CHALMERS

Automated Planners and Planning analysis with Durative Actions

Master of Science Thesis in the Programme Intelligent System Design

GHAZAL FAKHTEHYAVARI
YASAMAN VAZIRI

Department of Applied Information Technology
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, December 2012
Report No. 2012:095
ISSN: 1651-4769

Automated Planners and Planning with Durative Actions

GHAZAL. FAKHTEHYAVARI,
YASAMAN.VAZIRI,

Examiner: CLAES. STRANNEGÅRD

Chalmers University of Technology
University of Gothenburg
Department of Applied Information Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Applied Information Technology
Göteborg, Sweden December 2012

ABSTRACT

Today's need for efficient and effective automated planning has brought about a distinct set of planning approaches focused on different quantifiable results, namely time and resource consumption. The planning problems are diverse and touch many aspects of day-to-day life. Whether it is for finding the shortest route of travel in presence of traffic to selection of investment strategies, planning routines are in place to meet the needs of everyone and every machine. Most existing planning domains contain actions, which have common parameters such as duration, timing, resource consumption, and in some cases resource generation. This master thesis seeks to find the most optimal planner to solve Logistics problems. Logistics problems utilize time, space, resources and actions thus they represent an ideal case to assess planners and underlying algorithms in how they deal with complexity of issues, how much time they utilize in generating a plan and how many steps they use in reaching the goal. The combination of these factors points to the optimal planner suitable for logistics planning. Planners are reviewed, analysis is undertaken and the most optimal solution is identified through practical experiments.

ACKNOWLEDGEMENT

We would like to express our gratitude and sincere respect to all the people who supported us and gave us encouragement in overcoming difficulties throughout this journey. A very special thanks goes to our best friend Mr. Masoud Motamedi whose moral support and help encouraged us to complete this challenging journey. We sincerely appreciate the efforts of our examiners Dr. Urban Nulden and Dr. Claes Strannegård of Chalmers University of Technology for their mentorship during this effort. We are also deeply thankful to Dr. Son Cao Tran from New Mexico State University for his participation as an advisor and his helpful suggestions and comments, which undoubtedly encouraged us to improve this research. Our very special thanks go to Mr. Börje Johansson for all his helps and assistance through this work. And last but not least, we would like to pay tribute from bottom of our hearts to our dear parents for their staunch love, valuable guidance and support in numerous ways throughout our life. Hereby, we are kindly dedicating this to our parents, to whom we owe much of what we are now, to express our sincere appreciations.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Today's need for efficient and effective automated planning has brought about a distinct set of planning approaches focused on different quantifiable results, namely time and resource consumption. The planning problems are diverse and touch many aspects of day to day life. Planning problems are almost everywhere and their applications exist in different industries such as transportation, airlines, military, networking, etc.

It is obvious that each industry and each application has its own set of requirements. As such finding a general purpose planner that effectively and efficiently copes with durative actions is a difficult task. Although the history of this concept dates back to about 1998 and numerous research has been conducted in this area, it seems still there is a lot of interest in the growing field of temporal planner. There are a lot of gaps to overcome regarding introducing a new planner.

Research continues in planning and development of different planners, yet there are uncertainties that surface when it comes to the area of dealing with realistic planning domains. It is because such domains consist of several durative actions and all the actions either produce or consume resources. In fact, when the time and duration are the most critical factors to deal with, durative actions and their concurrency become more complicated components of the process. To clarify those actions, imagine that a passenger is interested in traveling from El Paso to Phoenix. From a planning point of view the passenger may consider several alternatives and seek to find the true cost of each. The problem in hand can be as simple as asking whether flying or driving would be the fastest option given the start time of the trip. Or, the problem can be very complicated if the passenger want to consider time to destination, arrival time, cost of travel, and possibly the impact to environment due to fuel consumption. This simple example highlights the fact that providing a simple planning system is a straightforward attempt however, when it comes to develop a planning system that can cooperate with actions which has duration and resources, it turns to be a complicated decision to make.

The area of Logistics represents a good challenge for planning problems as it includes the implication of planning on utilization of time, space, and resources. In this thesis the available planners that can potentially deal with Logistics problems are considered, their benefits and shortcomings are reviewed, and then suitable planners are applied to a set of logistics problems with the goal of finding an efficient and effective planner. Specifically the following question was the prime focus of this research: Of the numerous planners and search algorithms available today, which one is capable of effectively and efficiently solving logistics problems is a timely manner? This thesis answers this question.

1.1 STRUCTURE

Chapter 2 describes the methodology used in this thesis. In Chapter 3 a brief description of problem solving is presented and certain definitions are outlined. This chapter covers planning techniques and search strategies applicable to the problem in hand. Chapter 4 includes the results of experiments conducted using various planners operating on three real world logistics problems leading to the identification of the most optimal planner. Chapter 5 the thesis conclusions are presented and certain future work is identified.

2 METHODOLOGY

This thesis explores various planning and search methodologies to find an effective and efficient methodology capable of solving Logistics problems. This chapter outlines the methodology used in selecting a set of promising planners from many covered in literature, identifying methods to measure the performance of planners, applying the selected planners to a representative set of Logistics problems, and finally identifying, through analysis of experimental results the most effective and efficient planner for solving Logistics problems.

2.1 LITRETURE REVIEW

The study of available planners is to focus on all available planners and their underlying algorithms specifically considering them in terms of durative actions which is a key factor is solving Logistics problems. The various search algorithms available for use by the planners are to be reviewed in detail as their performance impacts the speed of searches that a planner takes and thus its performance. Finally given the importance of a clear definition of a problem and its domain, available computer languages used by planners are to be considered. All of the work is to concentrate on finding suitable planners capable of dealing with challenging Logistics problems. While the most critical issues are timing and actions with duration, the majority of the effort is to be concentrated on review of possible problems and difficulties regarding timing.

2.2 MEASUREMENT METHODS

To assure that an optimal planner is selected, four quantifiable factors were identified:

Search time	Time taken by a planner to generate a plan
Search length	Number of steps/notes identified in the solution
Expanded states	Number of states with potential solutions expanded upon
Generated states	Number of states considered

Generated States and Expanded States show the internal complexity of a planner thus its effectiveness whereas Search Time and Search Length represent the efficiency of the planner a planners goals. For the experiment and data analysis the follow terms are defined:

$$\text{Efficiency (p)} = \frac{Search\ Time(p)}{Expanded\ States\ (p)} \text{ in milliseconds/State}$$

$$\text{Effectiveness (p)} = \frac{Search\ Time(p)}{Search\ Length\ (p)} \text{ in milliseconds/Node}$$

Furthermore based on the size of the problems in hand, it is envisioned that certain planners could not complete the tasks in hand due to time and memory limitations. These behaviors formed the basis for some of the observations made in the thesis.

These performance factors are to be used first in a qualitative study and identification of planners and search algorithms with promising capabilities specifically focused on Logistics.

The measurement strategy is to employ real world logistics problems. Multiple problems are to be considered representing a range of complexity that can provide a good exercise for the planners under test. Each problem is to be solved by selected planners to form the basis for comparison and final selection of a planner.

2.3 EXPERIMENTATION

Based on the review of the available planners, a handful are to be selected and utilized on three distinct Logistics/Depot problems. Experiments are to quantify and qualify the impact of the size and complexity of the problem on the behavior of each planner. The planners are to be able to supply information that can be used in calculating Efficiency and Effectiveness thus allowing a fair comparison of the planners.

2.4 ANALYSIS OF DATA

The data is to be collected and presented in a common format by the authors. The results are to be analyzed in a qualitative manner and the all anomalies of the analysis are to be discussed. The data from each planner is to be presented individually and then compared as appropriate. In all cases relevant information such as the duration of the planning process, the resulting search length, the number of states considered, efficiency, and effectiveness are to be shown and form the basis for the comparison and the conclusion.

3 DEFINITION OF PLANNING

In general, planning is the process of controlling a group of tasks to complete what is described in the domain of the problem to achieve certain goals. Depending on how complex the domain is, the number of states required to complete the tasks increase. In Artificial Intelligence when faced with a problem, acting rationally is important to make an efficient plan to overcome that problem. Planning starts from certain point, which is called the initial state, and ends up at the goal state, which is the destination. In between there are steps to be taken, which are called actions. Basically, planning can be denoted as P and it can be formulated as P = (O, I, G) where the O denotes the set of actions, I stands for initial state, and G indicates the goal state. In general, planning is the process of choosing certain actions by considering the action's effect, employing strategies, prioritizing tasks, and combining them to make a reasonable plan capable to reach the goal.

3.1 TEMPORAL AND MERTIC PLANNING

While planning is the prognosticated way of finding a set of actions and being concerned about their effects, Temporal Planning is more concerned about the timing of reaching a solution rather than the sequence of actions. Timing issues may happen because of several reasons such as: deadlines, shortage of resources, or as an effect of some actions or events, which are not controllable in the scope of the problem.

Metric (numeric) planning on the other hand is not only concerned about the resource availability but also with the number of resources.

The real world problems, which deal with temporal and metric planning, are really hard problems to find a solution to as they do have to handle constraints of time as well as limited resources[1].

3.2 PLANNING PROBLEMS

Planning problems can be simple or complicated but all problems have at least one solution. In case of multiple solutions, they are not all the optimal answer to the problem but at least one is the reasonable answer to rely on. Indeed, the solution in general is the set of actions that are chosen to be performed in the different states to lead us to the goal state. To solve planning problems there are several alternatives available that vary in complexity. For instance, one possible way for the planner is to grab a path at random and then figure out if this is the one which can fulfill the desired outcome. If the answer is no, then it will try another one that seems most promising with the knowledge of its prior attempts. As a result, the next choice might be at least a reasonable solution or at most the optimal answer to the problem. The other possible alternative may be a brute force search where all possibilities are examined leading to the selection of the optimal solution. This approach seems to be easy but in reality it takes a long time to go through all the possible solutions. A third possibility is

1 http://planning.cis.strath.ac.uk/TEMPPLAN/

to define certain rules in order to find the solutions. For example, as a first rule all the resources should take part in a list; therefore, who comes first serves first. A second rule may be that encountering an anomaly allows the planning system to switch to the next rule which could be something very different such as who came in last serves first. The last but not least possibility is the use of a heuristic function that selects from several possible solutions the optimal answer to the problem.

One of the most famous planning problems is Blocks World Problem. The domain describes the actions an agent must take for approaching the goal. The domain of Blocks World problem, shown in Figure 3.1, is denoted by the table "t" which holds four removable blocks a, b, c, d. The rule is that one block can moves at a time, and they can either be placed on the top of each other or on the table. Each block can only move if there is no other block on top of it. The goal is to put the blocks on top of one another in an alphabetic order on the table. In order to reach the goal there are a number of actions to be considered. The initial state in blocks world problem can be formulated as a set of:

$$S_i = \{\text{On (a, b), On (b, c), On (c, t), On (d, t), Clear (a), Clear (d)}\}$$

The goal state is also formulated as a set of:

$$S_g = \{\text{On (a, b), On (b, c), On (c, d), On (d, t)}\}$$

Relevant actions are movements (a, b, d) and (b, c, a) which are formulated as follow:

{On (a, b), On (b, c), On (c, t), On (d, t), Clear (a), Clear (d)}
{On (b, c), On (c, t), On (a, d), On (d, t), Clear (a), Clear (b)}
{On (c, t), On (b, a), On (a, d), On (d, t), Clear (c), Clear (b)}

There exist some constraints and conditions named as pre and post conditions per each action to perform. Preconditions must be true before action runs in one step before its performance. The following part is the formulated form of pre and post conditions relevant to the actions:

Pre (MOVE (a, b, d)) = {Clear (a), On (a, b), Clear (d)}
Post (MOVE (a, b, d)) = {On (a, d), Clear (a) Clear (b)}
Pre (MOVE (b, c, a)) = {Clear (b), On (b, c), Clear (a)}
Post (MOVE ((b, c, a)) = {On (b, a), Clear (b) Clear (c)}

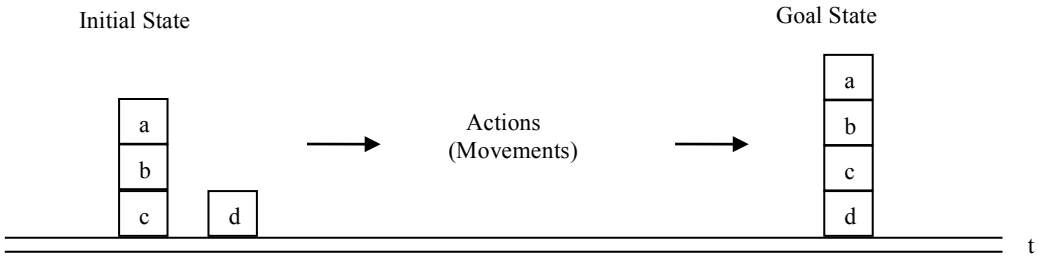


Figure 3.1 Blocks World Problem

The state space of the Blocks World Problem contains a set of all possible arrangement of blocks starting from the starting state to the goal state.

