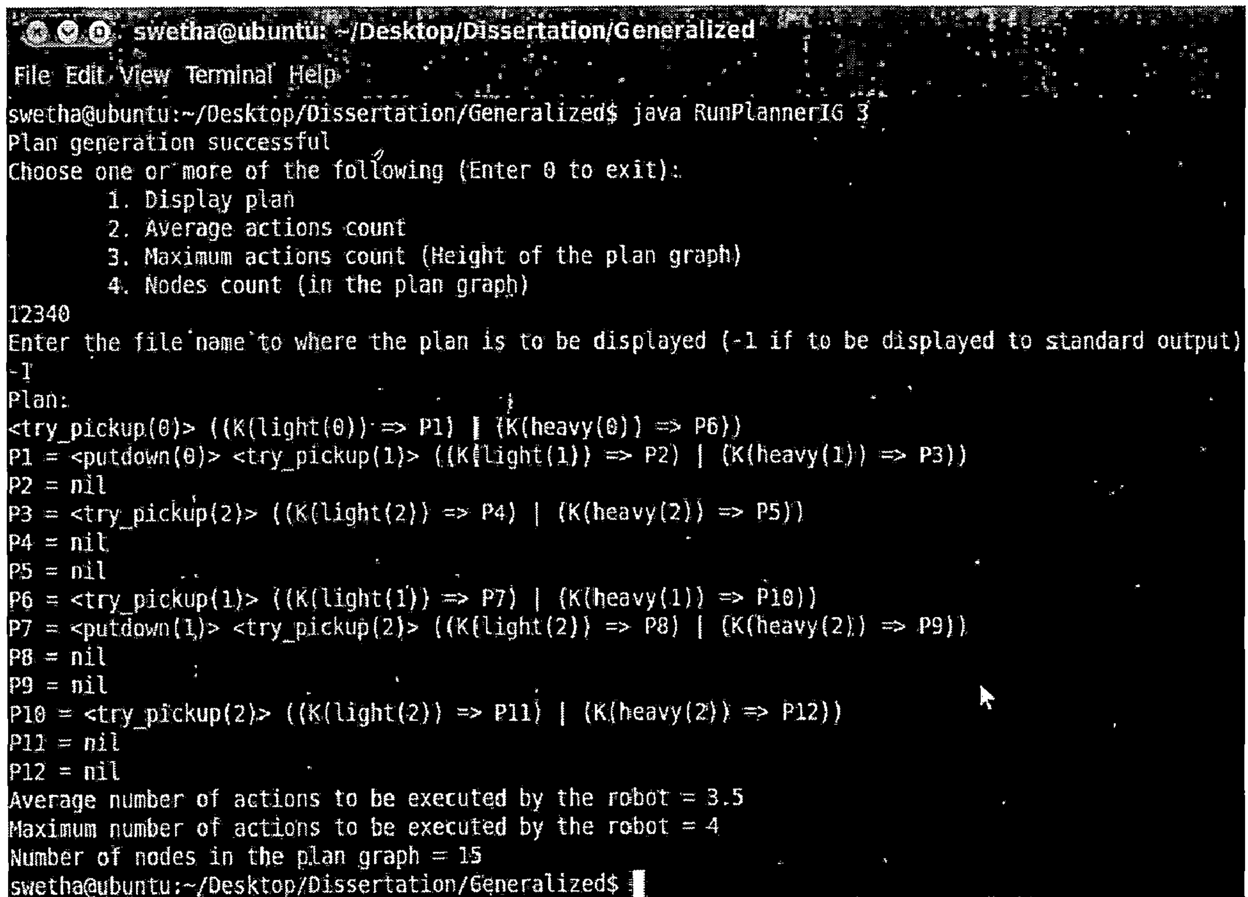


Figure 4.1: Screenshot for and while running the planner for information gathering from default initial state with input number of blocks as three.

