THE RELEVANCE SCORE: A LANDMARK-LIKE HEURISTIC FOR AUTOMATED PLANNING

by

OLIVER EDWARD KIM

A thesis submitted to the University of Birmingham and the University of Melbourne for the degree of

DOCTOR OF PHILOSOPHY

19 February 2025

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham

School of Computing and Information Systems
Faculty of Engineering and Information Technology
University of Melbourne

Abstract

Landmarks are facts or actions that appear in all valid solutions of a planning problem. They have been used successfully to calculate heuristics that guide the search for a plan. The potential uses for landmarks are limited by their definition relying on the existence of at least one valid plan. We investigate an extension to this concept by defining a novel *relevance score* $\Xi(l)$ that quantifies how often a fact or action will appear in partial plans. We describe a method to compute this relevance score that makes no reference to the initial state of a problem, ensuring that it is applicable even if no plans exist.

We define a heuristic h_{Ξ} that counts the amount of relevance between a state and the goal, that is analagous to landmark counting heuristics h_{LC}. We experimentally compare the performance of our approach to that of a state of the art landmark-based heuristic planning approach, using benchmark planning problems. While the original landmark-based heuristic leads to better performance on problems with well-defined landmarks, our approach substantially improves performance on problems that lack non-trivial landmarks.

Diagnosis can be described as finding explanations for observations that are not supported by an agent's model of the world. This can be treated as a planning problem, where the world model must be fixed by assuming facts, and explained by a sequence of actions that connects the new world model to the observations (ie a plan). Previous attempts at this have relied on providing the system with candidates for assumption. We show that because the relevance score is calculable in planning problems that do not permit a plan, it can also be used to identify potential candidates. This is achieved by defining an *assumability score* $\tau(\alpha)$ that uses the relevance score to assess fixes that are linked to the goal by backtracking actions.

Acknowledgements

Funding

This PhD was funded by a Priestley scholarship, which allowed me to spend time at both the University of Birmingham and the University of Melbourne.

Supervisors

I am especially grateful to Mohan Sridharan, who accepted me as a student part way through, and helped me to pick things up and organise my thoughts after each of the many interruptions to my study. I would also like to thank Adrian Pearce and Nir Lipovetzky for welcoming me to Melbourne and helping me to develop and articulate ideas. And I hope that Jeremy Wyatt, who inspired me to start a PhD on this topic, and (alongside Nick Hawes, however briefly) influenced my reading and early ideas on the topic, will be happy with where it has gone.

Discussions

I was honoured to be invited to present work derived from Chapters 3 and 4 at the ACS 2024 Conference. The (anonymous) detailed reviews helped me to refine the presentation of our ideas, particularly the mathematical representations in Chapter 3. Discussing the work after presenting it, particularly with Pat Langley, provided context and ideas for further development. Finally, I thank my examiners, Ron Petrick and Mauro Vallati for an enjoyable discussion of my work, and the detailed suggestions for improving its presentation and future development.

Contents

Chapter 1

Introduction

This thesis will explore the concept of relevance in the context of classical task planning. Classical planning will be formally defined later (Definition 1), but informally, is the task of using knowledge of the state of the world (facts), and interventions we can take (actions) to make a plan for how to acheive a goal. Despite being a relatively simple representation of a planning problem (all knowledge is known with certainty, and actions are deterministic), it permits a space of states and action sequences that is too large to search exhaustively. For this reason, approaches to solving classical planning problems typically involve the use of heuristics (Definition 2) - functions that estimate how close the planner is to the goal. This allows the planner to direct its search in the most promising direction.

Many heuristics can only exist when a plan exists, which limits their usabilty in variations of the problem where this is the case. We address this in the problem of diagnosis (Definition 13), which asks what change in our model of the world would allow us to find a sequence of events that explains an observation.

The aim of the work presented here is to provide new tools to help work with these problems. We focus on quantifying how relevant a piece of information is to the task, deriving a value we call the relevance score (Definition 26). We demonstrate that this value can be used to analyse and help solve problems in both classical planning and diagnosis.

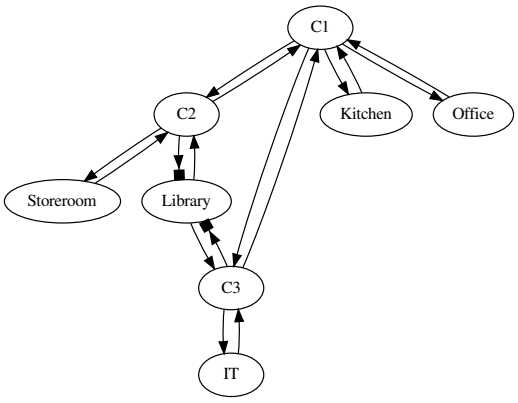
1.1 Motivating example

Consider a student who has to hand in a physical copy of an assignment. The actions available to them, and the places they can go are illustrated in Figure 1.1. With knowledge of the current state of the student (where they are, and what items they have), this could be considered a planning problem. The aim of a planning problem is to find a sequence of actions that take an agent from an initial state, to one in which their goals are achieved. In this case, the student must identify a sequence of actions that leads to them having a hardcopy of the assignment.

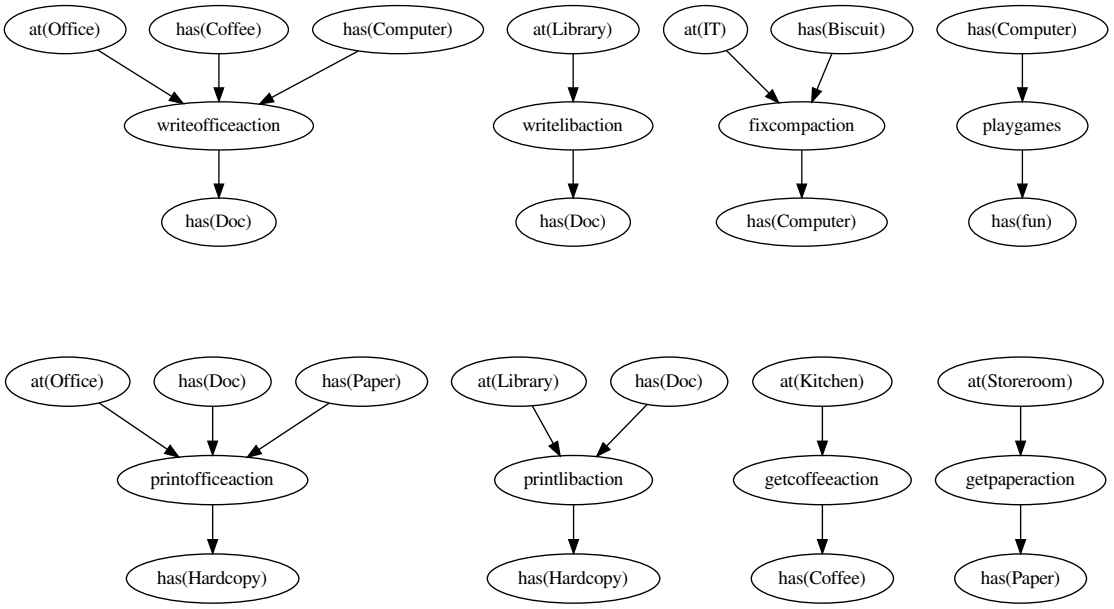
This thesis aims to help such a student complete their task, by providing tools to evaluate how the results of their actions are *relevant* to their goal. Is the student going to visit IT, despite already having a working computer? Is it *relevant* to the task whether or not the student $has(fun)$? The answers to these questions can be used to guide the search for a plan.