3.2.1 PLANNING PROBLEM IN TERMS OF DURATIVE ACTIONS

Real world problems have more constraints to overcome compared with simple planning problems. While handling planning problems is a complex task, dealing with temporal planners and durative actions make planning far more complicated. The introduction of every action's duration brings on an additional dimension of difficulty that the planner has to be concerned about. In addition to ordering of task for action and keeping track of the performance at every step, the planner is to keep track of the timing of each task to avoid conflicts with or interference to other ongoing actions. Resource and time constraints are the influencers of the performance of planning problems with durative actions.

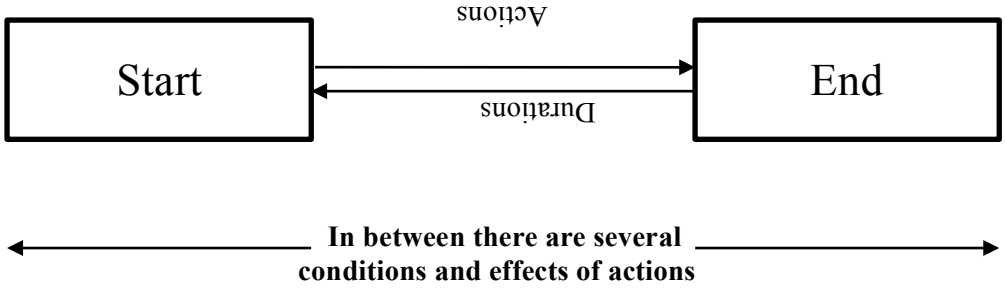


Figure 3.2 Durative actions schema

3.3 SEARCH STRATEGIES

An agent faced with a problem, needs to search for alternatives to find a solution. Possibly, the way to handle a problem is by searching an appropriate solution set to overcome any particular obstacle. Searching for a solution can take a form of a search tree where the initial state is the node from which the tree is formed and different states are placed as connected nodes underneath the initial state. Typically, states might be reachable through different path, but there is only one appropriate solution that meets the desired characteristics. It is search strategy's responsibility to select a node among group of nodes in a tree and expand upon it in order to find the best path to reach the goal state. The best search strategy is one that is aware of duplication in choosing the paths to follow therefore it can ignore certain nodes that could potentially get the search stuck in a never ending loop. The path cost is the cost of path from first parent node, which is the initiator to any node over the search tree. Also each search tree has depth which it recognized by the number of the levels from initiator to the end. The other fact which must be taken into consideration is selection of search algorithm's efficiency which has direct relation to time and space complexity. It is obvious that more complex problems need more memory space to keep track of all the nodes, which are generated in the tree. [13]

3.3.1 INFORMED SERACH

Informed search or heuristic search refers to a search that uses available information from

the problem definition so the domain turns to known area to search in order to find out the reasonable solution. It employs a heuristic function to help it in selecting a path among all possible paths considering the cost of such selection. In general, for each node there is a heuristic evaluation function, which qualifies the node for expansion. In fact, heuristic evaluation function has responsibility to calculate the distance from any node to the goal; therefore, surely if that node is carrying the minimum cost to reach the goal, then it is the one which goes for expansion.

3.3.1.1 BEST FIRST SEARCH

A second function, known as evaluation function, is responsible for allocating a value to each node. Therefore, Best First Search (BFS) employs the heuristic function to distinguish which node among the others is the best candidate to explore according to its approximate cost to the goal. Mainly in this algorithm two lists are involved. One is open list, which consists of all possible nodes for expansion, and the other is called the closed list, which is responsible to keep track of already explored nodes. While on one hand all the nodes are connected to each other, and produce different paths to the goal, it is possible that a node is closed yet its related nodes are waiting in the open list to explore. The remarkable feature of this algorithm is when it comes to the dead luck situation it does not give up; it simply makes attempts for exploring another node in the search graph.

3.3.1.2 GREEDY BEST FIRST SEARCH

Greedy is one of the special cases of best first search. In Greedy the heuristic function picks the nearest node to the goal instead of initiating the search from the starting point. In some cases this strategy works yet in other cases it turns problematic. As the algorithm‘s name implies it will pick that node greedily meaning that when it picks a node it does not consider the implications. This algorithm is neither complete nor optimal because it may fall into a loop and be locked there without finding a solution. It may get luck but generally it has time and memory space complexity.

3.3.1.3 A* SEARCH

A* Search is another special case for the best first search which looks for a complete and optimal solution to reach the goal. A* gets help from the nodes that are listed in the open and closed lists. Along the way from start node to the goal node there are a bunch of connected nodes to proceed through to reach the goal node. A* basically has two main functions to guaranty that it can find a reasonable solution in return, one function is working on the path cost from start node to any node that is in the path to achieve the goal, and the second function is the heuristic function dealing with the approximate cost from that particular node to the goal node. These two functions together represent the total cost from the node to the goal and formulize the most reasonable path to the goal state, which has least cost possible. The algorithm is optimal because it tries to find the lowest cost possible, and it is complete because it gives a solution to the search problem. A* has memory space complexity as it keeps track of nodes in either lists in its memory. It also has exponential time complexity.

3.3.2 UNINFORMED SEARCH

Unlike informed search, the uninformed search, which is also known as blind search, is one that has no information about the problem itself other than knowing the initial state. The search domain is fully unknown therefore it is imperative for the algorithm to explore different possibilities through the search domain. In this section some of well-known uninformed search algorithms are briefly introduced.

3.3.2.1 BREADTH FIRST SEARCH (BFS)

The Breadth First Search explores all the tree nodes one at a time. The execution starts from the initial node of the tree and evaluates all the nodes in every level until it reaches the last node in the level. It then moves forward to lower levels until it catches the very last node. It follows the First In, First Out (FIFO) rule, which means that any new added child goes to the end of the fringe. It is easy to observe that this algorithm is complete because it explores all the node one by one in each level from top to the bottom, and if there is a goal node in any "shallowest" level it can address it just after the search algorithm explore other "shallower" nodes. Although it is a good search algorithm to choose its time complexity and space complexity leave it to be inefficient. Since it goes through each and every node in search tree, it will take longer to complete the search process, especially if the goal is located at the very bottom level causing it to be discovered at the very end after the algorithm has examined all the higher levels. The algorithm keeps all the information in the memory spaces thus creating space complexity as well.

3.3.2.2 DEPTH FIRST SEARCH (DFS)

Whereas the Breadth First Search every node of one level before moving down to the next level, Depth First Search it starts from the initiator node and visits nodes from top to the end of each branch before exploring other branches. The type of fringe in this search algorithm is Last In First Out (LIFO) stack which indicates that the last node added to the fringe will be explored first. This search doesn't require as much memory space as it limits its storage to the path from initiator node to the leaves as well as keeping track of any unexpanded nodes. Depth first search algorithm is neither complete nor optimal as if it can't find a solution it never stops the search. If it makes a wrong decision it will continue in the wrong direction which leads the search process to get stuck in a loop or it might generate several possibilities as potential solutions.

3.3.2.3 BACKTRACKING SEARCH

Backtracking Search uses depth first search as a main tool to find a proper solution throughout paths and back offs whenever it makes a wrong selection in order to get a chance to explore another path, which may lead to the goal. This search does not seem sufficient to solve complex problems.

3.3.2.4 DEPTH LIMITED SEARCH (DLS)

Depth Limited Search is a special case of depth first search algorithm with the additional capability to overcome scenarios where it gets stuck in a loop. It limits the depth of the search tree by considering the information provided by problem description. As a result it only considers those states that are getting the chance of expansion. After exploring all the nodes, which belong to the certain depth of the tree search, the limitation will be updated by increasing the depth value. The algorithm is not complete if the limit is less than depth because it cannot manage to reach the goal. Nor is it optimal as it cannot guaranty that it can point out the goal even if the limit has a value bigger than the depth itself.

3.3.2.5 ITERATIVE DEEPENING DEPTH FIRST SEARCH (IDS)

This search puts all the best features of both breadth first search and depth first search algorithms together. It applies the depth first search to each level by assigning a limit to the depth of tree. Each time that it looks for the goal and fails, the depth first search will execute again for the next depth limit. This search is capable of finding the shortest answer to the problem, but compared to breadth first search algorithm it is much faster in finding the goal. The time complexity is a bit higher compare to the breadth first search and depth first search, but the space complexity is the same as depth first search.

3.3.2.6 BIDIRECTIONAL SEARCH

Indeed, the execution of this kind of search starts in two different directions at the exact same time. One process starts from initial state and goes forward toward the goal state, while another process starts from the goal state in search of the initial state effectively going backward. This search process is complete either when both of the processes approach each other in the middle or when they both point to a node that they both explored in their search space. As a result, when they both come to that point they can merge their path to bring along the final solution. The search has less time complexity but sometimes it requires having large memory space.

3.3.3 LOCAL SEARCH ALGORITHMS

In the search algorithms described so far, the solution for the problem is the path established from the start node to the goal node. Reaching the goal is the main concern of the algorithm. Local Search Algorithms only keep the recent state in their memory along with a value assigned to each state by a heuristic function. Each state is concerned with the assigned values of its neighbors. In the subsequent search, the state that is chosen as the best continues the process all over again until it satisfies the constraints. Like any other algorithms the local search algorithm is complete if it can find a solution for the problem. Local search algorithm can be optimal if it can find a local optimum where the state can't be any better than what it is according to its neighbors' calculations. Memory space complexity of local search algorithms is reasonable as they occupy less memory while finding a rational solution. In contrast the other classes of algorithm may not be able to deal with the type of problems local search algorithms handle. Local search algorithms are often better choices to handle real world problems.

3.3.3.1 HILL-CLIMBING

Hill-climbing, also known as greedy local search is one of the more famous algorithms under the class of local search algorithms. It promises to find local optimum solution, which is extracted from neighboring calculation, but it can't promise to bring along the global optimum, which is the best solution among all the possibilities. It performs as if it is in a loop which may start anywhere. It picks a good neighbor and tries to find a better solution by applying local search over surrounded neighbors. This procedure continually runs until the search improvement is achieved gradually by finding a reasonable solution. Although it improves its states by choosing better neighbors at any time and it is fast in its search, it sometimes stops when it hits local maxima. It happens when the local maximum, which is the biggest assigned number among the neighbors, is less than the global maximum. Therefore, hill- climbing can claim neither as being complete nor as being optimal. It doesn't have memory space complexity because it keeps track of few nodes at a time and discards the others not in its surrounding.

3.3.4 SUMMARY OF SEARCH ALOGORITHMS

The potential for using a specific search algorithm as the candidate to apply to all sorts of problems was studied in details. As depicted in Figure 3.3, the reviewed candidates have different characteristics that may make them suitable for diverse set of problems. There are advantages and disadvantages to each search algorithm thus leading us to the conclusion that one algorithm cannot satisfy all the search problems that one may encounter. Actually certain algorithms may even be completely wrong for some problems leading to wasted time as a solution may not be reached at all.

Search Algorithm	Complete	Time	Space	Optimal
Best First	No	$o(b^d)$	$o(b^d)$	Yes
Greedy Best First	No	$o(b^d)$	$o(b^d)$	No
A*	Yes	$o(b^{d+1})$	$o(b^d)$	Yes
Breadth First	Yes	$o(b^{d+1})$	$o(b^{d+1})$	Yes
Depth First	No	$o(b^m)$	$o(bm)$	No
Depth Limited	No	$o(b^l)$	$o(bl)$	No
Iterative Deepening Depth First	Yes	$o(b^d)$	$o(bd)$	Yes
Bidirectional	Yes	$o(b^{d/2})$	$o(b^{d/2})$	Yes
Hill Climbing	No	$o(b^d)$	$o(1) - o(b^d)$	No

Figure 3.3 Comparision of search algorithms.

3.4 PLANNING ALGORITHMS

There are two main areas in planning algorithm, classical planning, and non-classical planning. Classical planning are planning systems that have complete information about their initial state. “There are several restricted requirements for planning system to be as classical planning. The planning known as Classical planning if its environment would be finite, fully observable, static, and deterministic in terms of time, actions, effects, and objects.”[2] Non-classical planning, also known as conformant planning does not have enough information about problem’s initial state and they are based on indecisive action’s behavior and their state model is indefinite. As the description of these planning systems implies, the process of choosing sets of actions is based on their observations.

3.4.1 FORWARD SERACH ALGORITHM

The principle of Forward Search Algorithm is that it gets the problem description as an input to planning system. Thereafter, the planner processes the problem and if the problem seems to be solvable, then it returns the solution otherwise it fails when it approaches the branches it is supposed to delete. The search space is huge for forward search, but since it is not necessarily useful then there are different ways to control the branches by applying some changes in planning algorithm. Although the algorithm has space complexity it is complete. It is an optimal algorithm unless the optimal solution is part of deleted branches. The algorithm must be aware of useless branches, which are enlarging the search space of the problem very fast.

