Learning and Strengthening Meta-heuristics in Plan Space Planning

A THESIS

submitted by

SHASHANK SHEKHAR

for the award of the degree

of

MASTER OF SCIENCE

(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

December 2016

THESIS CERTIFICATE

This is to certify that the thesis entitled **Learning and Strengthening Meta-heuristics in Plan Space Planning**, submitted by **Shashank Shekhar**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science (by Research)**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Deepak Khemani
Research Guide
Department of CSE
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I express my sincere gratitude to my advisor Prof. Deepak Khemani for all his support, valuable suggestions, and the freedom he provided during this whole program. He is a great supervisor and a great person to work with. Since beginning his advice on my research carrier is just spot-on. He always kept his doors open for me to discuss even the small technical details I was struggling with. This contributed a lot in the quality of this thesis and of course its completion as well.

Further I would like to thank the General Test Committee (GTC) members (Prof. C. Chandra Sekhar, Dr. V. Krishna Nandivada, and Dr. Rahul Marathe) of this thesis. You provided very useful hints for improving this thesis manuscript, which I have gladly embraced. I convey my special thank to Prof. Narayanaswamy N. S. for all his support during these years.

A special thank goes to Dr. HLS Younes and Prof. Malte Helmert for being so kind in making their softwares (the automated planners; Versatile Heuristic Partial Order Planner (VHPOP), and Fast Downward (FD) respectively) available. I would also like to thank Dr. Nir Lipovetzky for his valuable advices.

I convey my special thank to Sai Ramesh, Stanley Simoes, and Baskaran Sankaranarayanan, from AIDB Lab for their unconditional support and helpful discussions in all aspects of my research. Their willingness to help me in sorting out the technical and non-technical difficulties, pushed me in the right direction was extraordinary. My most profound thank goes to the peers in AIDB and Dept. of CSE, with whom I enjoyed this lush green campus of IIT Madras. I will always miss the moments I have spent here with one or the other. You people were great to me for all your support. All of you are good researches and amazing human beings.

I would like to thank IIT Madras for providing the funds for attending a couple of conferences. This financial help is gratefully acknowledged. Thanks again goes to IIT Madras for having such a beautiful campus. You simply lose all your anger when you enter inside the campus. I cannot resist myself mentioning that I completed 3700 miles of running in last 3 years at this campus.

I would like to thank my parents and my elder brother for their constant support over the years, giving me all the freedom, and showing unconditional faith in my abilities. I could not have done this without the help and support from such great human beings.

“A man who has never gone to school may steal from a freight car; but if he has a university education, he may steal the whole railroad."

Theodore Roosevelt

“Most people are more concerned with doing things right than with doing the right things. The secret is to focus on doing the right things right."

Peter Drucker

ABSTRACT

KEYWORDS: Domain Independent Planning, Partial Order Causal Link Planning, Heuristic Search, Planning and Learning, Inadmissible Heuristics, Portfolios of Heuristics, Planning Graph, and Artificial Neural Network (ANN).

Planning is the reasoning side of acting. It is a model based approach to autonomous behavior as per the definition given by Hector Geffner. Planning is an explicit deliberation process of selecting and organizing actions by anticipating their possible outcomes. In Automated planning, we study this explicit deliberation process computationally.

The domain independent aspect of automated planning is a well studied field in Artificial Intelligence (AI). In recent years, heuristic search in Domain Independent Planning (DIP) has been widely studied. It has been one of attractive approaches used by the planning community in last ten years. The heuristic functions employed in DIP are domain independent too and do not consider domain specific information. A search algorithm used in DIP requires an informed heuristic function that could guide the search towards the goal while solving a planning problem. In the modern era of heuristic search, we are more concerned about designing domain independent heuristic functions. On the other hand, a domain specific heuristic is more informed during search for the problems belonging to that domain. The usage of domain independent heuristic functions often throws up experimental evidence that different heuristic functions perform well in different domains. Literature reveals that the informativeness of a domain independent heuristic function varies due to the varying nature of planning problems. This nature is often characterized by the degree of interaction between subgoals and actions. For a heuristic function, sometimes it is hard to capture the real nature of a planning problem.

Due to the varying nature of planning problems, often, a heuristic function cannot be equally effective in all planning domains. For improving the effectiveness of a search algorithm, the literature says that sometimes it is good to enhance the informativeness

of the heuristic function employed in the algorithm, using machine learning approaches. One possible approach could be learning from multiple, possibly inadmissible, heuristics for planning. During learning, we consider existing heuristic functions as features for learning. We apply several machine learning approaches that provide these heuristics with some individual weighted coefficients. These individual weighted coefficients can be different for different planning domains. The learning for planning literature shows that combining multiple heuristics, often, provides a more informed heuristic (also known as meta-heuristic) as compared to the individual feature heuristics. While another approach is to improve the informativeness of a heuristic function by capturing the single-step-error associated with the heuristic. A similar strategy is used by Thayer *et al.* based on TD learning, for learning inadmissible heuristics using an unsupervised learning approach in state-space planning.

In this thesis, we come up with effective ways of combining multiple heuristics together for Partial Order Causal Link (POCL) planning. We apply several machine learning techniques for coming up with domain specific combinations of existing heuristics. For the first time in the POCL framework, we use an adapted form of Temporal Difference (TD) learning that is proposed by Richard S. Sutton (1988), for enhancing the informativeness of heuristics on the fly.

We divide the discussion below into two major parts. In the first part of this thesis, we capture an approach of learning the characteristics of different domains in a supervised manner by employing a feed forward Artificial Neural Network (ANN) and its variants. We employ ANNs to combine different, possibly inadmissible, heuristic functions which learn domain specific combinations but the approach is domain independent. We also extend the heuristic functions derived from state-space based planning to POCL planning. Here our objective is to allow a planner to learn parameters over time in a supervised manner, to combine multiple heuristic in a given planning domain.

We introduce another approach for learning domain specific configurations of existing heuristics in a supervised manner. The proposed approach is domain independent and fully automated. We again focus on the POCL framework and endeavor to enhance the performance of a POCL planner by learning from existing non-temporal POCL heuristics in a supervised manner. The focus of this approach is to come across a domain specific combination of a meta-heuristic which can speed up the planning process.

In the second part, we discuss a promising approach that monitors and reduces the error associated with a given POCL heuristic function on the fly. This instance specific but domain independent approach is employed in the search algorithm when the POCL planner solves a particular problem. Here the goal is to allow a POCL planner to roughly estimate the effective average-step-error during search. The average-step-error is calculated using summation of all the single-step-error observed from the initial to current stage of the planning process. In the process, the planner tries to have more accurate estimates in the subsequent stages. This online heuristic tuning approach is a general purpose approach, and can be used to enhance the informativeness of any heuristic function. Of course, using this approach, an improvement cannot be guaranteed always as it depends on the nature of the heuristic. We also employ this approach to perform online tuning to minimize the error associated with the predictive models learned using our previous approach, thus enhancing their informativeness further as well.

To evaluate these approaches that are briefly introduced in the last paragraph, we select various planning benchmark domains like Logistics, Rovers, and Gripper, from different International Planning Competitions (IPCs). We perform separate experimentations for our approaches and compare them on standard planning benchmarks. The experimental evaluations demonstrate that our approaches are competitive with the state of the art planners and heuristics. However, they often perform better.

Contents

List of Tables

List of Figures

ABBREVIATIONS

AL	Active Learning
ANN	Artificial Neural Network
BSSP	Backward State-Space Planning
CEA	Context Enhanced Additive
CL	Causal Link
DIP	Domain Independent Planning
FD	Fast Downward
FDSS	Fast Downward Stone Soup
FF	Fast Forward
FSSP	Forward State-Space Planning
ILP	Integer Linear Programming
IPC	International Planning Competition
LAMA	Landmark
LMS	Least Mean Squared
LM-Count	Landmark Count
LM-Cut	Landmark Cut
LR	Linear Regression
MAP	Multi-Agent Planning
MC	Most Cost
MC-Loc	Most Cost Local
MLP	Multi-Layer Perceptron
MW	Most Work
MW-Loc	Most Work Local
OC	Open Condition
PDDL	Planning Domain Definition Language
PE-ANN	Penalty Enhanced Artificial Neural Network
POCL	Partial Order Causal Link
POP	Partial Order Planning

RegPOCL	Regression Partial Order causal Link
RPG	Relaxed Planning Graph
STN	Simple Temporal Network
STRIPS	Stanford Research Institute Problem Solver
TD	Temporal Difference
VHPOP	Versatile Heuristic Partial Order Planner

Chapter 1

INTRODUCTION

Planning is the reasoning side of acting (Ghallab *et al.*, 2004). It is an explicit deliberation process of selecting and organizing actions by anticipating their possible outcomes. In automated planning, we study this explicit deliberation process computationally. There are many practical and theoretical motivations for automated planning like designing a processing tool which can enable us to access planning resources efficiently. A combined theoretical and practical motivation for automated planning involves building an intelligent and fully autonomous system. Automated planning is a well studied field of Artificial Intelligence (AI). AI captures the intelligent aspect of computation where automated planning definitely plays a key role. Automated planning deals with solving problems by producing a valid sequence of actions which can take an agent from the given initial state to a desired goal state.

In general, planning is intractable. It is known to be PSPACE-hard, where the space and time complexities of the existence of a plan for a given planning problem are EXPSPACE-complete and NEXPTIME-complete respectively (Ghallab *et al.*, 2004; Bylander, 1994). Even a simple planning problem with propositional state variables is PSPACE-complete. In this thesis, we discuss classical planning where all actions are deterministic and only one agent is involved. An action is deterministic if a planner is fully aware of the effects of that action. The action is instantaneous in nature which means that it produces the effects of that action immediately after the planner executes it. Here, the whole state space is known to the planner where only the planner is allowed to make any changes in the state-space.

An important sub-area of automated planning is known as Domain Independent Planning (DIP) (Wilkins, 1984). Hector Geffner (2010) says that planning is a model based approach to autonomous behavior (Geffner, 2010). In DIP, we are interested in building an autonomous system that can solve a given planning problem. In other words, it is a task of finding a sequence of actions that leads the autonomous system from the given initial state to desired goal state, if the given problem is solvable. The autonomous system avoids using the domain specific knowledge in the process of finding a valid sequence of actions for the given problem. The system is also required to take independent decisions during the planning process (Bonet and Geffner, 2000).

To reduce the total effort required for a planner to solve a given planning problem, the study of certain properties and useful structures of the problem is very important. A tractable problem would become unsolvable if the planner ignores important properties and structures. As a result, the planner may go on a long tour in the search space as the space-space grows exponentially. Over the last fifteen years, planning as heuristic search has been an important approach for automated planning. In general, a heuristic search approach that is competent in DIP, uses domain independent heuristics to guide the search. A heuristic is domain independent if it is designed in a way such that it uses domain specific information (representation specific details), but overall its design does not change for different domains. The expectation is that its search guidance should be effective in each planning domain. Also, if it is rigorously designed, it should be able to exploit domain specific knowledge in domain independent manner. Such heuristics reduce the effort of a domain independent planner. The literature reveals that a given heuristic estimate is not informed in all planning domains as its informativeness varies over planning domains due to the nature of the domains. The set of almost all well informed heuristic functions from the literature which have been employed in planning as heuristic search, has been based on the tractable set of planning (Bonet and Geffner, 2001; Hoffmann, 2003; Helmert, 2006). The problem is that this known tractable set of planning is still very small. Almost all informed heuristics employed in the literature are domain independent in nature. They are designed in a way that they can extract domain specific structures and properties in domain independent manner. Perhaps, we all know that there is no universal heuristic estimate which is effective for heuristic search in all planning domains.

Partial Order Causal Link (POCL) planning is one form of performing domain independent classical planning for solving planning problems. The POCL framework dissociates the task of finding actions that constitute a plan and the placement of those actions in the plan. The search algorithm operates in a space of plans, starting with a null plan that represents all possible partial plans. Actions are added to the partial plan to achieve some objective, characterized by resolving flaws in the partial plan. In this thesis, by following Nguyen and Kambhampati (2001); Younes and Simmons (2002, 2003), we work with fully grounded actions *in lieu of* partially instantiated operators, trading delayed commitment for speed.

The POCL framework is one of the well studied areas of automated planning. POCL planning is also called as Partial Order Planning (POP). In general, the abbreviations POCL and POP are the same, and are used often interchangeably in the literature. The POCL framework utilizes the least commitment strategy during the planning process. The strategy avoids committing to any unnecessary constraints during the planning process which makes the framework different from almost all approaches employed in state-space planning. A POCL planning algorithm is usually domain independent which strictly uses delayed commitment strategy. An introduction to the delayed commitment planning (the POCL framework) can be found in (Weld, 1994). Our main focus in this thesis is to accelerate a POCL planner over different benchmarks. There have been many approaches proposed to make a POCL planner efficient in the past but these approaches are not competitive enough with many recent state-space planners. In the last decade, the POCL was a very attractive framework because of the plan flexibility it provides during the execution phase. However, current state-space planners are very efficient, generate consistent states fast, and use powerful state-based heuristics. But they often commit too early to ordering of actions, giving up on flexibility in the plans.

The state-of-the-art non-temporal POCL heuristics used in the framework are not competitive with these state-space heuristics, in terms of the plan quality and search time. The work done by the planning community researchers suggests that sometimes machine learning approaches have done well in state-space planning (Thayer *et al.*, 2011; Samadi *et al.*, 2008; Arfaee *et al.*, 2011). Instead of designing a new informed heuristic, these approaches can be adapted and employed to combine multiple POCL heuristics as well. However, the literature shows that often the combination of heuristics become more informed as compared to the individual ones (Röger and Helmert, 2010). To enhance the informativeness of state of the art POCL heuristics, we adapt previous successful machine learning approaches employed in the literature. We exploit planning domains' structures and properties using learning approaches, which enhances the effectiveness of the POCL planners. Although the approaches employed exploit domain specific knowledge but their designs make them fully domain independent.

Keeping the target of enhancing the efficiency of a POCL planner in mind, we come up with different strategies, introduced in the coming sections, which enhance the performance of our POCL planners. These planners are completely built upon Versatile Heuristic Partial Order Planner (VHPOP) (Younes and Simmons, 2003). This thesis gives a detailed overview of heuristic search enhancements in the POCL framework. The enhancements are done by generating, selecting, and combining multiple heuristics using different machine learning approaches. However, these approaches have been employed in the literature of automated planning. The community has seen the effectiveness of learning for planning approaches which drive us to use them in the POCL framework as well. We have observed the efficiency of our approaches on various International Planning Competition (IPC) benchmark domains, where Stanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971) style non-temporal planning domains and problems have been considered. The domains have been considered according to state-of-the-heuristics. It means, we consider most of those domains where latest heuristics have performed well, and we try to improve on that. In the next section, we quickly introduce our research objectives.

1.1 Research Objectives

The POCL framework alternates between two choices. First, a POCL planner decides which partial plan to select from the set of partial plans for further refinement. While in the second choice, the planner decides which flaw to be resolved first from the partial plan selected by the first choice, and with which resolver. Each resolver generates a new partial plan that gets stored in the set of partial plans. The planner needs good heuristics for both the choices during the planning process because a wrong selection of a resolver may end up searching more number of nodes in the plan-space.

It is very important to have an informed heuristic function during the planning process to evaluate each partial plan from the set of partial plans during planning. An informed heuristic enables the planner to select the most *suitable* (discussed later) partial plan from the set for further refinement. In this process, the heuristic estimates the total effort required to *refine* (discussed in the next chapter) each partial plan. The partial plan that requires the least effort will be the most suitable one to be picked next. Often, a good choice of partial plan leads the planner to minimal number of refinements with searching minimal number of nodes. Perhaps, this also generates shorter solution plans. The POCL literature specifies that the current POCL heuristics are not effective in all planning domains. As we said earlier, the informativeness of a heuristic estimate varies over the domains due to the nature of the heuristic. A heuristic is not usually very informed if it does not capture essential domain specific knowledge.

For the general purpose search algorithms employed in DIP, it is hard to find out which heuristic estimate will be informed in a given domain. However, machine learning approaches are quite successful in such scenarios where picking the most suitable heuristic for a particular domain is crucial. We discuss some of the successful recent approaches used by the planning community in the next chapter. The approaches provide separate priorities to the individual heuristics using regression models. The priorities vary with planning domain because the nature of these domains also vary. Machine learning techniques have been employed in the past to capture the actual nature of planning problems. Often, the regression models are learned over solved small sized and simple planning problems. Solving such problems is not a big overhead to the employed planning algorithms. The learned regression models over these small problems are also referred as meta-heuristics. In the literature, sometimes, they are also called as hyper-heuristics as sometime we learn hyper-parameters to combine multiple heuristics. In this thesis, we use these terms interchangeably.

In the next section, we briefly discuss the abstract of our approaches employed. They focus on designing good informed heuristic functions using a few well known machine learning techniques. These new heuristics are employed for selecting the most suitable partial plan from the set of partial plans, which is at the first choice point of the planning process. On the other hand, for selecting the most demanding flaw from that selected partial plan, we use existing informed heuristics from the literature of POCL planning (Younes and Simmons, 2003).

1.2 Contributions of the Thesis

We put forward some effective ways for generating new heuristics that are applicable in the POCL framework. In a situation where a POCL planner selects one from many options, the planner requires a strong heuristic estimate to enhance the plan quality which reduces the required search effort. In this thesis, we propose a couple of machine learning approaches to come up with more informed heuristic estimates (the meta-heuristics). The approaches discuss either about learning meta-heuristics by tuning parameters that are used to combine multiple heuristics together, or making an existing heuristic more informed. As stated earlier, we use the learned regression models using the following approaches at the first choice point during planning.

1.2.1 Learning from Multiple Existing Heuristics

We start with a discussion about the effectiveness of learning in planning. We combine multiple inadmissible heuristic functions in a supervised manner such that the combination gives a heuristic value that is close to the actual targets. Sometimes, it is hard to capture the complex nature of planning problems. To handle this complex nature, in some cases Artificial Neural Networks (ANNs) have been quite useful in capturing the useful structures of planning problems (Samadi *et al.*, 2008). Considering the success of ANNs, we deploy them for the learning combination heuristics. To enhance the informativeness of the learned meta-heuristic in the process we propose a cost function that is often helpful in getting heuristic estimates close to but smaller than the actual targets. Such combinations are deployed as heuristics in different search algorithms. In the literature, an ANN that uses a cost function like ours is called as Penalty-Enhanced ANN or in short (PE-ANN). We discuss PE-ANN in detail in the coming chapters. Currently, we have tested PE-ANN in a couple of planning domains. However, we often get the estimates close and smaller than the actual targets. The current experiments do not demonstrate that this approach help a POCL planner in achieving good plan quality with minimal search effort.

Apart from the above approach, there have been several learning approaches employed in planning literature which have shown good results. These approaches mostly cover supervised learning, unsupervised learning, active learning, and Bootstrap learning. Following the work done in the past (Arfaee *et al.*, 2011; Thayer *et al.*, 2011; Samadi *et al.*, 2008; Virseda *et al.*, 2013), in this approach, we again use supervised learning to learn meta-heuristic functions that are combinations of multiple heuristics. We use several regression techniques for learning. Our motivation for employing this approach is to learn meta-heuristics that can speed up the planning process, unlike the approach discussed in the last paragraph where we focus on learning admissible meta-heuristics.

We train regression models in a supervised manner for which we need training data sets. We propose a domain independent algorithm that generates datasets for each domain and learns regression models for our POCL planner. We call this planner **RegPOCL** that is based on VHPOP. We consider existing heuristics from POCL literature as the features for learning meta-heuristics. A few of them are quite weak due to the nature of their design. A meta-heuristic is represented as follows: for an example, suppose h_{hh} is the learned meta-heuristic, then h_{hh} = 3×h_1 + 0.5×h_1×h_2 - 4×h_3, where h_1 to h_3 are the three feature heuristics. Here, the coefficient of each h_i is a learned parameter (individual weight, w_i). Formally, it is,

$$h_{hh} = \sum_{i=1}^{i=k} w_i \times h_i$$

The approach is completely domain independent but RegPOCL learns relevant domain specific properties and structures.

In short, RegPOCL learns different meta-heuristics using different regression techniques from machine learning literature. This improves the effectiveness of heuristic search in the POCL framework.

1.2.2 Heuristics Tuning Using TD Learning

The study of heuristic search in classical state-space planning has received significant interest in the past. Computing different heuristic functions for different states or problems or domains is not suitable in DIP. There is no efficient universal domain independent heuristic estimate that is effective in all planning domains (Younes and Simmons, 2003).

It has been empirically observed that heuristics often commit some error in the estimation at each stage of DIP. Here, the *error* term considers the difference between the previous and current estimates. Therefore, in this approach, we examine the single-step-error associated with a given heuristic function on the fly during search. The approach monitors and reduces the step error, that is performed by the algorithm while it solves a problem. The approach of monitoring the single-step-error is based on Temporal Difference (TD) learning. It learns instance specific details in a domain independent manner. It also captures the current running average-step-error associated with a given heuristic function and minimizes the error estimate during the next prediction.

The basic idea behind this error monitoring approach is to avoid a heuristic calculating similar estimates for different planning problems. The approach starts with a heuristic, h. After each refinement, the approach corrects the error (ϵ) associated with h, *e.g.* from h to h-ϵ. Therefore, the approach avoids h arriving at similar estimates of distance-to-go for solving two different problems because the factor ϵ varies with natures of the problems. The fact is that, two problems may belong to one planning domain, and carry different nature (the orientations of the objects involved in that problem). We tested this technique on different IPC benchmarks for the POCL framework. It often leads to high quality plans and the search effort required to find the plans reduces too.

If we consider the supervised approaches employed for learning from multiple existing heuristics, due to the negative effects of the generalizing nature of supervised learning, an error tuning approach is quite useful for the learned meta-heuristics. Supervised learning generalizes the learned knowledge in the training phase to the testing phase. It learns from the training datasets generated by solving simple problems and considers that large problems carry similar natures, of course, that is not always the case for planning problems. Due to this generalization, the learned meta-heuristics may behave like a less informed heuristics for large sized problems. For sorting this issue out, we use the error tuning approach based on Temporal Difference (TD) learning. This two-fold approach has been evaluated and we find that RegPOCL is very efficient on benchmarks. Using this approach, the search often leads to high quality solution plans with less search effort.

1.3 Outline of the Thesis

We discussed automated planning, DIP, the POCL framework, and heuristic search in planning along with some machine learning strategies employed in the literature. We give motivation for the usage of machine learning approaches in POCL planning as these approaches are expected to capture the essential features of planning problems. Next, we give a brief outline of this thesis.

- In Chapter 2, we build a complete background and give motivating examples that drive us towards the approaches used. We talk about the importance of POCL planning and why this is different from state-space based planning. There are some negative aspects associated with the POCL framework too like current heuristics are not powerful, also sometimes the nodes in the search space are inconsistent due to looping of dependencies in a partial plan (node in the plan space). We also give examples that support our decision for improving the POCL framework. We end up discussing some recent advancements covered in the planning literature. This portion of the thesis is only distantly related to our employed approaches. Perhaps, it covers recent trends in the field heuristic search in automated planning. We confine our discussion to the classical and temporal planning perspectives.

- In Chapter 3, by following the current interests of researchers, we adapt and use some state-space based heuristics in the POCL framework with small modifications. The chapter also demonstrates some preliminary evaluations as the work is only partially done. We also adapt the PE-ANN (Samadi *et al.*, 2008) and empirically show that the modifications used in the existing approach, often positively affect the learning processes (Bishop, 2006). We give complete weight update rules using Gradient Descent algorithm for the new error function used in PE-ANN. In another approach, we demonstrate some adaptations of regression techniques like Linear Regression and Multi-Layer Perceptron, which have shown good results. Like previous approach, they are also helpful in learning combinations of domain specific meta-heuristics. We also show that often the performance of the combination heuristics is better than the individual heuristic estimates including the state of the art.

- This is followed by a discussion of an adapted approach based on Temporal Difference (TD) learning in Chapter 4. If we follow the literature of state-space based planning, a heuristic function commits some error at each step during search. Monitoring such error closely influences the performance of search algorithms resulting in the POCL planner becomes more efficient. The chapter also contains experimental results for the approach employed for monitoring and error correcting. In this chapter, we also employ TD learning approach to the learned models obtained using multiple existing heuristics. The employment of TD approach to the learned models, enhances their informativeness thus increasing the efficiency of RegPOCL. We employ TD learning because sometimes the poor generalizations takes place in the training phase of supervised learning which affects the performance of RegPOCL.

- Following this, in Chapter 5, we put a brief summery of this thesis in a nutshell where we also discuss some future avenues for further research.

Chapter 2

CLASSICAL PLANNING AND LEARNING

In this chapter, we give a detailed overview of automated planning. We describe preliminaries to classical planning. This is required to fully understand the content of the thesis. We discuss different kinds of search algorithms along with the heuristics used in the classical planning literature. We also give brief descriptions of the major, general purpose, learning approaches employed in automated planning by planning community researchers.

2.1 Classical Planning

Classical planning is one of the important fields of AI. There has been significant improvement in classical planning approaches in last fifteen years. Despite this fact, still, there are many problems that remain intractable for the state-of-the-art planners. It is a fact that solving planning problems, is among the hard problems in AI to solve (Bylander, 1994; Erol *et al.*, 1995). There are three different ways to represent classical planning problems. These representations are, (*i*) set-theoretic representation, (*ii*) classical representation, and (*iii*) state-variable representation. Each of them has equal expressiveness, and one of these representations can be represented as either of the other representations (Ghallab *et al.*, 2004). In this thesis, we describe in detail, the classical representation as the benchmark domains used for the evaluations by us solely follow the classical representation. However, we also use some extended representations of the classical representation. We do not describe those extensions in this chapter, however we describe them wherever it is required. In the next section, we formally describe the classical representation of classical planning problems.

2.1.1 Classical Representation

Major part of the text used in this section has been taken directly from the classical planning literature (Ghallab *et al.*, 2004; Chrpa, 2009), and rewritten in a different language. The notations used to represent the classical representation have been derived from first-order logic. Here, each state is represented as a set of logical atoms. The tautology and falsity of these atoms depend on some interpretation. Here, actions are shown as planning operators that change the truth values of these atoms.

States

To develop languages for classical planning like STRIPS and Planning Domain Description Language (PDDL), we start with first-order language, $\mathcal{L}$. There are many constant and predicate symbols but no function symbols in $\mathcal{L}$. In general, to represent predicates and constants we use alphanumeric strings, while for variables we use single character, possibly with subscripts like l and m_{12}. We represent a *state* as a set of grounded atoms of $\mathcal{L}$. The number of all possible states in the whole search space, $\mathcal{S}$, is finite as there are no function symbols used in $\mathcal{L}$. An atom *m* holds true in a state, *s*, if and only if $p \in s$. Another state s_1 satisfies *s* that is denoted as $s_1 \models s$, if for a possible substitution, σ, every positive literal of $\sigma(s)$ is in s_1 and no negated literals of $\sigma(s)$ is in s_1. We follow the closed-world assumption that says an atom holds in a state if and only if it is explicitly specified in that state. The truth value of an atom may vary from state to state as it depends on the interpretation. For example, an atom *at(robot1, location1)* is true in a state, *s*, if *robot1* is at *location1*, otherwise it is false.

Operators and Actions

In classical representation, a fully instantiated planning operator is called as an action. A planning operator is a triple of the form, *o* = ⟨name(*o*), precond(*o*), eff(*o*)⟩, where name(*o*) is the name of the operator, and precond(*o*) and eff(*o*) are respectively the preconditions and the effects of the operator. For example, a fully instantiated operator (an action) can be represented as:

```
load ($k_1$, $loc_1$, $c_1$, $rob_1$)
    ;; crane $k_1$ at location $loc_1$ loads container $c_1$ onto robot $rob_1$
    precond: belong($k_1$, $loc_1$), holding($k_1$, $c_1$), at($rob_1$, $loc_1$)
    eff: empty($k_1$), ¬holding($k_1$, $c_1$), loaded($rob_1$, $c_1$)
```

For this action the positive effects (eff$^+$) are empty(k_1) and loaded(rob_1, c_1), and the only negative effect (eff$^-$) is holding(k_1, c_1). In a similar way, we can figure out its positive preconditions (precond$^+$) and negative preconditions (precond$^-$).

As per the definition given in (Ghallab *et al.*, 2004), an action is any grounded instance of a planning operator. If *a* is an action and *s* is a state such that the set of positive preconditions, precond$^+$(*a*), is a subset of *s*, and the set of negated preconditions, precond$^-$(*a*), is disjoint to *s*, then, *a* is an applicable action to the state *s*. Eq. 2.1 shows the result of applying *a* in s which discovers a new state, s_1, in the search space using a state transition operator, γ (also known as a progression operator):

$$s_1 \leftarrow \gamma(s,a) = (s - \text{eff}^-(a)) \cup \text{eff}^+(a) \tag{2.1}$$

This can easily be computed using set operations.

Domains, Problems, and Plans

A classical planning domain in $\mathcal{L}$ is a restricted state-transition system, $\sum$ = (S, A, γ), such that, S is subset of $2^{\{\text{all grounded atoms of } \mathcal{L}\}}$, *A* is a set of all possible fully grounded instantiations of the operators of that domain, γ is defined as in Eq. 2.1. Here, *S* is closed under γ, which means the state, s_1 *s.t.* $s_1 \leftarrow \gamma(s,a)$, then $s_1 \in S$. This is true for all applicable actions in *s*, and for each state belongs to *S*.

A classical planning problem is represented as a triple $\mathcal{P}$ = ($\sum$, s_0, *g*), where s_0 is the starting state, could be any state in *S*, and *g* is the goal state. We also consider a term S_g that is a state in *S* such that S_g satisfies *g*. Apart from the notion of *applicable* actions, we have the notion of *relevant* actions as well when we regress from a given state. This is again performed using set operations using the regression operator, γ^{-1}. The general usage of γ^{-1} says that we regress from the goal state and try to discover the starting state in *S*. An action, *a*, is relevant for a goal state *g* if, (*i*) eff(*a*) contributes to *g* ($g \cap \text{eff}(a) \neq \phi$), and (*ii*) eff(*a*) does not have any conflict with *g* ($g^+ \cap \text{eff}^-(a) = \phi$, and $g^- \cap \text{eff}^+(a) = \phi$). For a relevant action *a* for *g*, the regression operator (γ^{-1}) is defined as,

$$s \leftarrow \gamma^{-1}(g,a) = (g - \text{eff}(a)) \cup \text{precond}(a) \tag{2.2}$$

Here, *s* is the resulting state after the execution of γ^{-1}. If we reach to the initial state using γ^{-1} further in *s*, *a* will be the last action of a valid solution *plan* found.