Suppose a friend is monitoring this student, and is concerned that they appear to have gone to the building without their belongings on a day their computer is not working. Based on this information, they expect the student to be unable to complete their assignment, but are later relieved to hear that it was handed in successfully. How can this friend diagnose the problems with their model of the situation, and fix them to explain what happened? Many facts about the world could be assumed (maybe the student $has(cat)$?), but only *relevant* assumptions can lead to explanations.

This example will be used to illustrate the concepts, problems, and solutions presented in this thesis. A PDDL (see Section 2.1.1) representation of this domain can be found in Appendix A.



(a) **Example map** Connections between rooms or corridors ($C[1, 2, 3]$) are shown as arrows, representing a move action with the precondition $at(?from)$ and the effects $[at(?to), \texttt{not}\ at(?from)]$, where $?from$ and $?to$ are the rooms at the front and end of the arrow. The $Library$ requires a keycard to access, which is shown as a square at the end of the arrows leading to it. The corresponding $move(?from, Library)$ actions have the additional precondition $has(Keycard)$.



(b) **Example actions** Each directed graph depicts an action (middle row), with preconditions (top row) and effects (bottom row) in a model planning problem.

Figure 1.1: **Example planning problem** An informal description of a simple planning problem. A student must write and print a hard copy of an assignment ($goal = [has(Hardcopy)]$). Their office has an unreliable computer (that IT will fix if brought their favourite biscuit), a distracting environment (the student needs coffee to focus), and a printer that frequently runs out of paper. On the other hand, working or printing in the library has fewer requirements, but is only accessible with a keycard. Facts F are fluents that describe current state of the world; the student may perform Actions A if the facts specified as their preconditions are satisfied, leading to the achievement of the facts in their effects. These will be defined more precisely in Section 2.1

1.2 Aims and motivation

I aim to address the problem of how to plan with an incomplete model of the world. If the observant friend (or even an inattentive student) lacks information about what items they have, or where they are, this is represented in the planning problem as elements (facts) missing from the world model. For example, if the student is unaware that they have the keycard, they would avoid making plans that involve the library, potentially concluding that the task is unsolvable. Planning with such an incomplete wolrd model will be addressed as a type of diagnosis, and will be formally defined in Section 2.3.3.

This presents challenges to agents trying to use the model, as it can make it seem unsolvable. An agent that wants to plan in such a domain must first fix (see Section 2.3.3) it, by identifying what missing facts are responsible for it being unsolvable. This could be achieved by making common sense assumptions that are both *relevant* (it contributes to the existence of a valid plan) and *reasonable* (it is supported by evidence found in a database of how the world often is). This project will concentrate on the first part of this, exploring the concept of relevance in the context of task planning.

Existing methods for addressing this type of problem rely on the provision of permissible fixes, whether explicitly, or in the form of default knowledge. Techniques exist to identify facts that, if they were made inaccessible by alterations to the model, would render it unsolvable. The use of such facts, known as *landmarks* (see Section 2.2.2) is prevented by their reliance on the existence of plans.

1.2.1 Constraining the problem

To ensure a finite size for the project, the problem was constrained in several ways. Here, we will consider the limitations of our approach, and concepts that we do not attempt to model or reason with.

Having an incomplete knowledge of the world means that the agent cannot be certain of what it will encounter. This ensures that there is at least once source of uncertainty,

and many real world applications will also include non-deterministic sensing and actions. As will be discussed further in Chapter 2, explicitly considering and maximising probabilities is computationally expensive, and is feasible for smaller domains than if the domain was deterministic. One approach to handling this is to determinise the domain, explore partial solutions in this more tractible form, and then use these partial solutions to constrain the probabilistic problem [Teichteil-Königsbuch et al., 2010]. The work described here will involve plans that operate in a deterministic space, with facts being either true, or not, in any state, and with actions that have fixed, known effects when they are performed.

We will assume that any facts in the domain given to the agent are correct and consistent, avoiding the need to check for these, and find ways to correct them. These would be difficult to fix without considering probability information, or sensing strategies.

Planning domains will be limited to task planning, and not consider any aspects of time or motion. These concepts are important for many real world applications, but do not directly relate to the questions we aim to answer. We will work with classically defined plans (see Section 2.1) that are linear sequences of discrete actions without considering how long they take, and assume that the motions required to perform an action can be correctly found by an appropriate method.

Given its compatibility with the simplifying constraints, and the benefits of having readily available test sets, we work with problems defined in PDDL 1.2 [McDermott et al., 1998] (see Section 2.1.1).

1.3 Contribution summary

In this thesis, we make 3 novel contributions to the field:

1. We define the relevance score, which measures how relevant a fact or action is to a planning problem. By representing the problem as a tree, we can compute probabilities of reaching parts of the tree if a particular tree traversal strategy is used. As facts about the world are represented as nodes within this tree, we argue that this probability corresponds to how likely facts are to be involved in a plan. We describe a method to calculate this score efficiently, without requiring knowledge of the initial state. The main previous attempt at quantifying a property like this was to identify landmarks, which offered a less nuanced, binary measure - a fact was either essential, or not. The relevance score quantifies this property as a probability, allowing inspection of a problem with greater resolution.

2. We show that this relevance score can be used as the basis for a heuristic to guide the search for a plan. We identify a class of planning problems, for which this heuristic allows more problems to be solved than other heuristics, by exploring fewer states. This validates our interpretation of the relevance score as capturing additional information (beyond that of landmarks) about how relevant a fact is, and that it is useful for solving planning problems. Code demonstrating has been published (see Section 6.1)

3. We make use of the relevance score's property of non-reliance on the initial state to fix planning problems from which some facts that were necessary to find a plan have been removed. We build on the observation that this is directly equivalent to solving a type of diagnosis problem where an incomplete world model prevents an explanation from being found. Unlike the vast majority of previous approaches to this problem, this is achieved without the provision of a precomputed list of permitted fixes. The application of the relevance score in this adjacent field (that traditionally uses first order logical representations and methods) demonstrates that its value extends beyond classical planning. Code demonstrating has been published (see Section 6.1)

1.4 Thesis outline

Chapter 2 reviews the current state of research in automated planning, with particular focus on the use of heuristics in classical planning. Established concepts and notation that will be used in later chapters are also defined here.

Chapter 3 defines the relevance score $\Xi(l)$, that calculates how likely a partial plan is to include l, if the initial state permits the existence of a plan. A method for calculating it efficiently is also presented.

Chapter 4 describes how the relevance score can be used as the basis for a novel heuristic for use in planning problems. Its performance is evaluated, and a class of problems is identified where it is able to find plans much more often than other heuristics, by performing calculations on a smaller portion of the search space most of the time.

An earlier version of the work described in Chapters 3 and 4 also appears as a technical report on arXiv [Kim and Sridharan, 2024].