3.4.2 BACKWARD SEARCH ALGORITHM

Backward search as its name implies is a planning algorithm which tries to find the reasonable path to solve the problem by starting the search from the goal to find out sub goals. Those sub goals are responsible to assure that the path found by them is the most promising path as goal state goes backward to the initial state. It is complete because it is guaranteed that a solution for the planning problem is reached.

3.4.3 PARTIAL ORDER PLANNING ALGORITHM

The partial order planning algorithm employs the technique “divide and conquer” to deal with sub problems separately; extracting sub goals, and generating sub plans and taking proper actions. In fact, the technique tries to make things easier, and applies more flexibility to the problem solving while it cuts problem into several sub problems, and from there each individual part has its own sub goals and sub plans to investigate. Generally, if two different actions are participating in a planning effort, regardless of their order they are combined at the end to form the resulting that is known as partial order planning. Indeed, every involved step in the plan can be interpreted as an action as none of the steps carry any ordering number for their performance.

[2] http://www.cs.umd.edu/~nau/cmsc722/notes/chapter02.pdf

3.4.4 PLANNING GRAPH AND GRAPHPLAN ALGORITHM

This is one of the most effective approaches to solve planning problems. Planning graph is a data structure, which helps to extract a reasonable heuristic function. The extracted heuristic function can be used in GraphPlan search algorithm to find a solution for planning problem. It might be possible to observe the solution from planning graph, which employs the GraphPlan algorithm. Planning graph has ordered levels which belong to "time steps". Levels order start from zero and go up. The level zero is dedicated to the initial state and there exists two different sets per level called "literals set and actions set." Basically, the planning graph is capable of working with propositional logic and more specifically it uses STRIPS, which also have propositional presentation. Figure 3.4 formulizes a planning problem at the top and the following part shows a relevant planning graph to the problem. Although GraphPlan decreases the time steps by allowing parallel actions, certain actions can't be performed in parallel. On the one hand, the planning graph shows that there is couple of levels, and each of which consist of several actions that possibly could happen in a particular state. The algorithm is complete and optimal. As shown in Figure 3.5 the GraphPlan algorithm is used to go through the given planning graph problem to generate the reasonable plan to reach the goal. This procedure runs until it finds the proper solution to the problem, or at least understands literally that there is no solution for this planning graph. [7,12]

Init(Have (cake))
Goal (Have (Cake) ^ Eaten (Cake))
Action (Eat (Cake)
PRECOND: *Have* (Cake)
EFFECT: ¬ *Have* (Cake) ^ *Eaten (Cake))*
Action (Bake (Cake)
PRECOND: ¬ *Have* (Cake)
EFFECT: *Have* (Cake))

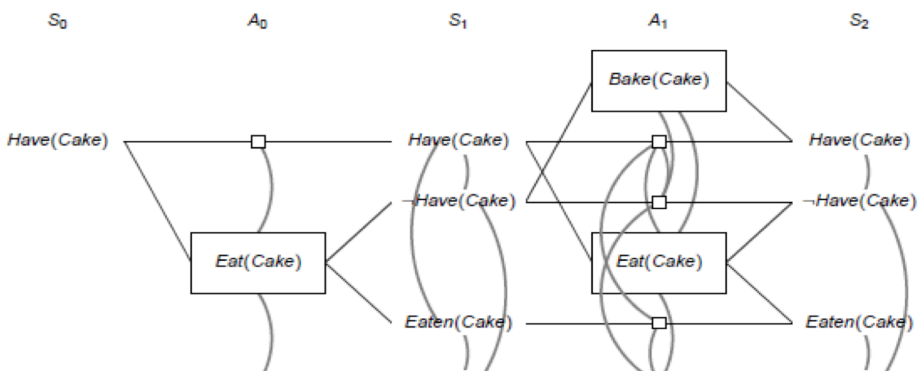


Figure 3.4 The "have cake and eat cake too "problem, and it's a plan graph

```
function GRAPHPLAN(problem) returns solution or failure
  graph ← INITIAL - PLANNING - GRAPH (problem)
  goals ← GOALS[problem]
  loop do
    if goals all non - mutex in last level of graph then do
      solution ← EXTRACT - SOLUTION (graph, goals, LENGTH (graph))
      if solution ≠ failure then return solution
      else if NO - SOLUTION - POSSIBLE (graph) then return failure
      end if
    end if
    graph ← EXPAND - GRAPH (graph, problem)
  end loop
```

Figure 3.5 Graph Plan algorithm

3.5 PLANNING LANGUAGES

In classical planning, the planning language is the language, which can describe initial state, goal state, and all belonging actions, which an agent must take for any planning problems. Planning languages are expressive formal ways to introduce the planning problems. In fact, the planning language interprets the knowledge given by problem description into an official language to display different states as well as a representation of the actions.

3.5.1 ACTION LANGUAGES[3] [4]

In the field of Artificial Intelligence, action languages are widely used to describe actions and their effects as well as the corresponding domain definition for each planning problem. The syntax of this type of language is the same as English and their semantics are like transition functions. There are a number of action languages such as ADL, STRIPS, PDDL, Action Language A, B, and C, where they are describing the set of possible actions that are modifying the states over the time. Among the list, the Planning Domain Definition Language (PDDL) is one of the well-known languages, which it is commonly used.

3.5.1.1 STANFORD RESEARCH INSTITUE PROBLEM SOLVER (STRIPS)

STRIPS was introduced in 1970 to control a robot called Shakey. Shakey was to move from room to room and pick up certain objects and carry them from place to place. Figure 3.6 shows Shakey's world. STRIPS uses A* algorithm's heuristic function to point out the

[3] An introduction to Action language, Tan Cao Son and Chiaki Sakama, January 9, 2010
[4] Action Languages, Michael Gelfond and Vladimir Lifschitz,May 7,1999

qualified node to explore in next attempt. It is a language for classical planning. STRIPS define the planning problem as a set of states. Basically, from the starting point of planning problem to the end point, there are three categories named as initial state, current state, and the goal state. The operators are in fact the set of actions, which are happening in different states causing some degree of change in different states. Each member of the set of operators can perform under certain conditions to change the states. As an example, with Shakey's movement task, information such as the relevant objects, the locations, and how these two factors are related to each other is to be extracted. Operators also have properties such as preconditions that must be satisfied before actions can be taken. STRIPS keeps two list, one contains the states which are added to list in terms of the result for employing certain operations and a delete list which keeps track of the nodes which are deleted from the problem.

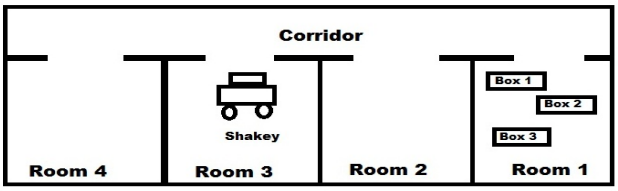


Figure 3.6 Shakey's world and its performance in his environment

- STRIPS formulation:

A person is at location A, in Las Cruces and has a car. There is a gas station in location C, which is El Paso. The person wants to meet his friend in location B, which is Phoenix, but it requires buying gasoline, and filling up the tank in order to reach to Phoenix.

Initial state: CarAt(Las Cruses), Fuel(Low), GasStationAt(El Paso), FriendAt(Phoenix)
Goal state: Meet(Friend)

Actions:

// drive from Las Cruces to El Paso

Drive(Las Cruse, El Paso)
Preconditions: At(Las Cruces), Fuel(Low)
Postconditions: not At (Las Cruces), At (El Paso)

// fill up the tank

FillUp(El Paso)
Preconditions: At(El Paso), GasstationAt(El Paso), Fuel(Low)
Postconditions:Feul(full), notFeul(low)

// drive to Phoenix

Drive(El Paso, Phoenix)
Preconditions: At(El Paso), Fuel(Full)
Postconditions: not At (El Paso), At (Phoenix), Fuel(low)

//meet a friend in Phoenix

MeetFriend(Phoenix)

Preconditions: At(Phoenix), FriendAt(Phoenix), Fuel(low)
Postconditions:Meet(friend)

3.5.1.2 ACTION DESCRIPTION LANGUAGE (ADL)

ADL is based on STRIPS and in some sense it is an upgraded version of STRIPS. It was coined in 1987 and it is basically an automated planning language which is actively used in robotic area. Action description language finds a specific condition that lead to a set of actions that promise to lead us to the goal. Each action has preconditions that must be satisfied in order to let the action start performing. Action performance causes some effects to the "environment" in order to modify it. The environment can be identified by particular states, which are either satisfied, or not. To help in better understanding of ADL, the following example is provided.

- ADL formulation:

Imagine the problem of bus transportation. In this example there are number of passengers who want to go from Las Cruces to El Paso by bus. The bus itself needs to get fuel and the fuel level in tank will change during each transport. Herein, according to the problem specification, there are several basic actions which are involved. They get in the bus, get off the bus, driving the bus, and fill up the tank:

Action (
  Getin (p: Passenger, b: Bus, c: City, g:Gas)
  Precondition: At(p,c) ∧ At(b,c) ∧ At(b,g)
  Effect: ¬ At(p,c) ∧ In (p,b) ∧ ¬ At(c,g)
)

Action (
  Getoff (p: Passenger, b: Bus, c: City, g:Gas)
  Precondition: In(p,b) ∧ At(b,c) ∧ At(b,g)
  Effect: At(p,c) ∧ ¬ In (p,b) ∧ ¬ At(b,g)
)

Action (
  Drive (b: Bus, from: City, to: City, g:Gas)
  Precondition: At(b, fromcity) ∧ At (fromcity ,g)
  Effect: ¬At(b,fromcity) ∧ At (b, tocity) ∧ At(tocity,nogas)
)

Action (
  Fillup (b: Bus, c: City, g:Gas)
  Precondition: At(b,c) ∧ ¬ In(b,g)
  Effect: At(b,c) ∧ In (b,g)
)

In Figure 3.7 STRIPS and ADL are compared.

STRIPS	ADL
- Only positive literals in states is allowed - All the unmentioned literals are counted as false literals, where this situation called ”Closed World Assumption ” - Goals are conjunctions - Effects are conjunctions - Does no support for types - Does no support for equality - Ground literals can be find in Goals - PSPACE complete and some restricted problems are transformed NP-complete - Propositional language	- Both positive and negative literals in states are allowed - All the unmentioned literals are counted as unknown literals. This situation called “ Open World Assumption” - Conjunction and disjunction are allowed for goals - Conditional effects allowed - Variables can have types - Equality is built in - Quantified variables can be find in Goals - PSPACE -Complete

Figure 3.7 Comparison between STRIPS and ADL

3.5.1.3 PLANNING DOMAIN DEFINITION LANGUAGE (PDDL)

Another standard language in AI field, PDDL is specialized to represent classical planning problem domains. There are different versions available from PDDL1.1 to the recent version called PDDL3.1. The extensions of original PDDL are for supporting the domains containing time and numeric resources. For example, PDDL 2.1 has four levels as follow:

Level 1: Propositional description of classical planning
Level 2: Numeric variables
Level 3: Actions which has duration but they don’t have continuous effects
Level 4: Actions which has duration and they do have continuous effects

PDDL is widely used for encoding problem domains and expressing the temporal planning problems. It consists of “objects”, “predicates” which are the attributes of objects, “actions” which are able to apply changes, “goal specification” which should be true to achieve the plan, and finally “initial state” which is the starting state. PDDL consists of two main files, domain and definition. Domain contains states and actions while Definition contains initial state, objects, and the goal specifications. Although PDDL has combination of STRIPS and ADL most of the planners are not able to support it.[5] [6] [5]

[5] http://www.cs.ust.hk/~qyang/221/introtopddl2.pdf
[6] http://cswww.essex.ac.uk/PLANET/summer-school-02/

- PDDL formulation:

Consider the problem of cargo transportation. In this problem instance there is a lift truck, which can move between Las Cruces and El Paso. It used to load with 5 large sizes of objects in one city and it will be unloaded in another city. In the beginning it is job all the objects are in Las Cruces whereas later on they must be in El Paso. Actions loading and unloading happens by a lifter.

Objects: The two cities, five objects, and a lifter.
Predicates: Is χ a city? Is object χ inside city Y? Is lifter χ empty? […]
Initial state: All objects and the lift truck are in the Las Cruces. The lifter is empty. […]
Goal specification: All objects must be in El Paso.
Actions/Operators: The lift truck can move between cities, load an object or unload an object.