A *plan*, π, is a valid solution plan for a given planning problem $\mathcal{P}$, if $\gamma(s_0, \pi)$ satisfies the goal state, g. A plan is redundant if a consistent sub-sequence of the plan is also a solution plan. If π is not redundant then the plan length, $|\pi|$ - the number of actions in π, is either minimal or the shortest.

Next, we discuss basics of some general purpose search algorithms used in classical planning. We just saw the classical (state-space) representation of classical planning. In general, state-space approaches solve planning problems by employing forward search algorithm that uses γ, and backward search algorithm that uses γ^{-1}. Forward state-space planning employs forward search algorithm, and backward state-space planning employs backward search algorithm. In the next subsections, we discuss these two approaches in detail.

2.1.2 Forward State-Space Planning

The state-space search algorithms are one of the simplest algorithms used in automated planning. Forward State-Space Planning (FSSP) applies forward search algorithm to reach the goal state from the initial state. The algorithm that uses progression operator, selects applicable actions and returns a solution plan if the algorithm finds a consistent sequence of actions that can transform from initial state to the goal state. A sequence of actions is consistent, if the search algorithm Forward-Search($\sum$, s_0, g) solves the given planning problem, $\mathcal{P}$ = ($\sum$, s_0, g). The algorithm returns a failure if it does not solve $\mathcal{P}$. The forward search algorithm is sound and complete, which respectively mean the solution plan found will be a real plan for the given problem (soundness), and the algorithm always finds one such plan if there exists at least one solution plan (completeness).

To solve a planning problem $\mathcal{P}$, a forward state-space planner expects domain definition ($\sum$) to solve $\mathcal{P}$. For example, the algorithm Forward-Search($\sum$, s_0, g) has been given the domain description, $\sum$. As defined earlier, this knowledge is in the form of operators, constants, and predicates. Given this, the search algorithm uses the progression operator (γ) to move to the next state in the state-space. For example, suppose an action a_1 which is applicable in the initial state then the next state (s_1) can be reached using the operator γ, which is represented as $s_1 \leftarrow \gamma(s_0, a_1)$. The algorithm progresses over the states in *S*, like $s_2 \leftarrow \gamma(s_1, a_2)$, $s_3 \leftarrow \gamma(s_2, a_3)$, and so forth. In a case, when $\mathcal{P}$ is solved after applying *k* actions, starting from s_0 in *S*, the search planner returns a valid sequence of these actions. As defined earlier, this valid sequence is a solution plan ($\langle a_1, a_2, ..., a_k \rangle$) for $\mathcal{P}$. This can also be represented in the form given below,

$$g \leftarrow \gamma(s_0, \langle a_1, a_2, ..., a_k \rangle)$$

The planner returns failure when the the goal state is not found after searching the entire search-space. The basic form of FSSP is highly computationally intensive in nature as the algorithm used in it has high branching factor. To make FSSP effective, the planning researchers have used informed heuristics to reduce the number of backtracks over different choices (Ghallab *et al.*, 2004). An appropriate selection of a heuristic estimator in FSSP often achieves a solution plan faster for a given planning problem as compared to when the planner has no sense of the direction toward the goal state.

2.1.3 Backward State-Space Planning

In Backward State-Space Planning (BSSP), the planner starts searching from the goal instead of initial state, in the backward direction for finding a solution plan for a given planning problem, $\mathcal{P}$ = ($\sum$, s_0, *g*). The Backward Search algorithm used in BSSP regresses (using a regression operator, γ^{-1}) from the goal state. The algorithm stops when it reaches a state s such that $s \subseteq s_0$. In this case the backward state-space planner either finds a solution plan or it returns failure (Ghallab *et al.*, 2004). At each stage of planning, the planner regresses to a new goal state that gets generated as shown as: $g \leftarrow \gamma^{-1}(g, a_g)$, where the action a_g is a relevant action for the goal state (g). The planner searches in the backward direction and generates a new partial plan that is $\pi \leftarrow a_g.\pi$. The partial plan π becomes a complete plan if the algorithm reaches a state s, such that $s \subseteq s_0$. Like FSSP, BSSP is also sound and complete due to its nature (Ghallab *et al.*, 2004). There are many issues associated with BSSP like the backward search algorithm generates lot of spurious states using γ^{-1}. However, often, the branching factor of the backward algorithm is small as compared to the branching factor of the forward search algorithm. For further details of BSSP and FSSP, we refer readers to (Ghallab *et al.*, 2004)

In the next sections, we introduce the POCL planning algorithm in brief and also compare the plan-space and state-space planning aspects.

2.1.4 Partial Order Causal Link Planning

In Chapter 1, in the brief introduction of POCL planning, we said that the POCL framework dissociates the task of finding actions that constitute a plan and the placement of those actions in the plan. The search algorithm operates in a space of plans, starting with the null plan that represents all possible plans.

A partial plan π is a 4-tuple (A, O, L, B), where A is a set of actions, O is a set of ordering constraints between actions, L is a set of causal links among actions and B is a set of binding constraints. If we consider only grounded actions then the set B becomes empty. A causal link between two actions a_i and a_j is represented as $a_i \xrightarrow{p} a_j$ states that the action a_i produces a proposition p which is consumed, as a precondition, by the action a_j. An ordering constraint between actions $(a_i \prec a_j)$ (in the set O) signifies that a_i is scheduled before a_j in the plan. An open condition *(OC)*, $\xrightarrow{p} a_j$ in POP is a proposition p that is a precondition for the action a_j and for which the supporting causal link is absent. An unsafe link *(UL)* (also called a threat) is a causal link, say $a_i \xrightarrow{p} a_j$, that can potentially be broken by an action a_k if it were to be scheduled in between a_i and a_j and a negative effect of a_k unifies with p. The set F of flaws is the set of all such *OC* and *UL* in a partial plan. That is, $F(\pi) = OC(\pi) \cup UL(\pi)$. Here, F is not an integral part of π.

A partial plan is a solution plan when the set F of flaws is empty. We adapt the UCPOP (Penberthy and Weld, 1992) algorithm which starts with a null partial plan with only open conditions and no threat. A null partial plan has two dummy actions a_0 and a_∞ along with a universal ordering constraint $(a_0 \prec a_\infty)$ between these two actions. For a given planning problem a_0 produces the propositions in the start state, and a_∞ has the goal propositions as its preconditions. The plan refinement procedure involves selection of open conditions, along with a resolver for each open condition. The resulting partial plan may have a threat. Either the provider of the chosen open condition may threaten some existing causal links or the new causal link(s) may be threatened by some other existing action(s). Suppose a causal link, $a_i \xrightarrow{p} a_j$, for which an action a_k is a threat.

This threat can be resolved by: (a) *promotion* - adding an ordering constraint ($a_k \prec a_i$) (b) *demotion* - adding ($a_j \prec a_k$), and (c) *separation* - avoiding a binding ($e.g.$ x = c or y $\neq$ d) that unifies *p* with a negative effect of a_k. However, in case of working with grounded actions we cannot use separation as a resolver.

2.1.5 State-Space Planning *vs* Plan-Space Planning

In this section, we compare the state-space planning and plan-space planning. The descriptions given in the earlier sections help us to point out the following differences.

- The state-space search algorithms perform the search in the space of states while the search is performed in space of partial plans in the case of POCL framework.
- The definitions of solution plans are completely different in these two approaches. POCL planning uses more general plan architecture as compared to simply a sequence of actions in FSSP.
- The state-space approaches only consider the choices of actions while on the other hand the POCL framework selects an action as well as an ordering of that selected action in the partial plan.
- The state-space approaches usually produce plans with total order due to the nature of the search algorithms used, but the POCL framework rarely produces a total order plan.

These contrasts make the POCL framework effective compared to the state-space planning as the former can provide plan flexibility during the plan execution phase. However, there are some disadvantages of the POCL framework as well. The framework is resource intensive, and also comparatively slow as compared the recent state-space planners.

2.2 The POCL Framework: An Example

To explain the general POCL framework, we consider an example below. We take a planning problem to be solved using a POCL planner in Example 1. The example effectively demonstrates that the planner requires good heuristic estimates to come up with good quality solution plans. As stated earlier, the requirement of rigorous heuristics is applicable for both the choices, either for selecting the most demanding partial plan or the most demanding flaw in the selected partial plan, during the planning process.

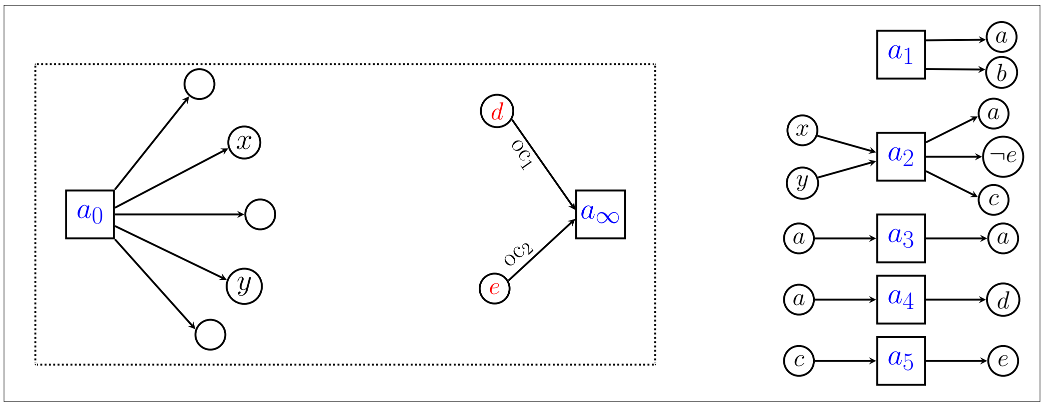


Figure 2.1: The null partial plan for $\mathcal{P}$ from Example 1, with the specified five actions.

Example 1. Consider a planning problem $\mathcal{P}$ with five fully grounded actions a_1, a_2, a_3, a_4, and a_5. The details about the preconditions and effects of these actions are as follows.

- a_1 has no preconditions and its effects are {a,b}.
- The preconditions of a_2 are $\{x, y\}$ and effects are $\{a, c, \neg e\}$.
- a_3 consumes $\{a\}$ and produces $\{a\}$.
- The actions a_4 and a_5 respectively consume $\{a\}$ and $\{c\}$ and produce $\{d\}$ and $\{e\}$.
- The initial state S_0 of $\mathcal{P}$ is {...,*x*,*y*,...} the goal state g is {*d*,*e*}.

Solution: For Example 1, the planner starts with the null partial plan. In the beginning, only the open conditions are present in the this partial plan as shown in Figure 2.1. The current partial plan has two unsupported causal links *d* and *e* which can be resolved by adding the resolvers a_4 and a_5 respectively. The resulting partial plan is shown in Figure 2.2. The figure shows that the planner has three choices to resolve an unsupported causal link a, therefore, three different partial plans can be generated using these three resolvers. The planner can select one resolver from the three in the next step of refinement. This selection depends on the informativeness of the heuristic function used by the planner in this stage of POCL planning. For example, a POCL heuristic estimate, possibly a weaker one, is based on the number of actions present in the partial plan ($|A|$). In Figure 2.2, using this heuristic, selection of a resolver for a will be random (it is the case of tie-breaking). If we select action a_1 the resulting partial plan will be having $\xrightarrow{c} a_5$ as a next unsupported *CL*. This can be resolved by adding a *CL*, $a_2 \xrightarrow{c} a_5$ to the current partial plan. The elements of set, precond(a_2) are available in the set, eff(a_0). This gives a solution plan with plan length 4. Similarly, if the planner selects a_2 as a resolver in place of a_1, in Figure 2.2, then it finds a solution plan of length 3 as a_2 can also provide c to action a_5. In the worst case, if the planner selects a_3 instead of a_1 and a_2, it may end up finding relatively a longer plan. ■

We continue with a further discussion about POCL planning. A POCL planner starts search with a null partial plan that keeps two dummy actions (a_0 and a_∞), and progresses over a search space of partial plans, by adding resolvers to a partial plan to refine it completely. POCL planning separates the task of finding actions and their placements for constituting a solution plan. We know that, a solution plan is a partial plan with no flaws in it. Here each consistent linearization of a partial plan with no flaws is a valid solution plan. For example, in Example 1, Figure 2.2, suppose the planner selects the action a_2 as a possible resolver which can provide a supportive causal link for the open condition c (a precondition of a_5) as well. In the above example, we saw that a_2 was capable of providing its support to both unsupported causal links $\xrightarrow{c} a_5$ and $\xrightarrow{a} a_4$. After adding action a_2, this becomes a partial plan with no flaws in it. The possible linearizations that are also the solution plans are a_2, a_4, and a_5 or a_2, a_5, and a_4.

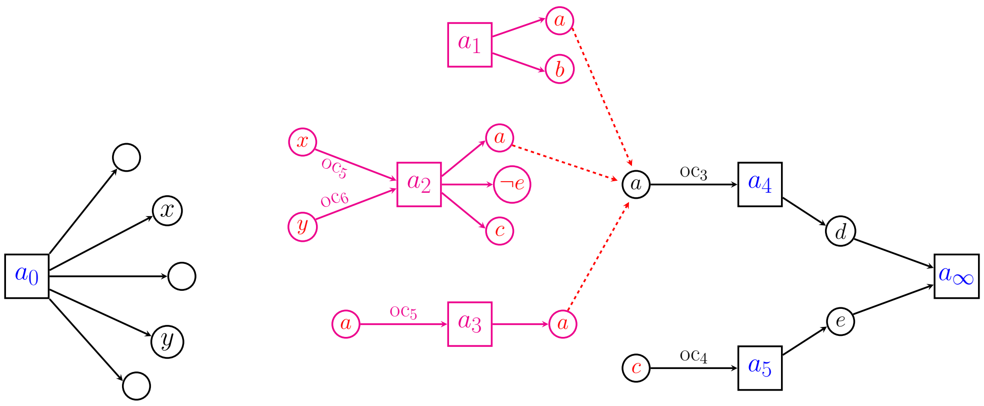


Figure 2.2: The resulting partial plan after adding the two resolvers a_4 and a_5.

2.2.1 Why POCL Framework?

The POCL framework has certain advantages over forward state-space search (FSSS). FSSS has the problem of premature commitment to an ordering between two actions which reduces the plan flexibility. It does so to avoid any mutual interference between actions, though there may not be any. As we discussed, the POCL framework avoids committing unnecessarily to ordering actions. FSSS faces problems while solving instances with deadlines. Deadlines may arise within the interval(s) of one or more durative actions. In general, the actions may produce some delayed effects, and this may have ramifications on deadlines as well, which creates deadlines relative to their starting points (Coles *et al.*, 2010). FSSS also suffers from significant backtracking as it may explore all possible plan permutations in the absence of effective guidance. In contrast to FSSS, the POCL framework has several advantages in temporal planning, specially in planning with complex temporal constraints beyond actions with duration (Benton *et al.*, 2012). These limitations of FSSS motivate us to explore the POCL framework.

An Illustrative Example

Suppose we are required to add four actions [a_1, a_2, a_3, a_4] to a plan, where a_2 is dependent on a_1 and a_4 is dependent on a_3. There is no interference between any two actions apart from the above dependencies. In this case, FSSS gives an ordering or timestamp [0, 1, 2, 3], with a makespan 4, whilst the delayed commitment strategy would give more choices with flexibility in the orderings like [2, 0, 1, 3] and [0, 2, 1, 3] (corresponding to [a_1 (0), a_2 (1), a_3 (2), a_4 (3)]). If parallel execution is allowed, makespan would be 2. If another action a_5, which is dependent on a_3, has to be introduced in the plan then FSSS will allot it a timestamp 4, whereas delayed-commitment strategy could allot it 1. We could consider some more examples which will strongly support our decision of exploring the POCL framework (Younes and Simmons, 2003).

However, if we ignore the absence of the flexibility and action parallelism in FSSS, it is very fast in recovering from a situation that would arise due to committing to some wrong choices during planning. FSSS has the advantage of faster search state generation and powerful state-based heuristics.

2.3 Non-Temporal POCL Heuristics

We deal with solving the STRIPS style planning problems effectively. However, there seems not much has been done in recent years by the community. But researches have investigated the POCL framework in Temporal Planning and Multi-Agent Planning quite frequently. Some specific work has been done in the last decade by Nguyen and Kambhampati (2001) and Younes and Simmons (2003) which handles solving classical planning problems. We can follow up some other work like (Penberthy and Weld, 1992), that is done in the early nineties after Systematic Non-Linear Planning (McAllester and Rosenblitt, 1991). In this thesis, we only discuss some good heuristics designed in RePOP (Nguyen and Kambhampati, 2001) and VHPOP (Younes and Simmons, 2003). To the best of our knowledge, the heuristics discussed in the following subsections as state-of-the-art heuristics.

The heuristic function described below, is based on the reachability analysis of the partial states (hypothetical states) from initial state *I* using the planning graph data structure (Blum and Furst, 1997). We treat all the *OCs* present in a partial plan as making up a state. One has to be careful here how to deal with sets of predicates that cannot be part of a state. Heuristics defined below in the rest of this section, are taken from the POCL literature, and they are inadmissible ones. However, POCL planning is not known for finding optimal plans. We work toward getting more effective plans by improving the informativeness of these heuristics.

2.3.1 Relax Heuristic

One simple approach is to just count the number of open conditions *(OC)* in the partial plan (Schubert and Gerevini, 1995; Joslin and Pollack, 1994).

$$h_{count}(\pi) = |OC|$$

h_{count} is usually an admissible heuristic, but when a set of *OC*s is a subset of initial state, the heuristic becomes inadmissible, and there are a few more cases where it is inadmissible. Various techniques have been proposed to cater to positive as well as negative subgoal interactions (Nguyen and Kambhampati, 2000; McDermott, 1999; Bonet and Geffner, 2001; Nguyen and Kambhampati, 2000; Hoffmann and Nebel, 2001). Nguyen and Kambhampati (2001) address positive subgoal interactions using a serial planning graph for the subgoal reachability analysis. Let *level(p)* be the level in the planning graph when the proposition *p* first appears. Given a set of open goals S, let q be the last opengoal in the set S that appears in the planning graph. Then,

$$level(q) = max_{q_i \in S} level(q_i)$$

The previous state S_{prev} is given as follows after applying regression operator a_q that produces q,

$$S_{prev} = S_{curr} + precond(a_q) - \mathit{eff}(a_q)$$

where S_{curr} is current state, a_q is the action which achieves the proposition q. The total cost of achieving all the open conditions of S_{curr} is,

$$cost(S_{curr}) = cost(a_q) + cost(S_{curr} + precond(a_q) - \mathit{eff}(a_q)) \tag{2.3}$$

then,

$$h_{relax}(\pi) = cost(S_{curr}),$$

where S_{curr} = $\{p \mid p \in OC\}$, and $cost(S_{curr})$ is computed using last recursive relation.

The planning graph data structure (Blum and Furst, 1997) facilitates detection of a proposition q in S_{curr} that has highest estimated cost from initial state *I*. We assert that the elements of $S_{curr} \cup \mathit{eff}^+(a_q)$ should be mutex free where a_q is the provider of q. We hypothesize that this assertion results in a more accurate estimate incorporating the negative interactions between the set S_{curr} and $\mathit{eff}^+(a_q)$. This gives us a new improved estimate as:

$$level(q) = max_{q_i \in \{S \cup \mathit{eff}^+(a_q)\}} level(q_i) \tag{2.4}$$

where $q \in S \cup \mathit{eff}^+(a_q)$, and it is expected that $cost(S_{curr} \cup \mathit{eff}^+(a_q)) \geq cost(S_{curr})$. The modified formula for *level(q)* is a variation of the approach used for ranking partial plans in RePOP. Since the assumption uses relaxed actions the $level(precond(a_q))$ is always strictly less than $level(a_q)$ therefore,

$$cost(a_q) = \begin{cases} 0 & \text{if } a_q \in A; \\ 1 & \text{Otherwise.} \end{cases}$$

where A is the set of actions in the partial plan. We would like to give some thought to this improvement in the near future.

2.3.2 Additive Heuristic

The additive heuristic (Bonet *et al.*, 1997) adds up the steps required by each individual open goal. The assumption of subgoal independence that it makes has worked well in many domains. However in many domains it has a tendency to overestimate the cost. The heuristic value is estimated recursively. Suppose *p* is a proposition and *A(p)* is a set of grounded actions that produce *p*. Then,

$$h_{add}(p) = \begin{cases} 0 & \text{If } p \text{ unifies with an atom in } S_0; \\ min_{a \in A(p)} h_{add}(a) & A(p) \neq \phi; \\ \infty & \text{Otherwise.} \end{cases} \quad (2.5)$$

and with the closed world assumption, the cost of an action is calculated by the formula given below,

$$h_{add}(a) = 1 + h_{add}(precond(a))$$

The $h_{add}(\pi)$ for POP is defined as:

$$h_{add}(\pi) = \sum_{\xrightarrow{p} a_j \in OC(\pi)} h_{add}(p)$$

an action a_p produces a proposition *p* and conditioned by *r*, so $h_{add}(r)$ will be added to the cost of a_p.

2.3.3 Accounting for Positive Interaction Heuristic

Younes and Simmons (2003) address the positive interactions among subgoals while ignoring the negative interactions. This estimation technique is used (as a variant of h_{add} (Haslum and Geffner, 2000)) for ranking the partial plans for the first time by Younes and Simmons (2003), which is defined in Eq. 2.6.

In Eq. 2.6, $h^r_{add}(\pi)$ is the substitute for $h_{add}(\pi)$ as the latter has no provision for actions reuse. The underlying principle of POCL planning is that the resolver already present in the partial plan will used, if it is consistent.

$$h^r_{add}(\pi) = \sum_{\xrightarrow{p} a_j \in OC(\pi)} \begin{cases} 0 & \exists a_k \in A \text{ such that an effect of } a_k \text{ unifies with } p, \text{ and } a_j \prec a_k \notin O; \\ h_{add}(p) & \text{Otherwise.} \end{cases} \quad (2.6)$$

For any further details about these heuristic functions, we encourage readers to refer these cited work (Penberthy and Weld, 1992; Nguyen and Kambhampati, 2001; Younes and Simmons, 2003).

2.4 A Review of Learning for Classical Planning

We can say that Prof. Richard S. Sutton is one of the researchers who brings learning into planning. In the last one and half decades, learning had appeared as an important aspect in automated planning. It has shown very good results to the planning community. The state-of-the-art learning strategies are mostly associated with state-space based approaches in planning. On the other hand, these successful strategies are not tried in plan space to enhance the effectiveness of search techniques. In this section, we discuss a few machine learning approaches successfully tried in the past. Some part of this thesis has influence of the machine learning techniques discussed in the following paragraphs. In the next paragraphs, we cover learning aspects in planning in detail.

This section covers some important literature related to learning in classical planning. At this stage, some part of the discussion might be only distantly related to our current learning approaches discussed in this thesis. We intend to apply a few of the remaining techniques discussed next in the recent future. We briefly discuss some future directions in the last chapter as well (Summary and Future Work).

2.4.1 Combining Multiple Heuristics *via* Learning

We discuss some machine learning techniques used in the planning literature like supervised learning, unsupervised learning. Later, we discuss Bootstrapping and Active Learning procedures employed by the community researchers.

Learning from Multiple Heuristics *via* Supervised Learning

When more than one heuristic functions are available, taking their maximum is always a good option in cost-optimal planning (Korf and Felner, 2002). However, in (Samadi *et al.*, 2008), the authors experimentally show that sometimes combining multiple available heuristics produces good results. This work was inspired from the evaluation function used in two-player games. The authors use Artificial Neural Networks (ANNs) to combine multiple heuristic functions to come up with a single heuristic estimate which has been tested on classical single agent domains, for example: 4-peg Towers of Hanoi, and sliding-tile puzzle. In this work, the authors claim that ANN estimates values close to the optimal values. ANN intelligently captures the mutual influence and correlation among the different input feature heuristics. It is possible that many of the feature heuristics used as input features to ANN, are overestimating (inadmissible by their nature of design). To manage the weighted factors for each feature heuristic in the combination obtained after tuning ANN, the authors give a modified cost function to optimize. In this thesis, we have adapted this cost function and applied Gradient Descent algorithm to optimize the modified cost function. The modified ANN is known as Penalty Enhanced ANN (PE-ANN). This modification penalizes the input feature heuristics more which are exceeding the target value during the training phase. The learned models are successfully tested by the authors on large sized problems that are solved by single-agent.

In another approach, Michael Fink (2007) proposes an approach that learns heuristics using an online learning approach which can find the shortest path between two nodes in a graph (Fink, 2007). It uses elementary heuristic functions and suggests that a weighted sum itself generates a dominating heuristic over the base ones. The assignment of weights to the elementary heuristic estimates is done empirically using an online approach. The approach finds shortest path between any two nodes in a given graph, and also reduces the required search effort. Michael Fink says that it is good to take different views using different elementary heuristics as their weighted combination can cover more of the underlying search space.

Thayer, Dionne, and Ruml (2011) give an online learning approach to combine multiple heuristics which generates a more informed, probably an inadmissible, heuristic estimate (Thayer and Ruml, 2009). Their work is an improvement over a previous work that effectively combines different admissible heuristics. The combinations might lead to inadmissibility but they would be more effective than the admissible ones on different planning standards. The previous offline approaches need training samples in the beginning before the actual planning process starts. As this pre-computation might be resource intensive for various planning domains, the authors use an online learning approach that also perform instance specific tailoring. Some parts of this thesis are based on the PhD dissertation of Jordan T. Thayer (Thayer, 2012). We extend a few approaches to POCL framework from this dissertation, *e.g.* the idea of TD learning. We adapt this instant specific learning approach in POP with some changes that we discuss in the coming chapters.

2.4.2 Bootstrapping and Active Learning

Bootstrapping has been one of the good approaches that the community has tried. It is used to design a more informed heuristic for search algorithms like Best-First Search, A^*, and IDA^*, or the heuristic search planners like Fast Forward, and Fast Downward. In one approach, from a given base heuristic h_0 and a set of unsolved planning problems, it generates a sequence of different heuristic functions (Arfaee *et al.*, 2011, 2010). The approach usually starts from a very weak heuristic and as the system solves more training problems, new heuristic h_1 gets generated. The assumption is that h_1 would be more effective compared to h_0. This is particularly based on bootstrapping approach of machine learning. In case, if a weak heuristic is not able to solve the given instance the system uses random walk from the goal state. We skip other specific details here, and encourage readers to refer (Arfaee *et al.*, 2011) for further clarification. However, we currently do not extend and use this approach. In future, unlike our current approaches, this could be an option for us where it is not required to have an informed heuristic estimate in the beginning of the planning process.

Domshlak, Karpas, and Markovitch (2010) claim that a heuristic function cannot be best in all planning domains (Domshlak *et al.*, 2010*a*). They say that, often, taking the maximum from a set of numerous heuristic values at every stage of planning is not a good idea in optimal planning. In such cases, it is required to compute each heuristic estimate at every stage which might be time consuming. However, there is always a trade off between the plan quality and number of states visited in finding a solution.

The authors give a novel approach that reduces the cost required to combine numerous heuristics together, while retaining the benefits of combining multiple heuristics. In this approach, to circumvent the trade-offs of combining multiple heuristics, a decision rule is used for selecting a heuristic function in a given state. An active online learning technique is applied to learn a model for that given decision rule. This method is called $Selective\ Max$, that is applicable for any form of search problems, and the authors have tested it on planning problems.

Another work uses Active Learning (AL) in an approach of improving the control-knowledge acquisition for automated planning (Fuentetaja and Borrajo, 2006). The approach solely depends on training examples. The training examples are generally extracted in form of search tree that are generated during the procedure solves the problems. In this work, the authors suggest some AL strategies for producing training problems applicable for machine learning *in lieu of* solving different natures of planning problems efficiently. They present a total three AL approaches that are independent from normal planning processes and the employed machine learning approaches in the literature. We refer readers to (Fuentetaja and Borrajo, 2006) for further details. Currently, we do not extend this approach as well, but plan to extend this approach in the POCL framework in the near future.

The next section captures the recent advancements in classical and temporal planning. We confine our discussion to some good heuristic approaches used in classical and temporal planning.

2.5 Recent Advancements in Heuristic Search

In this section, we discuss some recent topics covered in the literature of heuristic search in classical and temporal planning. Contents presented in here is only distantly related to our currently employed approaches. Perhaps, the section covers some recent advancements in the field of heuristic search. We have covered a few good heuristic estimation techniques proposed in last fifteen years, involved in the enhancement of planning processes, resulting in solving many problems efficiently that were intractable. We also discuss a few temporal planning approaches, where heuristic search techniques play important role. This section also covers a few temporal planners developed in the last one and half decades, where heuristic search has played a crucial role. The approaches discussed in this thesis are expected to be efficient in a temporal set up too.

Next, we discuss some recent advancements seen in the field of heuristic search in artificial intelligence planning. First, we cover approaches that have been extended and employed in classical planning.

2.5.1 A Classical Planning Perspective

Most of the classical planning algorithms are domain independent in nature, and use domain independent heuristics. A set of many informed and rigorously designed heuristic functions from the literature which have been employed in heuristic search for classical planning, has been based on the tractable set of planning problems. As we said in the introduction, the main problem with these heuristics is that the known tractable set of planning is still very small. This is specially the case if we consider cost-optimal search in planning, where the employed heuristic should be admissible in nature. A cost-optimal admissible heuristic function always underestimates the actual cost during heuristic estimations. It has been observed that often a planning problem is intractable for one set of heuristics, which is tractable for the other set. This happens due to informativeness of heuristic functions for the case of that particular planning problem, of course, the informativeness of a heuristic function strictly depends on its nature *or* the way it is designed.