Chapter 5 attempts to diagnose and fix planning problems that have had facts ablated. It does so by using relevance scores to define an assumabilty score $\tau(\alpha)$, that assesses how well a set of facts might perform if assumed to fix an unsolvable planning problem.

Chapter 6 summarises the findings presented here, and suggests ways that further research could build upon them.

Chapter 2

Literature Review

In this chapter, we perform a survey of approaches to task planning, and ways of representing it. We also provide formal definitions for concepts and notation that are used in later chapters.

2.1 Classical planning

Task planning, as the problem of finding a sequence of actions that take one from an initial state to a goal state, is most simply represented in classical planning [Ghallab et al., 2004]. This assumes the agent has a complete and accurate model of the world, and permits no uncertainty in the effects of the actions taken.

Definition 1 (Classical planning problem, $\Pi = \langle F, A, I, G \rangle$)**.** A classical planning problem is defined by the tuple (notation based on that used by [Keyder et al., 2010]):

$$\Pi = \langle F, A, I, G \rangle$$

where F is a finite set of ground predicates representing facts; A is a finite set of actions; $I \subseteq F$ is the set of facts true in the initial state; and $G \subseteq F$ is the set of facts representing goals. A state σ, is a set of facts that are true at a particular step in a plan. Facts that are not specified in σ are considered false. An action $a \in A$ has preconditions

$pre(a)$ and effects $eff(a)$, each of which is subdivided into sets of positive and negative facts: $pre^+(a), pre^-(a), eff^+(a), eff^-(a)$.

If an action's positive (negative) preconditions are true (false) in a particular state, it is applicable in that state:

$$Applicable(a, \sigma) \iff \begin{array}{l} \sigma \supseteq pre^+(a) \\ \sigma \not\supseteq pre^-(a) \end{array}$$

When an action is applied to a state in which it is applicable, the resultant state is given by:

$$Result(a, \sigma) = \sigma \cup eff^+(a) \setminus eff^-(a)$$

A sequence of actions $[a_1...a_n]$ is applicable in a state if each action is applicable in the state resulting from the previous sequence of actions, i.e., $Applicable(a_1, \sigma)$, $Applicable(a_2$, $Result(a_1, \sigma))$ etc. The result of applying a sequence of actions to a state in which it is applicable is the result of applying each action in that sequence, i.e., $Result([a_1...a_n], \sigma)$ $= Result(a_n, Result(a_{n-1}, ...Result(a_1, \sigma))$. A fact f_r is considered reachable if there exists a sequence of actions applicable to I, such that $f_r \in Result([a_1...a_n], I)$. We will discuss how whether a fact is reachable or not can be determined (under certain conditions) in Definition 7. A plan π^Π for problem $\Pi = \langle F, A, I, G \rangle$ is a sequence of actions $[a_1...a_n]$ that is applicable in I and results in a state where all facts in G are true, i.e., $Result(\pi^\Pi, I) = G$. A partial plan is a set of actions (maybe, but not necessarily fully or partially ordered) for which a plan exists that includes each member of that set. A forward chaining partial plan is a sequence of actions that is applicable to the initial state, that is hypothesised to form part of a plan.

Methods for solving classical planning problems are discussed throughout much of the rest of this chapter. A common feature is the use of graph search techniques, whether that takes place in plan-space [Sacerdoti, 1974] or state-space [Ghallab et al., 2004] (most modern approaches, and the focus of this thesis). State-space planning represents states as nodes on a

graph, and actions as edges connecting states in which they are applicable to the results of applying those actions to the original state. In such a representation, a plan is a path from I to a state that satisfies G. The search can be performed by finding reachable states (forward planning) or chaining actions backwards from the goal (backtracking/regression planning [Pollock, 1998]). In either case, the starting point of the search may be considered the root of a tree, as cycles are normally removed. Section 2.2 describes the main tools used for directing forward search (heuristics), which remains the most popular popular approach [Alcazar et al., 2013].

2.1.1 The Planning Domain Definition Language (PDDL)

The Planning Domain Definition Language (PDDL) is a set of popular languages used to define classical planning problems. Based on STRIPS [Fikes and Nilsson, 1971] (which was similar, but did not permit negative preconditions), PDDL uses a layer of abstraction to allow the specification of related facts and actions more compactly than enumerating them. It provides a standard and expressive format for defining problems, while remaining agnostic about how they might be solved.

The following overview of the components necessary to define a planning problem in PDDL1.2 [McDermott et al., 1998] uses examples extracted from the motivating example first introduced in Figure 1.1, and defined in full in Appendix A. Problems defined in earlier versions are typically forwards compatible. PDDL requires a domain file, which defines elements that will be used by multiple problems; and a problem file, which defines a specific instance and task. Terms are bounded by parentheses, and significant elements begin with a colon: `(:element contents)`.

Elements of a domain file:

- Requirements - lists optional elements of PDDL that are used. In practice, most parsers seem to ignore this and crash when something is used that they cannot handle. Eg: `(:requirements :strips :types)`

- Types (optional) - lists types that are used to group objects. These can be used to restrict the arguments of predicates or actions. If types are not used, then all objects are treated as a universal type, and may be used for all arguments of predicates and actions. The combinatorial nature of n-ary predicates permits a large number of possible groundings. Not all combinations will be reachable in most problems, and so parsers/planners typically only instantiate predicates that are needed. A hierarchy of subtypes can be defined with -. Eg:
  `(:types location object - concept)`

- Predicates - defines n-ary predicates, with arguments prefixed with a ?, and optionally associated with a type that restricts what objects may be used to ground them. Eg:
  `(:predicates (at ?loc - location) )`

- Constants (optional) - lists objects (see definition below) that will be present in all problems associated with the domain (optionally with their type if specified). Other than where they are defined (which enables constants to be used in action definitions), constants are equivalent to objects. Where we refer to objects, we also refer to constants implicitly. Eg:
  `(:constants Library - location Paper - object)`

- Actions - Aside from their name, actions have 3 components. Eg:
  `(:action move`

  - Parameters - defines parameters prefixed with a ?, and optionally associated with a type. These are only defined within the scope of that action. Eg:
    `:parameters (?from ?to - location)`

  - Precondition - lists predicates (defined earlier) with their arguments replaced by the actions parameters. Preconditions are combined with an AND logical operator. Supposedly other operators (such as OR) are available, although these are rarely used or supported. NOT can be used to negate a predicate, although not all

parsers or planners support this in preconditions. Eg:

`:precondition (and (at ?from) (connected ?from ?to))`

– Effect - same format as preconditions. `(not (predicate))` is much more widely used and supported for effects. Eg:

`:effect (and (at ?to) (not (at ?from))))`

Elements of a problem file:

- Objects - lists values that arguments to predicates (or action parameters) can take, and may be typed (as discussed above). They represent ways for other elements of the domain to be grounded. Eg:
  `(:objects Storeroom C1 - location)`
- Init - lists ground predicates (predicates with objects specified as arguments) that are true in the initial state. Eg:
  `(:init (connected Storeroom C1) (at C1)`
- Goal - lists ground predicates that the planner must make true in a single state in order to solve the problem. Eg:
  `(:goal)`

There are 3 main versions of PDDL (1 [McDermott et al., 1998], 2 [Maria Fox and Derek Long, 2003, Edelkamp and Hoffmann, 2004], 3 [Long and Gerevini, 2006]), and many extensions and alternative versions based on these. PDDL2 adds durative actions and numeric fluents, allowing some aspects of time and resource management to be represented. PDDL3 adds preferences and constraints in order to represent optimisation problems. Several other extensions have been made to the language, for example to allow the modelling of probability (eg. PPDDL [Younes and Littman, 2004], RDDL [Rao et al., 2016]), multiple agents (eg. MAPL [Brenner, 2003], MA-PDDL [Kovacs, 2012]), and decomposable tasks (eg. HDDL [Holler et al., 2020]).