Objects:
(: objects cityLas Cruces cityEl Paso
Object1 object2 object3 object4 object5
lifter)

Predicates:
(: predicates (CITY ? χ) (OBJECT ? χ) (LIFTER ? χ)
(at-lifttruck ? χ) (at- object ? χ ? Y)
(free ? χ) (carry ? χ ? Y))

Initial state:
(: init (CITY cityLas Cruces) (CITY cityEl Paso)
(OBJECT object1) (OBJECT object2) (OBJECT object3) (OBJECT object4) (
OBJECT object5)
(Lifter lifter) (free lifter)
(at- lifttruck cityLas Cruces) (at-object object1 cityLas Cruces) (at-object object2
cityLas
Cruces) (at-object object3 cityLas Cruces) (at-object object4 cityLas Cruces)
(at-object object5 cityLas Cruces))

Goal specification:
(: goal (and (at-object object1 cityElPaso)
(at-object object2 cityElPaso)
(at-object object3 cityElPaso)
(at-object object4 cityElPaso)
(at-object object5 cityElPaso)))

Action/Operator:
(: action move: parameters (? χ ? Y)
: precondition (and (CITY ? χ) (CITY ? Y)
(at – lifttruck ? χ))
:effect (and (at_lifttruck ? Y)
(not (at_lifttruck ? χ))))

Action/Operator:
(: action load: parameters (? χ ? Y ? Z)
: precondition (and (OBJECT ? χ) (CITY ? Y) (LIFTER ? Z)
(at-object ? χ ? Y) (at- lifttruck ? Y) (free ? Z))
: effect (and (carry ? Z ? χ)
(not (at- object ? χ ? Y)) (not (free ? Z))))

Action/Operator:
(: action unload: parameters (? χ ? Y ? Z)
: precondition (and (OBJECT ? χ) (CITY ? Y) (LIFTER ? Z)
(carry ? Z ? χ) (at-lifttruck ? Y))
: effect (and (at-object ? χ ? Y) (free ? Z)
(not (carry ? Z ? χ))))

3.5.1.4 ACTION LANGUAGE A

Introduced in 1991, Action Language A is the combination of STRIPS and ADL. In fact, it is STRIPS extension together with the propositional description part, which is borrowed from ADL. When any possible action has defined in language A then it would be deterministic action in all the states. In other words, action language A has a domain that contains dynamic propositions and conditions which must become true to make next action happen. As an example the "Yale Shooting Problem" shows how action language A is defined. There is a turkey called Fred who is alive and there is an unloaded weapon in starting state. Performing each action might cause some effects into this domain. It means that in order to use the weapon it required to load it and thereafter using the weapon means attempt for killing Fred. If shooting happens then it means that Fred comes dead and weapon status changed again to unload. There are two fluent here such as alive and loaded whereas each of which has a truth-values that can be possibly changes over the time and the corresponding actions are load and shoot. The domain and initial state of Yale Shooting Problem can be denoted as follow:

$$\boldsymbol{D_{YSP}} = \begin{cases} \boldsymbol{shoot} \text{ causes } \neg\boldsymbol{alive} \text{ if } \boldsymbol{loaded} \\ \boldsymbol{load} \text{ causes } \boldsymbol{loaded} \\ \boldsymbol{load} \text{ executablif } \neg\boldsymbol{loaded} \\ \boldsymbol{shoot} \text{ causes } \neg\boldsymbol{loaded} \end{cases}$$

$$\boldsymbol{I_{YSP}} = \{initialy\ \neg\boldsymbol{loaded}, \boldsymbol{alive}$$

Following is the transition function of this domain, which shows how the pair of action and state is connecting to its belonged states:

$$\boldsymbol{\Phi_D(a,s)} = \perp \boldsymbol{\ if\ a\ is\ not\ executable\ in\ s\ \ where\ \perp\ denotes\ a\ fail\ state;\ and}$$
$$\boldsymbol{\Phi_D(a,s)} = \left(\boldsymbol{s} \setminus \overline{\boldsymbol{e_a(s)}}\right) \cup \boldsymbol{e_a}\,(\boldsymbol{s})\boldsymbol{if\ a\ is\ executable\ in}\ \mathbf{s}.$$

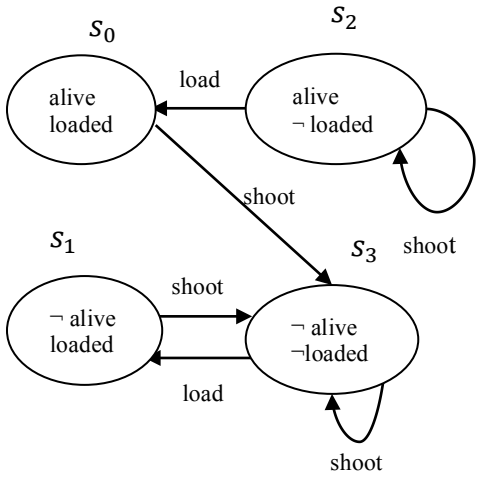


Figure 3.8 Presentation of problem in action langauge A

3.5.1.5 ACTION LANGUAGE B

In general, action language B is a more developed version of language A with some additional features like indirect effects of actions into the states. It is capable of recognizing the differences between static and dynamic rules. Back to Fred example, while alive and loaded has previously been defined as a set of fluents in action language A, here another fluent called dead which indicates "Fred is dead", is added. By adding this fluent it leads to a mutual exclusive situation whereas there is constraint in between these two.

$$\boldsymbol{state\ constrate\ and\ conditions} = \begin{cases} \neg\boldsymbol{dead} \text{ if } \boldsymbol{alive} \\ \neg\boldsymbol{alive} \text{ if } \boldsymbol{dead} \end{cases}$$

The domain and initial state of Yale Shooting Problem can be denoted as follow:

$$\boldsymbol{D_{YSP}} = \begin{cases} \boldsymbol{load} \text{ causes } \boldsymbol{loaded} \\ \boldsymbol{load} \text{ executablif } \neg\boldsymbol{loaded} \\ \boldsymbol{shoot} \text{ causes } \neg\boldsymbol{loaded} \\ \boldsymbol{shoot} \text{ causes } \boldsymbol{dead} \text{ if } \boldsymbol{loaded} \\ \boldsymbol{shoot} \text{ causes } \neg\boldsymbol{alive} \text{ if } \boldsymbol{loaded} \end{cases}$$

$$\boldsymbol{I_{YSP}} = \{initialy\ \neg\boldsymbol{loaded}, \neg\boldsymbol{dead}$$

The transition function for action language B is as follow: [2, 3, 6.8, 9, 10, 14]

$$\Phi_D(a, s) = \{\boldsymbol{s'} \mid \boldsymbol{s'\ is\ a\ state\ and\ s'} = \boldsymbol{Cl_{Dc}}\ (\boldsymbol{e_a}(\mathbf{s}) \cup (\boldsymbol{s} \cap \boldsymbol{s'}))\}$$

3.5.1.6 ACTION LANGUAGE C

Similar to above, action language C also adds indirect effects. Compared to language B, it is a more expressive language since the main focus is to have as an expressive language as possible. Alike languages A, and B the syntaxes are very similar with slight modifications. While language C can handle concurrent actions, language B cannot. Language C, can easily handle concurrent actions to be executed as well as it can cope with nondeterministic actions. Language C makes it possible to be flexible to non-constant fluent.

3.6 PLANNERS

Planners are "general problem solvers" which are acting as an automated solver system to numerous problems. They are aiming to find the reasonable solutions for the problem. It can be either a person acting as a planner to ensure that the given goal can be reachable through that solution or a machine. The plans instruction to reach the goal is the output of planner. In fact, the planner is a program, or it can take a form of planning algorithm to get the problem description as an input and return a plan as an output. The plan is executable path, which lead an agent to find out an optimal solution to the problem. It is like a map that shows the direction and helps people to move from certain place to their desired destinations. Other than planner systems, which are dealing with classical planning problems, there are some planners, which are interacting with durative actions, known as temporal planners. Usually, temporal planners are dedicated to single agent to provide them with an appropriate plan. There are many temporal planners available but here the well-known Single agent temporal

planner known as SAPA is discussed.

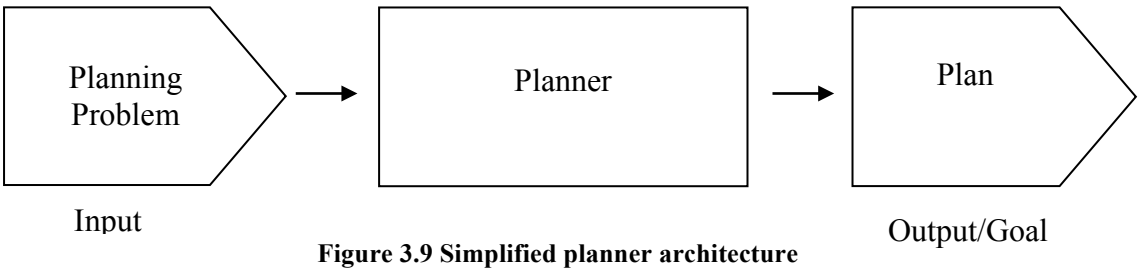


Figure 3.9 Simplified planner architecture

List of some well-known AI planning systems are tabulated as follow:

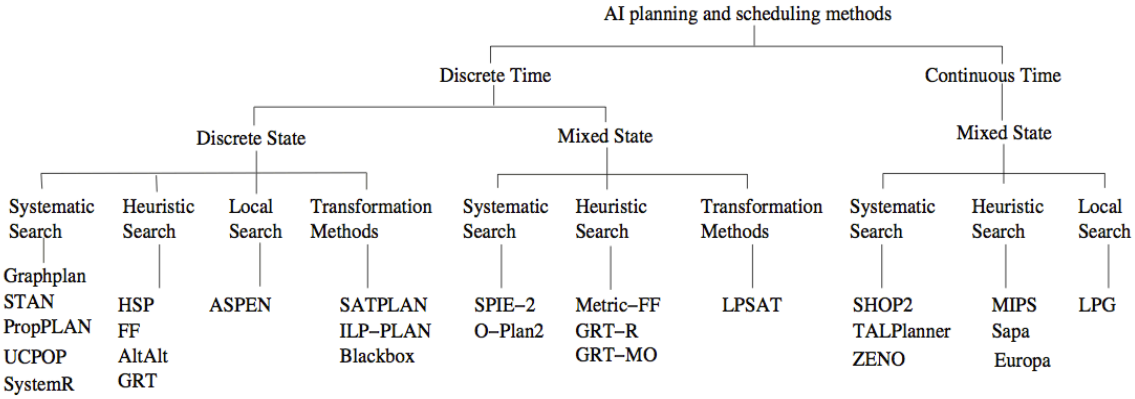


Figure 3.10 Classification of existing planning and scheduling systems

3.6.1 SAPA

SAPA is a temporal planner, which deals with durative actions and resources. “It is a multi-objective metric temporal planner.” The search strategy used by SAPA is forward chaining A* algorithm which get helps from heuristic function. This function is potentially aware of cost of the path as well as its length. However, heuristic function belonging to the other kind of planners is just concerned with only the paths length. SAPA is known as a good temporal planner because it finds reasonable solutions in the way it evaluates the quality of a solution as well as in finding multiple solutions to a problem. SAPA is capable of handling the reasoning process about resources, which are either producers or consumes. SAPA in contrast with other temporal planner is one of the best. There are several temporal planners handling durative actions and use resources almost the same as SAPA such as TLPlan, HSTS, and Zeno but the important ones are the ones that have information and understanding about the domain. However SAPA can interact without having that particular information. Among other temporal planners TGP can only deal with durative actions whereas LPSAT and GRT-R are able to cope with resources and actions, which are either using or producing them. There are only two temporal planners such as TP4 and Resource-IPP where they can work with both durative actions and resources. [4]

Moreover, there certain development has improved SAPA. One of those attempts called DRIPS and the other one is known as Sharaabi. DRIPS is also a Temporal metric planners. Its search space consists of time stamped states. Whenever any event happens, DRIPS, similar to

SAPA, looks for any possible actions from the list of possibilities. In fact, events result from performance of any action in its state either at the expected or at times with some delay. All these effects influence their states. All events are tracked and the times do change accordingly to the time stamp of the event. In developed version of SAPA the temporal planner can check the actions before they are completed. If DRIPS isn't faced with constrains such as time complexity and memory space complexity, it will produce a plan to solve the problem if such solution exists.