In the direction of heuristic search in planning, first, we discuss the work done by Patrik Haslum and Hector Geffner in 2000. Here, the authors discuss the designing of admissible heuristics required for optimal planning. This covers some highly rigorously designed heuristic functions proposed during that time. The work is based on the approach of overcoming the drawbacks associated with HSP and HSPr planners. However, these planners are competitive with Graphplan (Blum and Furst, 1995) and SAT planners, but are not the optimal ones. The main contribution of the work of Haslum and Geffner (2000) is to come up with a whole family of admissible and polynomial heuristic functions that trade efficiency and accuracy (Haslum and Geffner, 2000). To estimate an admissible cost for a set of atoms from initial state to the goal state, these new heuristic functions are employed. In a case, when the set has only one atom in it, for that, the heuristic behaves like h_{max} (Bonet *et al.*, 1997). Similarly, if the size of the set is two, for parallel planning, the estimate is similar to the inbuilt heuristic estimator of Graphplan. Perhaps, the heuristic does not consider the mutex relations, and also does not bother building a layered graph. One of the advantages of admissible heuristic is that we can safely prune a large part of state space, without losing on the optimality of the final solutions.

Like the additive heuristic, h_{add}, the authors use relaxed problems to calculate newly designed heuristics but the original problem and its relaxed version are formulated in a quite different way. The original problems are treated as single source shortest path problems. To solve such problems with action costs, $c(a) > 0$, they define a cost function, V^*, for a node s. Here, $V^*(s)$ shows the optimal path cost from initial state to s. The function is specified as a solution of Bellman Equation that is,

$$V^*(s) = \min_{\langle s',\ a\rangle \in R(s)} \{c(a) + V^*(s^{'})\}$$

where $R(s)$ contains state action pairs $\langle s^{'}, a\rangle$ s.t. action a maps state $s^{'}$ to s, and s = $f(a, s^{'})$. We skip further details from this thesis and refer readers to the original work (Haslum and Geffner, 2000). They come up with admissible heuristics like *max-pair* heuristic, and *higher-order* heuristic along with optimal heuristics for parallel planning. Note that, in this thesis, we discuss some approaches for systematic non-linear planning (McAllester and Rosenblitt, 1991), where actions are allowed to be executed in parallel. The designing strategies of inadmissible POCL heuristics like $h_{add}(\pi)$ and $h^r_{add}(\pi)$, are not extended from the strategies used for these optimal heuristics. But, these optimal heuristics are applicable in parallel planning like h_{add} and h^r_{add} in POCL (non-linear) planning.

Another approach proposed by Joerg Hoffmann, utilizes the problem structure to come up with a refined heuristic function (Hoffmann, 2003). Joerg devises an algorithm for local search that employs this refined heuristic. The investigation on the success of Fast Forward (FF) planner tells that most of planning benchmark problems share some kind of structures. The structures imply specific characteristic qualities to the fast forward heuristic, h_{FF}, employed in FF planner. The heuristic is based on delete-relaxation heuristics that use *relaxed* planning problems. To relax a planning problem, the general idea is to ignore all the negative effects of the actions for the given planning problem, which sometimes causes loss of important information, resulting in too much generalization. In this work, he suggests that it is important to exploit planning structures, where delete-relaxation is quite successful. Therefore, the approach could be *fast* in solving some intractable problems by exploiting the problem specific structures.

For the benchmark domains used to evaluate the performance of FF planner, he comes up with the following observations.

- A total 15 out of 20 planning domains do not contain unrecognized dead-ends during search.
- A total of 13 out of 15 domains do not contain any state that is not a solution, and the heuristic estimates of such states are smaller as compared to the heuristic estimates of their neighbors.
- An important observation made by him was related to *helpful* actions. In a total 10 out of 14 domains, if an action starts a shortest solution plan from a state *s*, it also starts the shortest relaxed solution plan from *s*.

Many other observations have also been made by him, and the same can be found in (Hoffmann, 2003), which are very effective in heuristic search for planning. In this thesis, we are also motivated towards exploitation of the structures and properties of planning problems using learning. However, there have been many other planners proposed after FF planner, but FF planner remained one among the fastest planners for quite some time. A couple of years later, a metric planner named Metric-FF comes, and is completely based on FF planner (J.Hoffmann, 2003). Metric-FF planner is able to handle metric fluents and optimizes metric cost functions. We refer readers to the cited journal for further details of this planner.

Now, we briefly discuss another approach for devising a heuristic function and a planning algorithm based on causal planning graph analysis (Helmert, 2004). The approach was influenced from the success of fast forward algorithm. However, in this work, Malte points out the negative effects associated with the usage of relaxed problems during the estimation of h_{FF}. The relaxation of a planning problem causes loss of too much vital information during the heuristic estimations for some planning domains, of course for those domains, the heuristic h_{FF} does not perform well. At that time, for all the known planners, detecting dead-ends for the STRIPS formalism was difficult. The STRIPS formalism is usually uncertain about the *causal structures* of planning problems, and their importance in problem solving. However, sometimes, these structures are quite evident to human beings. To expose such underlying structures like dead-ends detection for solving a planning problem, in this work, the author proposes an approach of translating STRIPS problems to a multi-valued state variables representation. This representation is also known as SAS+ planning representation (Bäckström and Nebel, 1995). Using this formalism, the approach proposed by him, exploits dead-ends in the search space by employing an algorithm. The algorithm uses the heuristic estimates which is completely based on hierarchical problem decomposition (Helmert, 2004).

In this thesis, we have seen the designing of a few informed non-temporal POCL heuristics. There have been several informed heuristics proposed for classical state-space planning in the literature. We briefly discuss a few of them, and also provide with the important references for those effective heuristics proposed by the community people in the past two decades. We avoid going into the details of these heuristics unlike the first two well described approaches by us proposed by Joerg Hoffmann and Malte Helmert, however, one can always follow the references for further details.

Figure 2.3 covers different classes of heuristic functions proposed in classical planning. The credit for this figure directly goes to Joerg Hoffmann, and it has been taken from the planning course he offers at Sarrland University. It depicts many heuristics designed and employed by the community researchers. Basically, the idea of heuristic search has been divided into four classes as shown in the figure. We briefly introduce each of them and also describe the relationships depicted between the heuristics in the figure. Recently, Malte Helmert and Blai Bonet and their colleagues have proposed potential heuristics, LP based heuristics (Seipp *et al.*, 2015*a*) and flow-based heuristics (Bonet and van den Briel, 2014), that have not been depicted in the figure, for which relevant references have been provided.

Figure 2.3 clearly depicts the four general purpose methods for obtaining heuristic estimates. They are: (*i.*) Delete Relaxation: solves simpler problems obtained from the relaxations of the original problems, as we discussed in the case of fast forward heuristic h_{FF}. We also call this class Ignoring Deletes. (*ii.*) Critical Paths: focuses on achieving *n* hardest atoms out of a total *m* atoms in a state *s*, where $n \leq m$ and $h^n(s) \leq h^m(s)$. (*iii.*) Abstractions: that solves small sized planning problems, and (*iv.*) Landmarks: that refers to the milestones achieved and remaining to be achieved. In further subsections, we will briefly discuss all the classes of heuristics shown in the figure along with some

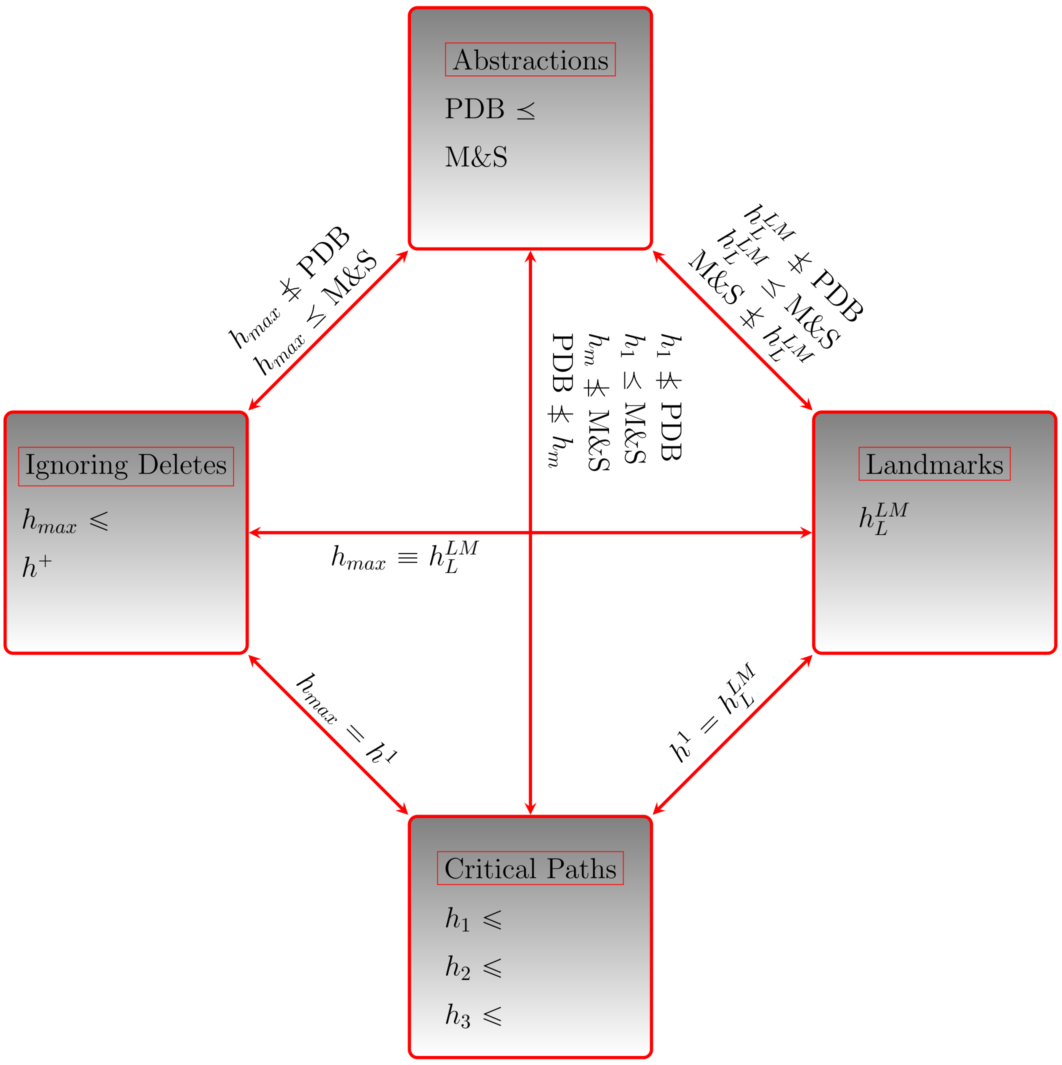


Figure 2.3: Compilability between lower-bound h.

of the mutual relationships between the heuristics like $\leq$, $\equiv$, $\preceq$, and $\npreceq$ that are built between these heuristics in the figure. We do not describe any class of heuristics in detail, perhaps some references have been provided to readers. Some delete relaxations (like h_{FF}) and landmarks based heuristics (like $h^{\text{LM-Cut}}$) have been used in this thesis for the several comparisons.

Delete Relaxations *or* Ignoring Deletes

There is an important aspect of heuristic search in planning, where the plan length of simplified version of the problems are used to calculate the estimates for the original planning problems. This simplification is done by relaxing some of the constraints for a given planning problems like ignoring the delete effects of the actions [1]. However, this does not allow the heuristic to capture the negative interactions between the actions, sometimes that resulting in less informed heuristic. In this context, there have been a few recent approaches employed in classical planning, and can be found in (Bonet and Geffner, 2000; Mirkis and Domshlak, 2007; Haslum, 2012; Keyder *et al.*, 2012; Imai and Fukunaga, 2014; Keyder *et al.*, 2014). Some of these approaches have been employed in cost-optimal planning using delete relaxation. *Note*: the inability of finding a solution for a relaxed problem implies that one cannot find a solution for the original planning problem.

Critical paths

Critical path heuristics focus on achieving m hardest propositions during evaluating a state S. It is an informative approach for which the hardness of estimating h^m increases with m, and it has been covered in the literature. To increase its informativeness, sometimes if the selected value of *m* is large, the heuristic value estimation resulting in a tedious task. This may cause an ineffective planning, specially for large sized problems. Also, for large *m*, it becomes almost as hard to evaluate as the original state.[2] The *critical paths* for heuristic search in planning can be followed for further details (Haslum *et al.*, 2005; Coles *et al.*, 2008*a*; Haslum, 2009) . This class of heuristics has not been purposely tested in the POCL framework.

Abstractions

As the name suggests, a planner estimates about an original planning problem by solving smaller version of it. At a very high level, the original problem is projected to a smaller search space. The estimator solves this projected version of the problem, and uses that solution to have *estimates* for the original problem (Haslum *et al.*, 2007; Helmert *et al.*, 2014; Sievers *et al.*, 2015; Helmert *et al.*, 2015; Sievers *et al.*, 2014). For example, if the original problem from Logistics domain, has 30 trucks, 100 packages, and 7 drivers. The abstraction resulting in a smaller and simpler problem that

[1]http://ai.cs.unibas.ch/_files/teaching/fs16/ai/slides/ai35.pdf

[2]http://fai.cs.uni-saarland.de/teaching/winter13-14/planning-material/planning08-critical-path-heuristics-post-handout.pdf

might have 10 trucks, 50 packages, and 3 drivers. Once a planner solves this abstract version, the solution found can be used for solving the original problem. The group led by Malte Helmert @University of Basel has found some excellent approaches on the usage of abstraction. To understand *abstraction* in a full fledged manner, readers are referred to some recent approaches discussed in the classical planning literature (Katz and Domshlak, 2008, 2009; Nissim *et al.*, 2011; Domshlak *et al.*, 2010*b*).

Landmarks

Hoffmann, Porteous, and Sebastia (2004) say that a landmark is a formula that must hold at some point in time during the execution of each plan for a given planning problem (Hoffmann *et al.*, 2004; Richter *et al.*, 2008). We can fix the order of execution of landmarks, while sometimes we also generate landmarks on the fly during the planning process. They are considered as landmarks to achieve, that also makes the search more directed towards the goal. For example, suppose if we are inside the campus of IIT Madras, and we need to go to the main gate from the department of CSE. For this problem, plans must satisfy *Gajendra Circle* as a landmark. There is possibility of designing admissible heuristics based on landmarks too (Helmert and Domshlak, 2011; Pommerening and Helmert, 2013)

There have been several heuristics proposed by the planning community based on landmarks (Bonet and Helmert, 2010). Most of the recent heuristics are quite informed in the search space, while they are capable of tackling large sized planning problems too. A landmarks could be either an *action* landmark or a *fact* landmark. The basic strategy of generating landmarks for a given problem is based on Backward Chaining algorithm. However, in general a landmark based heuristic function simply counts the number of milestones yet to achieve (Richter and Westphal, 2010; Karpas and Domshlak, 2009; Helmert and Domshlak, 2009). For a given problem, calculate a set (L) of landmarks, and for a given current state *S*, landmark heuristic $h(S)$ estimates the number of landmarks in (L), which have not been yet achieved on the route to S. In another extension, Keyder *et al.* talk about generating a sound and complete set of landmarks for an And/Or graph (Keyder *et al.*, 2010). Its one possible extension can be seen as generating a sound and complete set of landmarks for normal causal graphs.

In the previous paragraphs, at a very high level, we discussed the flavour of four different classes of designing heuristics. Now, we describe some of the relationships between heuristics depicted in Figure 2.3. There are several types of relationships shown in this figure like $\leq$, $\equiv$, $\preceq$, and $\not\preceq$. We discuss only a few, and for other relationships one can always refer to the online notes of Joerg Hoffmann (Saarland University), and (Helmert and Domshlak, 2009) could also be referred. One relationship in the figure depicts $h_L^{\mathrm{LM}} \preceq$ M&S, where the term $\preceq$ is a kind of simulation property. Here intuition is that whatever we could achieve using landmark heuristic that can be achieved using M&S as well. To be precise, each such estimate returns a lower bound on distance-to-goal. However, the notation $\preceq$, says that whatever lower bound we can calculate using h_L^{LM}, at least the same or a better lower bound can be calculated using M&S. Of course, that is not true for the case shown in the relationship $h_L^{\mathrm{LM}} \not\preceq$ PDB.

Apart from these classes of heuristics, now a days the community also explores the approaches based on Linear Programming (LP), and Integer Linear Programming (ILP) (Pommerening *et al.*, 2014, 2015), for designing heuristics. Roger *et al.* discuss some LP based approaches that are helpful in designing informed heuristics for cost-optimal planning. It is an interesting paper to be followed. We skip the details of the LP approaches used for the formulations of the current heuristics like landmark based heuristics. A couple of interesting work have been cited in this paragraph to be followed in this context.

Next, we discuss some of the recent advancements in temporal planning. Temporal planning is more suitable for modeling real world applications, and appears quite useful as compared to classical planning. Here, we cover some important approaches proposed in last one and half decades. This also includes the new search enhancements techniques introduced for temporal planners. The discussion in the next subsection is not confined only to the search techniques used in temporal planning. As per the context of this thesis, it is suitable to discuss these approaches in detail. The POCL framework has been quite successful for temporal planners in the past. We see our proposed approaches for POCL planning in this thesis as possible extensions, and they can be tested in the temporal planning setup too whether they are suitable. If we follow the current trend in this direction, it is not easy for a researcher to reduce the performance gap that separates out the partial order planning approaches involved in temporal planning, and the state of the art temporal planners.

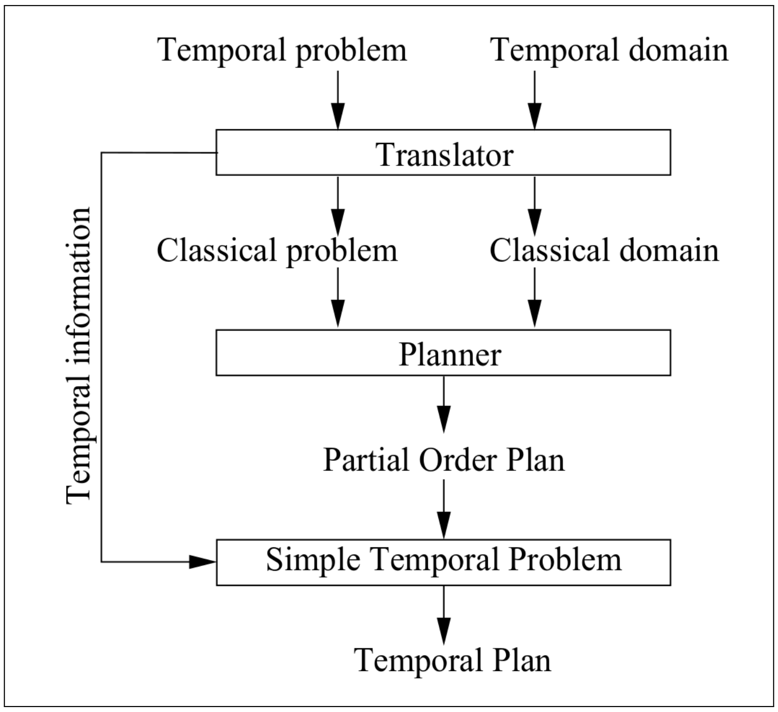


Figure 2.4: An old architecture for separating planning and scheduling. This high level approach is employed in CRIKEY, which is adapted from (Halsey, Long, and Fox 2004), and taken from (Planken, de Weerdt, and van der Krogt 2008) (Planken *et al.*, 2008)

2.5.2 A Temporal Planning Perspective

We start the discussion with an old temporal planner called CPT (Vidal and Geffner, 2006). The POCL planners generally lack on the effective search space pruning strategies, though they provide effective branching scheme. CPT introduces a Constraint Programming based approach that combines the POCL branching scheme with strong pruning rules. The novelty in the approach is the ability of the planner to reason out about the precedence, causal links, and supports involving actions that are not part of the plan. The evaluations demonstrate that, during 2004, it was one among the best parallel planners, specially when all the actions have the same duration.

Another temporal planner CRIKEY (Halsey *et al.*, 2004) got introduced during that time frame. It is able to handle those domains where planning and scheduling are tightly coupled. However, it solves the problems separately by separating out the temporal and logical reasoning. The authors employ action compression in Relaxed Planning Graph (RPG). It is also possible to allow a series of relaxations that can be tightened anytime if necessary, to generate a plan. In the procedure employed, the critical thing is to find the interaction point where the temporal and logical reasonings interact, as demonstrated in Fig. 2.4. The authors demonstrate the initial results in DriverLog domain and compared them with FF planning system (Hoffmann and Nebel, 2001), and SAPA (Do and Kambhampati, 2003). The reasoning takes place in CRIKEY is particular to PDDL2.1 representation (Fox and Long, 2003). The planner finds out a classical plan first, then it lifts the unnecessary constraints to make it a partial order plan. Later, it checks the consistency of the plan using Simple Temporal Network (STN). Here, one possible direction of extension is that instead of generating a total order plan, if we can focus on generating a partial order plan from the beginning. In CRIKEY, due to the nature of separating out the temporal and logical reasonings, the approach is not very effective in handling problems having deadlines, where actions required temporal coordination, and if the actions are concurrent. It does not handle linear numeric changes too where logical and temporal parts are interleaved.

Cushing *et al.* ask a question that is, "When can we say that temporal planning really temporal?" (Cushing *et al.*, 2007). They come up with answers for the issues like whether temporal benchmarks do capture the essential aspects of temporal planning. Most of the decision epoch planners enable the search to be carried out in state-space, so that they can utilize strong state-space based heuristics. Those heuristics are designed and developed for classical planning instead of temporal. However, there are severe weaknesses like incompleteness, are associated with these planners due to this practice. In a few cases, STN does not find any consistent orderings of the actions, specially when actions are concurrent, and where good reasoning is required. The authors raise another question whether we can make decision epoch planners complete while retaining all their advantages. In this regard, they come up with an effective state-space based temporal planning algorithm that helps decision epoch planners in achieving the completeness by retaining the quality performance (Cushing *et al.*, 2007).

Influenced from the above discussed works of Cushing *et al.*, to overcome the issues associated with CRIKEY, Coles *et al.* come up with a temporal planner called CRIKEY3 that is based upon CRIKEY. The planner is capable of reasoning with complete semantics of PDDL 2.1 along with numeric constraints (Coles *et al.*, 2008*b*). It uses heuristic forward search for managing durative actions. To make sure that all the temporal constraints are meeting CRIKEY3 employs STN, and for guiding the search the authors introduce a variant of Relaxed Planning Graph that can handle temporal actions. This variant is also known as Temporal Relaxed Planning Graph (TRPG).

However, CRIKEY3 does handles temporal and numeric interactions, also it does not handle Timed Initial Literals (TILs) (Hoffmann and Edelkamp, 2005) directly. TILs are exogenous events that happen to be TRUE at a particular time frame during the planning process like a shop opens at 10AM, and closes at 11PM. CRIKEY needs a compiled version of TIL (PDDL2.2 (Edelkamp and Hoffmann, 2004)) to handle it, perhaps on the other hand CRIKEY3 handles TIL directly (Coles *et al.*, 2008*b*). The techniques like detecting cycles in STN, employed in CRIKEY3 make sure that generated plan is sound. For further details we refer readers to (Coles *et al.*, 2008*b*).

In another approach of temporal planning the authors consider planning problems with continuous linear numeric changes. Such problems are quite interesting to model, which is not always possible using the older approaches like action discretization or action compressing employed in CRIKEY3 (Coles *et al.*, 2009). The authors show that it is possible to extend the forward-chaining planners to reason out with the continuous linear numeric changes. The authors continue by elaborating on temporal planning strategies which forms the basis of the temporal planner named COntinuous LINear (COLIN) process planner, which can handle linear changes. To track these changes effectively, they employ Linear Programming (LP) approaches (Coles *et al.*, 2012). The LP is used to examine the validity of search states as well as scheduling actions in a plan. They employ temporal numeric heuristics to guide the search which is required to solve planning problems with continuous changes. The work empirically demonstrates that their approach handles smartly the time dependent numeric changes.

A recent approach of finding Temporal Landmarks (TL) to solve a temporal planning problem, is quite interesting. Marzal, Sebastia, and Onaindia (2014) introduce the usage of TL for solving planning problems with specified deadlines. They propose a temporal landmark planner called TempLM that handles deadline constraints. There are several real world applications of such approaches that solve temporal planning problems with goal deadlines like delivery of goods, and supply-chain activities. TL could be either a fact landmark or an action landmark. A proposition (TL) must be achieved in a plan to satisfy the deadline constraints specified in the problems (Marzal *et al.*, 2014*a*). There are three annotations associated with each TL which describe the interval of its temporal occurrence along with logical and temporal consistencies. As more number of constraints violate with the progress of planning process, the planner prunes plans from the set of solution plans. TempLM builds temporal landmark graph for each problem which, structurally, looks like a solution plan. The authors employ an adapted version of $h^{\text{LM-Cut}}$ heuristic (Pommerening and Helmert, 2013) proposed by Helmert and Domshlak (2009), in the employed heuristic search algorithm.

Temporal landmarks based approaches have shown good improvement in the last couple of years in solving temporal planning problems. The procedure employed to find out all TLs in the previously discussed work is not effective in some cases like if there is no deadline associated with a planning problem. Hence, finding causal LMs, as proposed in (Marzal *et al.*, 2014*a*), would not be effective enough, resulting in yielding less benefits. To overcome this issue, Karpas *et al.* proposed an approach of finding LMs which suggests *which* LM should appear in a plan, and *when* (either a time interval or particular time during the plan execution) it must appear (Karpas *et al.*, 2015).

Karpas *et al.* come up with an effective procedure for generating sets of fact and action temporal landmarks for a given temporal planing problem. Usually Backward Chaining is employed in classical planning to generate sets of landmarks for a given problem. In this work, the authors also employ the same approach for TLs generation. For a given problem, the procedure of finding the sets of LMs is sophisticated due the derivation rules used to find out the temporal constraints over the occurrences of those LMs (Karpas *et al.*, 2015). The authors also compare their strategy with the one used in TempLM (Marzal *et al.*'s), they consider an example problem from Matchcellar planning domain, and theoretically demonstrate that TempLM struggles in finding out informative landmarks due to the absence of a deadline and some strong action dependencies in the problem.

Chapter 3

LEARNING FROM EXISTING INADMISSIBLE HEURISTICS: SUPERVISED APPROACHES

In recent International Planning Competitions (IPC) state-space based and total-order planners like LAMA (Richter and Westphal, 2010), Fast Downward Stone Soup (Helmert *et al.*, 2011), and Fast Downward Stone Soup 2014 (Röger *et al.*, 2014) have performed well. These planners are very efficient, generate consistent states fast, and use powerful state-based heuristics. But they often commit too early to ordering of actions, giving up on flexibility in the generated plans. In contrast, the POCL framework (Coles *et al.*, 2010) generates more flexible plans, but in general is computationally more intensive than the state-space based approaches. The POCL framework has found applicability in multi-agent planning (Kvarnström, 2011) and temporal planning (Benton *et al.*, 2012).

Researchers have recently investigated the applicability of state-space heuristic learning approaches (Sapena *et al.*, 2014; Coles *et al.*, 2010) in POCL planning. This revival of interest is due to the idea of delayed commitment of RePOP (Nguyen and Kambhampati, 2001) and VHPOP (Younes and Simmons, 2003). For this approach, we further investigate the adaptation of state-space approaches in POCL planners yielding quality plans over the same or even larger problems. As we stated earlier, in general, due to the diverse nature of planning problems characterized by the degree of interactions between subgoals, a heuristic function does not always work well in all planning domains. Various techniques have been devised to increase the informativeness of heuristics in the state-space arena. One approach strengthens the delete relaxation heuristic by incrementing lower bounds to tighten the admissibility of the heuristic, repeatedly by solving relaxed versions of a planning problem (Haslum, 2012). In another approach, to circumvent the trade-offs of combining multiple heuristics, a decision rule is used for selecting a heuristic function in a given state. An active online learning technique is applied to learn a model for that given decision rule (Domshlak *et al.*, 2010*a*).

We can say that, in recent years, the community has realized that techniques for learning from heuristic functions show great improvements in performance of state-space based planners. One could possibly use an approach of domain wise supervised learning to learn predictive models from existing heuristics. These learned models can be deployed as new heuristic functions.

3.1 Motivation

In the last one and half decades researchers have also investigated the use of heuristics derived from state space approaches (Nguyen and Kambhampati, 2001) in POCL planning (McAllester and Rosenblitt, 1991) (Penberthy and Weld, 1992). POCL planning has the advantage of greater flexibility during the plan execution (Muise *et al.*, 2011) relative to sequential planning. Following the current trends in learning for planning, we base our idea of developing a new informed heuristic function using old heuristics, and our focus here is to learn to combine different heuristics.

In the first learning approach that employs ANNs, where we intend to learn admissible combinations of heuristics though it is not guaranteed. We adapt the use of neural networks to combine heuristic functions (Samadi *et al.*, 2008) in the POCL framework. This approach uses a different error function, from the usual one where we try to minimize mean-squared-error, that penalizes those features more which have values higher than the target value. The complete goal is to incorporate a learning mechanism that further improves the performance of a RePOP (Nguyen and Kambhampati, 2001) or VHPOP like planner in a given domain by training the neural network to combine the available heuristic functions. Our results demonstrate that the planner does indeed learn a different way of combining heuristic functions over different domains. The basic idea behind this approach is that different heuristic functions take a different view of the current problem and arrive at different estimates of the distance to the goal. While some functions may grossly underestimate the distance in some domains, others may overestimate the distance in different domains. We adapt an approach that extends the usage of a set of solved examples in a domain to learn a suitable combination of heuristic functions for that domain. Further, when a search algorithm has partial plans as candidate nodes in the plan space, the notion of "distance" (a heuristic estimate) intuitively specifies that the amount of work needs to be done (number of refinements) by the planner to transform a node to a solution plan.

Here, we implicitly assume that the faster planning process will be resulting in better solution plans. The experiments reported here demonstrate that this approach of learning to combine heuristics does well on different domains.

A set of regression techniques is employed to combine multiple existing heuristics. In the second approach, we propose a learning technique applicable in the POCL framework. Unlike the first offline learning approach, here, we do not force on admissibility of the learned combination heuristics, rather we try to learn only effective combinations. They are tested very effectively on different planning benchmarks. Our approaches are influenced from the literature of learning for planning (Arfaee *et al.*, 2011; Thayer *et al.*, 2011; Samadi *et al.*, 2008; Virseda *et al.*, 2013), that improves the effectiveness of heuristic search in the POCL framework. We apply domain wise regression techniques in a supervised manner, using existing POCL heuristics as features. For generating the targets in each domain, we use our planner called RegPOCL, that is based on VHPOP and uses grounded actions to speed up the planning process. RegPOCL planner employs these two approaches and evaluation shows that it is more efficient on the benchmarks. We have confined the evaluation to non-temporal STRIPS domains.