The core version of PDDL1.2 used here can be viewed as a compact representation of classical planning, by observing the following mapping:

$$predicates \times types(objects) \to F$$
$$actions \times types(objects) \to A$$
$$init \to I$$
$$goal \to G$$

Here, we treat types as a filter on objects to only ground predicates and actions with objects matching the type specified in their definition. For example:

$$at(Storeroom) \in F$$

Is a valid grounding of the predicate `(at ?loc - location)`, because `Storeroom` is of type `location`. Whereas

$$at(Paper) \notin F$$

Because `Paper` is of type `object` (and `concept`), but not `location`.

In this way, objects are compiled away to make a finite set of ground predicates corresponding to facts, and ground actions. This direct mapping allows us to use the set notation of Classical planning (Definition 1) to explore mathematical properties of planning problems, while relying on PDDL as inputs to implementations of algorithms based on them.

While PDDL has a strict separation of domain from problem (although some elements like constants/objects could be in either), this is not followed in the literature. Unless PDDL is specified, we typically use the term planning problem to refer to the all the information

included in Definition 1. All experiments presented here use planning problems derived from those found in examples folder of the HSP2 repository [github: hsp2,], which are organised into folders containing many problems (often sharing a single PDDL domain file). We refer to these folders and the collection of problems they contain as a domain. In most cases, it does not matter whether the problem is specified in PDDL, some other formal language, or in mathematical notation like first order logic or set theory.

2.2 Planning with heuristics

The definition of a planning problem above lends itself to representation as a tree or graph, with a plan typically being a path between nodes. Methods for finding a path in a tree are either exhaustive (eg depth first, Dijkstra), which is prohibitively expensive for even moderately large problems, or make use of a heuristic to guide them. Because states are comprised of combinations of facts, the space of possible states grows exponentially with $|F|$, which further necessitates the use of a heuristic to avoid searching the whole space of possibilities.

The term "heuristic" has been used to convey various meanings, loosely connected to finding something. Our usage here is based on the third meaning proposed by [Langley, 2017], which is an evaluation criteria for selecting between partial solutions to a planning problem.

Definition 2 (Heuristic (general))**.** A function that maps a state to a value representing the utility of reaching it, in the context of reaching a state that satisfies the goal.

When used with search procedures like A* [Pohl, 1970], a narrower definition of heuristic is often used.

Definition 3 (Heuristic, $h(\sigma)$)**.** An estimate of the distance between a state σ and any state that satisfies all goal conditions $\sigma' \supseteq G$

This definition allows more theoretical analysis of heuristics, such as whether they are admissible, which provides a guarantee of finding an optimal (lowest cost) path.

Definition 4 (Admissible heuristic)**.** A heuristic is admissible if it never overestimates the distance between a state and a goal state.

Other fields of planning like path planning have cheaply computable, admissible heuristics such as Euclidian distance. This is not generally the case for task planning, although optimistic, or approximately admissible heuristics often perform well [Dechter and Pearl, 1985].

2.2.1 Simplifying approximations

A classical planning problem can be simplified for computing heuristics that will assist in solving the original problem. One such simplification is the *delete relaxation*, which removes all negative preconditions and negative effects from all actions. This makes it strictly easier to find a plan, as once a fact has been made true, it will remain true.

Definition 5 (Delete relaxation, Π^+)**.** The delete relaxation transforms a planning problem:

$$\Pi = \langle F, A, I, G \rangle \rightarrow \Pi^+ = \langle F, A', I, G \rangle$$

Where each action in the original problem $a = \{pre^+(a), pre^-(a), eff^+(a), eff^-(a)\} \in A$ is replaced with $a' = \{pre^+(a), \emptyset, eff^+(a), \emptyset\}$. This removes all negative preconditions and effects.

A range of successful heuristics estimate the minimum cost of a plan in the delete relaxation. The Heuristic Search Planner (HSP [Bonet and Geffner, 2001a, Bonet and Geffner, 2001b]) defines a heuristic $h_1(\sigma)$ (originally named $h_{max}(\sigma)$) that finds the minimum cost of the most expensive fact in G. This was shown to be part of a family of heuristics $h_m(\sigma)$ [Haslum and Geffner, 2000] that finds the minimum cost for the most expensive tuple of m facts (where $m = 1$ is the heuristic used in HSP).

Applying the delete relaxation to a planning problem involves a loss of information, and can lead to actions or action sequences that are impossible in the original problem being considered as possible in the relaxed problem. [Haslum, 2009] showed that by compiling facts into tuples

before applying the delete relaxation, they were able to preserve some of the information that is lost by the delete relaxation. By considering pairs ($m = 2$), triplets ($m = 3$) or larger conjunctions of facts, the requirement to achieve them simultaneously is preserved, in a new (larger) problem Π^m that is free from negative effects or preconditions. Unfortunately, this rapidly increases the space of possible tuples.

The Fast Forward (FF [Hoffmann and Nebel, 2001]) heuristic $h_{FF}(\sigma)$ also uses the delete relaxation, but instead uses GRAPHPLAN [Blum et al., 1997] to estimate the cost of a delete-relaxed plan that achieves all facts in the goal.

Definition 6 (Fast Forward heuristic, h_{FF})**.** The Fast Forward heuristic is calculated for a state by performing reachability analysis (Definition 7) on the delete-relaxed problem, and identifying the first state in which the goal is achieved σ_g. A delete-relaxed plan is extracted by backtracking from σ_g, taking the action that first achieves each fact, and the preconditions for those actions. h_{FF} is the number of actions in this delete-relaxed plan.

Definition 7 (Reachability analysis)**.** The set of all facts reachable from a state σ_0 under the delete relaxation can be found by applying all actions applicable to σ_0 to yield σ_1, then to σ_1 to yield σ_2 etc. This is continued until $\sigma_i = \sigma_i$

The Fast Forward planning system uses this heuristic, and also forces the choice of *helpful actions*. These are actions in the delete relaxed plan that are found between σ_0 and σ_1.

2.2.2 Landmarks

Another class of heuristic that has been used successfully is based on the concept of landmarks. Landmarks as a heuristic were initially described in [Porteous et al., 2001] and developed further in [Hoffmann et al., 2004], as an extension to goal ordering.