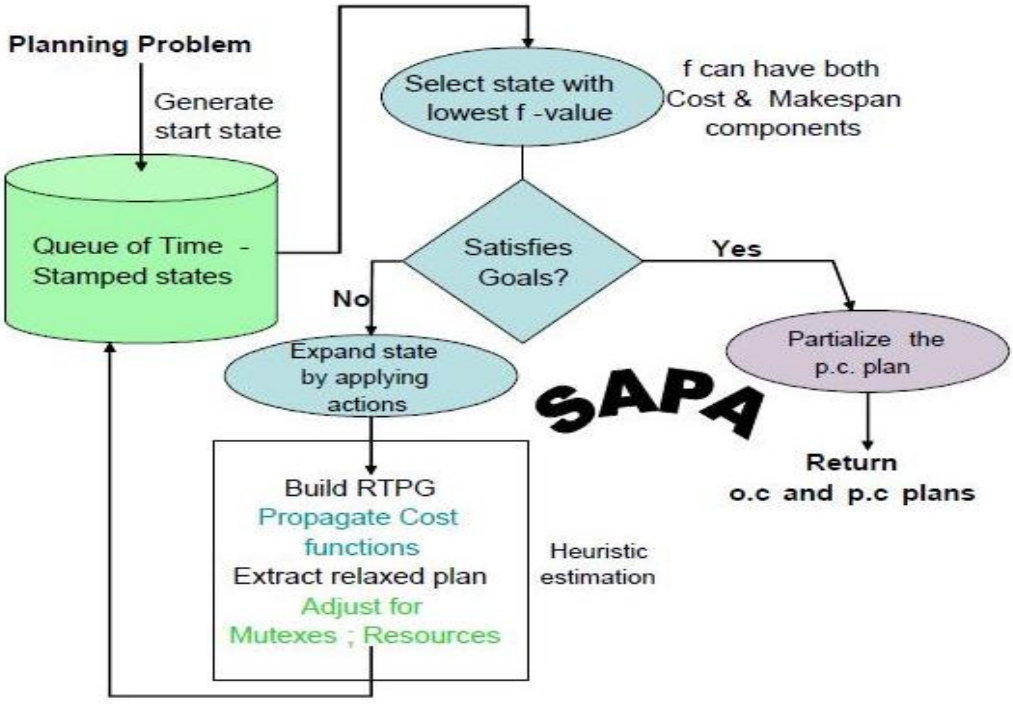
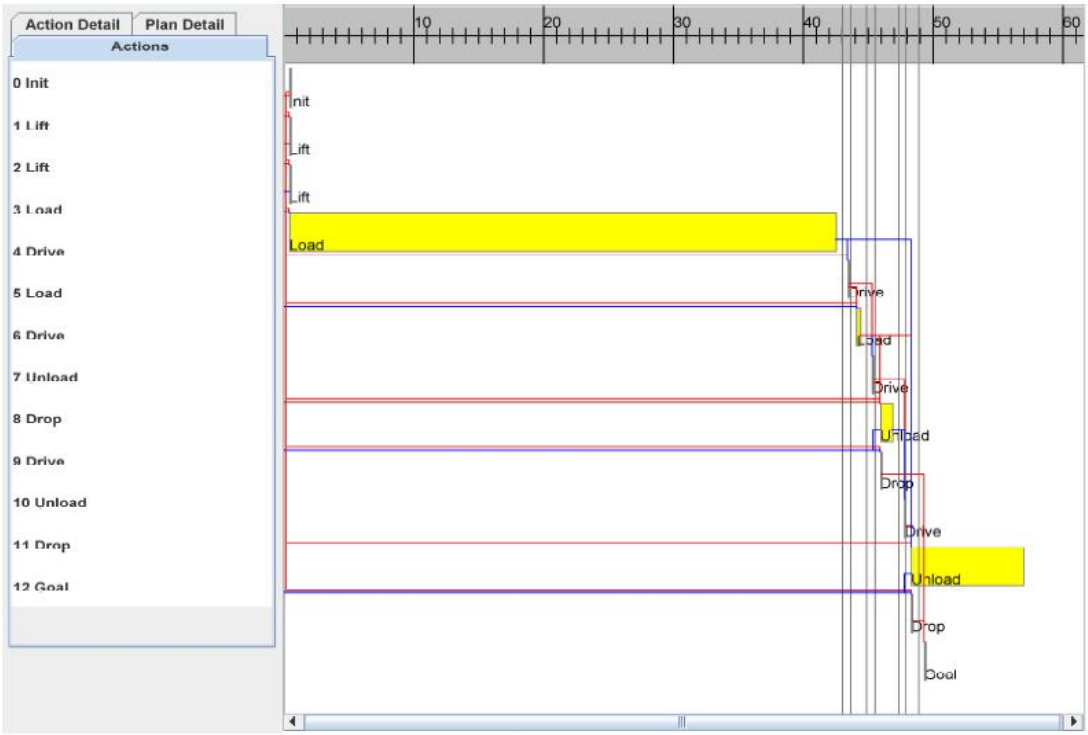
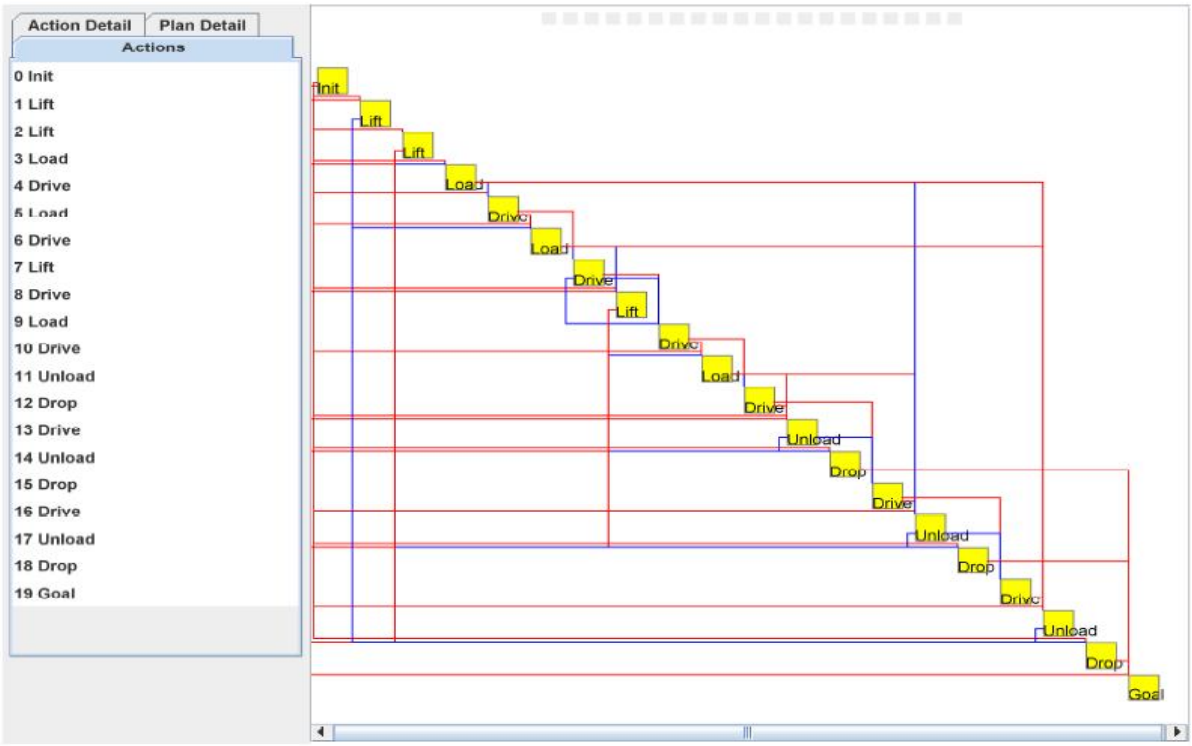


Figure 3.11 SAPA's architecture


Action Detail
Plan Detail
Actions
0 Init
1 Lift
2 Lift
3 Load
4 Drive
5 Load
6 Drive
7 Unload
8 Drop
9 Drive
10 Unload
11 Drop
12 Goal
10
20
30
40
50
60
Init
Lift
Lift
Load
Drive
Load
Drive
Unload
Drop
Drive
Unload
Drop
Goal


Action Detail
Plan Detail
Actions
0 Init
1 Lift
2 Lift
3 Load
4 Drive
5 Load
6 Drive
7 Lift
8 Drive
9 Load
10 Drive
11 Unload
12 Drop
13 Drive
14 Unload
15 Drop
16 Drive
17 Unload
18 Drop
19 Goal
Init
Lift
Lift
Load
Drive
Load
Drive
Lift
Drive
Load
Drive
Unload
Drop
Drive
Unload
Drop
Drive
Unload
Drop
Goal

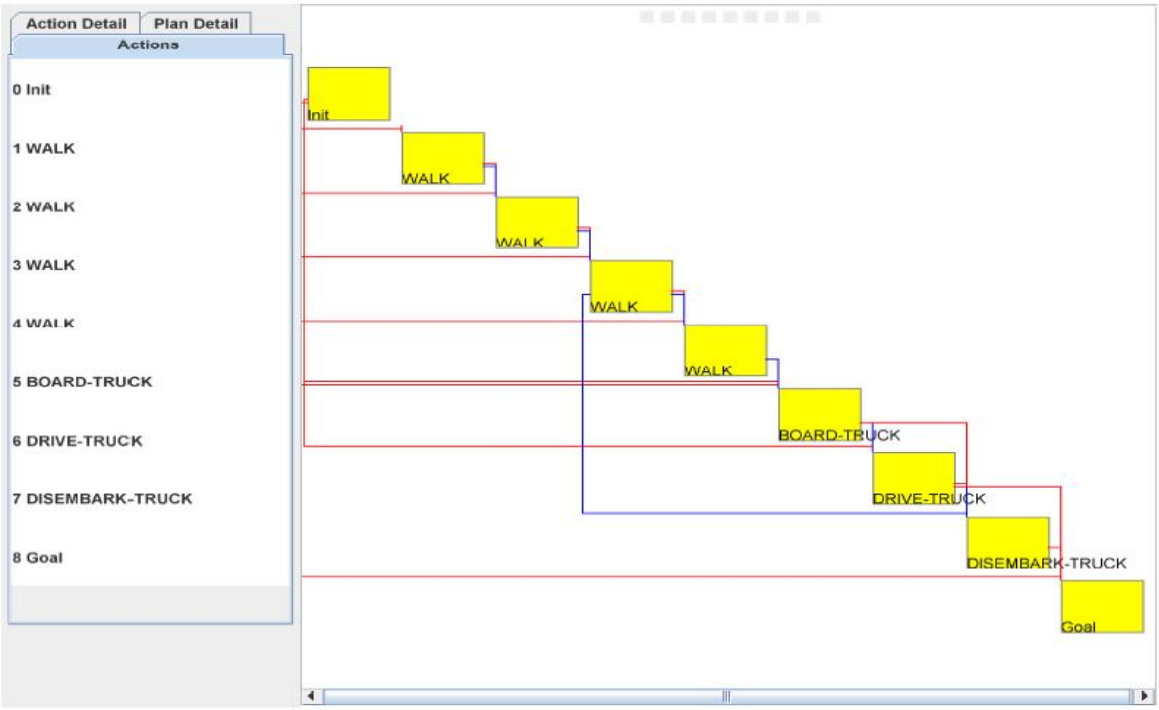


Figure 3.12 running SAPA resulted charts[7].

Sharaabi is an extension to DRIPS and SAPA. For each action to takes place, its preconditions are to be first satisfied. Once the Boolean value of a precondition has been checked, then that value will be kept in the state and there is no need to check its value later. The other modification to SAPA and DRIPS represent that the current time and duration of actions are known as a part of preconditions. Although there are several attempts happened to improve the execution of SAPA, still the contemporary approaches needs to improve their heuristic functions and they need to speed up the solution finding procedure.[18]

The performance of SAPA to solve the number of problems is tabulated as follow:

Planner	Number of solved problem	Number of attempted problem	Success ratio	Capabilities
SAPA	80	122	66%	Time, Complex

Figure 3.13 SAPA Performance

3.6.2 BLACKBOX PLANNING SYSTEM

It is a planning system developed by Henry Katuz and Bart Selman. Blackbox combines the best features of SATPlan(Littman) and GraphPlan(Blum) to come up with a flexible and

[7] http://rakaposhi.eas.asu.edu/sapa.html/

robust system to solve planning problems. The system runs SATPlan solver for a while to solve the problem and when it fails, then the planner switches to GraphPlan if needed. This remarkable functionality helps Blackbox to cover a range of planning problems. As the name of this planning system implies, neither plan generator nor problem solver in the system know anything about what is happening on the other side, whereby, both looks like blackbox to each other.

The acceptable language for Blackbox is PDDL and it uses logical expression to build a propositional frame by using SATPlan and GraphPlan. Based on experiments conducted using blackbox, its shortcoming of handling actions with conditional effects has been illuminated. It is notably capable to solve the complex logistic domains involving large problem description yet it can't scale up for solving very large size problems. Whilst such problems are solvable in less than 10 minutes, in contrast GraphPlan itself is not able to handle those problems at all and GraphPlan dealing with smaller problems may take four times to come up with solution to the problem. In particular, blackbox planning system often gets stuck in GraphPlan and can't find the solution. To overcome this limitation, it employs additional technique whereas that function can modify the heuristic function and cut off the search. It swiftly restarts the backtracking algorithm after a certain number of search algorithm executions. Therefore, it massively improves the search performance. Figure 3.14 shows in Blackbox planner the cutOff and its effect to search performance and Figure 3.15 illustrate the overview of Blackbox.