To run the planner for information gathering from default initial state, the input command on the terminal is *java RunPlannerIG <number of blocks>*; while to run that for tower

construction from default initial state the input command is *java RunPlannerDTC <numberof blocks>*. The planners also keep count of the number of actions for each permutation of the weights and the total number of possible states the domain can go through during the execution of the plans. Figure 4.1 and Figure 4.2 show the screenshots for and while running the planners for the 3-blocks MBW domain.

```
swetha@ubuntu: ~/Desktop/Dissertation/Generalized
File Edit View Terminal Help
swetha@ubuntu:~/Desktop/Dissertation/Generalized$ java RunPlannerDTC 3
Plan generation successful
Choose one or more of the following (Enter 0 to exit):
        1. Display plan
        2. Average actions count
        3. Maximum actions count (Height of the plan graph)
        4. Nodes count (in the plan graph)
12340
Enter the file name to where the plan is to be displayed (-1 if to be displayed to standard output)
-1
Plan:
<try_pickup(0)> ((K(light(0)) => P1) | (K(heavy(0)) => P6))
P1 = <putdown(0)> <try_pickup(1)> ((K(light(1)) => P2) | (K(heavy(1)) => P3))
P2 = <stack(1,2)> <pickup(0)> <stack(0,1)>
P3 = <try_pickup(2)> ((K(light(2)) => P4) | (K(heavy(2)) => P5))
P4 = <stack(2,1)> <pickup(0)> <stack(0,2)>
P5 = <apply_lever(2,1)> <pickup(0)> <stack(0,2)>
P6 = <try_pickup(1)> ((K(light(1)) => P7) | (K(heavy(1)) => P10))
P7 = <putdown(1)> <try_pickup(2)> ((K(light(2)) => P8) | (K(heavy(2)) => P9))
P8 = <stack(2,0)> <pickup(1)> <stack(1,2)>
P9 = <apply_lever(2,0)> <pickup(1)> <stack(1,2)>
P10 = <apply_lever(1,0)> <try_pickup(2)> ((K(light(2)) => P11) | (K(heavy(2)) => P12))
P11 = <stack(2,1)>
P12 = <apply_lever(2,1)>
Average number of actions to be executed by the robot = 6.25
Maximum number of actions to be executed by the robot = 7
Number of nodes in the plan graph = 33
swetha@ubuntu:~/Desktop/Dissertation/Generalized$
```

Figure 4.2: Screenshot for and while running the planner for tower construction from default initial state with input number of blocks as three.

4.2. *Analysis of the plans*

Various parameter values were computed for the plans generated for different number of blocks in the domain. Table 4.1 shows the different parameters computed and their respective values for blocks from 1 to 18. Figure 4.3, Figure 4.4 and Figure 4.5 are the graph plots for the various parameters.

The plans generated by both, procedure *plannerDIG* and procedure *plannerDTC*, are intuitively the optimal plans for information gathering and tower construction, respectively,

from the default initial state. The robot executing the plan for information gathering has to try to pick up each block and putdown a block every time it gets lifted. With increase in number of blocks, the average as well as the maximum (represented by the height of the plan graph) number of trials and number of putdowns increases linearly. Thus, the graphs in Figure 4.3 and Figure 4.4 show the linear variation of average number of actions and height of the plan graph respectively with variation in number of blocks, in case of information gathering. Similarly, for tower construction from the default initial state, the blocks are tried to be lifted, the heavy blocks stacked and the light blocks placed back on the table, then the light blocks are stacked. These are all linear operations. Thus, the graphs are linear for tower construction also, but with more slope than that in information gathering because of increase in average and maximum number of operations.

Table 4.1: Computed parameter values for different number of blocks

Number of blocks	Average number of actions over all permutations of blocks' weights		Height of the plan graph		Number of nodes in the plan graph	
	Information gathering	Tower construction	Information gathering	Tower construction	Information gathering	Tower construction
1	0	0	0	0	1	1
2	1.5	2.5	2	3	5	8
3	3.5	6.25	4	7	15	33
4	5.25	9.625	6	11	35	94
5	6.875	12.8125	8	15	75	235
6	8.4375	15.90625	10	19	155	552
7	9.96875	18.95313	12	23	315	1253
8	11.48438	21.97656	14	27	635	2786
9	12.99219	24.98828	16	31	1275	6111
10	14.49609	27.99414	18	35	2555	13276
11	15.99805	30.99707	20	39	5115	28633
12	17.49902	33.99854	22	43	10235	61398
13	18.99951	36.99927	24	47	20475	131027
14	20.49976	39.99963	26	51	40955	278480
15	21.99988	42.99982	28	55	81915	589773
16	23.49994	45.99991	30	59	163835	1245130
17	24.99997	48.99995	32	63	327675	2621383
18	26.49998	51.99998	34	67	655355	5504964

The third graph, in Figure 4.5, is for the number of nodes in the plan graph. The plan generated is for all the possible permutations of the weights of the blocks. Since there are 2^N possible permutations, the plan size is also of the same order for information gathering. For tower construction, the task after all the information is gathered is to stack all the light blocks. Since, the plan is generated for all possible permutations and the average number of light blocks over all possible permutations is of linear order, the number of nodes in the plan graph of tower construction are of the order of $N*2^N$.