Definition 8 (Landmark)**.** A landmark is defined as a propositional formula over which facts are true in the same state, that evaluates to true in at least one state that is visited during the application of any valid plan. Due to the difficulty in identifying propositional formulae over multiple facts, many planning systems (including LAMA [Richter and Westphal, 2010])

only make use of fact landmarks. Fact landmarks are single facts that are true in at least one state that is visited during the application of any valid plan.

Definition 9 (Trivial landmark)**.** Each fact in the goal and the initial state is a landmark, but is of little use to a planner; these are referred to as *trivial landmarks*.

Definition 10 (Non-trivial landmark)**.** Facts that must be true at some point other than at the start or end of a plan execution are considered *non-trivial*.

Illustrative example

Figure 2.1 depicts two plans that are applicable to the initial state shown, in the example planning domain described in Figure 1.1. A PDDL representation of this problem can be found in Appendix A.

Other plans exist (there is nothing to prevent the student running circles moving around the department, or playing games before completing the assignment), but those shown represent the two basic approaches that could be taken. Other than facts in the initial state and goal (ie trivial landmarks), the only fact that becomes true in both plans is $has(Doc)$. This fact is required by both actions ($printlibaction$, $printofficeaction$) that can achieve the goal fact $has(Hardcopy)$, and so must be achieved in any plan - ie it is a landmark.

Suppose $has(Computer)$ were removed from the initial state. The only action that could achieve it is $fixcompaction$, which has preconditions $[at(IT), has(Biscuit)]$. The

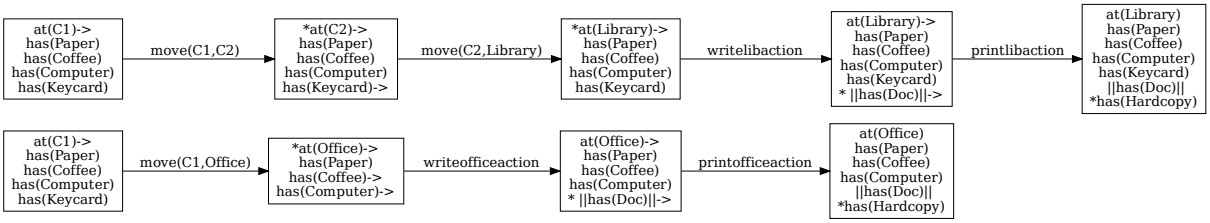


Figure 2.1: **Example plans to illustrate landmarks** Two plans applicable to the same initial state (shown far left). Boxes show states, defined by the facts that are true in them. Actions are represented by the arrows between states. As discussed in the text, the only landmark present in this problem is `has(Doc)`, which is surrounded by || In each new state, facts that were made true by the preceeding action are preceeded with a *. Facts that are preconditions of the next action are appended with $->$.

first, $at(IT)$ is reachable, but there is no action that can achieve $has(Biscuit)$. This leaves the bottom plan of Figure 2.1, (and variations based on the strategy of printing or working in the office) inapplicable. Removing this fact from the initial state also has the unintuitive effect of creating new landmarks. Because all plans must now follow the strategy of the top plan, they must all achieve $at(library)$ at some point. While $at(C2)$ also becomes true in the plan shown, it is still not a landmark, because the library can be accessed via C3.

Use as a heuristic

Early uses for landmarks ([Porteous et al., 2001, Hoffmann et al., 2004]) focussed on finding and using ordering constraints between landmarks. This partial ordering of landmarks was treated as a sequence of sub-goals to break a planning problem into sub-problems. Most (but not all [Büchner et al., 2021]) subsequent systems [Richter et al., 2008, Karpas and Domshlak, 2009, Richter and Westphal, 2010, Pereira et al., 2017] use a landmark counting heuristic, which assumes that each landmark that has yet to be reached will cost one action (although [Karpas and Domshlak, 2009] and [Richter and Westphal, 2010] allow this cost to vary with action cost). The estimate of distance to the goal is then given by the number of landmarks remaining to be found.

Definition 11 (Landmark counting heuristic, h_{LC})**.** The landmark counting heuristic [Richter and Westphal, 2010] calculates a set of landmarks in the initial state. This is propagated through states as they are explored, removing landmarks as they are accepted (ie become true in that state), and adding them back if the ordering information requires it:

$$h_{LC}(\sigma_i) = |L(\sigma_i)|$$
$$L(\sigma_i) = (L(\sigma_{i-1}) \setminus Accepted(\sigma_i)) \cup RequiredAgain(\sigma_i)$$

Methods for finding landmarks

Landmarks were originally [Porteous et al., 2001, Zhu and Givan, 2003, Hoffmann et al., 2004] found using a Relaxed Planning Graph (RPG [Hoffmann and Nebel, 2001]), which

performs reachability analysis (Definition 7) under delete relaxation (Definition 5) conditions followed by multiple steps to check that landmarks and their order are sound. Since then, other ways to compute landmarks have been explored.

One example is the translation of RPGs into AND/OR graphs for which landmarks are unique maximal solutions computed using Bellman-Ford methods [Keyder et al., 2010]. The AND/OR algorithm does not permit negative action effects, so is either used alongside the delete relaxation, Π^+, or with the method described in section 2.2.1 for compiling groups of facts into a domain description that maintains the information of negative preconditions and effects without containing those features [Haslum, 2009].

While most work that makes use of landmarks only use single fact landmarks, due to the complexity of using disjunctive landmarks, there have been recent attempts [Wichlacz et al., 2022] to represent ungrounded or *lifted* landmarks. Another recent innovation is based on the observation that cyclic dependencies among landmarks indicate that at least one such landmark must be required again [Büchner et al., 2021].

Applications of landmarks

Landmarks have also been used in several fields related to task planning, such as goal recognition [Pereira et al., 2017, Vered et al., 2018, Pereira et al., 2020] and contingent planning [Maliah et al., 2018, Segovia-Aguas et al., 2022]. But because the existence of landmarks depends on an initial state that permits the existence of plans, they can only be used in such problems. The only example we could find of landmarks being used in a planning problem with an incomplete world model is [Pereira, 2020], which modifies landmarks to work in an incomplete planning problem. However, their definition of an incomplete planning problem is a classical planning problem where actions are redefined as incomplete actions. In addition to classical actions, these also have possible preconditions and possible effects. Possible preconditions are ignored, and a *possible landmark* is then defined as a fact landmark that is extracted from a possible effect of an action. While interesting, this still relies on a completely

defined initial state (see Section 2.3.3), unlike the work presented here, as well as a different definition of actions.

2.2.3 Novelty

Search procedures can be improved by focussing their exploration in directions most likely to achieve the goal, or by increasing the rate at which they explore new areas of the search space. The category of novelty based heuristics began as a method for pruning the state space according to how novel a state is when reached [Lipovetzky and Geffner, 2012].

Definition 12 (Novelty, $novelty(\sigma, S)$)**.** The novelty of a state σ is the size (sometimes referred to as width) of the smallest tuple of facts in σ that is not present in any stored in the history of states S. A lower value is more novel, as a larger tuple indicates that a greater degree of precision is needed to find something new. If nothing is new about σ, then $|\sigma|$ is used as the maximum (worst) value possible.