In the coming sections, we discuss some supervised approaches that have been employed to learn from existing heuristics. We start with a regression approach based on ANN that focuses on admissibility of the learned predictive models. However, one cannot guarantee the admissibility of meta-heuristics as machine learning techniques have been employed. We then discuss another approach based on supervised learning. Similar to the earlier approach, we employ several regression techniques to learn effective domain wise combinations of existing heuristics. The focus of current approach is to make the POCL planning faster as compared to the state-of-the-art.

Next, we discuss a supervised learning approach for combining multiple existing heuristics by employing ANNs.

3.2 Supervised Learning: Artificial Neural Networks

This section covers the first supervised learning approach where we focus on admissibility of the learned combination heuristics. As we said, to combine multiple inadmissible heuristics we employ ANNs and their variants. For learning different regression models, here we use a different error function from the regular mean-squared-error function (explained later in this section). We give the reason behind this, and describe later in this section. While we train a network, we back propagate the associated error with each prediction for the given training instance. This changes the weight vector accordingly after every iteration in the training phase, here, we randomly provide the initial weight vector. For error optimization, we use Gradient Descent algorithm. As we said earlier, the error function focuses on the admissibility of generated meta-heuristics. An ANN that uses this new kind of error function (defined later) is called as Penalty-Enhanced ANN (PE-ANN). Currently, we do not test properly whether PE-ANN actually helps a POCL planner in achieving good quality plans with less effort. However, experiments demonstrate that these learned networks produce effective combinations of heuristics.

3.2.1 Feed-Forward Artificial Neural Networks

The basic idea in this part of our work is to estimate a heuristic value from the values returned by a set of existing heuristic functions. Following the work (Samadi *et al.*, 2008) we use an artificial neural network (ANN) (Mitchell, 1999) to define the estimate in terms of inputs. Given that some of the input values may be gross overestimates, we need to train a network so as to penalize inputs that are larger than the (known) target value in each training example. The estimates have been combined using a variant of ANN called Penalty-Enhanced Artificial Neural Network (PE-ANN) (Samadi *et al.*, 2008).

Artificial Neural Networks

Fig. 3.1 shows an ANN with Y as target variable and x_i for $i \in \{1, 2, ..., 6\}$ as input variables. The error function $E(t)$ is defined as $E(t) = \theta(t) - Y(t)$ where $\theta(t)$ is the predicted value and $Y(t)$ is the target value. The mean-squared error (MSE) is given as,

$$MSE = \frac{\Sigma_{t \in X} E^2(t)}{|X|}$$

where X is the training data. The MSE is a symmetric function that penalizes the error on both sides equally. To maximize the effects of algorithms like WA^* (Pearl, 1984),

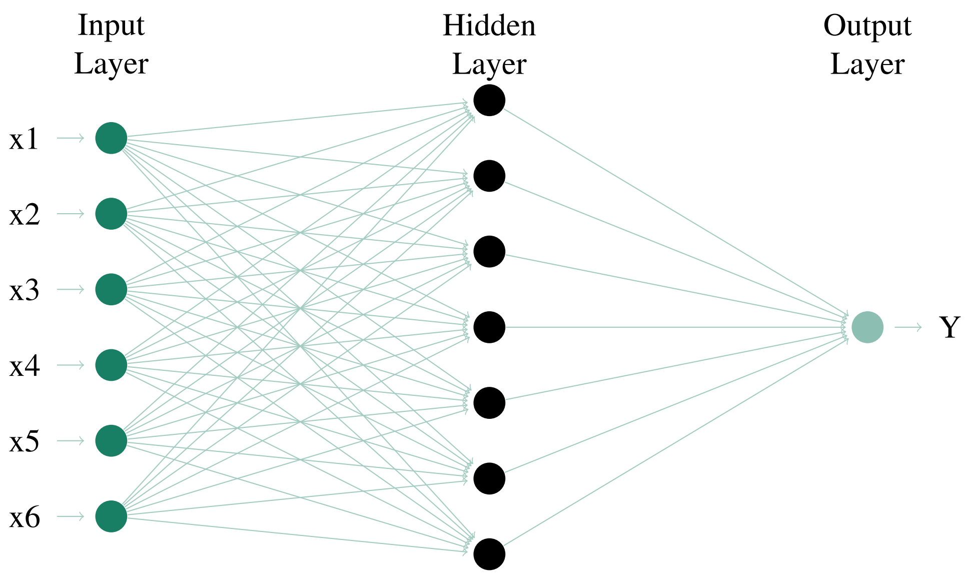


Figure 3.1: Artificial Neural Network

IDA^* (Korf, 1985) or dynamically weighted A^* (dWA^*) (Thayer and Ruml, 2009), we penalize the model predictions that are higher than the target value more than the ones that are lower. This results in our predications being by and large lower that the target value. Consequently we can use uniform weight factors for the heuristic functions, for combining them, over different planning domains. But, these weight factors will be learned for each domain, in a supervised manner. In future, we intend to test these newly learned meta-heuristics (learned from multiple heuristics) in planning algorithms that are used to solve planning problems.

We decide to penalize the overestimating heuristic values more than the underestimating heuristic values irrespective of whether the combined heuristics or the individual ones are admissible or not. The PE-ANN model addresses this concern and its error function (Samadi *et al.*, 2008) given below is biased towards underestimation,

$$E_{new}(t) = \left\{ a + \frac{1}{1 + \exp\big(-b \times E(t)\big)} \right\} \times E(t)$$

here a and b are experimental constants which decide the slope of the error curve, $E_{new}(t)$ regulates the penalization criteria for $(E(t) > 0)$ and $(E(t) < 0)$, where $E(t) > 0$ is penalized more.

We have used a variation of PE-ANN in which a regularization term (L2-norm) (Bishop, 2006) in the error function is introduced. This follows evidence that regularization decreases the tendency of overfitting when the size of the training set is smaller than what is ideally required. This is often the case in a planning domain (for example the Towers of Hanoi) where the number of different instances that can be solved are not many.

A brief description of the approach is as follows. For each planning problem in the training set, we have a set of n heuristic values along with the target value that is calculated using Graphplan (Blum and Furst, 1995). The training data is used to learn the weights for the PE-ANN using a suitable error function that determines the weights that are learnt. The error functions we use are described later. For a new (test) instance t, we input the n different values from the existing heuristic functions to the trained PE-ANN model to predict the heuristic value $\theta(t)$. The heuristic value at the output node is instrumental in deciding which of the candidate partial plans is selected for refinement.

A similar approach has shown very good results in state-space-planning in the past (Samadi *et al.*, 2008). By looking at the flexibility associated with POCL framework as we discussed in the previous chapter, we have implemented a version of plan space planning that uses fully grounded actions instead of partially instantiated operators. In doing so, we do give up on the least commitment strategy, but gain in speed-up of execution (Younes and Simmons, 2003, 2002; Nguyen and Kambhampati, 2001). Younes and Simmons (2003), and Younes and Simmons (2002) have shown the advantages of POCL planning with lifted actions as well. Therefore one extension could be to explore the possibility of reverting to partial plans with lifted actions. In the following section we describe the approach for training of neural networks.

Penalty Enhanced Artificial Neural Networks

The following properties are desired in a underestimating heuristic function. The larger the heuristic value, the more will be the search space pruned by the algorithm. In addition if we require the algorithm to be admissible it is imperative that the heuristic function underestimates the actual distance. Here the idea is not bound to strict admissibility of heuristic functions in any form (either individually or combined).

In POCL setup, a partial plan (π) is a node in a the space of partial plans, the function used by WA^* for selecting a candidate is: $f(\pi) = g(\pi) + w \times h(\pi)$. Here, $g(\pi)$ is the number of actions in π, and $h(\pi)$ is the heuristic value that estimates the number of refinements required to resolve the remaining flaws, *w* is the weight factor.

The heuristic value $h(\pi)$ thus plays a crucial role in the selection process of π from a set of candidate partial plans. Previous work has indicated that one heuristic function does not perform equally well in the domains. To address this drawback, we utilize PE-ANN model to yield often underestimating estimates for πs by combining multiple heuristic functions together.

The PE-ANN model uses the gradient descent approach in the algorithm Backpropagation to learn the weights starting from randomly initialized weights. The weight update rules with regularization using L2-norm, and the gradient descent process are described below.

3.2.2 The New Weight Update Rules

The error function defines a surface over which gradient descent seeks the minimum. The error function employed in the PE-ANN imposes higher penalties on edges that transmit the signal from overestimating inputs. The network then learns to suppress such inputs and generate an output that is lower than the target value. In addition since our PE-ANN model uses the L2-norm, the cost function is devised as follows,

$$\begin{aligned} E'(t) &= \sum_{i=1}^{N} \left\{ a + \frac{1}{1+\exp\big(-bE(t)\big)} \right\} \times E(t) \\ &\qquad + \gamma \times \left(\sum_m \beta_m^2 + \sum_m \sum_l \alpha_{ml}^2 \right) \end{aligned}$$

where $E'(t)$ is error for an instance *t*, a and b are the parameters that are set empirically, γ is the regularization coefficient, l is the number of nodes in the input layer, m is the number of nodes in the hidden layer, and α and β are the weight of the emanating edges.

We propose the weight update rules for the PE-ANN for newly introduced cost function with regularization. The squared error function used for the minimization problem is given below.

$$R(\theta) = \sum_{i=1}^{N} \left\{ \left(a + \frac{1}{1+\exp\big(-b(T-Y_i)\big)} \right) \times (T - Y_i) \right\}^2$$

$$R(\theta) \;=\; \sum_{i=1}^{N}\left\{\Big(a+\sigma\big(b(T-Y_i)\big)\Big)\times(T-Y_i)\right\}^2$$

where $R(\theta)$ is sum of the Squared Error that is also represented as $R(\alpha,\beta)$, T is the predicted output T = $\beta_0+\beta^T z$, Y_i is the target value, a and b are experimental parameters, and N is the number of training instances in the training set. In the above expression T is dependent on vector z, where m^{th} element of z is,

$$z_m=\sigma(\alpha_m+\alpha_m^T x)=\frac{1}{1+e^{-(\alpha_{0m}+\alpha_m^T x)}}$$

here, $\sigma(x)$ is a sigmoid function over x. The partial derivative of $R(\theta)$ *wrt* β_m is,

$$\frac{\partial R(\theta)}{\partial\beta_m}=2\times E'(t)\times\frac{\partial E'(t)}{\partial\beta_m}$$

The partial derivative of $\sigma(x)$ w.r.t. x is given as, $\frac{\partial(\sigma(x))}{\partial(x)}=\sigma(x)\times\big(1-\sigma(x)\big)$. In order to minimize $R(\theta)$, we are required to assess its sensitivity to each of the weights. We take the partial derivative of $R(\theta)$ with respect to each of the weight parameters to calculate the effect of changing weights. Simplification of the partial derivatives gives final weight update rules corresponding to α and β with *L2-norm* as regularization. The weight update for the weight vector β is quantified as,

$$\begin{aligned}\frac{\partial R(\theta)}{\partial\beta_m} \;&=\; 2\times\Big(a+\sigma(b(T-Y_i))(T-Y_i)\Big)\\&\times\Big\{b\times\sigma\big(b(T-Y_i)\big)\times\big[1-\sigma\big(b(T-Y_i)\big)\big]\times z_m\\&\times(T-Y_i)\;+\big[a+\sigma\big(b(T-Y_i)\big)\big]\times z_m\Big\}+\;\gamma\times2\beta\end{aligned}$$

Similarly, the weight update for the weight matrix α is quantified as,

$$\begin{aligned}\frac{\partial R(\theta)}{\partial\alpha_{ml}} \;&=\; 2\times\big(a+\sigma\big(b(T-Y_i)\big)\times(T-Y_i)\big)\\&\times\Big\{b\times\sigma\big(b(T-Y_i)\big)\Big(1-\sigma\big(b(T-Y_i)\big)\Big)\\&\times\beta_m\frac{\partial\big(\sigma(\alpha_m^T x)\big)}{\partial(\alpha_{ml})}(T-Y_i)+\;\big[a+\sigma(b(T-Y_i))\big]\\&\times\beta_m\frac{\partial\big(\sigma(\alpha_m^T x)\big)}{\partial(\alpha_{ml})}\Big\}+\gamma\times2\alpha\end{aligned}$$

where the regularization coefficient $\gamma > 0$, and

$$\frac{\partial \sigma(\alpha_m^T x)}{\partial \alpha_{ml}} = \sigma(\alpha_m^T x)\big(1 - \sigma(\alpha_m^T x)\big) x_l$$

From above equations, the gradient descent weight update rules for parameters α and β for a single training instance in r^{th} iteration are obtained and given as follows:

$$\begin{aligned}
\alpha_{ml}^{r+1} &= \alpha_{ml}^{r} - \eta_\alpha \times \frac{\partial R(\theta)}{\partial \alpha_{ml}} \\
\beta_m^{r+1} &= \beta_m^{r} - \eta_\beta \times \frac{\partial R(\theta)}{\partial \beta_m}
\end{aligned}$$

where η_α and η_β are the learning rates that may get different values for the gradient descent rules. The $\gamma * 2\beta$ and $\gamma * 2\alpha$ factor can be removed from the update rules in order to train PE-ANN without the regularized cost function. The specific details of the parameters values in our experiments are given in tables for each domain.

3.2.3 Multiple Heuristics

An efficient POCL planner does a controlled search in the plan space. It attempts to minimize search (the number of flaw resolutions) by making proper choices during planning process (Schubert and Gerevini, 1995). A good strategy is to select the most demanding flaw and a refinement that leaves the minimum refinements to be done subsequently. There are good flaw selection techniques based on Most-Cost or Most-Work etc, associated with the flaws (Younes and Simmons, 2003). One possible way of selecting a partial plan from the queue is based on the minimum number of actions needed to resolve all the open conditions in it (Nguyen and Kambhampati, 2001). We consider the definition of $h^*(\pi)$ from (Nguyen and Kambhampati, 2001) which says, for given a partial plan π, $h^*(\pi)$ gives the minimum number of new actions required to convert a partial plan to a solution plan. However, a similar definition can also be found in (Penberthy and Weld, 1992). In this work, the authors introduce a POCL planner named UCPOP. This is one of the oldest planners that handles Action Description Language (ADL) representation. Next, we briefly describe the nine heuristics used as features for learning. For further details we refer readers to the given pointers. The last six heuristics solely based on the reachability analysis done using planning graph data structures.

1. $h_{max}(\pi)$: **Max Heuristic**

The simplest heuristic h_{max} counts the number of steps required individually for each open goal, and takes the maximum value. In the state-space planning, this is likely to grossly underestimate the cost and is less informed than h_{add} described next. But in the POCL framework there are cases where it overestimates as well. The admissibility depends on the availability of *OCs* of a partial plan in the start state. It is considered a good choice in the case of overlapping states.

$$h_{max}(p) = min\{h_{max}(p), 1 + h_{add}(precond(a))\} \tag{3.1}$$

$$h_{max}(S) = max_{p \in S}\, h(p) \tag{3.2}$$

Here $h_{add}(precond(a))$ is $\sum_{q \in precond(a)} h_{max}(q)$.

2. $h_{add}(\pi)$: **Additive Heuristic**

We have already defined this heuristics in Chapter 2. It adds the total effort needed to achieve each proposition in a state. We avoid repeating the same text here. For further details refer to the section 2.3.2.

3. $h^r_{add}(\pi)$: **Positive Interaction Heuristic**

We have already defined this heuristics in Chapter 2. We avoid repeating the same text here. For further details refer to the section 2.3.3.

The heuristic functions defined below in this thesis, are based on the reachability analysis of the partial states (hypothetical states) from initial state *I* by planning graph data structure (Hoffmann and Nebel, 2001). We extend the use of state-space heuristic functions in plan space planning (Nguyen and Kambhampati, 2000), in order to impose a tight bound on the value. We treat all the *OCs* present in a partial plan as making up a state. One has to be careful how to deal with sets of predicates that cannot be part of a state.

4. $h_{relax}(\pi)$: **Relax Heuristic**

Again, we have already defined this heuristics in Chapter 2 as a part of base heuristics. We will not repeat the same text here. For any details about this heuristic refer to the section 2.3.2.

5. $h_{set-level}(\pi)$: **Set-level Heuristic**

We altered and use $h_{set-level}(S)$ proposed in (Nguyen and Kambhampati, 2000) for plan space search where its usage depends on the preprocessing step (the assumption of hypothetical state *S*). For a given partial plan, *S* contains all the open conditions present in the partial plan. The altered heuristic $h_{set-level}(\pi)$ is equal to $level(S)$.

$$h_{set-level}(\pi) = level(S).$$

6. $h_{partition-2}(\pi)$: **Partition-2 Heuristic**

Nguyen et al. have shown that in certain cases $h_{set-level}(S)$ estimates the same numerical values for two different states (Nguyen and Kambhampati, 2000). To overcome this, $h_{partition-2}$ is devised and that is adapted for POCL planning as, $h_{partition-2}(\pi) = \sum_{S_i} lev(S_i)$.

7. $h_{adjust-sum}(\pi)$: **Adjust-sum Heuristic**

We adapt the idea—proposed by Nguyen and Kambhampati (2000)—discussed in the previous heuristic function where we state that

$$level(S - p_1) \leq level(S)$$

p_1 is achieved first and by the time $(S - p_1)$ is achieved, it may clear the achieved proposition p_1 because of negative interactions. In this case, $level(S - p_1) \neq level(S)$ when we achieve p_1 again. Following this argument (Nguyen and Kambhampati, 2000) we state that,

$$cost(S) = cost(S - p_1) + cost(p_1)$$

If p_1 and any proposition p from $(S - p_1)$ are mutex, then taking the negative interaction between the propositions into the account we arrive at the following,

$$cost(S) = cost(S - p_1) + cost(p_1) + (level(S) - level(S - p_1))$$

In a similar manner, we can also write as specified in the next equation, and so on the chain continues until S becomes ϕ.

$$\begin{aligned} cost(S - p_1) &= cost(S - p_1 - p_2) + cost(p_2) \\ &\quad + (level(S - p_1) - level(S - p_1 - p_2)) \end{aligned}$$

Simplification of above sequence of relationships leads to,

$$cost(S) = \sum_{p_i \in S} cost(p_i) + level(S) - level(p_n)$$

The last proposition of the set S is p_n, and the corresponding equation for the POCL framework is,

$$h_{max}(\pi) = level(p_n)$$

$$h_{adjust-sum}(\pi) = h_{add}(\pi) + h_{set-level}(\pi) - h_{max}(\pi)$$

8. $h_{adjust-sum2}(\pi)$ **: Adjust-sum2 Heuristic**

The heuristic is adapted from Nguyen and Kambhampati (2000) and used for search in the plan space. The estimation of $Cost(S)$ is time consuming as compared to $h_{adjust-sum}(S)$ however it often produces near target solutions except for a few domains (Nguyen and Kambhampati, 2000). The altered heuristic $h_{adjust-sum2}(\pi)$ formulation is,

$$h_{adjust-sum2}(\pi) = h_{relax}(\pi) + h_{set-level}(\pi) - h_{max}(\pi)$$

The heuristic (in state-space search) performs better when we consider binary-mutexes (Nigenda *et al.*, 2000) also known as $h_{adjust-sum2M}$ heuristics (used in alt-alt planner (Nigenda *et al.*, 2000)) that combines planning graph data structure and heuristic search.

9. $h_{combo}(\pi)$: **Combo Heuristic**

We alter the definition of h_{combo} given by Nguyen and Kambhampati (2000) and consider its variation called $h_{combo}(\pi)$ defined as,

$$h_{combo}(\pi) = h_{relax}(\pi) + h_{set-level}(\pi)$$

We use $h_{relax}(\pi)$ as relax heuristic is more accurate in most of the planning domains compare to h_{add} (Younes and Simmons, 2003).

Even though Graphplan may sometimes give solutions that have more than the minimal number of actions in some domains, has been used for the calculation of target values (h_{target}) for smaller sized problems. It is difficult for the Graphplan algorithm to find solutions for bigger problems in the given time as it uses fully instantiated operators. Therefore, Graphplan takes inordinate amount of resources to grow fully, specially in the case of high correlations between the actions and the interactions among the subgoals. For some of the planning domains (*e.g.* Towers of Hanoi) we observe the lack of sufficient training instances for our model. We do not consider those domains but if sufficient training instances available we use basic regularization (L2 *norm*) factor in the PE-ANN error function.

3.2.4 Empirical Evaluation

We perform the experiments on Intel Core 2 Quad with 2.83 GHz 64-bit processor and 4Gb of RAM. These experimental results are the basis for the claims made. We present the results obtained by PE-ANN with and without regularization. The results demonstrate that penalizing the overestimates more does tend to keep the output heuristic within the target value.

The datasets used in our experiments are generated as follows. The partial plans for which the heuristic value is to be estimated are generated by our implementation of VHPOP. The target value for each partial plan is obtained by treating the open goals in the partial plan as the goal, which is then solved by the algorithm Graphplan. We use the relaxed version of Graphplan. In this version, actions used are relaxed by removing the list of their delete effects. However, this version is a part of many state-of-the-art work like latest variants of Fast-Downward (Helmert, 2006) use it for finding landmarks. It also helps in calculating the values of different heuristics discussed previously, where we consider a hypothetical state, S. For those sets of open goals that cannot be part of any consistent state, and for which Graphplan cannot find a plan, we set the target to infinity and we exclude them from the set of training examples. In a result, we exclude all partial plans for which Graphplan can not find a solution and hence all those for which S does not constitute a consistent state as well. This indicates that sometimes the extracted training instances for a partial plan will be removed from the training set because of the large cost. Consequently some instances extracted for some partial plans will not be accommodated in the training set. We intend to resolve this drawback with the approach of inadmissible heuristic functions learning (Thayer *et al.*, 2011) or bootstrapping (a completely unsupervised approach) (Arfaee *et al.*, 2011).

Dataset Preparation

Our datasets are made up of estimated and actual plan costs, with integer values. An example dataset is shown in Table 3.4. Each row in the table has the heuristic values computed by the functions described in the preceding section, and the last column is the target value (h_{target}) found by Graphplan. Observe that the datasets used for training are complete in all respects and do not have any missing values. The dataset preparation phase took some order of hours (4-5) for each domain. However, as mentioned earlier, problems like the Towers of Hanoi produce only small datasets, since even moderately sized problems cannot be solved by Graphplan within a time limit of 15 minutes. The details of the number of training instances and other parameters have been specified below the respective tables.

Performance Evaluation

The results of our experiments in three domains are presented in Tables 3.1 to 3.3, each with the data from five test instances (we generate all possible $\langle\langle h_1, h_2, ..., h_9\rangle,\ h_{target}\rangle$ pairs by solving a smaller sized problem, solve many such problems and pick randomly five instances from the validation set for testing). For each test instance, the tables show the lowest and the highest heuristic estimates, h_{low} and h_{high}, from the nine heuristic functions that provide the inputs to the neural networks. Also shown is the target value

Ins	h_{low}	h_{high}	h_{target}	$h_{closest}^{target}$	PE-ANN			Using L2 norm		
					$h_{predicted}$	$h_{closest}$	h_{val}	$h_{predicted}$	$h_{closest}$	h_{val}
1	3	18	9	h_{vhpop}	10.05	h_{vhpop}	7	8.97	h_{vhpop}	7
2	2	10	10	h_{combo}	11.32	h_{combo}	10	9.13	h_{combo}	10
3	6	15	12	$h_{partition-2}$	6.25	h_{vhpop}	6	7.0	$h_{set-level}$	7
4	4	19	11	$h_{set-level}$	6.27	h_{max}	4	6.95	$h_{set-level}$	9
5	6	14	11	h_{combo}	6.28	h_{vhpop}	6	6.98	$h_{set-level}$	7

Table 3.1: The Blocksworld domain, the fixed parameters: $|X|$ = 955, a = 1.0, b = 3.0, after cross-validation: γ = 0.1, $L_{rates} \in$ [0.6, 0) and 13 nodes in the hidden layer with all 9 features. The terms h_{low} and h_{high} are the minimum and maximum values of the features of the instance (t). h_{target} is the target value, calculated using Graphplan, while $h_{closest}^{target}$ indicates the feature heuristic function closest to h_{target}. Here, $h_{closest}$ indicates the feature which has its value closer to the predicted output and h_{val} is the value of the closest heuristic. The tables similar to this have condensed captions further in this section.

Ins	h_{low}	h_{high}	h_{target}	$h_{closest}^{target}$	PE-ANN			Using L2 norm		
					$h_{predicted}$	$h_{closest}$	h_{val}	$h_{predicted}$	$h_{closest}$	h_{val}
1	3	24	13	$h_{partition-2}$	5.82	h_{vhpop}	8	5.97	h_{vhpop}	8
2	5	20	17	$h_{partition-2}$	15.80	$h_{partition-2}$	15	15.11	$h_{partition-2}$	15
3	3	20	11	h_{vhpop}	5.78	h_{vhpop}	9	5.96	h_{vhpop}	9
4	3	12	12	h_{combo}	15.27	h_{combo}	12	12.12	h_{combo}	12
5	2	8	10	$h_{set-level}$	5.76	h_{vhpop}	7	7.87	h_{vhpop}	7

Table 3.2: The Elevator domain, the fixed parameters: $|X|$ = 724, a = 1.0, b = 3.0, after cross-validation: γ = 0.01, $L_{rates} \in$ [0.6, 0) and 10 nodes in the hidden layer with all 9 features. The terms h_{low} and h_{high} are the minimum and maximum values of the features of the instance (t). Here, h_{target} is the target value, calculated using Graphplan, while $h_{closest}$ indicates the heuristic function which has its value closer to the predicted output and h_{val} is its value.

h_{target} computed by the algorithm Graphplan with the set of open goals in the partial plan treated as the goal. For each test instance the heuristic estimate generated by both the PE-ANN and the regularized PE-ANN networks is shown along with the heuristic function that matches the predicted value the most. In cases where the predicted estimate is lower than h_{low} the corresponding heuristic function is left blank (like some entries in Table 3.2). We hazard a guess that the predicted value is lower than *all* the inputs because of the non-linear nature of the combination produced by the neural networks in which the penalty for some inputs pulls the output below h_{low}. The objective is to demonstrate that the estimates generated by the networks are (a) lower than the target and (b) high values.

Table 3.1 shows the outcomes of the experiments performed in the Blocksworld domain. The weights used in the networks, α and β, were initialized to random values. The number of nodes in the hidden layer is kept larger than the number of nodes input layer of PE-ANN as prescribed in the neural networks literature. The size of this layer was determined empirically. The learning rate L_{rate} is initially set high, so that it leads to early convergence to a region close to the minimum. When gradient descent approaches the minimum the L_{rate} is reduced to enable it to converge using smaller steps. The problem instances reported in Table 3.1, are taken from problems containing seven or eight blocks. All the predictions for all the instances by both the networks are closer to the target, and also the regularized network estimates are closer to target. One notable point that shows up in Table 3.1 is that two of our adapted heuristic functions *e.g.* h_{combo} and $h_{set-level}$ make an appearance as the functions producing estimates closest to ones generated by the networks.

Table 3.2 shows the results in the Elevator domain. In our experiments there was a paucity of training examples, because Graphplan was unable to solve many bigger problems that we generated. It may be observed that range of heuristic estimates generated by the nine functions is very high, and tending to be largely overestimates. In this domain the performance of regularized network is better than the unregularized one and at the same time the accuracy lies between 50 and 55. We get almost similar accuracy in the Blocksworld domain as well. Note that, by accuracy we mean the number of instances for which the values of $h_{predicted}$ were exceeding the targets, and become admissible by employing the proposed approach. The approach has been validated over **900** to **1000** instances generated from different stages of solving planning problems in each selected domain. Here the predictions are even more informed than the individual features used in the experiments performed in the earlier parts of POCL planning. The accuracy decreases as the value of the parameter γ increases, and some uneven behaviour is observed in the accuracy when $\gamma \in [90, 100]$. This uneven nature has not yet been studied further.

Table 3.3 shows outcomes of the travel domain. We do not have enough training samples for this domain as well. Here both the algorithms perform equally, though the performance could have been better. This may be due to the small number of training instances. The domain has high subgoal dependence as there is big difference between h_{target} and h_{high}. Predicting values lower than the h_{low} is not a desirable property. To overcome the drawback of small number of training instances, we reduce the number of features by removing highly correlated ones. In the end, 4 features (h_{max}, h_{add}, $h_{djust-sum2}$ and h_{combo}) are selected (right part of Table 3.3). We keep the same number

Ins	h_{low}	h_{high}	h_{target}	$h^{target}_{closest}$	PE-ANN			Using L2 norm			PE-ANN			Using L2 norm		
					h_{reg}	$h_{closest}$	h_{val}	h_{reg}	$h_{closest}$	h_{val}	h_{reg}	$h_{closest}$	h_{val}	h_{reg}	$h_{closest}$	h_{val}
1	3	10	5	h_{max}	3.43	h_{max}	3	3.48	h_{max}	3	3.96	h_{max}	3	4.12	h_{max}	3
2	5	14	17	h_{combo}	3.46	-	-	3.51	-	-	19.87	h_{combo}	14	18.21	h_{combo}	14
3	4	12	6	h_{vhpop}	3.45	-	-	3.49	-	-	4.02	h_{max}	4	4.36	h_{max}	4
4	3	10	5	h_{max}	3.44	h_{max}	3	3.48	h_{max}	3	3.96	h_{max}	3	4.13	h_{max}	3
5	4	12	12	h_{combo}	3.44	-	-	3.49	-	-	12.29	h_{combo}	12	11.81	h_{combo}	12

Table 3.3: This table shows the results in Travel domain. We divide this table into three major parts. The second and third parts capture the predictions by the learned models using all 9 features and the most uncorrelated 4 features, respectively. Here, h_{reg} shows the predicted values by the networks. In **second** part, the parameters are $|X|$ = 359, a = 1.0, b = 3.0, γ = 0.001, $L_{rates} \in$ [0.6, 0) and 7 nodes in the hidden layer. While in the **third** part $|X|$ = 359, a = 1.0, b = 3.0, γ = 0.001, $L_{rates} \in$ [0.6, 0) and 7 nodes in the hidden layer with 4 features.

of training instances with reduced number of features. Table 3.3 shows that the reduced network has better performance in the travel domain. It can be observed (in the training data set) that only h_{max} has values below target. In four out of five instances the network selects the correct options for the column $h_{closest}$ (the closest heuristic). The regularized network yields better predictions when it is a smaller network with only four inputs, probably because the number of training instances are sufficient for the smaller networks.