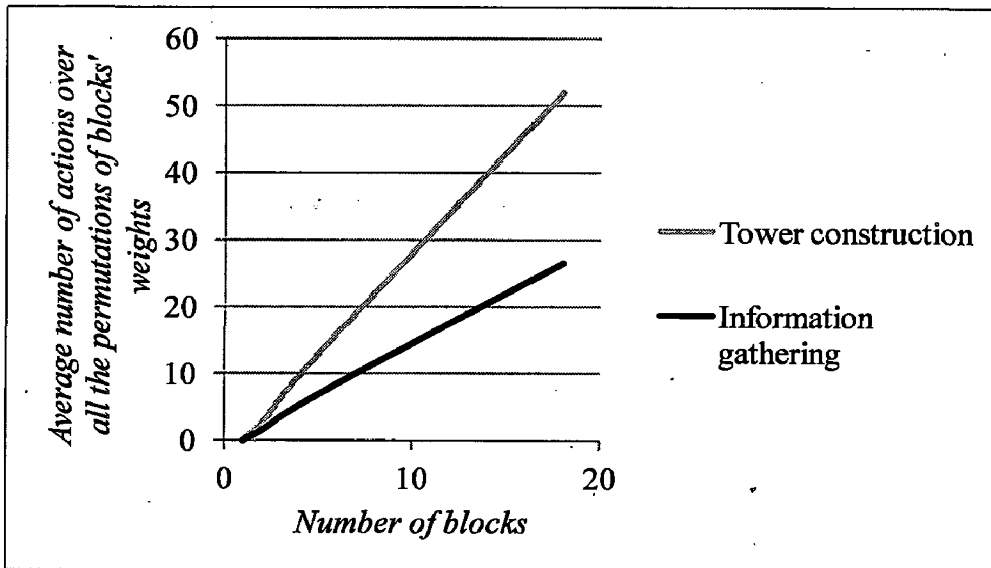


Figure 4.3: Average number of actions vs. Number of blocks

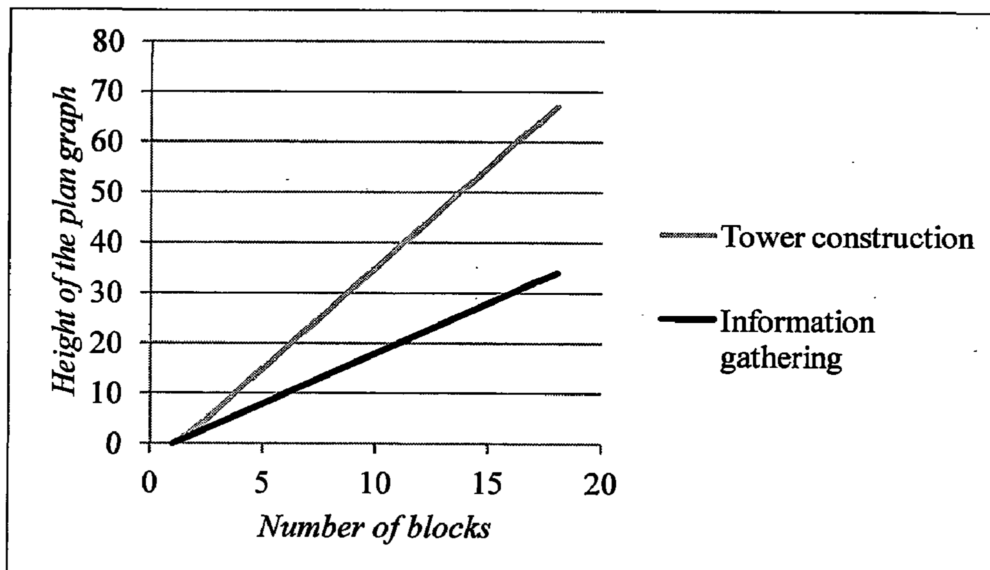


Figure 4.4: Height of the plan graph vs. Number of blocks

The graphs for times taken for generation of plans for information gathering and tower construction are shown in Figure 4.6. The times increase similarly as the number of nodes in the plan graph because for each of these states (nodes) choosing the next action takes more or less constant time.