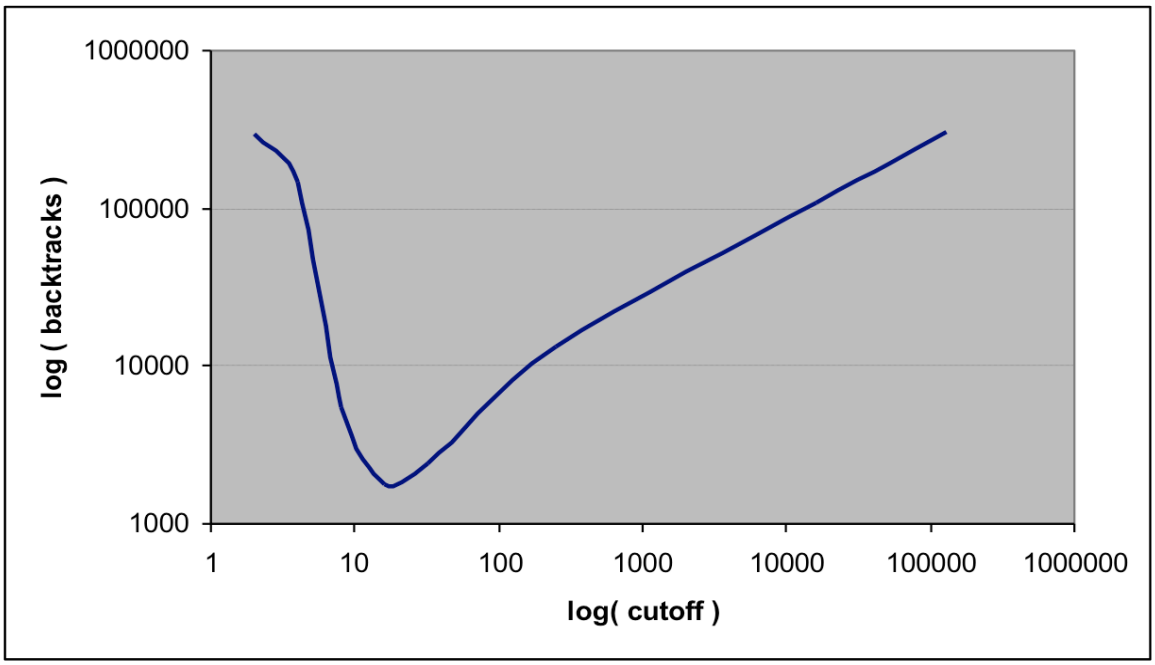


Figure 3.14 CutOff and its effect to search performance in Blackbox planner

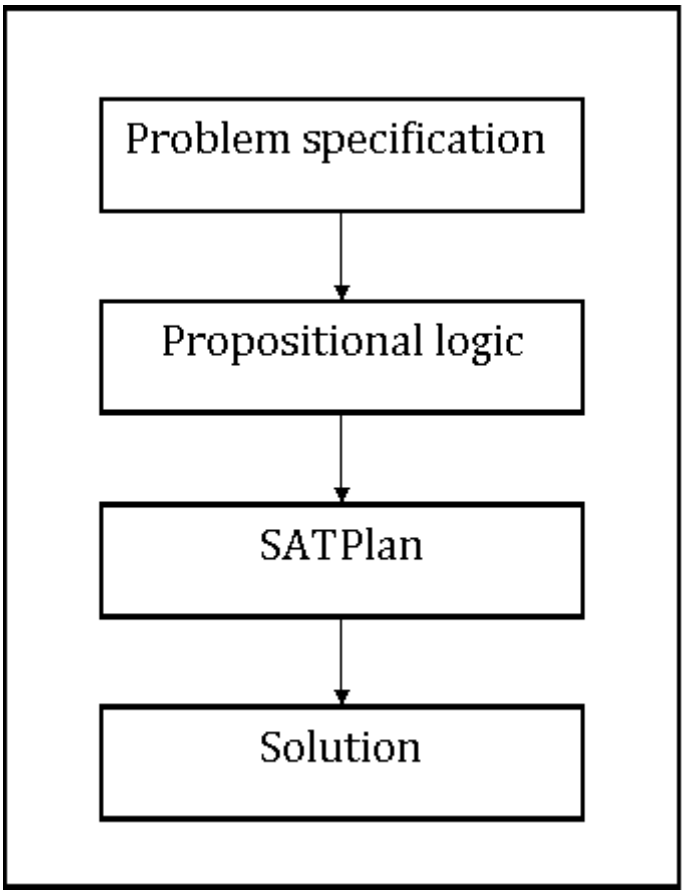


Figure 3.15 Blackbox simplified overview

3.6.3 HEURISTIC SEARCH PLANNER (HSP)

HSP is a planner coined by Bonet and Geffner. It is a domain independent planner, which solves STRIPS problems by using heuristic function from starting state to the goal state. Its implementation has been done in programming language C. Studies show that, the search algorithm used by HSP can be any of Best First Search, weighted A*, weighted BFS, Greedy BFS, or Hill Climbing search in order to find out the distance estimated from starting state to the goal state without considering on how good is the solution.

In contrast, it can't use A* algorithm as a search alternative to reach the goal while A* has space complexity since it gathers all the states in the memory and also A* is too slow to hit the goal state to solve the problem. Unlike A*, the Greedy search or Hill Climbing search could be a good approaches to take for both small size and large size problems. It is because they are usually very fast to generate the states and they are guaranteed to find an optimal or almost optimal solution to the problem. The heuristic function to the HSP can be observable from the problem description. Likewise the other heuristic search planning systems, in HSP also the search procedure will take time based on how large is the state space and the number of child at each state, the length of founded solution, and heuristic function.

HSP has time complexity since it recalculates its heuristics function in each new state and it doesn't keep going forward to find a solution to the problem with a constant heuristic

function. HSP is capable to handle actions that have conditional effects in return of those action executions.

The heuristic function in use by HSP can be either admissible (optimistic) heuristic, or non-admissible (non-optimistic) heuristic. The term admissible or optimistic applies when the function never overestimate the cost from current state to the goal state.

Figure 3.16 provides a brief comparison between HSP and Blackbox based on number of problems solved by them as well as it points out among all attempts by each planner how many times they were fastest to present the plan and how many times they brought shortest path from the initial state to the goal state. Studies show that HSP was faster than Blackbox in the way of solving logistic problems. Also the size of the plan presented by HSP was smaller compare to the one generated by Blackbox.

Planner	Average time	Solved problem	Fastest Planner	Shortest Planner
HSP	1.49	63	16	55
Blackbox	35.48	82	19	61

Figure 3.16 Comparison between HSP and Blackbox

3.6.4 LAMA

LAMA is a classical planner that uses heuristic search to solve the problems. It also uses another source for searching the solution called landmarks. The term landmark implies that there are a number of propositions in the provided solution whereas their truth-value must be set as true in all the solution. Landmarks are propositional sub goals that should be available in any possible solution to the problem. LAMA uses landmarks to estimate the distance from current state to the goal state to help with heuristic search. Specifically LAMA can obtain the distance to the goal from the current state by keeping track of the number remaining sub goal, which are not achieved. The sequential numbers assigned to all the remaining landmarks tell which landmark should be achievable next. LAMA uses weighted A* search algorithm whereas it looks for solution continuously and it updates the weights by reducing assigned value. Therefore, LAMA keeps searching iteratively to find the best possible plan until search procedure is terminated. In order to find a good quality of solution, LAMA uses the cost to reach the goal from initial state together with distance from the current state to the desired goal state. However the searching procedure starts by greedy best first search to find a quick solution to the problem. The first phase of searching will be done as soon as the first solution is found. In the second phase of search, the weighted A* will starts searching for solutions iteratively to observe any better solution considering a reduction of the weights.

The core structure of LAMA has three different programs: translator program, which is implemented in Python, the “knowledge compilation” program, which is implemented in C++, and the search program, which is also implemented in C++. LAMA planner invokes each program where they are sorted in sequential order. The communication between programs is done through text files.

In fact, the translator programs interpret the problem to PDDL to give it as an input to the planning system. The knowledge compilation is generating date structures that are vital for

further sub goal generation, which enhances the heuristic search. Therefore, it is time for search engine to play its role to find a solution to the problem accordingly. [19]

3.6.5 CRIKEY

The temporal planner CRIKEY is a forward heuristic search planner that uses both best first search and enforced hill-climbing search algorithms in the planning phase whereas the scheduling phase uses greedy search algorithm. CRIKEY supports PDDL but not any other languages. It handles both metric and time planning problems. It draws the line between planning and scheduling phases. It can also establish the connection between them as necessary. This can be done by using the relaxed problem, which is able to load the search procedure. The relaxed problem is a sub problem of the real world problems in which the constrains are reduced. Moreover, the quality of relaxed problem is directly influenced by the planner's execution. In some cases it may affect the search procedure to find a solution to the problem. If that is case, then it is CRIKEY's responsibility to observe those situation to modify the relaxed problem based on its observation to omit the deadlock and prevent the generation of plan which is not schedulable otherwise the planner will fails. CRIKEY uses Forward Heuristic Search over constructed graphPlan and it is capable to handle both metric temporal planning problems. CRIKEY's implementation has been done in Java.

As it was mentioned earlier when CRIKEY is faced with a temporal planning problem, it first attempts to divide it into the planning and scheduling phases. Combining these two phases, if necessary, may avoid a failure in solving a relaxed problem. In order to detect failing situations it looks for such a case in the domain description of the temporal planning problem. In other words it looks at whole domain to figure out where both phases must be combined and when they should be separated. It starts modifying the domain and solves two phases together if necessary. CRIKEY is guaranteed to find a solution if there exists one for any particular temporal planning problems. In that sense CRIKEY is called complete but it can't be called optimal since it can't guaranty the quality of the solution, as it can't assure that the identified solution is the best one considering time complexity and resource impacts. It is necessary to double-check the plan to make sure that is schedulable as well. Therefore, Mini-Scheduler which is consists of three elements such as: A Simple Temporal Network (STN) (Dechter, Meiri, and Pearl, 1991) containing a set of " time- point variables and binary constraints" [8] between those variables is employed. In general STN indicates the constraints between the tasks that an agent is supposed to perform, durative actions constraint, temporal priority of actions, and temporal constraint between actions. Mini-scheduler will starts reasoning about set of chosen actions by performing those set of actions within the specified length of time and if they do have enough time for their performance, the Mini-Scheduler will consider those set of actions as appropriate actions otherwise it will discard them from the problem search space. The Mini-Scheduler uses same algorithms as the one uses by CRIKEY in its scheduling phase. The Mini- Scheduler will finish its task whenever the set of approved actions are accomplished in the plan, then it will terminates. CRIKEY needs to improve the quality of its search. Better heuristic function is required in order to guaranty the quality of search. The heuristic function is depending on the number of participated actions in relaxed problem. CRIKEY is not able to handle the resources whereas they are producing or

[8] http://www.cs.vassar.edu/~hunsberg/__papers__/ss-thesis-parts/ss-thesis-chap4.ps

consuming as well as they does have dependency on their states. Figure 3.17 is showing how these two phases are separated in CRIKEY. [15][16][17]

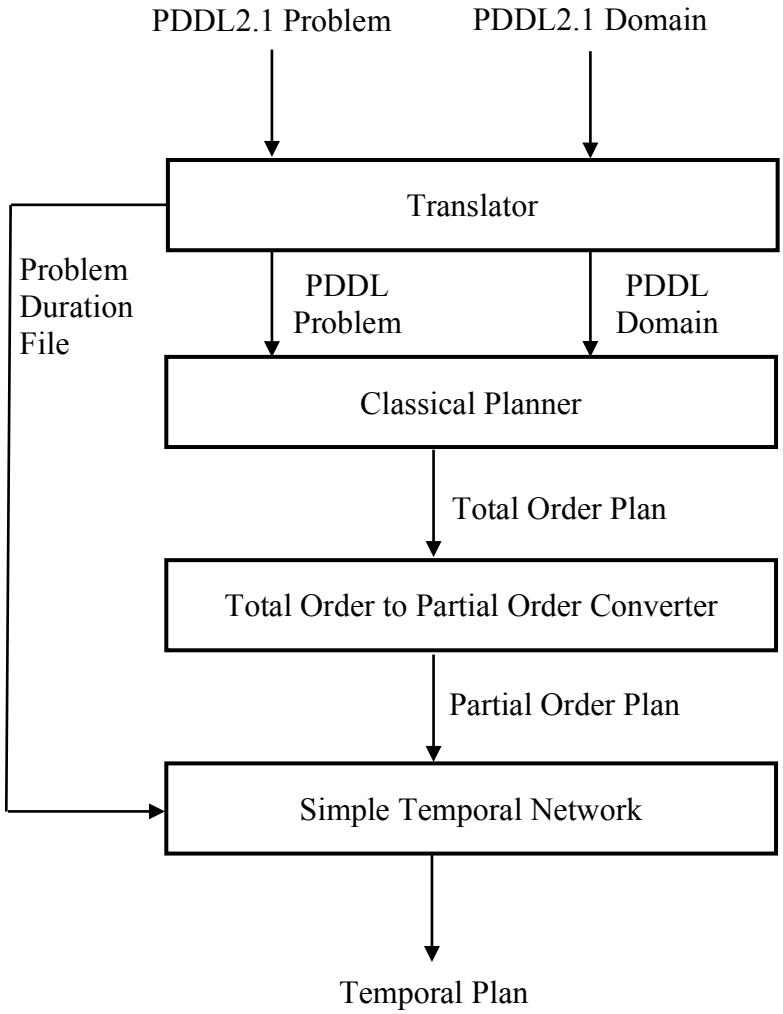


Figure 3.17 Architecture of CRIKEY in terms of seprating two phases

3.7 PLANNING AND SCHDULING

Scheduling is slightly different from planning. During planning some sequential actions are taken to find a reasonable path from initial state to the goal state. Scheduling on the other hand is mainly dealing with who has the possibility to solve the problem, which contains "temporal information" as well as thinking about which order of action must be taken to meet the "deadlines." A very famous example of scheduling is "job shop scheduling problem" which has set of "jobs" and each job has its own sequence of actions and corresponding constraints. All involved actions have duration and they are using certain resources. The utilization of the resources, the applicability of each resource to the task, as well as availability of resources must all be put into consideration. Whether a resource produces or consumes must be defined. To solve the "job shop scheduling problem" the time to start for any actions must be denoted in advance and all the constraints must be carried through. Literally, time as a resource is a critical issue to deal with it. Therefore, the combination of

planning and scheduling is required to cope with production and consumption by different actions in the problem domain. [1]
Scheduling and planning are done in two separate phases to handle durative actions and resource assignment. Figure 3.18 illustrates two phases. [9] [10]