The Iterated Width (IW) search procedure [Lipovetzky and Geffner, 2012] performs exhaustive breadth first search while pruning states that have a novelty of greater than i, where i is increased at each iteration. For goals containing multiple facts, this works better when serialised - resetting the algorithm each time a goal is achieved, carrying forward the state containing the newly achieved goal as the initial state for the next call.

Later work [Lipovetzky and Geffner, 2017a, Lipovetzky and Geffner, 2017b] combine the use of novelty with heuristics to define variations of the Best First Width Search (BFWS) procedure. This selects states with the lowest novelty (ie most novel), and breaks ties according to other heuristics - in those papers, h_{FF} (Definition 6) and h_{LC} (Definition 11) are used.

[Katz et al., 2017] uses novelty as a heuristic directly, by adapting it to various forms of quantified novelty. These count the number of facts (or tuples of facts) that are novel in each state, allowing the heuristic to distinguish finer degrees of novelty, avoiding the many-way ties that [Lipovetzky and Geffner, 2017a, Lipovetzky and Geffner, 2017b] use a different heuristic to break.

The novelty based approach requires the maintenance of a history of states that have been visited, which restricts its use to applications that search through state space (making it incompatible with Chapter 5). Novelty does not attempt to estimate the distance to the goal, and so is a heuristic according to Definition 2, but not Definition 3.

2.3 Planning with incomplete world models

We use the term *world model* to refer to the planning agents model of the contents and dynamics of the world. This is mostly contained in Definition 1, but may also include representations of uncertainty that will be described in more detail in Section 2.4. We focus on the case that world models are incomplete because their initial state is incompletely defined. Although [Pereira, 2020], (discussed in Section **??**), [Sohrabi et al., 2010] (Section 2.3.3) and some systems in Section 2.5 work with modifiable action models.

[Baier et al., 2014] defines 3 classes of planning problem, based on whether the initial state is (in)completely defined, and whether there are sensing actions. Classical planning problems have a completely defined initial state (and therefore no need for sensing actions). Conformant planning problems have an incompletely defined initial state, but no sensing actions, and so must compute a plan that will work for all possible initial states. Conditional (also known as contingent) planning problems have an incompletely defined initial state, and sensing actions. In order to plan with unknown states, we must modify the action part of Definition 1 to specify what happens if an action is applied to a state to which it is not applicable. Typically, actions have no effect when applied to an inapplicable state, although other conditional effects are possible. This system also does not consider planning problems that have actions with stochastic effects.

Generalised planning is a related problem, that also has an incompletely defined initial state, and may [Srivastava et al., 2011] or may not [Segovia-Aguas et al., 2019] include sensing actions (and so does not fit neatly into the [Baier et al., 2014] system). When given a set of planning problems that share action definitions (and sensing actions if used),

a generalised planner finds a set of plans that map onto the problems [Hu et al., 2011]. Approaches include seeking a compact representation of the problem set or plans such as a policy (see Section 2.4) [Segovia-Aguas et al., 2022].

This section will review literature according to the system used by [Baier et al., 2014]. Classical planning has already been described in Section 2.1. One of the key points made by [Baier et al., 2014] is that while the problem of diagnosis initially appears to fall into the category of conformant or conditional planning, it can be reframed as fixing a classical domain.

2.3.1 Conformant planning

Conformant planners seek to find a plan that will achieve a goal state regardless of which of a finite set of possible initial states is given to it. They often use belief states, that specify possible states the world could be in (eg [Maliah et al., 2018]). Alternatively, they may reason about what they do or do not know, differentiating between True, False, and Unknown. This allows a more compact representation of possible states [Brafman and Hoffmann, 2004]. The probability distributions of possible states of the world are often considered [Hyafil and Bacchus, 2003, Cimatti and Roveri, 2000], and will be discussed further in Section 2.4.

2.3.2 Conditional planning

Planning systems that respond to sensory inputs can respond to unexpected situations. Whether or not they explicitly model them, they are able to handle non-standard action effects as these are equivalent to conditional actions acting on hidden facts in the initial state. Pseudo-random action effects could thus be modelled as deterministic conditional actions acting on a set of random variables in the initial state.

Using $[True, False, Unknown]$ as possible evaluations of ground predicates allows sensory actions to be treated like regular actions (with effects that change the values of facts from $Unknown$ to a definite value. This permits the use of classical heuristics [Hoffmann and Brafman, 2005]. Sensory actions provide information about the world, but special treatment is needed for handling logically inconsistent statements that can arise when false information

is received, or the world changes after a sensor reading. Architectures for integrating this information must often employ techniques to handle non-monotonic logic in case observations contradict information in the currrent world model [Schwering et al., 2015, Hanheide et al., 2017, Sridharan et al., 2019]. As with generalised planners, probability is often considered explicitly.

2.3.3 Fixing a domain

Many of the problems and approaches described in the previous sections can be characterised as fixing a planning problem. This can happen before (assumptive planning), during (sensing) or after (diagnosis) a plan has been executed. All of these challenges involve selecting a 'fix', usually from a predefined set of allowable fixes, and proving that it fixes the flaw. A flaw here means a goal (in addition to conventional planning goals, this could be an observation that needs to be explained) that could not be reached from the flawed domain description. Fixes can generally be grouped into fixing an incomplete initial state, or fixing action models that do not correspond to real world dynamics.

Fixing the domain before plan execution - assumptive planning

There are many possible reasons for a planning problem to appear unsolvable, that could permit a solution in the real world. Maybe the agents world model is missing some information (eg that information needs to be sensed, or was encoded incorrectly). Giving up when faced with an apparently unsolvable problem means accepting a bad outcome, that could be improved upon by accepting some risk by introducing potentially incorrect facts. This type of missing information can be reintroduced into the world model by assumption, which may be respresented as an action with no preconditions, and just the assumed fact as an effect. Many representations have a precondition *initial*, that is true in the initial state, negated as an effect of assumptive actions, and has no other way to achieve it. This ensures that assumptions are only made at the start of a plan, and cannot bypass negative action effects.

While there is a close relationship between assumptive and conformant planning, the recognition that not all potential fixes that have been provided will support viable plans leads to the challenge of selecting ones that do. This selection can be translated into a planning action, allowing the problem to be compiled into a classical planning problem [Davis-Mendelow et al., 2013]. This requires the pre-specification of what fluents are assumable in order to avoid trying to plan over all combinations of possible fluents.

[Hanheide et al., 2017] uses assumptive actions generated from default knowledge to populate the world outside a known area. This is not limited to a predefined list, but able to draw on information in a database of default knowledge. The combination of different types of domain knowledge at different levels of abstraction is further developed in [Sridharan et al., 2019]. The flexibility of these systems relies on reasoning with probabilities and non-monotonic logic that is beyond the scope of this thesis.

Fixing the domain during plan execution - conditional planning

Receiving corrective information about the domain during the execution of a plan can be treated as a trigger for replanning (with assumptive or diagnostic actions - see previous or subsequent paragraphs) or handled as part of a plan that involves sensing (ie a conditional planner - Section 2.3.2).