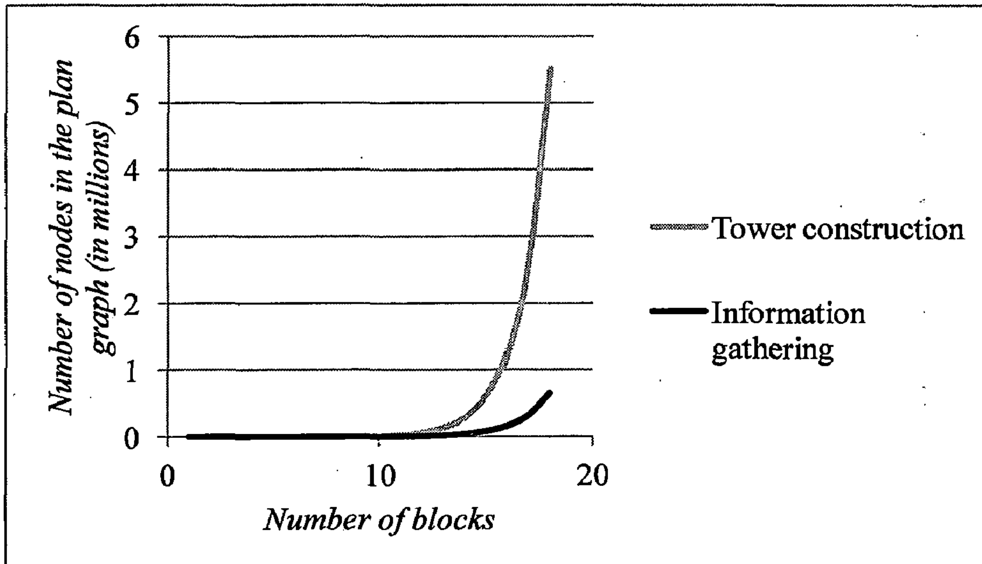


Figure 4.5: Number of nodes vs. Number of blocks

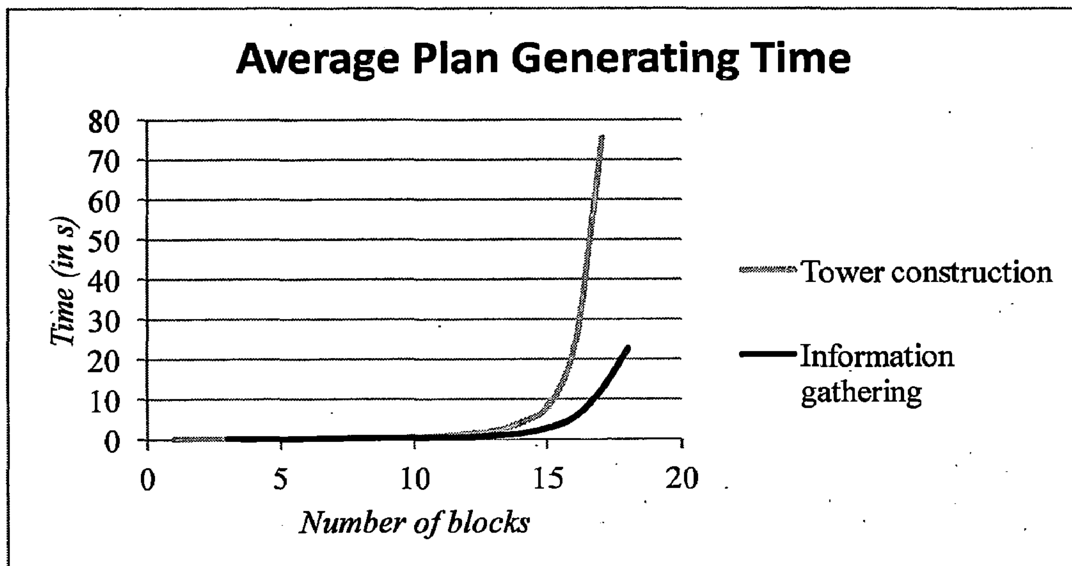


Figure 4.6: Average plan generating time vs. Number of blocks

CHAPTER 5. INTRODUCTION TO JOINT TRIALS

So far, the try actions considered have been trials by single agents. This chapter has been added as a minor part of the dissertation to introduce the concept of joint trials, try actions by multiple agents. This chapter begins with brief discussion about what planning in a multi-agent domain is. It then moves towards extension of the notion of try actions to multi-agent domains introducing joint trials, and finally goes on to introduce a couple more concepts related to trials which are valid for both single-agent trials and multi-agent trials but explaining them through joint try action examples

5.1. Multi-agent Planning

When there are multiple agents in an environment, it is a multi-agent planning problem for each agent in the environment. An agent in a multi-agent domain tries to achieve it own goals with the help or hindrance from other agents in the domain. The following discussion in this section is from [2].

The issues involved in multi-agent planning are:

i. Representation and planning for multiple simultaneous actions
ii. Cooperation, coordination and competition

5.1.1. Planning with multiple simultaneous actions

In the multi-agent setting, a single action a is replaced by a joint action $<a_1\ldots, a_n>$, where a_i is the action taken by the ith agent. (For simplicity perfect synchronization is assumed). As a planning problem, a transition model for different joint actions remains to be described. Also it is a planning problem with the branching factor of exponential order in the number of agents. With such high branching factor, the principal focus of research in the multi-agent planning is to decouple the agents to an extent, so that the complexity becomes of a linear order rather than the exponential order.

The standard approach to loosely coupled problems is to pretend that problem is completely decoupled. This now requires the action schemas to be written as if the agents act independently.

Example 5.1:

A and B are two players of a double tennis' team. Initially, A is at the left baseline and B is at the right net. The goal of the team (at some point in the game) is to return the ball that has been hit to them and is coming towards the right baseline, ensuring that at least one of them is covering the net. A joint plan that works for this goal is:

PLAN 1: A: [Go(A, RightBaseline), Hit(A, Ball)]
B: [NoOp(B), NoOp(B)]