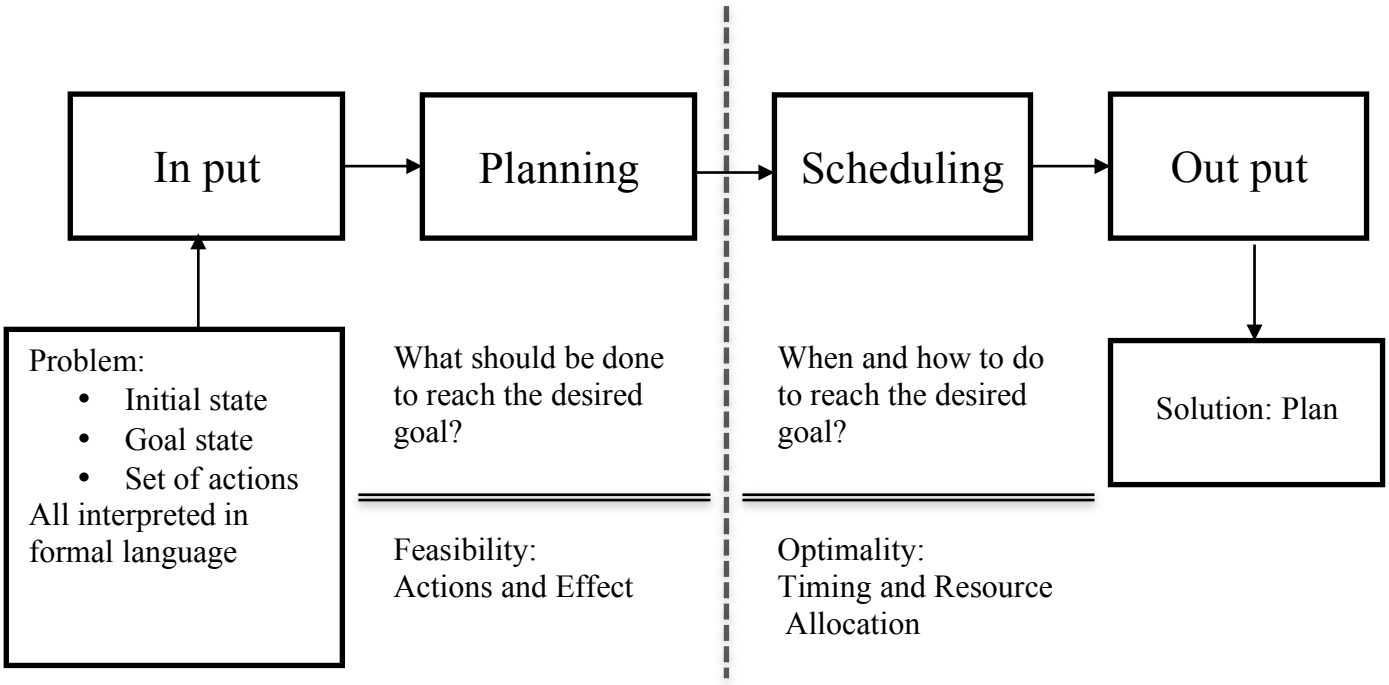


Figure 3.18 Planning versus Scheduling

9 http://prometeo.ing.unibs.it/sschool/slides/ghallab/ghallab-slides.pdf

10 http://ai-lab-webserver.aegean.gr/EETN/planning_and_scheduling

4 EXPERIMENTAL RESULTS

SAPA, Black Box, HSP, LAMA, and CRIKEY planners are subjects of the experiments conducted as part of this thesis. The planners were selected based on that they all use A* or other types of local search to reach the goal and they all are capable of handling PDDL as an input language. Three problems with varying degree of difficulty were selected to exercise the selected planner. For each planner and each planning problem the research focused on whether the planner solves the problem, and if so, the time taken by the planner, the number of actions in the solution, the planners efficiency and its effectiveness. This Chapter provided the details of the findings when the above planners were applied to three distinct problems.

4.1 Logistic Problem – Package Delivery

The domain of the problem consists of trucks, airplanes, airports, and packages. The airplanes are used to transport number of packages from one airport to another while trucks do the local intra city deliveries. The planning process includes actions such as load, unload, drive, and fly. Every package delivery includes all aforementioned actions therefore combination of these actions in sequence leads to the transportation of a package to an airport and delivery of it from an airport to another. The goal is to devise a plan for minimal time to make the deliveries.

Assumptions:

Three airport	Three Depots	Three Points of Origins
3 trucks	2 airplanes	3 packages

Goal:

Minimize time of delivery

Process:

Truck picks up at point of origin and delivers to airport
Airplane carries from one airport to another
Truck delivers from airport to depot

Figure 4.1 Package Delivery Domain

Table 4.1 depicts the performance of five competing planners all solving the Package Delivery Problem. Using STRIPS the performance of CRIKEY, HSP, LAMA, SAPA, and Black Box running this logistic problem was evaluated.

	CRIKEY	**HSP**	**LAMA**	**SAPA**	**Black Box**
Search time	120 ms	28 ms	238.78* s	168 ms	53 ms
Search length	25	27	25	25	25
Expanded states	N/A	41	105	45	N/A
Generated states	N/A	343	1386	154	N/A
Efficiency	N/A	**0.7**	**2274.1**	**3.7**	N/A
Effectiveness	**4.8**	**1.0**	**9551.2**	**6.7**	**2.1**

Table 4.1

* *High value may not have been caused by planner.*

The results show that all the planners were able to handle the problem but in different ways. Their search times varied from 28 Milliseconds to almost 239 seconds but their search lengths were almost equal. The generated and expanded states varied significantly with HSP demonstrating the most efficiency and most effectiveness. Unfortunately neither Crikey nor Black Box outputted the states counts.

With an Efficiency of 1.0 and an Effectiveness of 0.7, by far HSP was the best performer in solving the Package Delivery problem. It generated 343 states and expanded on 41 in just 28 milliseconds. Its search algorithm combines searches know for reaching best solution with searches known for quickness in reaching solutions. Specifically to be as quick as possible HSP starts its search with some Greedy search and as soon as it found the first solution it starts to investigate for not only the quicker one but also for the optimal one. HSP significantly reduced the time consumed by planner to find a solution even though it took 2 more actions in less time comparing the other evaluated planners in this study. The closest competitor was Black Box with an Effectiveness of 2.1.

4.2 Depot Problems

Combining Logistics and BlockWorlds domains, a new domain called Depot domain has been formulated. In this problem the trucks are moving boxes from their point of origination to a depot where the boxes are stacked with a priori knowledge of the desired stacking order. The lifters on the trucks utilize this knowledge in stacking the boxes inside the trucks and at the depot. The possible actions defining the states of this domain are:

Drive	The movement of trucks from the origination point to the depot. Position and the duration are dependent on the speed of the trucks.
Lift	The action to move a box from a truck to the stacking position. Certain conditions have to be met one being that the boxes and the lifter are in the same place and the same time and that the lifter is available.
Drop	The action of the lifter to drop a box in its appropriate position. The action could be in stacking the box at the depot or inside the truck based on the stacking requirement for each box.

This domain has 4 different forms: Strips, Numeric, Simple-Time, and Time:

Strips	Simplified form of the domains. Strips limit the size of search space when the search space is too big.
Numeric	Optimizes usage of fuel by trucks and lifters
Simple-Time	Optimizes the time utilization by the truck and by lifter. Simultaneous actions requiring truck and lifter are taken into account.
Time	Optimizes the time utilization by the truck and by lifter considering the speed of the truck and lifter as well as the weight of the boxes to be transported and stacked. Scheduling based on the speed of the trucks and their positions is considered.

This case considers the availability of the trucks, their proximity to the destination, and the weight of the boxes and the order of stacking. Faster truck can be available to serve first but

they need to be in the right position, Therefore an accurate plan to solve the problem is needed to schedule resources and redirect them in an efficient way. Depots domain can be considered as a sequential domain where time is varied in different versions.

As described earlier the domain itself is the combination of two other domains and they all consists of certain objects called trucks, lifters, boxes, and number of boxes which are to be stacked on top of each other in certain place. But as each form has its own criteria, the initial state would be varying from one to other. While Strips concentrates on basic features, the initial state for Simple-Time is duration of each step, for Time duration depends on distance between locations, and for Numeric it is based on catching the goal state and using the minimal fuel possible. Regardless of the initial states of the four forms, the common goal is to move boxes and stacking them in particular places.

Two different stances of the depot problem were considered, a single depot and a multi depot.

4.2.1 Single Depot

The parameters utilized in this problem are outlined in Figure 4.2.

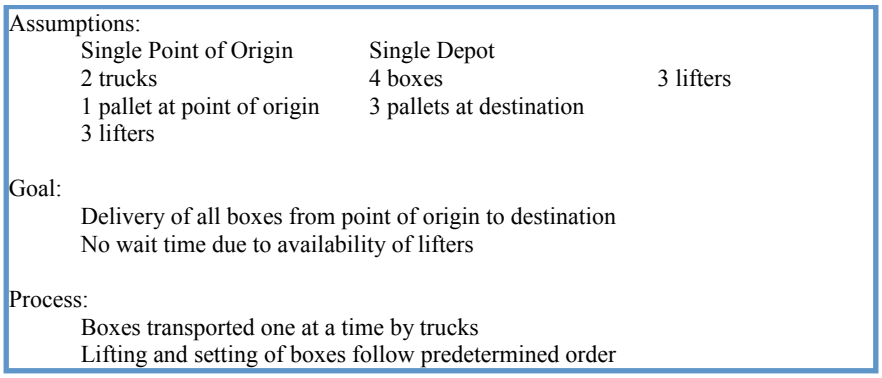


Figure 4.2 Single Depot Domain

In Tables 4.2 to 4.6 the results of five planners working on the Single Depot problem are tabulated.

CRIKEY	**Time**	**Strips**	**Simple time**	**Numeric**
Search time	130ms	0.00s	90 ms	10 ms
Search length	15	17	13	17
Efficiency	**N/A**	**N/A**	**N/A**	**N/A**
Effectiveness	**8.7**	**0.0**	**6.9**	**0.6**

Table 4.2

HSP	**Time**	**Strips**	**Simple time**	**Numeric**
Search time	Timed Out	28ms	Timed Out	Timed Out
Search length		16		
Expanded states		16		
Generated states		120		
Efficiency		**1.8**		
Effectiveness		**1.8**		

Table 4.3

LAMA	Time	Strips	Simple time	Numeric
Search time	Timed Out	3.95s	Timed Out	Timed Out
Search length		15		
Expanded states		71143		
Generated states		539983		
Efficiency		**0.05**		
Effectiveness		**263.3**		

Table 4.4

SAPA	Time	Strips	Simple time	Numeric
Search time	86ms	60ms	78ms	Timed Out
Search length	17	17	85	
Expanded states	32	31	39	
Generated states	65	76	85	
Efficiency	**2.7**	**1.9**	**2.0**	
Effectiveness	**5.1**	**3.5**	**0.9**	

Table 4.5

Black Box	Time	Strips	Simple time	Numeric
Search time	Timed Out	Timed Out	Timed Out	Timed Out
Search length				
Expanded states				
Generated states				
Efficiency				
Effectiveness				

Table 4.6

Crikey was the only planner cable of solving this planning problem using all of its four versions. The results show that the search time varied from almost 0 to 130 Milliseconds with search lengths as low as 13 and as high as 17. No information about the states is generated by Crikey therefore the Efficiency of the various versions of this planner cannot be compared. Its Effectiveness however shows that Numeric and Simple time versions outperformed the other versions.

Three of the four versions of HSP timed out before any results could be generated. Strips was the only version that produced results reaching a search length of 16 in 28 Milliseconds. Similarly LAMA's results were limited to Strips as well but as shown in Table 4.4, the search time reached almost 4 seconds with more than half a million states generated in this ineffective process.