First, we observe that the heuristic $h_{set-level}(\pi)$, $h_{partition-2}(\pi)$, $h_{adjust-sum}(\pi)$, $h_{adjust-sum2}(\pi)$ and $h_{combo}(\pi)$ that we have adapted and used. It can be seen in the datasets that the extended heuristic functions perform well. By and large the value given by these functions is closer to the target value than the existing functions, for example as in Table 3.4 (however, the whole dataset cannot be shown) which is a part of actual dataset created for the Blocksworld domain. Though in some domains they underperform at few stages of planning process as compared to other state-of-the-art heuristics of POCL planning.

Our second observation concerns the idea of combining these heuristic functions together, wherein both the overestimated as well as underestimated values of the heuristic functions are combined in a certain domain specific ratio. In all our experiments the heuristic values generated were high underestimating values, though there were instances where the values were much lower than the optimal value, for example as in Table 3.3. The poorer results in this table pertain to travel domain, where the number of training instances we could generate was far fewer. We tried training a smaller neural network here with only four inputs instead of nine, and as depicted in Table 3.3, the performance improved.

h_{max}	h_{add}	h^r_{add}	h_{relax}	h_{set}	h_{part2}	h_{adjsum}	$h_{adjsum2}$	h_{combo}	h_{target}
4	13	9	9	9	13	18	14	18	14
4	11	7	9	9	12	16	14	18	15
5	11	5	6	6	10	12	7	12	12

Table 3.4: A sample dataset in Blocksworld domain, The attributes are the heuristic functions *Max, Add, Positive Interaction, Relax, Set-level, Partition-2, Adjust-sum, Adjust-sum2 and Combo* defined in the last section along with *target value* retrieved from Graphplan

3.2.5 Conclusions

Our main motivation of the approach in this section (Section 3.2) was that whether we could learn domain independent way of combining heuristic functions by training over examples from that domain. We have evaluated a combination heuristic by comparing it with the constituent heuristic functions from literature, and also against the known target value (optimal targets) for the training data. We have not tested this using a POCL planner. Here, another motivations was to explore whether given a set of diverse input heuristic values if we can learn combinations are underestimating the (assumed) optimal cost. The learned models generated have been tested on about **1000** test instances in each selected domain. We demonstrate the overall accuracy (the aggregate) of each learned model which is more than 50% for each domain. It means using our adapted cost function (using PE-ANN with L2 norm), at least half of the total samples used for testing were forced to underestimate the actual targets in the testing phase. We randomly choose only a few of the test points as we cannot show all of them.

However, for the statistical significance, comparison does not include the time needed to generate the training samples or learning combinations of heuristic functions. For getting a general idea of the complete approach, we demonstrate the overall accuracy in each domain. A possible extension to this approach would be to compare a POCL planner using the combination heuristics against other planners.

3.2.6 Summary and Future Work

This approach describes an attempt to learn ways to combine different heuristic values to arrive at consistently better estimates over a range of planning domains and problems.

The efficiency of a search algorithm is dependent upon the quality of the heuristic function that it employs.

Various planning algorithms using different domain independent heuristic functions have been studied in the literature. Heuristics adapted from state-space planning approaches have been applied to partial order planning with considerable success. And yet it is not always easy to design a good heuristic function that performs consistently over different domains (Weld, 2011). We extend an approach of learning combinations of heuristic functions (meta-heuristics) that are employed in plan space planning, using a set of existing POCL heuristics, and a set of some adapted heuristics derived by exploration of the state-space.

One can see from Tables 3.1 to 3.3 that not only do different heuristic functions perform better in different domains; even in a single domain different heuristic functions perform better on different instances of problems. This is probably due to the fact that planning by search is essentially a process of non-monotonic reasoning, in which the reasoner asserts and retracts fluents describing the (current) state of the world. This is something that a simple heuristic function is unable to make predictions about. Thus a case for a non-linear function to combine the different estimates, for example as done by a neural network, can be made.

We deploy a penalty enhanced artificial neural network that employs error functions that penalize overestimates more than underestimates. We also explore a version with regularization that is designed to work with smaller training sets. The inputs to the networks are the values predicted by the available heuristic functions and the output is a new, hopefully improved, estimate. The networks are trained separately for each domain. The cost of achieving the goal is computed by running the algorithm or the original planning problem but with the set of open goals treated as the goal to be achieved. This value is fed back as the cost of resolving the open goals into the backpropagation algorithm for training the network.

Once trained, the network is used to arrive at a heuristic value as a function of the values predicted by the existing heuristic functions. Our experiments demonstrate that this approach consistently arrives at estimates that are closer to but smaller than the target value. The additional cost one has to pay is two fold. One, that a set of training examples has to be solved for each domain to train the network. And two, that at each step, heuristic evaluation is done first by the different (input) functions and then as an output of the neural network. However we feel that the benefits in search might outweigh these costs. Our future work is to validate this thesis experimentally using a partial order planner.

Another possibility is to reintroduce some aspects of partially instantiated operators, to try and cut down on the space required to completely instantiate all operators. Recent work (Ridder and Fox, 2014; Ridder, 2014) has shown that this can work, specially when dealing with large problems, where the grounding phase itself takes up most of the computation cycles.

In the next section, we introduce another approach for learning from existing heuristics. We give an algorithm for dataset preparation and training regression models. These models are employed as new heuristics to the search algorithms used in RegPOCL. The intension of this approach is to speed up the planning process. As this is further extended using another approach of heuristic tuning in the next chapter, we do not provide a complete evaluation in this section. However, we empirically demonstrate the utilities of this approach, and we further discuss that why it should be extended by employing the *error tuning* approach. This tuning approach has been discussed in Chapter 4.

3.3 Supervised Learning: Regression Methods

Like our previous approach where we focus on the admissibility of the learned meta-heuristics. Here, in this approach we focus on providing some speed up to the POCL framework using machine learning techniques. For speeding up the process, it is not mandatory to have an admissible heuristic function. Therefore, in a similar way, one could possibly use an approach of domain wise supervised learning to learn predictive models (meta-heuristics) from existing heuristics. These learned models can be deployed as new heuristic functions.

3.3.1 Learning Approaches Used

We propose a two fold approach to learn better heuristic functions. First, existing heuristic function are combined by a process of offline learning that generates learned predictive models. This is followed by an online technique of adjusting the step-error associated with these models during partial plan refinement. We divide this section into two parts: first describes the offline learning techniques to perform regression, and then a technique of further strengthening the informativeness of a learned predictive model.

Offline learning is an attractive approach because generating training sets in most planning domains is a fast and simple process. The learning process has three phases: (*i*) dataset preparation, (*ii*) training, and (*iii*) testing. The training instances gathered by solving small problems become the inputs to the used regression techniques like Linear Regression (LR) and M5P that are described later, which learn effective ways of combining the existing heuristic functions. The testing phase is used to validate the best ways of combining the known heuristics. Algorithm 1, a high-level code described below is fully automated, embodies the complete training phase.

Dataset Preparation

The procedure DATASET-PREP() Line 15 in Algorithm 1 is used to solve a set (S) of planning problems. We consider only small problems that are gathered from each planning domain selected from previous IPCs. We consider each problem from S for the dataset preparation in each domain (line 18). In this algorithm, a seed partial plan is a new partial plan that gets generated due to a possible refinement of the selected partial plan. We select a seed sp from a set of seed partial plans Π (line 20). sp will be provided to RegPOCL for its further refinements. If RegPOCL is able to generate at least one consistent solution by refining sp completely using *Solve()* function (line 19), then the flag F will be true. As shown in Figure 1, we capture the newly generated partial plans in local instance of Π called Π_{loc}. The target $\mathcal{T}$ captures the number of new actions that get added in sp. $\mathcal{T}$ is calculated when the planner refines sp completely using heuristics from H. Note that, $\mathcal{T}$ (the target) is not a heuristic value but actual the number of new actions added during the refinement process using an h_i (line 21). The value of $\mathcal{T}$ is also the plan length found which might not be optimal. Since sp is refined completely, Π_{loc} is updated to Π (line 23). Line 22 computes a training instance Ins, using *Comp-inst()* function. For a given sp, the planner generates a training instance of the form $t(sp) = \langle\langle h_1(sp), h_2(sp), h_3(sp), h_4(sp), h_5(sp), h_6(sp)\rangle, \mathcal{T}\rangle$, where h_1 to h_6 are the feature heuristics. As we explained in our first approach, these features

Algorithm 1 : This algorithm is a domain independent procedure for generating meta-heuristics, that learns domain specific meta-heuristics. The main procedure called TRAINING-ALGORITHM() that calls another procedure DATASET-PREP(). This algorithm first prepare training datasets and later it learns different predictive models. We also call these domain specific learned models as meta-heuristics because they are deployed as heuristics in the testing phase.

```
1:  Input
2:      $AS$ - Attribute Selection; $T$ - Training Datasets;
3:      $S$ - Problem Set; $L$ - Learning Techniques;
4:      $H$ - Heuristic Set; $RegPOCL$ - The Planner.

5:  Output
6:      Predictive Models, M;        // functions of $h_i$s from $H$

7:  procedure TRAINING-ALGORITHM($AS$, $T$, $L$)
8:      for $ind \leftarrow 2$; $ind \leq |H| - 1$; $ind$++ do
9:          $T[i] \leftarrow$ DATASET-PREP($RegPOCL$, $S$, $H$, $ind$)      // Sub-procedure call
10:     end for
11:     The best two $T[i]$s are selected on highest $|T[i]|$
12:     TrainIns$[i] \leftarrow$ Apply($AS$, $T[i]$)
13:     return $M[k] \leftarrow$ Apply(TrainIns$[i]$, $L[j]$)
        // For each dataset & learning technique ($2\times5 = 10$ per domain).
14: end procedure
```

There is another procedure called DATASET-PREP() that will be called by the main procedure TRAINING-ALGORITHM(). For each selected planning domain, DATASET-PREP() will be called four times. This creates four different training datasets out of which two are selected on the basis of highest $|T|$.

```
15: procedure DATASET-PREP($RegPOCL$, $S$, $H$, $ind$)
16:      $F$ - Check; $\mathcal{T}$ - Target Value; $T \leftarrow \phi$.
17:      $\Pi$ - A set of seed partial plans; $h_i \leftarrow H[ind]$.
18:     for each $p \in S$ do
19:         $\Pi \leftarrow$ Null partial plan for the problem "$p$"
20:         for a random $sp \in \Pi$ do        // Limited iterations
21:             $(F, \mathcal{T}, \Pi_{loc}) \leftarrow$ Solve($RegPOCL$, $sp$, $h_i$)
22:             if $F$ then        // sp refines completely
23:                 $\Pi \leftarrow \Pi \cup \Pi_{loc}$
24:                 Ins $\leftarrow$ Comp-inst($RegPOCL$, $H$, $sp$, $\mathcal{T}$)
25:                 $T \leftarrow T \cup$ Ins
26:             end if
27:         end for
28:     end for
29:     return $T$
30: end procedure
```

considered here for learning are also the existing heuristics from the POCL literature. To maintain consistency, we update the training set T (line 25) only when RegPOCL refines the current seed sp completely. If its complete refinement is not possible, all new seeds from Π_{loc} are dropped, even though it might be the case that Π_{loc} contains some consistent seeds that can be completely refined, particularly in the case of time out. To maintain diversity in T, for a given domain we randomly select a fixed number of seeds for the complete refinement process (line 20).

Algorithm 1, a domain independent approach, gets executed once for each selected domain with a given set of feature heuristics. Note that learning does not guarantee optimal predictive models even though optimal targets have been used in the training (Virseda *et al.*, 2013). Algorithm 1 hunts for a well informed heuristic using learning and does not bother about its admissibility. Since the state-of-the-art POCL heuristics are not optimal in nature (Younes and Simmons, 2003), the usage of RegPOCL for generating training instances should not affect the performance of RegPOCL on large problems in the testing phase. The selection of RegPOCL for generating training sets might deteriorate the actual target values, as the targets calculated by RegPOCL are not optimal in general. Thus there is a possibility of learning inaccurate predictive models in the training phase, which might reduce the informativeness of the models. To alleviate this, we enhance the informativeness of the models by correcting the step-error associated with them using an online heuristic tuning approach. We describe this approach of finding single-step-error in the next chapter. In the next chapter, we also demonstrate the results of of our current approach (the second offline learning approach). We also show that how single-step-error alleviate the performance of a POCL planner further.

Training

Once Algorithm 1 finishes generating different datasets for a given domain, it moves to the next step (line 11). We define a regression model $\mathrm{R} : \mathrm{T} \rightarrow \mathbb{R}$, where T is a training set and $\mathbb{R}$ is a set of real numbers. Following the general model training strategy, we use WEKA (Hall *et al.*, 2009) to remove irrelevant or redundant attributes (line 12) from the training set. This reduces the effort of the planner because the planner must calculate the selected feature heuristics at each step of the planning process. The output of Algorithm 1 is a set of different trained predictive model (as shown in line 6). Next, we apply model training process (line 13). We feed the selected datasets to different machine learning approaches to learn different predictive models.

Testing

We test the predictive models on large problems. The models are directly compared to the current best heuristics in the POCL framework. For using machine learning approaches in planning efficiently, we select the best learned regression models and test the efficiency of RegPOCL by using them over different benchmarks. These models help in selecting the most suitable partial plan for refinement.

Offline learning learns a better model in terms of search guidance and accuracy than online learning (Samadi *et al.*, 2008; Thayer *et al.*, 2011). An offline learned predictor is more accurate than an online one because in the offline case a complete training set is available. Another alternative to the above approaches would be bootstrapping methods (Arfaee *et al.*, 2011), where a set of problems is solved using a base heuristic within a specified time limit. Later, the solutions obtained for learning are used to generate a new more informed heuristic.

3.3.2 Experiment Design

In this section we describe the evaluation phase settings. This includes (*i*) the heuristics selected as features, and (*ii*) the domains selected. The features used for learning are non-temporal heuristics from the literature of POCL planning. Considering the applicability of some of the POCL heuristics in the literature (Younes and Simmons, 2003; Nguyen and Kambhampati, 2001), we select six different heuristic functions. Some of these heuristics are informed but their informativeness varies over different planning domains. Our aim is to learn a more informed combinations from these heuristics.

We now discuss six feature heuristics out of which we have already discussed a few.

G Value ($h_{g\text{-}val}$)

This returns the number of actions in a selected partial plan π not counting the two dummy actions (a_0 and a_∞). This heuristic signifies how far the search has progressed from the starting state. During the planning process, alone this heuristic function is not much effective and it hardly solve some problems. We do not cover any experimental details specially for this heuristic.

Number of Open Conditions (h_{OC})

This is total number of unsupported causal links present in a partial plan, $h_{OC}(\pi) = |\mathrm{OC}|$ (Nguyen and Kambhampati, 2001). During the planning process, like first, alone this heuristic function is also not much effective and it hardly solve some problems. We do not cover any experimental details specially for this heuristic. For further details refer to the section 2.3.1. There we also talk about how we can use each open condition effectively using serial planning graph.

Additive Heuristic (h_{add})

The additive heuristic h_{add} (Haslum and Geffner, 2000), adds up the steps required by each individual open goal. Younes and Simmons (2003) use an adapted version of additive heuristic in POCL planning for the first time. We have described this heuristic function in detail, refer to 2.3.2

Additive Heuristic with Effort ($h_{add,w}$)

The estimate is similar to h_{add} but it considers the cost of an action as the number of preconditions of that action, plus the linking cost 1 if the action supports any unsupported causal link (Younes and Simmons, 2003). We call it $h_{add,w}$ as its notation is not used earlier. Here, w signifies the extra work required.

Accounting for Positive Interaction (h^r_{add})

This returns an estimate which takes into account the positive interactions between subgoals while ignoring the negative interactions. This is represented as h^r_{add} that is a variant of h_{add} (Younes and Simmons, 2003). For further details refer to the section 2.3.3.

Accounting for Positive Interaction with Effort ($h^r_{add,w}$)

This is similar to the above heuristic which considers the total effort required (Younes and Simmons, 2003). A standard notation of this heuristic is also not used in the literature. For further details refer to (Younes and Simmons, 2003).

3.3.3 Domains Selected

We consider the following domains: Logistics and Gripper from IPC 1, Logistics and Elevator from IPC 2, Rovers and Zenotravel from IPC 3, and Rovers from IPC 5. One of the reasons of selecting these domains is, in our experiments, we do not consider other domains from these competitions because either the state-of-the-art heuristics are not able to create enough training instances for learning, or RegPOCL does not support the domain definition language features. IPC 4 domains are not selected since the planner is not able to generate enough instances to initiate offline learning. The domains from IPC 6 and later are not supported by RegPOCL because the representations use action costs, fluents, and hard and soft constraints. Some of them can be included by preprocessing like removal of the cost of the actions from the domain description files.

For each selected domain, we consider problems that are represented in STRIPS style. We select small sized problems for learning and test the learned predictive models over large sized problems in the same domain. We have a total of 109 small sized problems from the selected domains. The last four feature heuristics from the previous subsection have been used for calculating targets in each domain. This means that we generate four different datasets in each selected domain from which best two are selected. We choose satisficing track problems for generating training instances. For the training set preparation, we fix a time limit of 3 minutes and an upper limit of 500,000 on the node generation. We generate a few thousand training instances except for the Zenotravel domain where the total instances are 950.

3.3.4 Selected Regression Approaches

In this section, we discuss in brief a procedure for feature selection in each dataset for training regression models, and different regression techniques with their references.

Feature Selection

In general, the training sets contain irrelevant or redundant attributes (out of the six selected heuristics). To reduce the training effort and increase the efficiency of our planner, we discard them from the training set. The planner is bound to calculate all the selected features at each stage of refinement. The correlation based feature selection technique (Hall, 2000) is used to find the correlated features.

Regression Techniques

We use the following regression techniques to learn predictive models. These techniques have been applied in planning for learning in recent years (Samadi *et al.*, 2008; Thayer *et al.*, 2011; Virseda *et al.*, 2013).

1. **Linear Regression (LR):** The regression model learns a linear function that minimizes the sum of squared error over the training instances (Bishop, 2006).
2. **M5P:** M5P gives more flexibility than LR due to its nature of capturing non linear relationships. M5P technique learns a regression tree (Quinlan *et al.*, 1992) that approximates the class value.
3. **M5Rules:** Similar to M5P but generates rules instead of modeling regression trees (Quinlan *et al.*, 1992). It generates a decision list for regression problems using separate-and-conquer. In each iteration it builds a model tree using M5 and makes the "best" leaf into a rule.
4. **Least Median Squared (LMS):** LMS is similar to LR with median squared error. Functions are generated from subsamples of data with least squared error function. Usually a model with lowest median squared error is selected (Rousseeuw and Leroy, 2005).
5. **Multilayer Perceptron (MLP):** MLP can learn more complex relationships compared to the other four regression techniques (Bishop, 2006). The key difference in this kind of network from the previously employed networks in our first approach is, we use the regular error function ($E(t) = \phi(t) - Y(t)$). Also, our aim here is to combine multiple heuristics effectively instead of forcing the combinations to be admissible. As we said earlier too, learning does not guarantee admissibility, but most often we demonstrate it in the experiments.

In this work, the techniques discussed above are used to learn models through WEKA (Hall *et al.*, 2009) using a 10-fold cross-validation in each domain. However, it has not much influenced planning processes in the past (Virseda *et al.*, 2013).

As we said earlier, Algorithm 1 is completely automated for the training phase, where it learns domain wise suitable regression models using the above discussed regression approaches. In automated planning, we do not have any standard dataset for any planning domains. Therefore, to apply machine learning approaches, we need to create our own datasets according to our need. So, the general purpose algorithm says thatover we consider small problems in each domain to create training instances, later we apply regression techniques on those training sets. This resulting in regression models, that are deployed as new heuristics in the search algorithms used in RegPOCL.

3.3.5 Environment

We perform the experiments on Intel Core 2 Quad with 2.83 GHz 64-bit processor and 4GB of RAM. To evaluate the effectiveness of learned models and to correct the single-step-error associated with the models, a time limit of 15 minutes and a node generation limit of 1 million is used.

3.3.6 Experiments

We use RegPOCL to compare the performances of the offline predictive models h^l, and the last four base features that are also the state-of-the-art non temporal heuristics). We exclude the first two base features from the comparison since they are weak ones, and RegPOCL does not solve enough number of problems using them. However, they are useful while working jointly with other good heuristics as we demonstrate later in this section. Next, we discuss the observations made during the training phase.

Training

Using Algorithm 1, we prepare datasets and learn different predictive models by applying the various regression techniques discussed earlier. We select each of the last four features to solve a set of problems. The target value is the plan length found by RegPOCL using the base features. The dataset preparation phase took less than two hours on an average in each domain, for each of the four features. Once we have enough instances, we begin the training process. We define a regression model $\mathrm{R} : \mathrm{T} \rightarrow \mathbb{R}$,

Domain	**#**	**POCL Heuristics**					
		State-of-the-art				**Via Learning**	
		h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	h^l_{add}	$h^l_{add,w}$
Gripper-1	20	16	**20**	1	1	**20**	**20**
Rovers-3	20	19	19	**20**	**20**	**20**	**20**
Rovers-5	40	28	31	32	36	**39**	$^+$36
		h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	$h^{r,l}_{add}$	$h^{r,l}_{add,w}$
Logistics-1	35	25	1	32	28	**32**	22
Elevator-2	150	148	14	**150**	58	**150**	**150**
Logistics-2	40	36	12	36	34	**40**	**40**
Zenotravel-3	20	5	4	9	10	**16**	**16**
Coverage	325	277	101	278	187	**317**	$^+$304

Table 3.5: Number of problems solved using each feature heuristic, and is compared with the learned combination heuristics via supervised leaning. **#** is the number of problems selected in each domain. Best results are shown in **bold**, and a number with "+" mark (*e.g.*$^+$36) shows competitive performance by the learned models and their enhancements over each base heuristic. We give the total *coverage* **score** in the last row of each table for making the overall comparison easier. For a note, the numerical values with the domain names in the table depict the IPC. Similar column representations have been followed in the next other .

where T is a training set and $\mathbb{R}$ is a set of real numbers. Note that, in Figure 4.1, different heuristics for calculating target values will prefer different paths. Therefore, the four base features will generate four different datasets. The attribute selection strategy allows us to select a subset of attributes in the training set by removing correlated and redundant features. We learn a total of 70 (7 domains $\times$ 2 datasets $\times$ 5 regression techniques) regression models. The training phase took 20ms (milliseconds) using LR, 270ms using LMS, 600ms using MLP, 82ms using M5Rule, and 58ms using M5P on an average per model. All the learned models have high accuracy but LR is the fastest, followed by M5P and M5Rule. For a note - these training times are not used for the comparison. We consider learned models as new heuristics, and in the testing phase they are employed as new heuristics like a base heuristic, in other words, like other heuristics for them the time factor starts from *zero* too. Next, we test these models on different benchmarks.

Testing

We test the effectiveness of our approaches by selecting partial plans (π) for refinement using RegPOCL. We assume that an informed heuristic leads to minimal possible refinements needed for π. Next, for comparison we compute score as in IPC for satisficing track problems. The standard followed by the community for scoring is given below. The formula describes to generate score ($0 \leq\ score_p\ \leq 1$) for each problem in a given planning domain. For the exact details on how these scores are computed, readers are referred to `https://helios.hud.ac.uk/scommv/IPC-14/`.

$$score_p = \begin{cases} best_{val} \Big/ val_{app} & \text{: if a solution is found} \\ 0 & \text{: otherwise} \end{cases}$$

These **scores** will be unit-less, and the better score on each standard benchmark signifies the better performance. We compare the performance of the learned models with the selected base features h_{add}, $h_{add,w}$, h^r_{add}, and $h^r_{add,w}$. The comparison is done on the basis of (*i.*) the number of solved problems, and (*ii.*) the score obtained on plan quality, and execution time.

Before we move to evaluation part, for example, the offline learned model $h^{r,l}_{add}$ used as a column in Table 3.5 and Table 3.6, shows that it is learned on a dataset that is prepared using h^r_{add}. In other words, RegPOCL uses h^r_{add} for calculating the target values in the dataset. It is also similar for other offline learned heuristics in these tables. These models are applied in the POCL framework for selecting the most suitable partial plan, followed by the heuristic MW-Loc (Younes and Simmons, 2003) for selecting the most adverse flaw in the selected partial plan.

Empirical Evaluation

In Table 3.5, we show the *coverage* on the total number of problems solved. Including all the domain, we consider a total 325 problems. This table demonstrates that using the supervised learning approach, RegPOCL is able to solve at least 39 more problems when the learned predictive models have been employed. This also shows the effectiveness of the combined heuristic functions, in which the weights given to each heuristic should be dependent on the utility of that heuristic in a given domain.

Based on the *coverage* shown in Table 3.5, the following inferences are made. In Elevator-2, h_{add} and h^r_{add} are among the good performers from the base heuristics, perhaps learned heuristics solved all the problems. We could see that $h_{add,w}$, and $h^r_{add,w}$ have not performed well. It happened because in this domain, an action resolver can

Domain	POCL Heuristics					
	State-of-the-art				Via Learning	
	h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	h^l_{add}	$h^l_{add,w}$
Gripper-1	16.0	14.9	0.7	0.7	**20.0**	**20.0**
Rovers-3	17.3	16.2	18.1	16.9	17.8	17.8
Rovers-5	25.7	26.3	28.0	30.2	**33.1**	30.1
	h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	$h^{r,l}_{add}$	$h^{r,l}_{add,w}$
Logistics-1	24.3	0.8	31.2	26.1	$^+$31.2	20.9
Elevator-2	142.8	11.8	144.9	49.7	142.7	142.7
Logistics-2	33.8	10.7	34.9	30.7	**38.1**	**38.1**
Zenotravel-3	4.6	3.7	8.5	8.8	**13.5**	**13.5**
Quality Score	264.5	84.4	266.3	163.1	**296.4**	$^+$283.1
Time Score	204.4	76.5	197.7	148.6	**272.9**	$^+$258.6

Table 3.6: Scores on plan quality and overall time. We compare state-of-the-art POCL heuristics with learned ones. The second last row *sums up* the *total quality score* and the last row demonstrates the overall *time score* of each heuristic. Our approaches have performed well.

resolve multiple unsupported causal links at one time. Therefore, in these heuristics, the extra effort considered for each unsupported causal link predicts larger values than the real estimates. A similar trend can also be seen for Logistics-1 and Logistics-2 domains. But in Logistics-2, once we allow action reuse in the heuristic ($h^r_{add,w}$), the performance improves as it improves the heuristic's accuracy, and it could solve **34** problems out of 40. However, when the action reuse was not allowed during evaluating a partial plan, it was able to solve only 12 problems in the given time. Once learning is applied, the selected predictive models get higher weighted coefficients for h_{add} and h^r_{add}. The supervised approach suggested the best learned heuristic for RegPOCL, and hence in Logistics-2 domain it solved all **40** problems. In Zenotravel-3, the base heuristics perform better if action reuse is applied, shows that actions in this domain are not tightly coupled. These heuristics also get higher weights in the best learned models as expected from a supervised approach, resulting in solving up to 16 problems, where the best base heuristic could solve only 10 problems. In Logistics-1, $h^{r,l}_{add}$ solved 39 out of 40 problems, but $h^{r,l}_{add,w}$ could only solve 22. Sometimes it affect the performance of the planner using an over generalized learned model on bigger problems, which happened in this domain. We further enhance the informativeness of these kind of over generalized models in the next chapter.

In Table 3.6, we show the scores obtained on the plan quality and overall time required. The *coverage* shown in the previous table directly affects the scores in this table. The scores obtained using the learned heuristics is quite good as compared to when the base features have been used. The new scores are far ahead as compared to the old scores. The trend we saw in the previous table, we can see almost a similar trend in this table too. As we can infer in Zenotravel-3, even though the learned models have solved 16 problems but they are not the best for those problems that are solved using all the heuristics. This is because the two base heuristics h^r_{add} and $h^r_{add,w}$ were the best for relatively smaller sized problems. Using h^r_{add}, RegPOCL solved 9 problems and obtained the plan quality score 8.5. This justifies that the plans obtained for these 9 problems were among the best plans. We also infer that for these problems, the two learned heuristics $h^{r,l}_{add}$ and $h^{r,l}_{add,w}$ can obtain scores maximum up to 7.5. Therefore, learning techniques are good for bigger sized problems in this domain. A similar trend can also be seen in Elevator-2 domain. It is inferred that if using a base heuristic solves a problem, it is highly probable that a shorter plan will be found. Which is justified by the scores obtained by h^r_{add} (144.9), and the two learned models (142.7) in Elevator-2 domain. Rovers-5 domain also shows a similar trend if we do not consider the extra 3 problems solved using h^l_{add}. The winners for both on the plan quality score and total time score are the learned models, h^l_{add} and $h^{r,l}_{add}$. A few other inferences can be made further which are quite straight forward to visualize. We mention some of them in the next chapter.

We discussed the problem of over generalized learned predictive models like the cases in Logistics-1 and Rovers-5 domain. We work for enhancing the informativeness of such models in the next chapter. In this direction, we base our approach upon Temporal Difference learning. From Table 3.5 and Table 3.6, we at least assess that the offline learned predictive models are doing well in the planning process. In the next chapter, in the *testing* phase of the experiment section (4.4), for an easier overall comparison, we consider these demonstrated results again.

Statistical Significance

Similar to the previous supervised learning approach, it is important to mention the following things for a clear differentiation between the approaches employed in this section and the approaches they have been compared with. When we use scores to compare, specially, on the standard benchmark of the total time taken, it is important to mention a couple of points. In our cases, an ML approach is performed in three phases: (a) dataset preparation, (b) training different models and their comparison for picking the best ones from them, and (c) testing. Each phase is independent and performed in a sequence, therefore each phase has their own time of completion. For the comparison of the learned predictive models with the feature heuristics, the time taken in the dataset preparation and training phases is not considered. This also means once a combination of heuristics is learnt, it is used like a regular base heuristic. An ML approach also involves in cherry-picking of domain wise best heuristics, however base heuristics do not have this flexibility.