In an action schema, the preconditions of an action restrict the states from which the action can be executed successfully, and then there are effects that are caused by the successful completion of action. In a multi-agent domain, executing an action from a state that satisfies the preconditions does not ensure success of the action. There might be a case that at the same time, there is another agent executing an action that is in conflict with its own action. In Example 5.1, if both A and B hit the ball at the same time (for both the agents, the preconditions for execution of the action are satisfied), the execution of the action fails instead of being successful. Thus, the action schemas in loosely coupled multi-agent domains need to incorporate this idea of an action executed by an agent being messed up (not successful) due to the simultaneous execution of some action by another agent in the domain. This can be solved by augmenting action schemas with a new feature: a concurrent action list that states which actions should not be executed simultaneously with the action for which the action schema is. For other actions, it might be that an action is successful only when another agent(s) executes some action simultaneously. For example, it may be that the action of moving a cooler from one position to another is successful only if another agent is also executing the action of moving the cooler to the same position.

5.1.2. Planning with multiple agents: cooperation and coordination

Consider a multi-agent domain where each agent makes it own plan. Initially, it is assumed that the goals and knowledge base are shared. In this situation each agent could execute the joint solution and execute its own part of the solution. But, the problem here is that there may exist more than one joint solution that achieve the goal, and each agent may choose different joint solutions.

Example 5.2:

Consider the same problem as in Example 5.2. Another joint plan that exists here is the PLAN 2.

PLAN 2: A: [Go(A, LeftNet), NoOp(A)]

B: [Go(B, RightBaseline), Hit(B, Ball)]

The agreement of a joint plan by both the agents would help achieve the goal. But if A chooses plan 2 and B chooses plan 1, then nobody'll return the ball. Conversely, if A chooses 1 and B chooses 2, they'll both try to hit the ball. The agents may both realize this, but how do they coordinate to make sure both agree on the same plan.

An option in such cases where the agents in the domain have to all agree on a common joint plan is to set some conventions (or protocols). A convention like "stick to your side of the court" would serve the purpose for example.

When there are no conventions, agents can use communication to decide on the common plan. In the Example 5.2, a tennis player could should "Mine" or "Yours" or the other player to indicate the common plan to the other player.