Similar to Crikey, SAPA was able to reach results in three of the versions in reasonable time and excellent Efficiency and Effectiveness. Time and Strip versions demonstrated good performance however Simple time did have the best overall performance.

Overall the most optimal planner for this Logistics problem with an Effectiveness of 0.6 was Crikey using the Numeric version. Within 10 milliseconds it reached a solution. The closest competitor to it was SAPA's Simple Time version.

4.2.2 Multi Depot

The parameters utilized in this problem are outlined in Figure 4.3.

Assumptions:

Single Point of Origin	Three Depots	
2 trucks	6 boxes	6 lifters
1 pallet at point of origin	1 pallet at each destination	

Goal:

Delivery of all boxes from point of origin to three destinations
No wait time due to availability of lifters

Process:

Boxes transported one at a time by trucks
Lifting and setting of boxes follow predetermined order

Figure 4.3 Multi Depot Domain

In Tables 4.7 to 4.11 the results of five planners working on the Multi Depot problem are tabulated.

CRIKEY	**Time**	**Strips**	**Simple time**	**Numeric**
Search time	Timed Out	7.26s	Timed Out	10.84s
Search length		31		31
Efficiency		N/A		N/A
Effectiveness		**234.2**		**349.7**

Table 4.7

HSP	**Time**	**Strips**	**Simple time**	**Numeric**
Search time	Timed Out	10.56s	Timed Out	Timed Out
Search length		26		
Expanded states		93199		
Generated states		975941		
Efficiency		**0.11**		
Effectiveness		**406.2**		

Table 4.8

LAMA	**Time**	**Strips**	**Simple time**	**Numeric**
Search time	Timed Out	3.99s	Timed Out	Timed Out
Search length		24		
Expanded states		27770		
Generated states		431066		
Efficiency		**0.14**		
Effectiveness		**166.3**		

Table 4.9

SAPA	**Time**	**Strips**	**Simple time**	**Numeric**
Search time	1373ms	1623ms	1227ms	Timed Out
Search length	26	27	26	
Expanded states	170	100	146	
Generated states	1863	1238	1681	
Efficiency	**8.07**	**16.23**	**8.40**	
Effectiveness	**52.8**	**60.1**	**47.2**	

Table 4.10

Black Box	Time	Strips	Simple time	Numeric
Search time	Timed Out	Timed Out	Timed Out	Timed Out
Search length				
Expanded states				
Generated states				
Efficiency				
Effectiveness				

Table 4.11

For this more complex depot problem, Time and Simple time versions of Crikey could not reach a solution and timed out. Strips and Numeric did manage to reach solutions with equal search lengths with search times of 7.26 and 10.84 seconds respectively deeming Strips as the more effective of the two.

As was seen in the Single Depot scenario, three of the four version s of HSP timed out before any results could be generated for this second depot problem. Strips did produce results reaching a search length of 26 in 10.5 seconds. Its Efficiency was great but its Effectiveness was extremely poor. Similarly LAMA's results were limited to Strips but as shown in Table 4.9, the search time reached almost 4 seconds with less than half a million states generated in this efficient but ineffective process. It is interesting that the number of states generated and the expanded states by LAMA for the multi depot problem were less than the single depot case.

Across the board SAPA was the best performing planner for this complex problem. Three of the versions led to solutions in far less time than the other planners while reaching relatively the same search length with far more effectively but less efficiently. The most optimal one of the three was Simple Time version.

4.3 Discussion of results

CRIKEY was able to produce a solution for the majority of the cases applied. For the Logistic problems as well as Single Depot, it was able to reach solutions in a timely and effective way. It did however time out in running the Multi Depot problem using its Time and Simple Time versions. CRIKEY by default uses Enforced Hill-Climbing which may fail while doing a local search however, adjusting the preset algorithm to Breadth-First Search will bring solution to all different version of the depots problem. CRIKEY is not an optimal algorithm for finding the best solution but is one to find a quick solution however its quickness was not seen in the Multi depot problem as it took significant time to reach a solution. It was also observed that while SAPA handled some actions simultaneously, CRIKEY took a single action at the time. Interestingly all actions taken by SAPA were eventually taken by CRIKEY.

As for HSP, it was only able to run in the Strip version for all three problems. During the search procedure it first employs Greedy Best First Search and then Best First Search to reach the goal in both directions, forward and backward. It did fine with the simplest of the three problems but had a slightly ineffective process in solving the multi depot problem.

LAMA also ran only in Strip version. Its performance from a time and effectiveness point of view was the poorest of all the planners. The LAMA planner uses Lazy Greedy and Lazy

Best First searches in its first iteration in order to find a quickest possible solution to the problem at hand. Subsequently and immediately after finding an initial solution, it will move on to find a more optimal solution by replacing the initial search algorithms with the non-continuous Weighted A*. The search starts from the beginning to improve the quality of the search procedure in the planner. The process is too complex for three of the four versions while the simplest one, Strips, takes a long time to converge on a solution. Observations tell that the solution generated by LAMA take much longer whereas per each run of search there are more states generated than the one expanded in the process of examining different possibilities. In solving the problems about half a million states were generated and a high percentage were expanded upon. LAMA consists of three different programs that are invoked as needed to solve the task in hand. Although in theory LAMA is good in satisfying the sequential domains, in this case at times it crashed and at times it wasn't able to proceed as it hit a deadlock situation. Based on the observation the problem occurred either because of memory usage, which is calculated in rang of 55.3% up to 72.7% or CPU usage in rang of 8% to 95%. Note that CPU usage never exceeds 95% regardless of memory usage. Other potential explanations are (1) overloaded memory when huge graph is generated, or (2) blocked memory or possible memory leakage due to assignment parts of memory to a variable or task which is not really in use but for which the memory remains active until released.

According to the results presented in the above tables, the solutions found by SAPA was a well performing planner for all the three problems and in a majority of its versions. This can be attributed to the search algorithm used by SAPA, Breadth First Search. SAPA uses forward chaining for its search direction and supports all the versions except Numeric. This is because metric temporal planners deal with more complex constraints whereas classical planners only handle constraints between actions. Although SAPA handles both durative actions and the resource consumer actions, still it is not capable to handle Numeric since SAPA requires more time to produce a solution with less cost. However, in contrast Numeric version requires less fuel usage to achieve the goal. SAPA is a sound and complete planner, which means that it guaranty that if there exists any solution to the problem in hand it will generate it.

Unlike other planers Blackbox is not capable to handle depots problems. It uses the SATPlan's local search and the GraphPlan's engine to search the plan graph. Blackbox also applies another strategy to improve its performance. It actually restarts the search procedure after the fixed number of backtracking, i.e. it adds a randomization into the heuristic in the search in order to reduce the amount of time consumed to produce the proper solution. However neither of these features is able to give it the possibility of solving depots problems.

The version that worked on all the three problems was Strips. In Tables 4.12 to 4.15 the performance of four out of five planners using Strips is outlined. It works quicker than the other versions and all but the Black Box case lead to reasonable solutions. The more complex the problem, the more actions the planner needs to apply to reach its goals. This requires more memory to allocate to handle therefore the more complex problems require more time and memory. Clearly Strips capability to minimize the size of the domain to a manageable one gives it the possibility to run quickly before any timeout or memory limitations are reached. Out of the four planners whose Strip version worked HSP and SAPA clearly have the better performance when one considers time, efficiency and effectiveness.

To solve logistics problems HSP shows the best performance overall with SAPA having slightly better effectiveness on harder multi depot problems.

Crikey	*Logistic*	*Single Depot*	*Multi Depot*
Search time	120 ms	0.00s	7.26s
Search length	25	17	31
Efficiency	N/A	N/A	N/A
Effectiveness	**4.8**	**0**	**234.2**

Table 4.12 Crikey using Strips

HSP	*Logistic*	*Single Depot*	*Multi Depot*
Search time	28ms	28ms	10.56s
Search length	27	16	26
Expanded states	41	16	93199
Generated states	343	120	975941
Efficiency	**0.68**	**1.75**	**0.11**
Effectiveness	**1.0**	**1.8**	**406.2**

Table 4.13 HSP using Strips

LAMA	*Logistic*	*Single Depot*	*Multi Depot*
Search time	238.78* s	3.95s	3.99s
Search length	25	15	24
Expanded states	105	71143	27770
Generated states	1386	539983	431066
Efficiency	**2274.1**	**0.05**	**0.1**
Effectiveness	**9551.2**	**263.3**	**166.3**

* *High value may not have been caused by planner.*

Table 4.14 LAMA using Strips

SAPA	*Logistic*	*Single Depot*	*Multi Depot*
Search time	168ms	60ms	1623ms
Search length	25	17	27
Expanded states	45	31	100
Generated states	154	76	1238
Efficiency	**3.7**	**1.9**	**16.2**
Effectiveness	**6.7**	**3.5**	**60.1**

Table 4.15 SAPA using Strips

5 CONCLUSION

This thesis answers the following question: Of the numerous planners and search algorithms available today, which one is capable of effectively and efficiently solving logistics problems is a timely manner? Through selection of appropriate planners and search algorithms, the most promising ones were selected and utilized in solving logistics problems. It was explicitly shown that that HSP has superior performance in solving various logistics problems. It was quick, efficient and effective in solving all three problems using its Strips version.

In this thesis planning problem along with search algorithms and strategies have been studied in detail. A brief comparison between search algorithms has shown the power of these algorithms in finding reasonable solutions for planning problems. The thesis also covered the planning as a core along with all the related concepts. Particularly, classical planners were compared to temporal planners and usability of each was highlighted. Scheduling as it relates to planning was also discussed in detail along with some computer languages to represent planning problems. Specifically, five temporal planners SAPA, CRIKEY, Blackbox, HSP, and LAMA were selected and utilized in solving three distinct logistics planning problems with the goal of finding the most effective and efficient planner to solve any logistics problem. Methods to measure the performance of planners were presented and it was demonstrated how these planners and their respective versions dealt with the problems, explicitly showing their strengths and weaknesses. All and all, through experiments conducted it was shown that HSP has superior performance in solving various logistics problems. It was quick, efficient and effective in solving all three problems using its Strips version. The overall high performance is attributed to its use of Greedy Best First Search followed by Best First Search to reach the goal in both directions, forward and backward.

It was also observed timing and potentially memory issues that deserve further look in future research. As expected time plays an essential role in concept of planning and scheduling problems.

5.1 FUTURE WORK

As an avenue for future research, the current planners with complementary capabilities can be combined to achieve better performing planners suitable for Logistics and other real world problems. It is the recommendation of the researchers that a common hardware and software platform should be used to launch the various planners and demonstrate all planner performance under a common platform and under similar circumstances. Furthermore using a common platform planning problems are to be run multiple times to remove any anomalies that may exist in the data. The authors will put more effort on programming our own idea to take part in competition with existing planners. Thinking about search space in planning domain and finding the way to prevent going through any paths which are not providing a solution can be another point to put it in consideration.

BIBLIOGRAPHY

[1] Stuart Russell and Peter Norvig. Artificial Intelligence A Modern Approach Third Edition/2010.

[2] Hector Geffner. Heuristics, Planning and Cognition.

[3] Tran Sao Son and Chiaci Sakama. An Introduction to Action Language, January 2010.

[4] Minh B. Do and Subbarao Kambhampati. SAPA: A Multi Objective Metric Temporal Planner, 2003.

[5] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Velose, Daniel Weld, and David Wilkins. PDDL- The Planning Domain Definition Language, October 1998.

[6] Patrik Haslum and Hector Geffner. Heuristic Planning with Time and Resourses.

[7] Avrim L. Blum and Merrick L. Furst. Fast Planning Through Planning Graph Analysis, 1997

[8] Alex Coddington, Maria Fox, and Derek Long. Handling Durative Actions in Classical Planning Frameworks, UK.

[9] Mausam and Daniel S. Weld. Planning with Durative Actions in Stochastic Domains, Seattle 2008.

[10] Blai Bonet and Hector Geffner. Planning as Heuristic search, Venezuela.

[11] Hector Geffner. Perspectives on Artificial Intelligence Planning, Spain.

[12] Avrim L. Blum and John C. Langford. Probabilistic Planning in the Graph Plan Framework.

[13] Malte Helmert. The Fast Downward Planning System, Germany 2006.

[14] Jorn Hoffmann and Bernhard Nebel. FF: The Fast- Forward Planning System.

[15] Keith Halsey, Derek Long, and Maria Fox. CRIKEY-A Temporal Planner Looking at the Integration of Scheduling and Planning, Glasgow, UK.

[16] Keith Halsey, Derek Long, and Maria Fox. Multiple Relaxations in Temporal Planning.

[17] Keith Halsey. The Workings of CRIKEY- a Temporal Metric Planner, Glasgow, UK

[18] Bharat Ranjan Kavuluri. Extending Temporal Planning for the Interval Class, India.

[19] Silvia Richter,and Mathias Westphal. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks,Australia, Germany.