3.3.7 Discussion

The literature of learning for planning empirically shows that sometimes the offline learned models generalize the learned knowledge. Therefore, due to too much generalization, they are not effective on large sized problems. The reason is that planning problems often carry different natures even though they belong to the same domain. To overcome such effects of offline learned approaches, we introduce a online heuristic tuning approach in the next chapter. We give a general formula for tuning a given heuristic on the fly when RegPOCL solves a problem. This is based on TD learning proposed by Richard S Sutton (1988). The heuristic tuning approach captures the error associated with the given heuristic, and corrects that error for the subsequent stages of search. This makes the heuristic fine tuned for that particular problem. The approach is problem independent but it learns instance specific details, so that the informativeness of the heuristic can be enhanced for a given problem with the progress of search.

At this stage in this chapter, we do not demonstrate the complete empirical results of offline learning that is covered in this particular section. However, we have shown the utilities of the learned predictive models on a couple of important planning benchmarks, in the domains selected. As we deploy the general purpose online heuristic tuning approach to these learned models, it would be more meaningful if we compare all the empirical results together, after describing the heuristic error tuning approach.

Note that for this approach in this section, we have discussed the domains selected, the feature heuristics, and the regression approaches employed to learn the offline predictive models, and some basic results. In the next chapter, for the evaluations, we will start with comparisons between the base features (the feature heuristics), offline learned predictive models and their further enhanced version obtained using the online heuristic tuning approach.

Chapter 4

ONLINE HEURISTIC TUNING USING TEMPORAL DIFFERENCE LEARNING

Learning new heuristic functions (meta-heuristics) from existing heuristic has been a matter of investigation in last ten years. Apart from the supervised approaches, a promising approach is to monitor and reduce the error associated with a given heuristic function on the fly even as the planner solves a problem. In this part of the thesis, we extend an approach for calculating single-step-error associated with a heuristic function in the POCL framework.

4.1 Introduction and Motivation

The performance of a domain independent planner is critically influenced by the design of the heuristic function. The study of heuristics in classical state-space planning has received significant interest in the past. There are many good heuristic evaluation functions with varying performances on different domains. Our approach is to examine the single-step-error associated with the different heuristic functions during search. The single-step-error also varies for individual domains or even for individual problems. The last decade has also seen a revival in use of heuristics derived from state-space approaches (Nguyen and Kambhampati, 2001; Bercher *et al.*, 2013; Bercher and Biundo, 2013) to POCL planning (McAllester and Rosenblitt, 1991; Penberthy and Weld, 1992).

We consider the importance of heuristic search in POCL planning and the tendency of often overestimating the actual cost by the state-of-the-art POCL heuristics. To overcome this tendency, we adapt and modify a procedure for monitoring single-step-error associated with some state-of-the-art heuristic functions based on Temporal Difference learning also called as TD learning (Sutton, 1988; Thayer *et al.*, 2011). As we saw in Example 1 from Chapter 2, the POCL framework has the advantage of greater flexibility during the planning execution process (Muise *et al.*, 2011). A partial plan is quite complex structurally and therefore developing a well informed heuristic function is comparatively a more tedious task (McAllester and Rosenblitt, 1991; Weld, 2011). In this work, our focus is to investigate the use of the average-step-error associated with some powerful state-of-the-art POCL heuristic functions during search. The basic idea behind this monitoring approach is to avoid a heuristic function taking a similar view of the problems in different domains by measuring and trying to correct the error in the heuristic value on-the-fly.

We discuss the pros and cons of capturing the single-step-error on-the-fly during search, and empirically show that the performance of a planner like Versatile Heuristic Partial Order Planner (VHPOP) (Younes and Simmons, 2003) can be enhanced. We perform experiments over various domains with different degree and nature of interactions between the subgoals and actions. Experiments show that this approach results in more informed decision in different planning domains during search, even being competitive on the time required as well. In some domains we find shorter length plans, good scores on time, and number of nodes visited by the planner, specially for large planning instances.

4.2 Enhancing The Informativeness of a Heuristic

Thayer, Dionne, and Ruml (2011) adapt the idea of TD learning (Sutton, 1988) in state-space planning for capturing the error committed by a given heuristic (Thayer *et al.*, 2011). They use this approach for generating targets on the fly to perform online learning. In the POCL framework, we also adapt the TD learning approach for estimating error made by a POCL heuristic during refinement of a partial plan. The original derivations for capturing the step-error and average-step-error are credited to Thayer, Dionne, and Ruml (2011) (Thayer *et al.*, 2011).

In the POCL framework (Figure 4.1), the minimum number of total refinements needed for a partial plan π_i to make it a solution plan, goes through its best child π_{i+1} that is obtained after refinement R_i. A child π_{i+1} is the best child when it has the lowest prediction of the number of new actions needed for its complete refinement among its siblings. We break ties in favor of minimum number of actions in the children partial plans. The set of successors of a partial plan is potentially infinite. This is due to the introduction of loops in the plan which simply achieve and destroy subgoals like $\langle(Stack\ A\ B), (Unstack\ A\ B)\rangle$ or $\langle(Pickup\ A), (Putdown\ A)\rangle$, in Blocksworld domain. Such loops are common during the refinement process, specially when the heuristic is not well informed. The enhancement explicitly avoids generating such loops which is crucial in real world scenarios. For example, a pair $\langle(LoadTruck\ obj\ truck\ loc), (UnloadTruck\ obj\ truck\ loc)\rangle$, in Driverlog domain could be expensive. In general in plan space planning there is no backtracking (Ghallab *et al.*, 2004), leads you to a space containing infinite nodes. Which means, each refinement of a partial plan leads to a different node in the plan space. This necessitates that we explicitly consider the issue of getting into a loop.

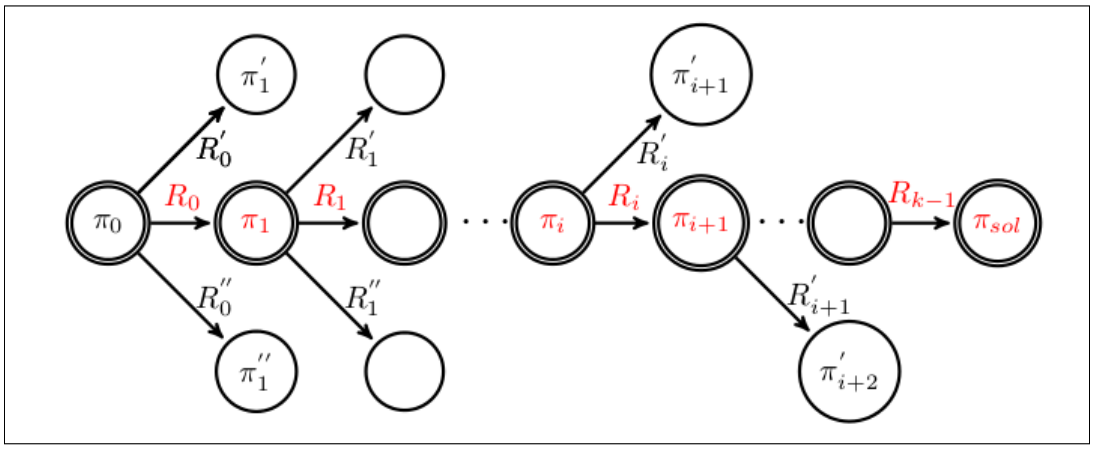


Figure 4.1: A POCL framework - The refinement (R) starts from π_0 and it goes to the solution plan (π_{sol}). At each step, for a selected partial plan, many refinements are possible like refinements of π_0 which lead to π_1, π_1', and π_1''. Here, the best child is shown in the horizontal refinements.

Following the TD learning approach, for capturing the single-step-error, we define the minimum number of refinements needed for a partial plan (π_i) as the sum of the cost of its current best possible refined partial plan (π_{i+1}) and the refinement R_i,

$$h^*(\pi_i) = cost(R_i) + h^*(\pi_{i+1})$$

The runtime estimation of $h^*(\pi_i)$ is a tedious task, also not feasible sometimes, and therefore we approximate it as,

$$h(\pi_i) \approx cost(R_i) + h(\pi_{i+1})$$

Suppose a partial plan π_0 requires minimum k refinements (only resolution of unsupported causal links) as shown in Figure 4.1. In this figure, each refinement R_i to π_{i+1} in the series is the best possible refinement (locally $i.e.$ π_{i+1} is best child among its siblings for the parent π_i). This estimation of the best candidate is based on the heuristic values. The smaller heuristic estimate for a candidate partial plan suggests that it requires lesser number of total possible refinements for the partial plan to be a solution plan. Before resolving an unsupported causal link we make sure that there is no threat in π_i. In Figure 4.1, π_{sol} shows a solution plan (a partial plan with no flaws).

We give a definition (Definition 4.2.1) below which captures the total error committed by a heuristic function (h). The new heuristic estimation, h^e: expected to be a stronger version of h, in the definition is supposed to have more realistic estimate as compared to h. Our adaptation captures more essential details about the single-step-error correcting approach in the POCL framework. Some specific issues arise in the POCL framework which makes this approach tedious to be employed in this framework like handling of achieving and deleting of an open condition, as explained in the beginning of this section. Such issues are not associated with state-space approaches, makes the planing process easier for the state-space planners. In this approach, we directly adapt TD learning from the reinforcement learning literature. However, the adaptation may show some side effects during the heuristic estimation, but such side effects which can affect the performance negatively have not been observed during the experimentations. In the discussion section of this chapter, we clearly mention the possibilities when this way of enhancing the power of a heuristic function can adversely affect the overall performance of a POCL planner. Next, we give a *definition*, a formula which is used for calculating the overall error associated with a heuristic function. We discuss about this by keeping Figure 4.1 in mind.

Definition 4.2.1. *For a given heuristic function (h) and partial plan (π_i) which leads to the solution plan (π_{sol}) after certain refinement steps, the enhanced version of the heuristic (h^e) is computed as,*

$$h^e(\pi_i) \;=\; h(\pi_i) \;+ \sum_{\pi' \text{ from } \pi_i \rightsquigarrow \pi_{sol}} \epsilon_{h(\pi')} \tag{4.1}$$

where $\pi_i \rightsquigarrow \pi_{sol}$ is a path that considers refining each partial plan (π') generated along the path between π_i and π_{sol}. The path includes π_i and excludes π_{sol}. The term ϵ_h is single-step-error associated with h during refinement.

Now, using the Principle of Mathematical Induction, we show how effectively the total *single-step-error* associated with a given heuristic function *h*, is captured. This also justifies why this update rule should be used.

Basis: We assume that π_i needs only one refinement to become π_{sol} that would also be π_i's best child. Here, the best child always keeps the lowest estimate of requirement of new actions for its refinement among its siblings. One possible refinement of π_i is, $\pi_i \xrightarrow{R_i} \pi_{sol}$. Using Eq. 4.1, we say,

$$h^e(\pi_i) \quad = \quad h(\pi_i) \; + \; \epsilon_{h(\pi_i)} \tag{4.2}$$

The term $\epsilon_{h(\pi_i)}$ is the single-step-error associated with *h* that estimates the total effort required for π_i to refine it completely. For unit cost refinements (cost(R_i) = 1), $\epsilon_{h(\pi_i)}$ is computed as,

$$\epsilon_{h(\pi_i)} = (cost(R_i) + h(\pi_{i+1})) - h(\pi_i) \tag{4.3}$$

Here, the partial plan π_{i+1} is also the π_{sol}, therefore $h(\pi_{i+1})$ = 0. By using Eq. (4.2) and Eq. (4.3) together, we get $h^e(\pi_i)$ = 1. Therefore, the base step holds.

We assume that after refinement step R_i, to be the best child, there is no unsupported causal link present in π_{i+1} and a threat (if any) will be resolved by the planner immediately to make it a solution plan. If there is an unsupported causal link then there must be an existing action in π_{i+1} to support it. Here the estimate for new resolvers is still be 0.

Inductive step: We select an arbitrary π_{i+1} and assume that Eq. (4.1) holds for it, and we show that Eq. (4.1) holds for π_i too.

$$\begin{aligned} h^e(\pi_i) &= cost(R_i) + h^e(\pi_{i+1}) \\ &= cost(R_i) + h(\pi_{i+1}) + \sum_{\pi' \text{ from } \pi_{i+1} \rightsquigarrow \pi_{sol}} \epsilon_{h(\pi')} \\ &\qquad\qquad \text{...by the induction hypothesis.} \\ &= h(\pi_i) + \epsilon_{h(\pi_i)} + \sum_{\pi' \text{ from } \pi_{i+1} \rightsquigarrow \pi_{sol}} \epsilon_{h(\pi')} \text{ by Eq. (4.3).} \\ &= h(\pi_i) + \sum_{\pi' \text{ from } \pi_i \rightsquigarrow \pi_{sol}} \epsilon_{h(\pi')} \end{aligned}$$

Therefore, the the relationship holds for the parent partial plan π_i as well. Thus, by induction for all partial plans π, our assumption is correct.

A General Formula for Heuristic Enhancement

In Definition 4.2.1, $h^e(\pi_i)$ is an online approximated version of $h(\pi_i)$. This approximation uses the parent and the best child relationships to measure the step-error of each refinement in the path and correct it for the evaluations of further refinements.

Using Definition 4.2.1 and Figure 4.1, we calculate the average-step-error associated with h, denoted by ϵ_h^{avg}, along the path from π_i to π_{sol} as,

$$\epsilon_h^{avg} = \sum_{\pi' \text{ from } \pi_i \rightsquigarrow \pi_{sol}} \epsilon_{h(\pi')} \Big/ h^e(\pi_i) \tag{4.4}$$

Rewriting Eq. (4.4),

$$\sum_{\pi' \text{ from } \pi_i \rightsquigarrow \pi_{sol}} \epsilon_{h(\pi')} = \epsilon_h^{avg} \times h^e(\pi_i) \tag{4.5}$$

Using Eq. (4.5), Eq. (4.1) simplifies to,

$$h^e(\pi_i) = h(\pi_i) + \epsilon_h^{avg} \times h^e(\pi_i) \tag{4.6}$$

Further simplification of Eq. (4.6) yields,

$$h^e(\pi_i) = h(\pi_i) \Big/ (1 - \epsilon_h^{avg}) \tag{4.7}$$

Another possible expansion, using infinite geometric progression, of Eq. 4.7 would be,

$$h^e(\pi_i) = h(\pi_i) \times (1 - \epsilon_h^{avg})^{-1} \tag{4.8}$$

Therefore,

$$h^e(\pi_i) = h(\pi_i) \times \sum_{i=0}^{\infty} (\epsilon_h^{avg})^i \quad : \text{if } |\epsilon_h^{avg}| < 1 \tag{4.9}$$

We use RegPOCL to test the effectiveness of $h^e(\pi_i)$ in the POCL framework, where it selects the best partial plan. Here, Eq. 4.8 shows a general formula which can be used for an enhancement in the informativeness of a heuristic function. In this chapter, we demonstrate the results obtained by using this equation for enhancing h_{add} and h^{r}_{add} heuristics, as they are the state-of-the-art non temporal POCL heuristics and improvements over these heuristics are sought. We also employ this approach to enhance the informativeness of the offline learned predictive models, discussed in the last section of the previous chapter. First, we see the performance of online tuning approach on general POCL heuristics.

4.3 Online Tuning of Non-Temporal POCL Heuristics

First, we discuss the additive heuristic (h_{add}) and its adaptation to the POCL framework.

4.3.1 $h^{e}_{add}(\pi)$: Improved Additive Heuristic

We have discussed h_{add} in Section 2.3.2. On the lines of previous discussion, the additive heuristic (Bonet *et al.*, 1997) adds up the steps required by each individual open goal. The assumption of subgoal independence that it makes has worked well in many domains. However in many other domains it has a tendency to overestimate the cost. The heuristic value is estimated recursively using the recursive relation given below. Suppose p is a proposition and $A(p)$ is a set of grounded actions that produce p. Then,

$$h_{add}(p) = \begin{cases} 0 & \text{If p unifies with an atom in I;} \\ min_{a \in A(p)} h_{add}(a) & \text{If } A(p) \neq \phi; \\ \infty & \text{Otherwise.} \end{cases}$$

The cost of an action is calculated by the formula given below,

$$h_{add}(a) = 1 + h_{add}(precond(a))$$

The $h_{add}(\pi)$ for POP which is given by Younes and Simmons (2003) is defined as:

$$h_{add}(\pi) = \sum_{\xrightarrow{p} a_j \in OC(\pi)} h_{add}(p)$$

an action a_p produces a proposition *p* and conditioned by *r*, so $h_{add}(r)$ will be added to the cost of a_p.

The additive heuristic often overestimates the actual distance-to-go due to the nature of its design. h_{add} often commits significant error in the actual estimation at each refinement stage in the POCL framework. Using the online heuristic tuning approach, we further enhance the additive heuristic. We apply h_{add} in Eq. 4.9, therefore the new estimate generated is represented as h^e_{add}. The improved additive heuristic based on is,

$$h^e_{add}(\pi_i) = \frac{h_{add}(\pi_i)}{1 - \epsilon^{avg}_{h_{add}}} \tag{4.10}$$

This is an inadmissible heuristic like h_{add}, but it is more informed as compared to h, where $\epsilon^{avg}_{h_{add}}$ (given below in Eq. 4.12) is the average-step-error associated with h_{add}. In a similar way, we capture the single-step-error associated with the h_{add} as,

$$\epsilon_{h_{add}(\pi_i)} = (cost(R_i) \; + \; h_{add}(\pi_{i+1})) \; - \; h_{add}(\pi_i)$$

where $\epsilon_{h_{add}(\pi_i)}$ is either positive or negative since h_{add} is an inadmissible heuristic. We approximate the total effort required to refine π_i as,

$$h^e_{add}(\pi_i) = h_{add}(\pi_i) \; + \sum_{\substack{\pi^* \text{ from} \\ \pi_i \rightsquigarrow \pi_{sol}}} \epsilon_{h_{add}(\pi^*)} \tag{4.11}$$

where $\epsilon^{avg}_{h_{add}}$ is an estimate of average-step-error committed by h_{add}, that is defined as,

$$\epsilon^{avg}_{h_{add}} = \sum_{\substack{\pi^* \text{ from} \\ \pi_i \rightsquigarrow \pi_{sol}}} \epsilon_{h_{add}(\pi^*)} \Big/ h^e_{add}(\pi_i) \tag{4.12}$$

In this derivation, we skip some of the important intermediate steps and they have already been shown in the general formula in detail. We refer readers to the previous section in this chapter for a detailed understanding of the formulas derived in Eq. 4.8. Simplification by using Eq. 4.11 in Eq. 4.12, we get an inadmissible but improved estimate for a given heuristic function (*e.g.* additive heuristic in this case) for POCL planning,

$$h^e_{add}(\pi_i) \quad = \quad h_{add}(\pi_i) \; + \; \epsilon^{avg}_{h_{add}} \times h^e_{add}(\pi_i)$$

$$\begin{aligned} h^e_{add}(\pi_i) &= \frac{h_{add}(\pi_i)}{1-\epsilon^{avg}_{h_{add}}} \qquad \text{(after simplification)} \\ &= h_{add}(\pi_i) \times \sum_{i=0}^{\infty} \left(\epsilon^{avg}_{h_{add}}\right)^i \end{aligned}$$

We use h^e_{add} as an improved alternative in place of h_{add}.

4.3.2 $h^{r,e}_{add}$: Improved Positive Interaction Heuristic

On the lines of the discussion in Section 2.3.3, we begin with a brief discussion of accounting for positive interaction heuristic which is represented as h^r_{add} Younes and Simmons (2003). Younes and Simmons (2003) address the positive interactions among subgoals while ignoring the negative interactions. This estimation technique is used (as a variant of h_{add} (Haslum and Geffner, 2000)) for ranking the partial plans for the first time by Younes and Simmons (2003), which is,

$$h^r_{add}(\pi) = \sum_{\xrightarrow{p} a_j \in OC(\pi)} \begin{cases} 0 & \exists a_k \in A \text{ such that an} \\ & \text{effect of } a_k\text{, unifies with} \\ & p \text{ and } a_j \prec a_k \notin O; \\ h_{add}(p) & \text{Otherwise.} \end{cases} \tag{4.13}$$

In Eq. 4.13, $h^r_{add}(\pi)$ is the substitute for $h_{add}(\pi)$ as the latter has no provision for actions reuse. The underlying principle of POCL planning is that the resolver already present in the partial plan will used, if it is consistent.

We consider single-step-error associated with accounting for positive interaction heuristic because it is one of the poor performers in gripper domain (Younes and Simmons, 2003). The direct expression, similar to the definition of h^e_{add} (Eq. 4.10), is,

$$h^{r,e}_{add}(\pi_i) = cost(R_i) + h^{r,e}_{add}(\pi_{i+1})$$

We use $h^{r,e}_{add}$ as the other improved inadmissible heuristic function close to h^*, that is,

$$h^{r,e}_{add}(\pi_i) = \frac{h^r_{add}(\pi_i)}{1-\epsilon^{avg}_{h^r_{add}}} \tag{4.14}$$

where the term $\epsilon^{avg}_{h^r_{add}}$ (similar to $\epsilon^{avg}_{h_{add}}$ as in Eq. 4.12) is the average single-step-error

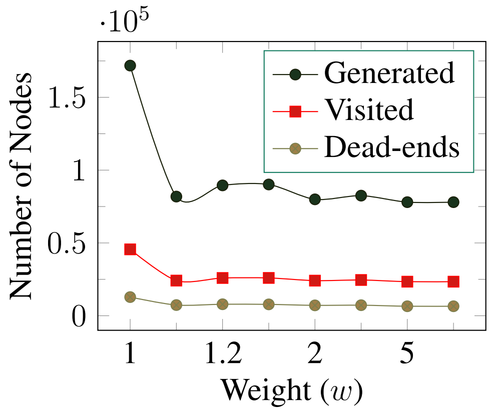
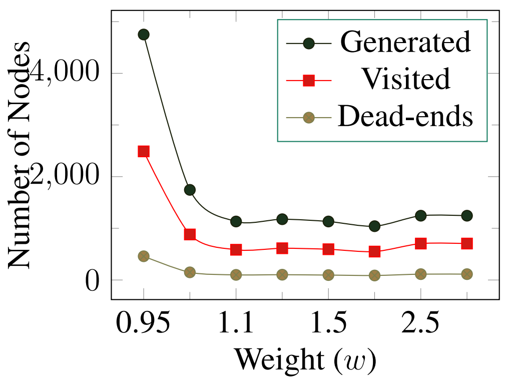
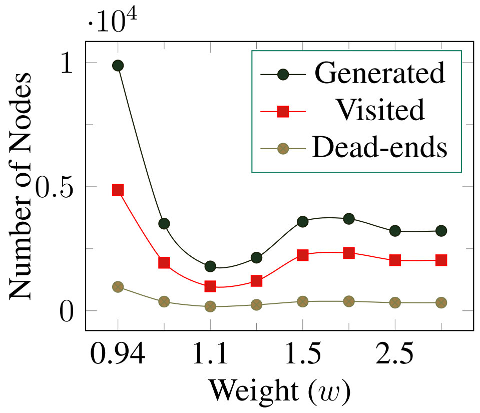


Figure 4.2: Performance of wA^* algorithm in gripper domain from IPC-1. The graph shows the behavior of h^e_{add} across different weights. MW-Loc is used as the flaw selection heuristic.

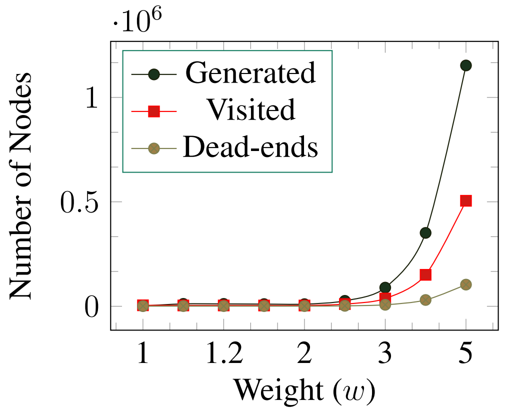


(a) Using h^e_{add} heuristic

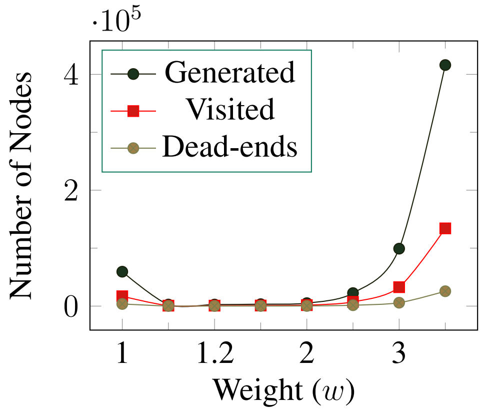


(b) Using $h^{r,e}_{add}$ heuristic

Figure 4.3: Performance of wA^* algorithm in logistics domain from IPC-2. The figure 4.3a shows the behavior of h^e_{add} and the figure 4.3b shows the behavior of $h^{r,e}_{add}$. MC-Loc is used as flaw selection heuristic.



(a) Using h^e_{add} heuristic



(b) Using $h^{r,e}_{add}$ heuristic

Figure 4.4: Performance of wA^* algorithm in driverlog domain from IPC-3. The figure 4.4a shows the behavior of h^e_{add} and the figure 4.4b shows the behavior of $h^{r,e}_{add}$ across different weights. MW-Loc is used as flaw selection heuristic.

Instance	h^r_{add} with effort						$h^{r,e}_{add}$ with effort					
	Algo	Partial Plans					Algo	Partial Plans				
		Gen	Vis	PL	CPU	DE		Gev	Vis	PL	CPU	DE
gripper-06	-	–	–	–	–	–	(IDA)	(1.5K)	(880)	(15)	(12)	(183)
gripper-08	-	–	–	–	–	–	(wA)	(1.8K)	(1.0K)	(15)	(20)	(208)
	h_{add} with effort						h^e_{add} with effort					
gripper-06	wA	493	243	19	4	51	HC	326	142	15	3	21
gripper-08	IDA	1.1K	605	21	12	153	IDA	640	274	21	4	54
gripper-12	IDA	3.4K	2.0K	43	40	612	(IDA)	(1.9K)	(767)	(33)	(24)	(195)
gripper-20	IDA	20K	11K	79	365	3.7K	(IDA)	(8.7K)	(3.3K)	(59)	(168)	(1.0K)

Table 4.1: Performance comparison in Gripper domain from first International Planning Competition (IPC). MW-Loc is used for flaw selection during planning process. The top half of the table shows performance for h^r_{add} and $h^{r,e}_{add}$ *with effort*. Only first 8 problems were solvable. And second half is for h_{add} and h^e_{add} *with effort*. *Algo* indicates best of A, wA, IDA and HC. *Gen* is nodes generated, *Vis* is nodes visited, *PL* is plan length, *CPU* is execution time in milliseconds, *DE* is dead-ends. **Overall** best results are shown in parentheses. **Dash** indicates no solution was found within spoverecified limits. Tables similar to this have condensed captions further in this chapter.

Instance	h^r_{add} with effort						$h^{r,e}_{add}$ with effort					
	Algo	Partial Plans					Algo	Partial Plans				
		Gen	Vis	PL	CPU	DE		Gen	Vis	PL	CPU	DE
logistics-a	IDA	317	174	51	76	13	IDA	308	169	51	76	13
logistics-b	HC	244	127	42	88	8	wA	243	130	42	88	8
logistics-c	IDA	307	158	50	116	9	IDA	309	159	50	116	9
logistics-d	IDA	1.4K	678	73	420	68	(IDA)	(654)	(338)	(70)	(412)	(42)
	h^r_{add} without effort						$h^{r,e}_{add}$ without effort					
logistics-a	IDA	1.8K	1.4K	59	108	172	(IDA)	(634)	(463)	(51)	(72)	(61)
logistics-b	IDA	1.4K	904	48	104	113	(A)	(826)	(467)	(43)	(100)	(71)
logistics-c	IDA	1.7K	1.1K	57	260	138	(IDA)	(1.0K)	(764)	(52)	(136)	(108)
logistics-d	-	–	–	–	–	–	(wA)	(3.5K)	(1.9K)	(70)	(484)	(367)

Table 4.2: Performance comparison in logistics domain from IPC-2. MC-Loc and MW-Loc are used for flaw selection in the top half and bottom half of this table respectively. The top half of the table shows performance for h^r_{add} and $h^{r,e}_{add}$ *with* effort. And second half is for h^r_{add} and $h^{r,e}_{add}$ *without* effort.

associated to the additive heuristic with actions reuse. The term $h^{r,e}_{add}$, is derived in a way similar to the derivation of h^e_{add}.

In the next section, we use the two improved heuristics mentioned in Eq. 4.10 and Eq. 4.14 for comparison with h_{add} and h^r_{add} respectively.

4.3.3 Experimental Evaluation

We describe the performance of RegPOCL using these heuristics. The planner uses fully grounded actions and some existing heuristics along with their improvements proposed by us. For the flaw selection, we use the heuristics from (Younes and Simmons,

Instance	h_{add} with effort						h^e_{add} with effort					
	Algo	Partial Plans					Algo	Partial Plans				
		Gen	Vis	PL	CPU	DE		Gen	Vis	PL	CPU	DE
P-15	(wA)	(1.1K)	(702)	(44)	(12)	(44)	wA	1.3K	826	43	20	45
P-25	A	2.3K	983	35	60	10	wA	2.3K	933	34	48	34
P-30	wA	84K	57K	162	8K	20K	(HC)	(16K)	(9.5K)	(130)	(632)	(920)
P-35	wA	447K	351K	466	210K	102K	(wA)	(186K)	(118K)	(377)	(38K)	(17K)
P-40	-	-	-	-	-	-	(wA)	(93K)	(66K)	(325)	(9.5K)	(1.8K)

Table 4.3: Performance comparison in rovers domain from IPC-5. MW-Loc is used for flaw selection during planning process. This table shows performance for h_{add} and h^e_{add} *with* effort. The selection of heuristics is dependent on the performance of the base heuristics. We can see that only h_{add} has been selected here unlike the previous tables, this is because h^r_{add} and its enhancement are not helpful in solving enough number of problems.