The most difficult problems are those that involve both cooperation with member of one's own team and cooperation against members of opposing teams, and that without any centralized control.

5.2. Joint trials

Trials in multi-agent domains have been discussed in [5], [14], but with trials being carried out by each agent independently and if required the agent carrying out the trial communicates whatever it learns to other agents. In this section, joint trials, trials carried out by multiple agents'll be discussed.

Joint trials are trials of joint actions. The joint actions are to be executed simultaneously by all the agents involved in the joint action simultaneously, .i.e. as discussed in the earlier section, apart from the satisfying of preconditions in the current state, a joint trial requires the simultaneous execution of the trial by the agents involved in the trial. Like trials, joint trials can be used when there is uncertainty in state information. In [5], where the notion of try actions has been introduced, it was motivated from the fact that sensing actions are not available, and that another way of gaining information about the state of the world is by using

trials. Some domains where joint trials are possible are mentioned below. These domains are all adaptations of domains that are already present in literature.

1. The blocks world domain. Modified form of the domain is already discussed in section 2.4.1 where try actions were included into the domain. Now suppose it is known that a block cannot be lifted by a single agent, but it might be possible if two agents try to lift the block together. If the lifting of the block by two agents together is a joint action, then both of them trying together to lift the block is a joint trial. Once the block gets lifted, the two agents can perform several joint actions like moving the block to another position, or placing the block back on the table. The failure to lift the block, would suggest that the block is much heavier than the capacity of the two agents together to lift the block, and that some other technique is to be used to lift the block.
2. Consider the example domain from [15] where there is a heavy door and it opens only when there are two agents pushing together or pulling together the door. Now, that it is known that the door opens only when both the agents push the door together or pull it together. Let Joint_PushDoor and Joint_PullDoor be the respective actions. An additional condition may be added that the door opens only if it is not locked, and whether the door is locked or not is not known. The joint trials Try_Joint_PushDoor and Try_Joint_PullDoor can now be used as trials for pushing or pulling, respectively, the door open. When the door does not open, it is known (or learnt) that the door is locked and that the door should be unlocked first so as to be opened.
3. The grid world domain is a popular multi-agent planning domain where there are agents and obstacles in some of the blocks of the grid world. Each grid can have either an agent or an obstacle or may be free at any instant of time. The obstacles in this domain, as far as used in the literature, have fixed positions. Now an adaptation of the grid world domain is considered where some obstacles can be moved by two agents who are in positions adjacent to that of the obstacles. Therefore, if moving an obstacle is a joint action, then trial of this action is a joint trial. The success or failure of the trial gives whether the obstacle is loose or fixed respectively at its position.

The notation $a^{(i, j)}$'ll be used to denote a collective joint action a by agents i and j. To represent the trial of this joint action, $Try_a^{(i,j)}$'ll be used.

Now, consider the modified blocks world domain as discussed in section 2.4.1 but with all the blocks as such that at least two agents are required to be able to lift or move the blocks.

Let A, B, and C be 3 blocks in the domain. The fluents and operators (along with preconditions and effects) can now be modified from that in section 3.1 to be written as.

Fluents (*x*, *y* are distinct blocks) –

1. Relational:
   - $ArmEmpty^{\{i\}}$: true if agent i's arm is empty, false otherwise.
   - $ArmEmpty^{\{j\}}$: true if agent j's arm is empty, false otherwise.
   - OnTable(x): True if x is on table, false otherwise.
   - On(x, y): True if x is on y, false otherwise.
   - Clear(x): True if there's no block on x, false otherwise.
   - $Holding^{\{i, j\}}(x)$: True if the agent i and j are having x in their arms, false otherwise.
2. Knowledge:
   - $K^{\{i, j\}}(Light(x))$: True if agents i and j know that x is light with respect to both of them, false otherwise.
   - $K^{\{i, j\}}(Heavy(x))$: True if agents i and j know that x is heavy with respect to both of them, false otherwise.