Fixing the domain after plan execution - diagnosis

The problem of diagnosis had been studied for several years [Reiter, 1987, de Kleer et al., 1992] before it was considered in the context of planning [Sampath et al., 1995, McIlraith, 1998]. It is still predominantly studied with logic based approaches rather than planning [Rodler, 2023], where it is often referred to as abduction [Denecker and Kakas, 2002, Bochman, 2007] (in contrast to other logic-based terms like deduction and induction). It has also been referred to as postdiction [Chopra and Zhang, 2001, Eppe et al., 2013] due to its relation to the terms use in psychology. While these terms carry slightly different associations, postdiction with speculating as to what has happened based on current observations;

diagnosis with more focus on current observations that differ from an expected outcome; and abduction treating the passage of time as an ordering of causation; they are also used interchangably.

Nearly all interpretations of the problem of task planning can be described as finding a sequence of actions applied to an initial state that result in some target (goal) state. Although the temporal relationship between the application of the algorithm and the execution of the actions is different, diagnosis by abduction can be descibed in the same way, and therefore has a direct correspondence to classical planning. This correspondence is made explicit by [Sohrabi et al., 2010] which encodes the observations requiring explanation as goals for a planning problem, and permits fixes of two types: completions of the initial state (implemented as assumptive actions applicable only in the initial state) and substitution of actions with alternative ("faulty") actions. Both are provided as a set of permissible fixes to the planner, and as such are encoded directly in the domain definition. Preferences for simpler explanations (ie fewer fixes used) as well as domain specific preferences for what fixes are more acceptable, are encoded as action costs. By reframing diagnosis as an assumptive planning problem, the complexity of the problem becomes that of classical planning rather than conformant (despite the incomplete specification of the initial state as noted in [Baier et al., 2014]).

Here, we use the definition of diagnosis as a planning problem described by [Sohrabi et al., 2010], expressed in the notation for planning based on [Keyder et al., 2010] that was defined in Section 2.1. Figure 2.2 illustrates the task of diagnosis on the example domain from Figure 1.1.

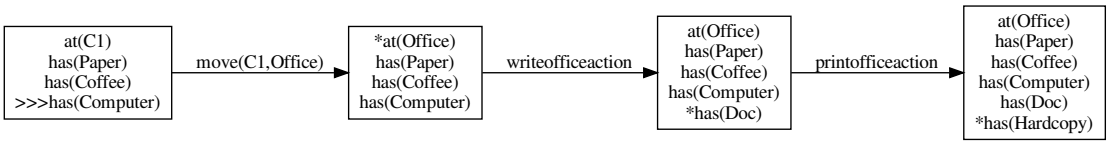
Definition 13 (Diagnostic planning problem, $\Pi_{diag} = \langle F, A, I, G_{obs} \rangle$)**.** A diagnostic planning problem $\Pi_{diag} = \langle F, A, I, G_{obs} \rangle$, consists of sets of ground predicates F and actions A, a model of the initial state of the world I, and a set of observations G_{obs} to be explained. In contrast to a classical planning problem, there does not exist a plan applicable to I that achieves G_{obs}.

Definition 14 (Fix, α)**.** A fix α, is a set of facts that are not true in the initial state $\alpha \subset F \setminus I$ that, when added to I, permit the existence of a valid plan. A fix is minimal if removing any facts from α makes the problem unsolvable again: $\forall f_i \in \alpha : \nexists \pi^{\langle F, A \cup \alpha \setminus f, I, G_{obs} \rangle}$. A candidate

```
(:init
    at(c1)
    has(paper)
    has(coffee)
    )
```

```
(:goal
    has(Hardcopy)
    )
```

(a) I **(left) and** G **(right) for a diagnosis problem** Based on the example domain described in Figure 1.1. There is no plan that is applicable in I that can achieve G.



(b) **Fix and explanation for the diagnosis problem above** Boxes show states, defined by the facts that are true in them. Actions are represented by the arrows between states. In each new state, facts that were made true by the preceeding action are indicated with a *. The fix $\alpha = \{has(Computer)\}$ is indicated in the modified initial state by $>>>$. The 3 actions displayed constitute a plan applicable to $I \cup \alpha$, and when combined with α, provide an explanation for how the student could have achieved the goal.

Figure 2.2: **Example diagnosis problem** No plan exists for the initial state defined above - without a keycard, the student cannot access either library action, and without a computer (or biscuit to have the computer fixed in IT), cannot write in the office either. However, if the student is observed as having the Hardcopy, an observer might attempt to diagnose what is missing from I. The fix that their computer was working ($has(Computer)$), allows the explanation that they went to their office, where they wrote and printed the document.

fix is a set of facts hypothesised to be a fix, but that has not yet been proven to permit a valid plan.

Definition 15 (Explanation, η)**.** An explanation $\eta = (\alpha, \pi^{\langle F,A,I\cup\alpha,G_{obs}\rangle})$, is a fix and a plan applicable to $I \cup \alpha$ that achieves G_{obs}.

[Göbelbecker et al., 2010] describes a similar problem, which they refer to as *coming up with excuses*. While they acknowledge a connection to abduction and diagnosis, they claim that the lack of a logic based framework marks it as a fundamentally different problem. Here we note that classical planning can be represented and reasoned with in the framework of first order logic [Russell and Norvig, 2016, Gelfond and Kahl, 2014, Asai, 2019], and conclude that a difference in representation does not constitute a difference in the fundamental problem being

solved. There are direct parallels between several terms used in [Göbelbecker et al., 2010] and those defined above:

- An (acceptable) excuse is a (minimal) fix (Definition 14).
- The relevant domain is the set of all fluents (ground predicates) that are connected to the goal by an operator/action by backtracking.
- Problems are generated by deleting a fluent (element of the initial state), or by removing an object/constant (and all ground predicates referring to it in the initial state). The latter may require multiple excuses.

Candidate excuses are found using only causal relationships found with causal graphs. Candidate excuses are then selected by performing a plan search using Fast Downward [Helmert, 2006]. When this finds a plan (equivalent to our definition of an explanation), the excuse(s) included in the plan is accepted as an excuse.

2.4 Planning with probabilities

Planners that explicitly model and take account of uncertainty are computationally expensive . At each timestep, an action may lead to many possible states, each of which must be kept track of, and may be followed by a different action. Rather than a plan being a linear sequence of alternating actions and states, the number of states increases exponentially with the length of the plan. Although finding ways to mitigate this is an active area of research, there is a fundamental difference in the size of domains that can be handled by deterministic and probabilistic planners. While this thesis will not make use of strategies for modelling the probabilities of facts being true in a state, or action effects, they represent alternative representations for similar concepts. Problems that involve probabilities are often divided into Fully Observable, Non-Deterministic (FOND) or Partialy Observable, Non-Deterministic (POND), according to whether or not the planner has direct information about the state of the world, or must rely on imperfect sensors. FOND problems can then be divided according to whether

they seek a strong solution (a policy that will always reach the goal) or a weak solution (a policy that could reach the goal). Distinguishing what type of solution exists for a problem is often referred to as reachability analysis, which is related to, but not the same as the classical planning technique described in Definition 7. POND problems typically do not provide enough information to the planner to enable this distinction. Instead, they model the effect of actions on the world as a Markov Decision Process - ie they assume that the history of actions taken does not matter, only the current state of the world (which may include hidden variables).