2003). LIFO is used for breaking ties selection of partial plan during the planning process.

Environment

We perform the experiments on Intel Core 2 Quad with 2.83 GHz 64-bit processor and 4GB of RAM. These experimental results on planning domains from different international planning competitions (IPC) are the basis for claims made. The domains are: Gripper, Logistics, Driverlog, Rovers , and Storage. We present the graphical analysis for gripper, logistics and driverlog domains since the existing heuristics are not very successful. We use an upper limit of $1,000,000$ on the number of nodes generated, apart from a time limit of 900 seconds.

Experimentation

The results (Tables 4.1 to 4.4) demonstrate that the planner becomes more informed as search progresses. During the evaluation, we consider non-temporal only STRIPS style problem instances as we are improving upon the heuristics involved in solving non-temporal STRIPS style problems effectively in the POCL literature. The tables contain different sized planning problems and the effort required by the planner to find the plans.

We employed four well known search algorithms - A* (A), wA* (wA), Iterative Deepening A* (IDA), Hill Climbing (HC). These table bring out the pros and cons

Instance	h^r_{add} without effort						$h^{r,e}_{add}$ without effort					
		Partial Plans						Partial Plans				
	Algo	Gen	Vis	PL	CPU	DE	Algo	Gen	Vis	PL	CPU	DE
P-06	-	–	–	–	–	–	(IDA)	(1.3K)	(1.2K)	(8)	(40)	(304)
P-09	-	–	–	–	–	–	(IDA)	(40K)	(18K)	(11)	(1.9K)	(3.7K)
	h_{add} with effort						h^e_{add} with effort					
P-05	IDA	2.9K	1.8K	13	44	349	(IDA)	(836)	(523)	(11)	(16)	(101)
P-06	IDA	60K	47K	12	1.7K	9.7K	(IDA)	(1.2K)	(701)	(12)	(28)	(152)

Table 4.4: Performance comparison in storage domain from IPC-5. MW-Loc is used for flaw selection during planning process. The first part of the table shows performance for h^r_{add} and $h^{r,e}_{add}$ *without* effort. And the second part demonstrates for h_{add} and h^e_{add} *with* effort.

of the adapted approach in POCL planning. We found some volatile behavior of the wA algorithm during evaluation, and further investigated it with improved heuristics in gripper, logistics and driverlog domains. Some interesting results can be seen in Fig. 4.2 to Fig. 4.4. We consider four heuristics: h_{add} and h^r_{add} with their respective improvements: h^e_{add} and $h^{r,e}_{add}$. They are evaluated *with effort* and *without effort*. The usage of a heuristic with effort means we consider the number of preconditions as the cost of a unit cost action. For example, suppose an action has four preconditions and all are present in the initial state. A heuristic with effort will estimate the action cost 4 instead of 0. This ends-up with good estimates sometimes as for refining a partial plan we look for an action resolver for each *OC*. If interaction between actions is very less, heuristic with *effor* gives a very good estimate. The estimates MC, MW, MC-Loc and MW-Loc are used for the flaw selections (Younes and Simmons, 2003).

Table 4.1 shows the evaluation in the gripper domain. The first half of the table depicts the performances of h^r_{add} and $h^{r,e}_{add}$ *with effort*. For larger problems a heuristic gets more opportunity to see the overall global *average-step-error*. For instances like gripper-06 and gripper-08, the new heuristic has performed well whereas its counterpart is not able to solve these instances. The second part of Table 4.1 demonstrates the performance of h_{add} and h^e_{add} *with effort*. On instances from gripper-10 to gripper-20 (only gripper-12 and gripper-20 are shown in the table), the improved heuristic comprehensively outperforms the h_{add}. However, in Table 4.5, we also demonstrate the overall performance of the POCL planner considering all the problems in an aggregate manner. In Table 4.1, for the last two instances IDA has come up with the best results. Performance of both the current and improved versions of h^r_{add} either *with effort* or *without effort* are incomparable (as the performance is very dull) to the results mentioned in

IPC Domain	Instances	Improved Heuristics (h^e)				Current Heuristics (h)			
		Solved	Average			Solved	Average		
			Gen	Vis	CPU		Gen	Vis	CPU
gripper	20	20	(26K)	(8.3K)	(840)	20	51K	27K	1.4K
logistics	75	(72)	14K	1.8K	66K	61	5.3K	2.3K	57K
rovers	60	(60)	50.2K	32.6K	(8.5K)	57	37.8K	29.3K	10.1K
storage	30	(9)	17K	9.5K	337	5	6.1K	3.3K	126

Table 4.5: The performance of the planner is aggregated (average) here in all the selected domains. We follow the same notion for best results and other specifications as per the previous tables. This table shows the best performance of the most improved version of the base heuristics (h^e) corresponding to the best performance from the base heuristics used (h). Table 4.3 has 40 problems, but here rovers domain has 60. This shows that for aggregation, planning problems in this domain have been selected from more than one IPC.

this table, therefore they are not shown. Fig. 4.2 shows the behavior of wA algorithm with h^e_{add}. It shows that the number of generated nodes, visited nodes and the dead-ends becomes constant with the large weights. The best solution is found at *weight* = 1.10.

Table 4.2 shows the experiments performed in the logistics domain. First part of this table shows the performance of h^r_{add} and $h^{r,e}_{add}$ *with effort*. We do not see much improvement when the improved versions are used though the effort required from the planner is less for the instance logistics-d. It takes reasonable time to perform the search to solve logistics-d. The second part shows the performance of h^r_{add} and $h^{r,e}_{add}$ *without effort*. Performance of $h^{r,e}_{add}$ for the partial plan selection, shows good results compared to h^r_{add}. It gives shorter length plans, while also being competitive on the time required. The estimate $h^{r,e}_{add}$ solves instance logistics-d (minimum plan length = 70) with exploring approximately half the search space explored by h^r_{add} for solving the instance logistics-c (minimum plan length = 57). Fig. 4.3 shows the behavior of wA algorithm with h^e_{add} and $h^{r,e}_{add}$ respectively. Subgraphs show that the search speed becomes constant with large weight factors.

Table 4.3 shows the evaluations performed in the rovers domain. The performances of the heuristics h_{add} and h^e_{add} *with effort* are shown in this table. The heuristic h^e_{add} solves all 40 problem instances though h_{add} solves 37 instances as well. Table 4.4 shows the experiments performed in the storage domain. First part of this table, shows the outcomes of h^r_{add} and $h^{r,e}_{add}$ without effort. It indicates that taking care of *average-*

step-error associated with underperformed heuristics into an account, can solve a few bigger sized problems. The second part shows the performances of h_{add} and h^{e}_{add} with effort. The improved heuristic has enhanced the effectiveness of the planner as well. Specifically, it take very less time to reach to π_{sol} (the solution plan). IDA is good for both instances but it takes more time.

Fig. 4.2 to 4.4 show the behavior of wA algorithm using h^{e}_{add} and $h^{r,e}_{add}$ respectively, in the driverlog domain from IPC-3. The subgraphs depict that search becomes very difficult even for the smaller values of *w*. In this domain, state-of-the-art heuristics are among the worst performers along with one of the improved heuristic h^{e}_{add}, though the other improved heuristic $h^{r,e}_{add}$ solves 13 problem instances out of 20. The state-of-the-art heuristics do outperform in some domains but not on the number of solved problems. More observations can be made directly from Tables 4.1 to 4.4, excluded here from the discussion. In Table 4.5, we aggregate the overall performance of the planner. We consider each domain which has been considered individually too. We observe that the improved versions have beaten state-of-the-art heuristics in these domains.

In this section, we discussed an online heuristic tuning approach. We give a general formula to enhance the informativeness of a heuristic function in plan space planning. We demonstrated some experimental results that show the efficiency of the tuning approach. Considering the results shown by this approach, we employ it to enhance the learned predictive models. Note that we have already discussed the approaches of generating offline learned models in the previous chapter. Where we show empirical evaluations on certain benchmarks. The results obtained were quite convincing.

We had skipped a major evaluation of the offline regression approaches employed in the previous chapter. Therefore, in the next section, we further test the offline learned models along with their enhanced version achieved using the general formula (4.9). Then we compare the results obtained using learned predictive models and their enhanced versions. We also compare these evaluations with some latest state-of-the-art strong heuristics from state-space planning.

4.4 Evaluation: Predictive Models and Their Tuning

In the subsection 3.3.6, we saw the effectiveness of leaned predictive models that are learned using supervised learning. We tested the approach in 7 different domains from various IPCs. In the previous section, we demonstrated that the informativeness of a given heuristic function can be enhanced using an online tuning approach which corrects the error committed by that heuristic. This has also been tested in several domains.

Next, we work for improving the performance of the offline learned predictive models using the online heuristic error tuning approach. In this section, we project a learned predictive model as a given heuristic (h^l), and its enhanced version using our tuning approach as $h^{l,e}$. As we stated earlier, these improvements are only employed to select the most suitable partial plan from the set of partial plans. Whilst we use MC-Loc and MW-Loc (Younes and Simmons, 2003) as flaw selecting heuristics for refining a partial plan. They give higher preference to the local flaws present in the partial plan. We employ Greedy Best First Search algorithm for selecting the next partial plan for refinement.

4.4.1 Environment

We perform the experiments on Intel Core 2 Quad with 2.83 GHz 64-bit processor and 4GB of RAM. To evaluate the effectiveness of learned models and to correct the single-step-error associated with the models, a time limit of 15 minutes and a node generation limit of 1 million is used.

4.4.2 Experiments

We briefly discuss the procedure used in the subsection 3.3.6, and use almost the same text. We use RegPOCL to compare the performances of the offline predictive models h^l, their corresponding enhanced models $h^{l,e}$ and the last four base features. The enhancement ($h^{l,e}$) can be obtained in a similar way using the online heuristic tuning approach. For example: the way we obtain h^e_{add} and $h^{r,e}_{add}$ in the previous section. These are also compared with some of the recent effective state-space based heuristics and approaches that are introduced later. We exclude the first two base features from comparison since they are weak heuristics and RegPOCL does not solve sufficient problems using them. However, they are useful while working jointly with other informed heuristics. Next, we discuss the observations made during the training phase.

Training

Using Algorithm 1, we prepare datasets and learn different predictive models by applying the various regression techniques discussed earlier. We select each of the last four features to solve a set of problems. The target value is the plan length found by RegPOCL using the base features. The dataset preparation phase took less than two hours on an average in each domain, for each of the four features. Once we have enough instances, we begin the training process. We define a regression model $\mathrm{R} : \mathrm{T} \rightarrow \mathbb{R}$, where T is a training set and $\mathbb{R}$ is a set of real numbers. Note that, in Figure 4.1, different heuristics for calculating target values will prefer different paths. Therefore, the four base features will generate four different datasets. The attribute selection strategy allows us to select a subset of attributes in the training set by removing correlated and redundant features. We learn a total of 70 (7 domains $\times$ 2 datasets $\times$ 5 regression techniques) regression models. The training phase took 20 ms (milliseconds) using LR, 270 ms using LMS, 600 ms using MLP, 82 ms using M5Rule, and 58 ms using M5P on an average per model. All the learned models have high accuracy but LR is the fastest, followed by M5P and M5Rule. Next, we test these models on different benchmarks.

Testing

We test the effectiveness of our approaches by selecting partial plans (π) for refinement using RegPOCL. We assume that an informed heuristic leads to minimal possible refinements needed for π. Next, for the comparison we compute score[1] as in IPC for satisficing track problems. The better score on each standard signifies the better performance. We compare performance of the learned models with the selected base features h_{add}, $h_{add,w}$, h^r_{add}, and $h^r_{add,w}$. The comparison is done on the basis of (*i.*) the number of solved problems, and (*ii.*) the score obtained on plan quality, execution time, nodes (partial plan) visited, and makespan quality.

[1]`https://helios.hud.ac.uk/scommv/IPC-14/`

For example, the offline learned model $h^{r,l}_{add}$ is learned on a dataset prepared using h^{r}_{add}. In other words, RegPOCL uses h^{r}_{add} for calculating the target values in the dataset. Here, $h^{r,l}_{add}$ can be enhanced to $h^{r,l,e}_{add}$ using the online heuristic tuning approach which is expected to be more informed than $h^{r,l}_{add}$. It is similar for other learned heuristics. Please note that $h^{r,l,e}_{add}$ is obtained by employing the general tuning approach mentioned in Eq. 4.8. These models are applied in the POCL framework for selecting the most suitable partial plan, followed by the heuristic MW-Loc (Younes and Simmons, 2003) for selecting the most adverse flaw in it .

We also compare our approaches with state-space based approaches on the basis of the number of problems solved, and score obtained on plan quality and total execution time. We select fast forward heuristic (FF) (Hoffmann and Nebel, 2001), context-enhanced additive heuristic (CEA) (Helmert and Geffner, 2008), and landmark-cut heuristic (LM-Cut) (Helmert and Domshlak, 2011). We also use these heuristics together by applying them in multi-heuristic first solution strategy (MHFS) (Röger and Helmert, 2010). In general, the strategy performs better with alternating usage of different heuristics instead of combining them. We also compare the effectiveness of our techniques with LAMA11 (Richter *et al.*, 2011); the winner of IPC-2011 in the sequential satisficing track. LAMA11 applies FF and LM-Count (Richter *et al.*, 2008) heuristics together using multi-queue search. We set a 20 minutes time limit while evaluating LAMA11 over these domains, since it has an internal time limit of 5 minutes for the invariant synthesis part of translator. All the state-based approaches are evaluated using Greedy Best First Search algorithm in the fast downward planning system (Helmert, 2006). We use "eager" and "greedy" types of evaluations with no preferred operators. The regression models selected in the chosen domains are trained using, (*i*) M5P in Gripper-1 and Elevator-2, (*ii*) LR in Rovers-3, Rovers-5, Logistics-2, and Zenotravel-3, and (*iii*) M5Rule in Logistics-1.

Note that we now evaluate the enhanced versions of the models learned using supervised approaches that are discussed in the previous chapter. We further discussed the evaluations of those offline learned predictive models from Table 3.5 and Table 3.6. Here, we again consider those results to get a clearer and easier overall evaluations of the models learned using offline learning from multiple inadmissible heuristics, followed by the online error tuning which enhances the informativeness of those models. In Table 4.6, we compare (*i*) the base features, (*ii*) offline learned models and their

Domain	**#**	**POCL Heuristics**								**Evaluation using Fast Downward (FD)**								**LAMA**
		State-of-the-art				**Via learning approaches**				**Lazy**				**Eager**				
		h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	h^l_{add}	$h^{l,e}_{add}$	$h^l_{add,w}$	$h^{l,e}_{add,w}$	FF	CEA	LM-Cut	MHFS	FF	CEA	LM-Cut	MHFS	
Gripper-1	20	16	**20**	1	1	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Rovers-3	20	19	19	**20**	**20**	**20**	**20**	**20**	**20**	18	17	14	**20**	19	18	15	**20**	**20**
Rovers-5	40	28	31	32	36	$^+$39	$^+$39	$^+$36	$^+$39	22	25	15	28	24	29	17	30	**40**
		h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	$h^{r,l}_{add}$	$h^{r,l,e}_{add}$	$h^{r,l}_{add,w}$	$h^{r,l,e}_{add,w}$	FF	CEA	LM-Cut	MHFS	FF	CEA	LM-Cut	MHFS	
Logistics-1	35	25	1	32	28	$^+$32	$^+$32	22	$^+$32	25	34	21	28	28	34	16	25	**35**
Elevator-2	150	148	14	**150**	58	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**	**150**
Logistics-2	40	36	12	36	34	**40**	**40**	**40**	**40**	36	36	35	36	36	36	36	36	**40**
Zenotravel-3	20	5	4	9	10	$^+$16	$^+$16	$^+$16	$^+$16	**20**	**20**	17	**20**	**20**	**20**	16	**20**	**20**
Coverage	325	277	101	278	187	$^+$317	$^+$317	$^+$304	$^+$317	291	302	262	302	297	307	260	301	**325**

Table 4.6: Number of solved problems using each heuristic. **#** is the number of problems in each domain. State-of-the-art POCL heuristics are compared with the learned ones in the left half. The state-based heuristics FF, CEA, and LM-Cut and their combination using MHFS strategy are also compared with POCL heuristics. The last column captures the performance of LAMA11. Best results are shown in **bold**, and a number with "+" mark (*e.g.*$^+$36) shows competitive performance by the learned models and their enhancements over each base heuristic. We give the total score (*coverage*) in the last row of each table for making the overall comparison easier. For a note, the numerical values with the domain names in the table depict the IPC. Similar representations are followed in other tables.

Domain	POCL Heuristics								Evaluation using Fast Downward (FD)								LAMA
	State-of-the-art				Via learning approaches				Lazy				Eager				
	h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	h^l_{add}	$h^{l,e}_{add}$	$h^l_{add,w}$	$h^{l,e}_{add,w}$	FF	CEA	LM-Cut	MHFS	FF	CEA	LM-Cut	MHFS	
Gripper-1	16.0	14.9	0.7	0.7	**20.0**	**20.0**	**20.0**	**20.0**	15.45	14.9	15.5	15.5	15.5	14.9	15.5	15.5	**20.0**
Rovers-3	17.3	16.2	18.1	16.9	17.8	$^{+}$18.6	17.8	$^{+}$18.6	17.1	15.9	12.5	18.8	18.1	16.9	13.9	19.0	**19.8**
Rovers-5	25.7	26.3	28.0	30.2	$^{+}$33.1	$^{+}$36.3	30.1	$^{+}$33.6	20.8	23.4	13.5	26.0	22.7	27.3	15.7	28.3	**39.8**
	h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	$h^{r,l}_{add}$	$h^{r,l,e}_{add}$	$h^{r,l}_{add,w}$	$h^{r,l,e}_{add,w}$	FF	CEA	LM-Cut	MHFS	FF	CEA	LM-Cut	MHFS	
Logistics-1	24.3	0.8	31.2	26.1	$^{+}$31.2	$^{+}$31.2	20.9	**32.0**	23.7	30.3	19.7	27.4	27.0	30.8	15.1	24.6	30.8
Elevator-2	142.8	11.8	144.9	49.7	142.7	$^{+}$144.9	142.7	$^{+}$144.9	117.1	112.0	117.1	116.9	144.1	136.0	144.2	141.0	**148.2**
Logistics-2	33.8	10.7	34.9	30.7	**38.1**	$^{+}$36.3	**38.1**	$^{+}$36.3	33.72	29.5	32.4	33.3	34.1	30	33.7	35.6	37.8
Zenotravel-3	4.6	3.7	8.5	8.8	$^{+}$13.5	$^{+}$13.7	$^{+}$13.5	$^{+}$13.7	17.3	17.7	14.7	18.6	18.1	14.3	16	18.8	**19.2**
Quality Score	264.5	84.4	266.3	163.1	$^{+}$296.4	$^{+}$301	$^{+}$283.1	$^{+}$299.1	245.2	243.7	225.4	256.5	279.6	270.2	254.1	282.8	**315.6**
Time Score	204.4	76.5	197.7	148.6	$^{+}$272.9	$^{+}$260.8	$^{+}$258.6	$^{+}$264.3	255.6	**280.0**	228.0	233.8	259.6	271.3	223.0	222.5	137

Table 4.7: Scores on plan quality and overall time. We compare state-of-the-art POCL heuristics with learned ones. The effectiveness of the POCL heuristics is compared with some latest state based approaches. The second last row *sums up* the *total quality score* and the last row demonstrates the overall *time score* of each heuristic. Our approaches have performed well. Note that the numerical values with the domain names in the table depict the IPC.

Domain	**State-of-the-art**				**Via learning approaches**			
	h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	h^l_{add}	$h^{l,e}_{add}$	$h^l_{add,w}$	$h^{l,e}_{add,w}$
Gripper-1	7.0	14.0	0.0	0.0	$^+$18.0	**19.7**	$^+$18.0	**19.7**
Rovers-3	8.6	8.3	12.6	**19.0**	13.3	13.6	13.3	13.6
Rovers-5	8.2	14.7	13.5	30.9	**31.4**	**31.4**	24.7	**31.4**
	h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	$h^{r,l}_{add}$	$h^{r,l,e}_{add}$	$h^{r,l}_{add,w}$	$h^{r,l,e}_{add,w}$
Logistics-1	5.1	0.6	16.5	21.6	**29.9**	**29.9**	20.7	**29.9**
Elevator-2	41.4	8.2	58.9	39.7	**145.0**	$^+$135.0	**145.0**	$^+$135.0
Logistics-2	24.5	6.5	23.4	26.7	$^+$33.4	**35.3**	$^+$33.4	**35.3**
Zenotravel-3	0.0	1.0	2.4	7.7	7.4	**11.07**	7.4	**11.07**
Total Score	94.8	53.3	127.3	145.6	**278.4**	$^+$276	$^+$262.5	$^+$276

Table 4.8: Results of refinement score (the number nodes visited) of RegPOCL using each heuristic. We compare state-of-the-art heuristics with the learned models (h^l) and their further enhancements ($h^{l,e}$). The best results are in **bold**.

enhanced versions (*iii*) state-space based heuristics FF, CEA, and LM-Cut, and (*iv*) the strategies used in MHFS, and LAMA11, on the basis of number of problems solved. In this table, RegPOCL solves equal number of problems as LAMA11 in Gripper-1, Rovers-3, Elevator-2, and Logistics-2 using each of learned heuristics and their enhanced versions. The base features have performed well in some domains but are not consistent overall. In Rovers-5, our approaches solved 1 problem less than LAMA11, but they beat other state-space based competitors comprehensively. Also, each learned model has performed better than all the base features. For Logistics-2, we are competitive with LAMA11 and solve at least 4 more problems than other good heuristics like CEA and LM-Cut. In Zenotravel-3, RegPOCL solved 6 problems more by applying our approaches but loses to the state-based competitors. Our second approach improves the performance of the learned models in Rovers-5 by solving 3 more problems, and in Logistics-1 where it solves 10 more problems. This approach could not increase the coverage in other domains. LAMA11 wins on the basis of the total number of problems solved in each domain.

In Table 4.7, we compare the score obtained on plan quality by each of the base features, learned models with their corresponding enhancements, and state-space based heuristics and techniques. LAMA11 is an anytime planner which gives solution close to the optimal but takes more time to compute shorter plans. For Gripper-1 and Logistics-2, RegPOCL using the learned heuristics solves equal number of problems but finds shorter plans compared to LAMA11. In Logistics-1, RegPOCL lost to LAMA11 by 3 problems in the number of problems solved, but obtained a higher score in the other problems it

Domain	h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	h^l_{add}	$h^{l,e}_{add}$	$h^l_{add,w}$	$h^{l,e}_{add,w}$
Gripper-1	16.0	9.8	0.5	0.5	**20.0**	**20.0**	**20.0**	**20.0**
Rovers-3	**17.9**	15.5	17.8	15.8	17.4	17.7	17.4	17.7
Rovers-5	26.1	24.2	28.4	26.5	**30.5**	$^+$29.8	$^+$27.6	**30.5**
	h_{add}	$h_{add,w}$	h^r_{add}	$h^r_{add,w}$	$h^{r,l}_{add}$	$h^{r,l,e}_{add}$	$h^{r,l}_{add,w}$	$h^{r,l,e}_{add,w}$
Logistics-1	21.8	0.9	25.2	20.8	$^+$30.4	$^+$30.4	17.6	**30.5**
Elevator-2	147.1	11.7	**149.1**	50.5	**149.1**	146.8	**149.1**	146.8
Logistics-2	26.3	7.0	33.0	19.3	$^+$33.5	**35.4**	$^+$33.4	**35.4**
Zenotravel-3	3.7	2.7	7.5	7.7	$^+$12.1	**13.5**	$^+$12.1	**13.5**
Total Score	258.9	71.8	261.5	141.1	$^+$293	$^+$293.6	$^+$277.2	**294.4**

Table 4.9: Makespan quality score of the heuristics. We compare state-of-the-art heuristics with the learned models (h^l) and their further enhancements ($h^{l,e}$). The best results are in **bold**.

solved as it produces shorter plans using $h^{r,l,e}_{add,w}$, $h^{r,l}_{add}$, and $h^{r,l,e}_{add}$. The effectiveness of $h^{r,l,e}_{add,w}$ wins in this domain with best plan quality. $h^{r,l,e}_{add,w}$ also increases the plan quality score from 20.9 that is obtained by $h^{r,l}_{add,w}$ to 32.0 using our online error correcting strategy. However, in Logistics-2, the performance has decreased after further enhancements of $h^{r,l}_{add}$ and $h^{r,l}_{add,w}$.

In general, the performance of RegPOCL using either of the offline learned models and their enhancements is often better than that using the base features. In most of the cases, the online error adjustment approach has further enhanced the performance of these learned models. The last row of Table 2 gives the score obtained on total time taken by the process. If a planner takes less than 1 second for solving a problem then it gets full score. On the total time score the winner is CEA with “lazy” evaluation. The learned models and their enhanced versions have obtained better scores than other competitors except CEA. These scores are very close to the winning score and almost twice that of LAMA11.

In Table 4.8, we compare the score obtained on the number of nodes RegPOCL visits for solving the planning problems. This is obtained for the base features, the learned heuristics, and their enhanced versions. The models obtained after offline learning are more informed toward goals and refines fewer partial plans. The score obtained by the learned models is further increased by a good factor in Zenotravel-3 using the error correcting approach. For Elevator-2, the error correcting approach has shown some negative effect which continues in Table 4.9 too. In this table we demonstrate the score obtained on the makespan quality. Higher score signifies smaller makespan and more flexibility in the generated solution plan. In Elevator-2 and Rovers-5, the scores of h^{l}_{add} and $h^{r,l}_{add}$ have decreased due to the negative effects of our error adjustment approach, while the score obtained by $h^{r,l,e}_{add,w}$ is almost 1.5 $times$ the score of $h^{r,l}_{add,w}$ in Logistics-1. In general, the offline learned models have generated more flexible plans with shorter makespan than the base features. These qualities are further improved using the enhanced versions of these models.

The results demonstrated in this chapter show that most of the times the enhancement strategy based on TD learning has performed as per the expectations. The reason is that this approach improves upon the informativeness of a heuristic as the search progresses. But there are some domains in which the performance of this approach is not as per the expectations. For example, Logistics-2 domain in Table 4.7, and Elevator-2 and Rovers-5 domains in Table 4.9. In the next section, we discuss the possible drawbacks associated with this *tuning* approach. This will also cover the possible reasons that led to the adverse performance in these domains.

4.5 Drawbacks of The Error Tuning Approach

The experiments demonstrate that the online tuning approach does not always enhance the overall performance of the offline learned predictive models. There are certain reasons for this behavior. The online error adjustment approach performed poorly in a few domains. If the cases when the orientation of objects in a domain is such that $h(\pi_{i+1})$ is larger than $h(\pi_i)$ then $\epsilon_{h(\pi_i)}$ may not be accurate. The inaccuracy in $\epsilon_{h(\pi_i)}$ is compounded if the above condition holds at the beginning of the planning process. This results in an inaccurate ϵ^{avg}_{h}, leading to wrong selection of the partial plan to refine next. Consequently, the planner ends up finding longer and less flexible plans. In Table 4.9, this nature can be seen in Elevator-2 and Rovers-5 domains. Another limitation is that a refinement may change the existing priorities of partial plans in the set due to the single-step-error adjustment. Considering the time factor, we avoid changing the decided priorities of those partial plans. This sometimes also leads to inaccurate ϵ^{avg}_{h}. In Table 4.7, this could also be a reason for the adverse performance of the enhanced versions of the learned predictive models like $h^{l,r,e}_{add}$and $h^{l,r,e}_{add,w}$ in Logistics-2 domain.

These are the possible cases when the approach adversely affect the informativeness.

However, this is not the usual cases with heuristics. Most of the results demonstrated in this chapters suggest that often the approach leads to a better estimate.

4.6 Discussion

We have already discussed the advantages of our approaches but they also have limitations. In our offline approach, we are bound to do some poor generalization while learning heuristics. Current literature supports the idea of selecting a large feature set for more accurate learning (Roberts *et al.*, 2008). Accuracy can also be improved using an empirical performance model of all components of a portfolio to decide which component to pick next (Fawcett *et al.*, 2014). In our work, a large feature set may have some drawbacks. For example, computing the features at each refinement step during the planning process is computationally expensive.

Our approaches do not utilize the advantage of strategies like alternation queue, and candidate selection using concept of pareto optimality (Röger and Helmert, 2010). Recently, the planning community has tried coming up with effective portfolios of heuristics or planners. Techniques of generating good portfolios are not new to theoretical machine learning. A follow up work done in the past is combining multiple heuristics online (Streeter *et al.*, 2007). One could form a portfolio of different algorithms to reduce the total makespan for a set of jobs to solve (Streeter and Smith, 2008). The authors provide a performance bound for the portfolio approaches. For example, an execution of a greedy schedule of algorithms cannot exceed four times the optimal schedule.

In planning, a sequential portfolio of planners or heuristics aims to optimize the performance metrics. In general, such configurations automatically generate sequential orderings of best planning algorithms. In the portfolio the participants are allotted some timestamp to participate in solving problems in the ordering. A similar approach is used in (Seipp *et al.*, 2015*b*). The authors outline their procedure for optimal and satisficing planning. The procedure used in this work starts with a set of planning algorithms and a time bound. It uses another procedure $OPTIMIZE$ that focuses on the marginal improvements of the performance. Here, the quality of the portfolio is bounded by $(1 - (1/e)) \times \text{OPT}$, and the running time cannot exceed 4 OPT. The components can be allowed to act in a round-robin fashion (Gerevini *et al.*, 2014).

The state-of-the-art planners exhibit variations in their runtime for a given problem instance, so no planner always dominates over others. A good approach would be selecting a planner for a given instance by looking at its processing time. This is done by building an empirical performance model (EPM) for each planner. EPM is derived from sets of planning problems and performance observation. It predicts whether the planner could solve a given instance (Fawcett *et al.*, 2014). The authors consider a large set of instance features and show that the runtime predictor is often superior to the individual planners. Performance wise sorting of components in a portfolio is also possible (Núñez *et al.*, 2015). The portfolio is sorted such that the probability of the performance of that portfolio is maximum at any time. Experiments show that performance of a greedy strategy can be enhanced to near optimal over time. The planning community has also used Integer and Linear Programming (ILP) approaches. A few useful LP-based heuristics for optimal planning can be found in (Pommerening *et al.*, 2013). A further follow up work is (Pommerening *et al.*, 2015). There is also see a unifying framework of LP-based heuristics for optimal planning (Pommerening *et al.*, 2014).