Operators (x, y are distinct blocks) –

- $PickUp^{\{i, j\}}(x)$
  - Preconditions: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge OnTable(x) \wedge Clear(x) \wedge K^{\{i, j\}}(Light(x))$
  - Effects:
    - Delete: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge OnTable(x)$
    - Add: $Holding^{\{i, j\}}(x)$
- $PutDown^{\{i, j\}}(x)$
  - Preconditions: $Holding^{\{i, j\}}(x) \wedge K^{\{i, j\}}(Light(x))$
  - Effects:
    - Delete: $Holding^{\{i, j\}}(x)$
    - Add: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge OnTable(x)$
- $Stack^{\{i, j\}}(x, y)$
  - Preconditions: $Clear(y) \wedge Holding^{\{i, j\}}(x) \wedge K^{\{i, j\}}(Light(x))$
  - Effects:
    - Delete: $Clear(y) \wedge Holding^{\{i, j\}}(x)$
    - Add: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge On(x, y)$

- $UnStack^{\{i,j\}}(x, y)$
    - Preconditions: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge On(x, y) \wedge Clear(x) \wedge K^{\{i,j\}}(Light(x))$
    - Effects:
        - Delete: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge On(x, y)$
        - Add: $Clear(y) \wedge Holding^{\{i,j\}}(x)$
- $Try_PickUp^{\{i,j\}}(x)$
    - Preconditions: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge OnTable(x) \wedge Clear(x) \wedge$ not $K^{\{i,j\}}(Light(x)) \wedge$ not $K^{\{i,j\}}(Heavy(x))$
    - Effects: i $\vee$ ii
        i. (Success of trial)
            - Delete: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge OnTable(x)$
            - Add: $Holding^{\{i,j\}}(x) \wedge K^{\{i,j\}}(Light(x))$

        ii. (Failure of trial)
            - Delete: None
            - Add: $K^{\{i,j\}}(Heavy(x))$
- $ApplyLever^{\{i,j\}}(x, y)$
    - Preconditions: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge Clear(x) \wedge Clear(y) \wedge K^{\{i,j\}}(Heavy(x)) \wedge K^{\{i,j\}}(Heavy(y))$
    - Effects:
        - Delete: $Clear(y)$
        - Add: $On(x, y)$
- $RevLever^{\{i,j\}}(x, y)$
    - Preconditions: $ArmEmpty^{\{i\}} \wedge ArmEmpty^{\{j\}} \wedge On(x, y) \wedge Clear(x) \wedge K^{\{i,j\}}(Heavy(x)) \wedge K^{\{i,j\}}(Heavy(y))$
    - Effects:
        - Delete: $On(x, y)$
        - Add: $OnTable(x) \wedge Clear(y)$

5.3. Other concepts of trials

- Learning about capabilities of agents: Earlier, it was seen that a block is heavy or light can be decided by an agent or the agents performing trial on the block. Note that a block being heavy or light is relative to the agent and is not fixed for all agents. If there are a

number of agents, the capacities of the agents may be compared by making them perform trials of some actions. For example, suppose there are two agents whose capacities are known to be equal and they can lift a block together. Now if one of these agents is replaced by another agent whose capacity is not known and the trial by them to lift the same block fails, it is known that the latter agent has lifting capacity lower than the replaced agent.

- Repetition of trials: So far, it has been assumed that a trial if one succeeds resp. fails means the action which was tried would always succeed resp. fail. Now, suppose that an action can succeed after being tried a number of times. The plans in case of repeated trials are iterative. For example, consider again the grid world domain but with obstacles that are not totally fixed or totally lose. The obstacles are such that the trial of moving them can succeed after a number of trials by both the agents i and j in the domain. Let $Move^{\{i, j\}}(o, x_2, y_2)$ be the operator that moves an obstacle o to position (x_2, y_2) in the grid when the obstacle is loose in its position and the agents are at required positions. Now, suppose the obstacle O is to be moved from location (c, d) jointly by the agents. The plan, when the agents are at the required positions can then be written as:

  While (not At(o, c, d))

  <Try_Move$^{\{i,j\}}$(o, c, d)>

CHAPTER 6. CONCLUSION AND FUTURE WORK

The major part of the dissertation has been to develop planner(s) for the modified blocks world domain. The algorithms for planners for information gathering and tower construction from the default initial state of the order of 2^N and $N*2^N$ respectively (N is the number of blocks in the domain) have been given in the dissertation. The complexity orders are same as that of the possible states that the robot can go through when trying to achieve the respective goals in the optimal way. The plans generated by the planners are intuitively the most optimal, although not in their size, but, in the average number of actions, over all the possible permutations of weights of blocks, that'll be executed on execution of the plan.

The planner for tower construction from any initial state has also been developed. Although the algorithm has not been given, various cases that are to be taken care of while writing the planner have been explained with example plans generated by the planner developed. As a part of future work, the planner can be generalized to generate plans for any goal configurations, after developing the specification for representing goal states in the domain.

As another (minor) part of the dissertation, the concepts of joint trials – trial actions by multiple agents –, repeated trials, and learning about capabilities of agents through trials have been introduced. As a part of future work, these concepts are to be refined and the concept of joint trials can also be extended to cases when the knowledge of the agents are different and when there are agents with different abilities.

REFERENCES

[1] E. Rich and K. Knight, *Artificial Intelligence*, 2nd ed. New York: McGraw-Hill, Inc., 1991.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. New Jersey: Pearson Educ., Inc., 2010.

[3] S. Koenig and Y. Liu, "Representations of Decision-Theoretic Planning Tasks," in *Proc. 5th Int. Conf. Artificial Intelligence Planning Systems*, Breckenridge, Colorado, 2000, pp. 187-195.

[4] L. Iocchi, T. Lukasiewicz, D. Nardi, and R. Rosati, "Reasoning about Actions with Sensing under Qualitative and Probabilistic Uncertainty," *ACM Trans. Computational Logic*, vol. 10, no. 1, pp. 5:1-5:41, Jan. 2009.

[5] R. Niyogi, "Planning with Trial and Errors," in *Proc. Int. Conf. Intelligent Agent & Multi-Agent Systems 2009*, Chennai, 2009.

[6] F. v. Martial, *Coordinating Plans of Autonomous Agents*. Germany: Springer-Verlag, 1992.

[7] J. S. Rosenchein and G. Zlotkin, "Designing Conventions for Automated Negotiation," *AI Mag.*, vol. 15, no. 3, pp. 29-46, 1994.

[8] K. Matsubayashi and M. Tokoro, "A Collaboration Mechanism on Positive Interactions in Multi-agent Environments," in *Proc. 13th Int. Joint Conf. Artifical Intelligence*, vol. 1, Chambery, 1993, pp. 346-351.

[9] W. S. Briggs and D. J. Cook, "Modularity and Communication in Multiagent Planning," Dept. Comput. Sci. and Eng., UTA, Arlington, 1996.

[10] H. J. Levesque, "What is planning in the presence of sensing?," in *Proc. 13th Nat. Conf. Artificial Intelligence*, vol. 2, Portland, 1996, pp. 1139-1146.

[11] T.C. Son and C. Baral, "Formalizing sensing actions— A transition function based approach," *Artificial Intell.*, vol. 125, no. 1-2, pp. 19-91, Jan. 2001.

[12] H. J. Levesque, "Planning with loops," in *Proc. 19th Int. Joint Conf. Artificial*

Intelligence, Edinburgh, 2005, pp. 509-515.

[13] L. Spalazzi and P. Traverso, "A Dynamic Logic for Acting, Sensing, and Planning," *J. Logic and Computation*, vol. 10, no. 6, pp. 787-821, Dec. 2000.

[14] R. Niyogi and R. Ramanujam, "An epistemic logic for planning with trials," in *Logic, Rationality, and Interaction: Second International Workshop, LORI 2009, Chongqing, China, October 8-11, 2009, Proceedings*, X. He, J. F. Horty, and E. Pacuit, Eds. Germany: Springer-Verlag, 2009, pp. 238-250.

[15] A. Dovier, A. Formisano, and E. Pontelli, "An investigation of Multi-Agent Planning in CLP," *Fundamenta Informaticae*, vol. 105, no. 1-2, pp. 79-103, Jan. 2010.

PUBLICATIONS

[1] Swetha Jain Kothari and Rajdeep Niyogi, "Generation of conditional plans for a modified blocks world domain," in *Proc. Int. Conf. Recent Advances in Engineering & Technology (ICRAET 2012)*, Hyderabad, Apr. 29-30, 2012, pp. 28-31.

[2] Swetha Jain Kothari and Rajdeep Niyogi, "Generation of conditional plans for a modified blocks world domain," Special Issue *Int. J. Syst., Algorithms & Applicat. (IJSAA)*, vol. 2, no. ICRAET12, pp. 156-159, May 2012. ISSN Online: 2277-2677.