The last two paragraphs cover recent literature in brief which explain previous strategies of combining different base methods. The literature shows that they have performed well over different benchmarks. Our current settings do not capture any such ideas for combining different components of heuristics. A direct comparison with any of the above mentioned works is therefore out of scope for our current work. This is because, we are more concerned about working with unit cost based POCL heuristics in isolation. On the other hand, we suspect that many of these strategies, in some adapted form, would likely be beneficial in the POCL framework.

Different learning techniques have been tried to enhance the efficiency of planning processes in the past. Our learning techniques are inspired from the following approaches. Samadi, Felner, and Schaeffer (2008) try to combine several features by employing ANNs, inspired from single agent games. For optimal targets in the training sets, they use optimal solution cost (h^*) found for relaxed problems. However, our offline approach does not need optimal targets, also obtaining them is a tedious task in POCL planning. They consider planning problems similar enough for employing the learned models in the training phase to testing phase, which is not always the case. Another approach is to learn from domain independent heuristics in each domain for cost-based planning (Virseda *et al.*, 2013). This is focused on enhancing the informativeness of real cost heuristics. However, like the previous approach, this is also exposed to the drawbacks of offline learning. We focus on learning to improve POCL heuristics and are exposed to all the drawbacks of offline learning discussed earlier. Unlike others, we alleviate the effect of the drawbacks using online tuning. As a result, we avoid restricting our combined approach only to the similar planning problems.

4.7 Related Work

Different learning techniques have been tried to enhance the efficiency of planning processes in the past. Our learning techniques are inspired from the following approaches. Samadi, Felner, and Schaeffer (2008) try to combine several features by employing ANNs, inspired from single agent games. For optimal targets in the training sets, they use optimal solution cost (h^*) found for relaxed problems. However, our offline approach does not need optimal targets, also obtaining them is a tedious task in POCL planning. They consider planning problems similar enough for employing the learned models in the training phase to testing phase, which is not always the case. Another approach is to learn from domain independent heuristics in each domain for cost-based planning (Virseda *et al.*, 2013). This is focused on enhancing the informativeness of real cost heuristics. However, like the previous approach, this is also exposed to the drawbacks of offline learning. We focus on learning to improve POCL heuristics and are exposed to all the drawbacks of offline learning discussed earlier. Unlike others, we alleviate the effect of the drawbacks using online tuning. As a result, we avoid restricting our combined approach only to the similar planning problems.

Arfaee, Zilles, and Holte (2011) give an approach for learning stronger heuristics by solving small combinatorial problems with single goal state. It starts with domain dependent ad-hoc heuristics, which creates easy problems using Random Walk in case the original problems are unsolvable using these heuristics. In our work, the main contribution is to go from domain specific to domain independent way of POCL planning. Our learning approach in Algorithm 1, is fully automated and domain independent, however it learns domain specific details. Unlike theirs, Algorithm 1 does not depend on single goal node during dataset preparation. Furthermore, we use domain independent informed heuristics as features in the training sets. Our offline approach takes much less time in training as compared to their approach.

Thayer, Dionne, and Ruml (2011) do not use general form of learning in an online approach that does not need a huge training set. It avoids generalizing the nature of planning problems too. In general, an offline approach learns better models than online, however, the evaluation shows that this is not always the case as the online learned models provide better guidance. Unlike ours, their approach avoids using the real targets for training which might expose it towards learning inaccurate models. To mitigate the effects of this drawback, we begin by offline leaning with more accurate targets and obtain informed predictive models. To reduce the negative effects of offline learning, we use online error tuning based on the parent-child relationship (Definition 4.2.1) (Sutton, 1988). This improves the informativeness of those predictive models after each refinement during planning, and provides an overall better search guidance than individual models. Unlike their online approach, this one handles explicit looping of actions that achieve and destroy subgoals, thence does not include those refinements in computing ϵ_h^{avg}. This is crucial in POCL planning because you may never achieve a solution plan. On the whole, our evaluation shows that it is good to have an informed heuristic (a predictive model) in the beginning which can further be strengthened using online tuning. However, our online approach is a general form of $\mathrm{TD}(\lambda)$, where λ is 0, hence its quintessence is similar to the quintessence of one in Thayer *et al.* In future, we intend to perform the error tuning using different values of λ.

4.8 Summary and Future Work

We demonstrate the use of different regression models to combine different heuristic values to arrive at consistently better estimates over a range of planning domains and problems. We extend some recent attempts to learn combinations of heuristic functions in state-space based planning to POCL planning. We also show that the learned models can be further enhanced by an online error correction approach.

In future we intend to explore online learning further, and continue our experiments with combining heuristic functions. We also aim to explore the use of an optimizing planner in tandem with bootstrapping methods. Apart from these, we will be giving a complete generalization of our current learning approaches for temporal planning and planning with deadlines.

In the next chapter, we discuss an overall summary of the thesis, and also discuss related possible future extensions in brief.

Chapter 5

SUMMARY AND FUTURE WORK

The degree of informativeness of a heuristic function is critical for a heuristic search algorithm to be successful. It becomes more important when we employ such heuristic search algorithms (*e.g.* A^* and IDA^* etc) in domain independent planning as a domain independent heuristic does not perform well in *all* planning domains. In this thesis, we presented two major approaches that enhances the effectiveness of the POCL framework employing different types of learning. The first approach learns to combine multiple heuristics together effectively using a feed forward ANN with a more effective cost function, $R(\theta)$. The error minimization is done using Gradient Descent algorithm. This is followed by another supervised learning approach for coming up with a combination of multiple heuristics, and finally the learnt heuristic function (meta-heuristics) is further tuned using TD learning. By employing TD learning, a heuristic learns from its own mistakes and tries to be more accurate in the subsequent steps of the planning process. Next, we present a summary of this thesis below, followed by a discussion on future extensions.

5.1 Summary and Conclusions

In Chapter 1, we introduce, formally, some specific issues addressed in this thesis, which is followed by the background discussion in Chapter 2. In Chapter 2, we talk about the importance of POCL framework and how this is different from state-space based planning. There are some negative aspects associated with the POCL framework too like current heuristics are not powerful enough, also sometimes the nodes in the search space are inconsistent due to looping of dependencies in a partial plan (node in the plan space). We give two examples in support of opting for the POCL framework.

In Chapter 3, we show the adaptation of some of the regression techniques, for example: Linear Regression, Multi-Layer Perceptron, PE-ANN etc. The adaptations have shown good results. Motivated by the current interests of planning researchers, we also adapted and employed some state-space based heuristics into the POCL framework with small modifications. The adaptation demonstrates good results. We also adapt the PE-ANN (Samadi, Felner, and Schaeffer (2008)) and empirically show that the regularization often affects the learning processes (Bishop, 2006). We give complete weight update rule for the new cost function used in PE-ANN. We also demonstrate that often the performance of the combination of heuristics is better than the individual heuristic functions including the state of the art. This adaptation is still required to be tested in the POCL framework. As of now, we cannot say anything currently about the overall performance of a POCL planner using the learned heuristics, but our primary motivation of getting better estimates, gets fulfilled. In the future, we intend to work upon this.

In Chapter 4, we discuss an adapted approach from Temporal Difference (TD) learning in the POCL framework. On the lines of the state-space based literature, we observe that a heuristic function may commit some error in each step during search. Monitoring and correcting such error closely might influence the performance of the search algorithm. The adaptation shows good results, demonstrated in the previous chapter. TD learning is also employed to enhance the learned predictive models that are learned from multiple existing heuristics. The reason behind employing TD learning for tuning these learned heuristics is the poor generalizations that often occur during the offline training. Due to the poor generalization during model learning, the predictive models are prone to commit mistakes (step-error) at each refinement step as the nature of planning problems varies, even they belong to same planning domain.

5.2 Future Work

However, there is still much to test in this POCL framework, in future, we intend to test a complete online approach to enhance the effectiveness of heuristic search algorithms by designing a well informed heuristic function. In that case we do not need a huge training set in the beginning of the learning process. We also plan for applying bootstrapping techniques, though they have not been tested much by the planning community.

In Section 3.2, we discuss that currently the approach used is highly memory intensive because of the employment of the planning graph data structure. In future, we also intend to cut down the space required to grow a planning graph fully. We will be using TIM (Fox and Long, 1998), which is also adapted in (Fox and Long, 1999). Ridder and Fox (2014) show that capturing the object symmetries during the planning process reduces the space required to solve a planning problem. The approach used in their work uses TIM as well.

The literature shows that the POCL framework is highly useful in Multi-Agent Planning (MAP) (Torreño *et al.*, 2015). We could also follow some of the recent efforts in MAP put by the community researchers (Nissim and Brafman, 2013, 2014). We strongly feel that the POCL framework is highly useful in temporal planning (Karpas *et al.*, 2015; Marzal *et al.*, 2014*b*) too (e.g. POPF (Coles *et al.*, 2010) and POPF2 (Coles *et al.*, 2011)), more specifically with complex temporal constraints that are beyond durative actions. We would also like to extend our previous approaches using online learning or bootstrapping for speeding up the planning process in these frameworks.

LIST OF PUBLICATIONS BASED ON THESIS

1. **Shekhar, S.,** and **D. Khemani** (2016). Learning and Strengthening Meta-heuristics in Plan Space Planning. In *the Israeli Association for Artificial Intelligence (IAAI)*, Technion, Israel, October 2016.

2. **Shekhar, S.,** and **D. Khemani** (2016). Learning and Tuning Meta-heuristics in Plan Space Planning. *CoRR abs/*1601.07483*,* 2016.

3. **Shekhar, S.,** and **D. Khemani** (2015). Extending and Tuning Heuristics for a Partial Order Causal Link Planner. In 3^{rd} *international conf. on Mining Intelligence and Knowledge Exploration (MIKE)*, pages 81-92, India, December 2015.

4. **Shekhar, S.,** and **D. Khemani** (2015). Improving Heuristics on-the-fly for Effective Search in Plan Space. In 38^{th} *German Conference on Artificial Intelligence (KI)*, pages 302-308, Germany, September 2015.

Bibliography

1. **Arfaee, S. J.**, **S. Zilles**, and **R. C. Holte**, Bootstrap learning of heuristic functions. *In Proceedings of the Third Annual Symposium on Combinatorial Search, SOCS 2010, Stone Mountain, Atlanta, Georgia, USA*. 2010.

2. **Arfaee, S. J.**, **S. Zilles**, and **R. C. Holte** (2011). Learning heuristic functions for large state spaces. *Artificial Intelligence*, **175**, 2075–2098.

3. **Bäckström, C.** and **B. Nebel** (1995). Complexity results for sas+ planning. *Computational Intelligence*, **11**(4), 625–655.

4. **Benton, J.**, **A. J. Coles**, and **A. Coles**, Temporal planning with preferences and time-dependent continuous costs. *In Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil*. 2012.

5. **Bercher, P.** and **S. Biundo**, Encoding partial plans for heuristic search. *In Proceedings of the 4th Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*. 2013.

6. **Bercher, P.**, **T. Geier**, and **S. Biundo**, Using state-based planning heuristics for partial-order causal-link planning. *In KI 2013: Advances in Artificial Intelligence*. 2013, 1–12.

7. **Bishop, C. M.**, *Pattern Recognition and Machine Learning*, volume 1. Springer New York, 2006.

8. **Blum, A.** and **M. L. Furst**, Fast planning through planning graph analysis. *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada*. 1995.

9. **Blum, A. L.** and **M. L. Furst** (1997). Fast planning through planning graph analysis. *Artificial intelligence*, **90**(1), 281–300.

10. **Bonet** and **Geffner**, Planning as heuristic search: New results. *In Recent Advances in AI Planning*. Springer, 2000, 360–372.

11. **Bonet, B.** and **H. Geffner** (2001). Planning as heuristic search. *Artificial Intelligence*, **129**, 5–33. URL `http://dx.doi.org/10.1016/S0004-3702(01)00108-4`.

12. **Bonet, B.** and **M. Helmert**, Strengthening landmark heuristics *via* hitting sets. *In Proceedings of European Conference on Artificial Intelligence, ECAI, Lisbon, Portugal*. 2010.

13. **Bonet, B.**, **G. Loerincs**, and **H. Geffner**, A robust and fast action selection mechanism for planning. *In IAAI AAAI*. 1997. URL `http://www.aaai.org/Library/AAAI/1997/aaai97-111.php`.

14. **Bonet, B.** and **M. van den Briel**, Flow-based heuristics for optimal planning: Landmarks and Merges. *In Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS, Portsmouth, New Hampshire, USA*. 2014.

15. **Bylander, T.** (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, **69**, 165–204.

16. **Chrpa, L.** (2009). *Learning for classical planning*. Ph.D. thesis, Charles University, Prague.

17. **Coles, A.**, **A. Coles**, **M. Fox**, and **D. Long** (2011). POPF2: A forward-chaining partial order planner. *The 2011 International Planning Competition*.

18. **Coles, A.**, **A. Coles**, **M. Fox**, and **D. Long** (2012). COLIN: Planning with continuous linear numeric change. *J. Artif. Intell. Res. (JAIR)*, **44**, 1–96.

19. **Coles, A.**, **M. Fox**, **D. Long**, and **A. Smith**, Additive-disjunctive heuristics for optimal planning. *In Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS, Sydney, Australia*. 2008*a*.

20. **Coles, A.**, **M. Fox**, **D. Long**, and **A. Smith**, Planning with problems requiring temporal coordination. *In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, 2008, Chicago, Illinois, USA*. 2008*b*.

21. **Coles, A. J.**, **A. Coles**, **M. Fox**, and **D. Long**, Temporal planning in domains with linear processes. *In Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), USA*. 2009.

22. **Coles, A. J.**, **A. Coles**, **M. Fox**, and **D. Long**, Forward-chaining partial-order planning. *In Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada*. 2010.

23. **Cushing, W.**, **S. Kambhampati**, **Mausam**, and **D. S. Weld**, When is temporal planning really temporal? *In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*. 2007.

24. **Do, M. B.** and **S. Kambhampati** (2003). SAPA: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)*, **20**, 155–194.

25. **Domshlak, C.**, **E. Karpas**, and **S. Markovitch**, To max or not to max: Online learning for speeding up optimal planning. *In Proc. AAAI*. 2010*a*.

26. **Domshlak, C.**, **M. Katz**, and **S. Lefler**, When abstractions met landmarks. *In Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS, Toronto, Ontario*. 2010*b*.

27. **Edelkamp, S.** and **J. Hoffmann** (2004). PDDL2.2: The language for the classical part of the 4th international planning competition. *4th IPC*.

28. **Erol, K.**, **D. S. Nau**, and **V. S. Subrahmanian** (1995). Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, **76**, 75–88.

29. **Fawcett, C.**, **M. Vallati**, **F. Hutter**, **J. Hoffmann**, **H. H. Hoos**, and **K. Leyton-Brown**, Improved features for runtime prediction of domain-independent planners. *In Proc. ICAPS*. 2014.

30. **Fikes, R.** and **N. J. Nilsson**, STRIPS: A new approach to the application of theorem proving to problem solving. *In Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK*. 1971.

31. **Fink, M.**, Online learning of search heuristics. *In Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS 2007, San Juan, Puerto Rico*. 2007.

32. **Fox, M.** and **Long**, The detection and exploitation of symmetry in planning problems. *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI, Stockholm, Sweden*. 1999.

33. **Fox, M.** and **D. Long** (1998). The automatic inference of state invariants in TIM. *J. Artif. Intell. Res. (JAIR)*, **9**, 367–421.

34. **Fox, M.** and **D. Long** (2003). PDDL2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)*, **20**, 61–124.

35. **Fuentetaja, R.** and **D. Borrajo**, Improving control-knowledge acquisition for planning by active learning. *In Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, Proceedings*. 2006.

36. **Geffner, H.**, The model-based approach to autonomous behavior: A personal view. *In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI, Atlanta, Georgia, USA*. 2010.

37. **Gerevini, A.**, **A. Saetti**, and **M. Vallati** (2014). Planning through automatic portfolio configuration: The PbP approach. *J. Artif. Intell. Res. (JAIR)*, **50**, 639–696.

38. **Ghallab, M.**, **D. Nau**, and **P. Traverso**, *Automated Planning: Theory & Practice*. Elsevier, 2004.

39. **Hall, M.**, **E. Frank**, **G. Holmes**, **B. Pfahringer**, **P. Reutemann**, and **I. H. Witten**, The WEKA data mining software: An update. *In SIGKDD Explorations*, volume 11. 2009.

40. **Hall, M. A.**, Correlation-based feature selection for discrete and numeric class machine learning. *In Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA*. 2000.

41. **Halsey, K.**, **D. Long**, and **M. Fox**, CRIKEY - A temporal planner looking at the integration of scheduling and planning. *In Workshop on Integrating Planning into Scheduling, ICAPS*. 2004.

42. **Haslum, P.**, $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. *In Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS, Thessaloniki, Greece*. 2009.

43. **Haslum, P.**, Incremental lower bounds for additive cost planning problems. *In Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil*. 2012.

44. **Haslum, P.**, **B. Bonet**, and **H. Geffner**, New admissible heuristics for domain-independent planning. *In Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, Pittsburgh, Pennsylvania, USA*. 2005.

45. **Haslum, P.**, **A. Botea**, **M. Helmert**, **B. Bonet**, and **S. Koenig**, Domain-independent construction of pattern database heuristics for cost-optimal planning. *In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia*. 2007.

46. **Haslum, P.** and **H. Geffner**, Admissible heuristics for optimal planning. *In AIPS*. 2000.

47. **Helmert, M.**, A planning heuristic based on causal graph analysis. *In ICAPS*, volume 16. 2004.

48. **Helmert, M.** (2006). The fast downward planning system. *J. Artif. Intell. Res. (JAIR) 2006,*, **26**, 191–246.

49. **Helmert, M.** and **C. Domshlak**, Landmarks, critical paths and abstractions: What's the difference anyway? *In Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS, Greece*. 2009.

50. **Helmert, M.** and **C. Domshlak**, LM-Cut: Optimal planning with the landmark-cut heuristic. *In Seventh IPC*. 2011.

51. **Helmert, M.** and **H. Geffner**, Unifying the causal graph and additive heuristics. *In Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia*. 2008.

52. **Helmert, M.**, **P. Haslum**, **J. Hoffmann**, and **R. Nissim** (2014). Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *J. ACM*, **61**(3), 16:1–16:63.

53. **Helmert, M.**, **G. Röger**, and **E. Karpas**, Fast downward stone soup: A baseline for building planner portfolios. *In Proc. ICAPS PAL*. 2011.

54. **Helmert, M.**, **G. Röger**, and **S. Sievers**, On the expressive power of non-linear merge-and-shrink representations. *In Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS, Jerusalem, Israel*. 2015.

55. **Hoffmann, J.**, *Utilizing Problem Structure in Planning: A Local Search Approach*, volume 2854 of *Lecture Notes in Computer Science*. Springer, 2003.

56. **Hoffmann, J.** and **S. Edelkamp** (2005). The deterministic part of IPC-4: an overview. *J. Artif. Intell. Res. (JAIR)*, **24**, 519–579.

57. **Hoffmann, J.** and **B. Nebel** (2001). The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)*, **14**, 253–302.

58. **Hoffmann, J.**, **J. Porteous**, and **L. Sebastia** (2004). Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)*, **22**, 215–278.

59. **Imai, T.** and **A. Fukunaga**, A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. *In ECAI*. 2014.

60. **J.Hoffmann** (2003). The Metric-FF planning system: Translating ”ignoring delete lists” to numeric state variables. *J. Artif. Intell. Res. (JAIR)*, **20**, 291–341.

61. **Joslin, D.** and **M. E. Pollack**, Least-cost flaw repair: A plan refinement strategy for partial-order planning. *In Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, Volume 2.*. 1994.

62. **Karpas, E.** and **C. Domshlak**, Cost-optimal planning with landmarks. *In IJCAI, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA*. 2009.

63. **Karpas, E.**, **D. Wang**, **B. C. Williams**, and **P. Haslum**, Temporal Landmarks: What Must Happen, and When. *In Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS, Jerusalem, Israel*. 2015.

64. **Katz, M.** and **Domshlak**, Structural-pattern databases. *In Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS, Thessaloniki, Greece*. 2009.

65. **Katz, M.** and **C. Domshlak**, Optimal additive composition of abstraction-based admissible heuristics. *In Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS, Sydney, Australia*. 2008.

66. **Keyder, E.**, **S. Richter**, and **M. Helmert**, Sound and complete landmarks for and/or graphs. *In ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, Proceedings*. 2010.

67. **Keyder, E. R.**, **J. Hoffmann**, and **P. Haslum**, Semi-relaxed plan heuristics. *In Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS, Atibaia, São Paulo, Brazil*. 2012.

68. **Keyder, E. R.**, **J. Hoffmann**, and **P. Haslum** (2014). Improving delete relaxation heuristics through explicitly represented conjunctions. *J. Artif. Intell. Res. (JAIR)*, **50**, 487–533.

69. **Korf, R. E.** (1985). Depth-First Iterative-Deepening (DFID): An optimal admissible tree search. *Artificial intelligence*, **27**(1), 97–109.

70. **Korf, R. E.** and **A. Felner** (2002). Disjoint pattern database heuristics. *Artif. Intell.*, **134**(1-2), 9–22.

71. **Kvarnström, J.**, Planning for loosely coupled agents using partial order forward-chaining. *In Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS, Freiburg, Germany*. 2011.

72. **Marzal, E.**, **L. Sebastia**, and **E. Onaindia**, On the use of temporal landmarks for planning with deadlines. *In Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS), New Hampshire, USA*. 2014*a*.

73. **Marzal, E.**, **L. Sebastia**, and **E. Onaindia**, On the use of temporal landmarks for planning with deadlines. *In Twenty-Fourth International Conference on Automated Planning and Scheduling*. 2014*b*.

74. **McAllester, D. A.** and **D. Rosenblitt**, Systematic nonlinear planning. *In Proceedings of the 9th National Conference on Artificial Intelligence, AAAI*. 1991.

75. **McDermott, D.** (1999). Using regression-match graphs to control search in planning. *Artificial Intelligence*, **109**(1), 111–159.

76. **Mirkis, V.** and **C. Domshlak**, Cost-sharing approximations for h+. *In Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS, Providence, Rhode Island, USA*. 2007.

77. **Muise, C.**, **S. A. McIlraith**, and **J. C. Beck**, Monitoring the execution of partial-order plans *via* regression. *In IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22. 2011.

78. **Nguyen, X.** and **S. Kambhampati**, Extracting effective and admissible state space heuristics from the planning graph. *In IAAI AAAI*. 2000.

79. **Nguyen, X.** and **S. Kambhampati**, Reviving partial order planning. *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA*. 2001.

80. **Nigenda, R. S.**, **X. Nguyen**, and **S. Kambhampati** (2000). AltAlt: Combining the advantages of graphplan and heuristic state search. *Knowledge Based Computer Systems*, 409–421.

81. **Nissim, R.** and **R. I. Brafman**, Cost-optimal planning by self-interested agents. *In Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*. 2013.

82. **Nissim, R.** and **R. I. Brafman** (2014). Distributed heuristic forward search for multi-agent planning. *J. Artif. Intell. Res. (JAIR)*, **51**, 293–332.

83. **Nissim, R.**, **J. Hoffmann**, and **M. Helmert**, Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. *In Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain*. 2011.

84. **Núñez, S.**, **D. Borrajo**, and **C. Linares López**, Sorting sequential portfolios in automated planning. *In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI, Buenos Aires, Argentina*. 2015.

85. **Pearl, J.**, *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc., Reading, MA, 1984.

86. **Penberthy, J. S.** and **D. S. Weld**, UCPOP: A sound, complete, partial order planner for ADL. *In Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*. 1992.

87. **Planken, L.**, **M. De Weerdt**, **R. van der Krogt**, **J. Rintanen**, **B. Nebel**, **J. C. Beck**, and **E. Hansen**, P^3C: A new algorithm for the simple temporal problem. *In ICAPS*. 2008.

88. **Pommerening, F.** and **M. Helmert**, Incremental lm-cut. *In Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS, Rome, Italy*. 2013.

89. **Pommerening, F.**, **G. Röger**, and **M. Helmert**, Getting the most out of pattern databases for classical planning. *In IJCAI, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China*. 2013.

90. **Pommerening, F.**, **G. Roger**, **M. Helmert**, and **Bonet**, LP-based heuristics for cost-optimal planning. *In Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS, Portsmouth, New Hampshire, USA*. 2014.

91. **Pommerening, F.**, **G. Röger**, **M. Helmert**, and **B. Bonet**, Heuristics for cost-optimal classical planning based on linear programming. *In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI, Buenos Aires, Argentina*. 2015.

92. **Quinlan, J. R.** *et al.*, Learning with continuous classes. *In Proc. Australian Joint Conference on Artificial Intelligence*, volume 92. 1992.

93. **Richter, S.**, **M. Helmert**, and **M. Westphal**, Landmarks revisited. *In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI, Chicago, Illinois, USA*. 2008.

94. **Richter, S.** and **M. Westphal** (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, **39**, 127–177.

95. **Richter, S.**, **M. Westphal**, and **M. Helmert**, LAMA 2008 and 2011. *In Seventh IPC*. 2011.

96. **Ridder, B.** (2014). *Lifted Heuristics: Towards More Scalable Planning Systems*. Ph.D. thesis, King's College London (University of London).

97. **Ridder, B.** and **M. Fox**, Heuristic evaluation based on lifted relaxed planning graphs. *In Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS), Portsmouth, New Hampshire*. 2014.

98. **Roberts, M.**, **A. E. Howe**, **B. Wilson**, and **M. desJardins**, What makes planners predictable? *In Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS, Sydney, Australia*. 2008.

99. **Röger, G.** and **M. Helmert**, The more, the merrier: Combining heuristic estimators for satisficing planning. *In Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS, Toronto, Ontario, Canada*. 2010.

100. **Röger, G.**, **F. Pommerening**, and **J. Seipp**, Fast Downward Stone Soup 2014. *In Eighth IPC*. 2014.

101. **Rousseeuw, P. J.** and **A. M. Leroy**, *Robust regression and outlier detection*, volume 589. John Wiley & Sons, 2005.

102. **Samadi, M.**, **A. Felner**, and **J. Schaeffer**, Learning from multiple heuristics. *In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI, Chicago, Illinois, USA*. 2008.

103. **Sapena, O.**, **E. Onaindıa**, and **A. Torreno**, Combining heuristics to accelerate forward partial-order planning. *In Proc ICAPS COPLAS*. 2014.

104. **Schubert, L.** and **A. Gerevini**, Accelerating partial order planners by improving plan and goal choices. *In Proceedings of Seventh International Conference on Tools with Artificial Intelligence*. 1995.

105. **Seipp, J.**, **F. Pommerening**, and **M. Helmert**, New optimization functions for potential heuristics. *In Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS, Jerusalem, Israel*. 2015*a*.

106. **Seipp, J.**, **S. Sievers**, **M. Helmert**, and **F. Hutter**, Automatic configuration of sequential planning portfolios. *In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, Texas, USA.*. 2015*b*.

107. **Sievers, S.**, **M. Wehrle**, and **M. Helmert**, Generalized label reduction for merge-and-shrink heuristics. *In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, Québec City, Québec, Canada.*. 2014.

108. **Sievers, S.**, **M. Wehrle**, **M. Helmert**, **A. Shleyfman**, and **M. Katz**, Factored symmetries for merge-and-shrink abstractions. *In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, Texas, USA.*. 2015.

109. **Streeter, M. J.**, **D. Golovin**, and **S. F. Smith**, Combining multiple heuristics online. *In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada*. 2007.

110. **Streeter, M. J.** and **S. F. Smith**, New techniques for algorithm portfolio design. *In UAI, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland*. 2008.

111. **Sutton, R. S.** (1988). Learning to predict by the methods of temporal differences. *Machine Learning,*, **3**, 9–44. URL `http://dx.doi.org/10.1007/BF00115009`.

112. **Thayer, J. T.** (2012). *Heuristic Search Under Time and Quality Bounds*. Ph.D. thesis, University of New Hampshire, USA.

113. **Thayer, J. T.**, **A. J. Dionne**, and **W. Ruml**, Learning inadmissible heuristics during search. *In Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS, Freiburg, Germany*. 2011. URL `http://aaai.org/ocs/index.php/ICAPS/ICAPS11/paper/view/2710`.

114. **Thayer, J. T.** and **W. Ruml**, Using distance estimates in heuristic search. *In Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece*. 2009.

115. **Torreño, A.**, **E. Onaindia**, and **O. Sapena** (2015). FMAP: distributed cooperative multi-agent planning. *CoRR*, **abs/1501.07250**. URL `http://arxiv.org/abs/1501.07250`.

116. **Vidal, V.** and **H. Geffner** (2006). Branching and pruning: An optimal temporal pocl planner based on constraint programming. *Artificial Intelligence*, 298–335.

117. **Virseda, J.**, **D. Borrajo**, and **V. Alcázar**, Learning heuristic functions for cost-based planning. *In Proc. ICAPS PAL 2013*. 2013.

118. **Weld, D. S.** (1994). An introduction to least commitment planning. *AI Magazine*, **15**, 27–61. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/1109`.

119. **Weld, D. S.** (2011). AAAI-10 Classic Paper Award: Systematic NonLinear Planning - A Commentary. *AI Magazine*, **32**(1), 101. URL `http://www.aaai.org/ojs/index.php/aimagazine/article/view/2341`.

120. **Wilkins, D. E.** (1984). Domain-independent planning: Representation and plan generation. *Artif. Intell.*, **22**, 269–301.

121. **Younes, H. L. S.** and **R. Simmons** (2003). VHPOP: versatile heuristic partial order planner. *J. Artif. Intell. Res. (JAIR)*, **20**, 405–430. URL `http://dx.doi.org/10.1613/jair.1136`.

122. **Younes, H. L. S.** and **R. G. Simmons**, On the role of ground actions in refinement planning. *In Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, Toulouse, France*. 2002. URL `http://www.aaai.org/Library/AIPS/2002/aips02-006.php`.