2.4.1 POMDP

Representing probabilistic planning problems as Partially Observable Markov Decision Processes (POMDPs) allows algorithms to search for the solution most likely to succeed. Because these must track the probability of being in multiple states after each action (meaning that a solution is a policy, deciding what is the best action to take from each state) , the memory and computation cost grows rapidly with the number of possible states [Kaelbling et al., 1998] and actions [Pineau et al., 2003]. The computational cost and complexity of various aspects of POMDP are surveyed in [Mundhenk, 2000].

One way to reduce this search space is to combine groups of states into important points, and only track belief states within the range of those points. These are known as point-based POMDP solvers. The *importance* of points is used as a heuristic, the appropriateness of which is critical to the effectiveness of the planner. Choosing points that have a large information gain available [Ma and Pineau, 2015], causes the planner to seek situations which constrain the belief state. Using highly connected nodes as waypoints [Kurniawati et al., 2008] [Kurniawati et al., 2011] can allow calculations on that region to be reused in similar situations. Such waypoints can be considered highly relevant to the problem, and as such, this approach is somewhat similar to the use of landmarks (see Section 2.2.2) or the relevance score that we define in Section 3.6.

Reuse of combinations of actions as macro-actions is another way of reducing the action space [Kearns et al., 2000,He et al., 2011,He et al., 2010]. This approach can be combined with

value iteration [Smith and Simmons, 2005] to combine histories of actions, further reducing the computational cost. Seeking compact representations in this way is equivalent to the approach used for generalised planning discussed in Section 2.3.1.

Despite these attempts to make search over POMDPs more efficient, there are still limits to the size of the state-action space that can be searched. One way to reduce this space before application of POMDP solvers is to utilise deterministic planners on a simplified version of the problem to find reachable solutions before searching those possible solutions for an optimal one. Reachability analysis approaches refine the state space to only include states that have a finite probability of being reached by a sequence of actions. This can be achieved by reducing a POMDP to a Deterministic Finite Automata [Lacerda et al., 2015] before relabelling arcs with probabilities to convert it back into a POMDP.

A limitation of standard approaches to POMDP planning is that they require discrete states, with any change in the value of a state variable being represented as another state. This makes it difficult to efficently represent many-valued variables like time or resources, and requires discretisation for continuous variables. These sorts of variables typically represent a limit, or constraint (see Section 2.4.2) on resource usage, making it inefficient to treat each value for that variable as a different state. This can be addressed in versions of POMDP that use state-variables [Wang et al., 2014] by applying a large reward penalty to entering states that do not satisfy constraints. Another approach is to set transition probabilities to constraint-violating states to 0 [Brázdil et al., 2016]. However it is worth noting that POMDPs aim to optimise policies rather than search for viable plans. Therefore they still return a plan even if no plan is likely to succeed.

Plans can also be generated efficiently in such a deterministic space, and then their probability of success assessed to decide between plans. [Teichteil-Königsbuch et al., 2010] finds deterministic plans using h_{FF} [Hoffmann and Nebel, 2001] before aggregating them into a policy. Similarly, [Zhang et al., 2014] finds deterministic plans using ASP [Elkhatib et al., 2004] for a high level representation of the planning problem, and uses a POMDP to work out the details at a low level.

2.4.2 Constraint satisfaction planners (CSP)

While basic constraint satifaction planners (CSPs) do not model probability, they have been adapted for use alongside representations that do. We first describe the general form of a constraint satisfaction problem (using notation and definitions based on those in Chapter 8 of [Ghallab et al., 2004]) before describing some versions that use them to represent probabilistic planning problems.

A set of variables X represents the decisions that need to be made (ie which action taken at each time step). Each variable is associated with a domain D, that specifies the set of values that the variable may have. Finally a set of constraints C defines the permitted combinations of values that may be assigned to tuples of k variables. While k can be any number (up to the $|X|$), if $k = 1$ (a unary constraint), it is equivalent to reducing the domain of the variable involved.

An assignment of values to some or all variables is consistent if it does not violate any constraints. A solution is a consistent assignment to all variables. The main advantage of CSPs over other searchable representations of problems is the ability to reduce the search space by checking the consistency of constraints and propagating them.

Because the computational complexity of consistency and search algorithms tend to be polynomial dependent on the size of domains, the number of variables, and the number of constraints, large gains in efficiency can be achieved by dividing CSP into smaller ones [Xing et al., 2006] [Li and Epstein, 2010]. This does not affect the consistency of the problem if there are no constraints linking the smaller CSPs. Once all variables that link two subnetworks have been assigned, they can be treated separately.

It is possible to exploit this property to reason about a problem at multiple levels of abstraction and from different perspectives. This can be described as a metaCSP [Mansouri and Pecora, 2016], in which high level decisions about what actions to take form the variables of the top level CSP, and the consistency of these decisions is checked by a collection of lower level CSPs corresponding to metaconstraints such as temporal, spatial and resource availability constraints. If an inconsistency or flaw is discovered in one of these metavariables,

it can be resolved by altering the assignments in the top level CSP, or by adding appropriate metaconstraints. The hierarchical structure of metaCSP allows uncertainty to exist at lower levels, bounded by the metaconstraints.

Temporal CSP

In many situations, plans are required that contain an explicit representation of time. For example if different actions have different durations and must be performed within different deadlines or time windows. If multiple actions can be performed at once, there may be restrictions on what can be done simultaneously. Many frameworks for planning with time are based on the Temporal Constraint Satisfaction Problem [Dechter et al., 1991], which encodes timepoints that must be decided as CSP variables (ie nodes in a temporal constraint network TCN), and intervals/constraints between those timepoints as directed (forwards in time), labelled arcs. Consistency is determined by searching for the shortest path between a pair of nodes. If a negative cycle is found then the network is inconsistent.

Probabilistic temporal CSPs

The time it takes to achieve something is often uncertain, so temporal planners have been developed based on [Dechter et al., 1991] to handle uncertainty. [Vidal and Fargier, 1999] splits simple temporal network intervals into controllable and uncontrollable. Solutions to this STNU (Simple Temporal Network with Uncertainty) can then be categorized as follows:

- Strongly controllable - There exists an assignment of controllable intervals that satisfies all possible assignments of uncontrollable intervals
- Weakly controllable - There exists an assignment of controllable intervals for each uncontrollable assignment, but different uncontrollable assignments may require different assignments of controllable assignments.
- Dynamically controllable - At each time point, there exist satisfactory assignments of future controllable variables

Probabilistic Simple Temporal Problem (pSTP) [Tsamardinos, 2002] extends the representational capacity of [Vidal and Fargier, 1999] by associating probability density functions to uncontrollable variables. They use gradient descent to look for the maximum probability of success in a static schedule. Chance constrained probabilistic Simple Temporal Problem (cc-pSTP) [Fang et al., 2014] sets a bound on the probability of failure, and then chooses constraints on time points that satisfy this bound. This reduces the problem to an STNU. [Wang and Williams, 2015] allows the planner to decide what parts of a schedule are permitted to show variation by assigning a risk bound for each temporal constraint, and iteratively reassigns them to find plan that satisfies all of the temporal constraints validated using [Fang et al., 2014]'s algorithm.

2.4.3 Policy learning

Given the complexity and computational cost of planning while explicitly modelling uncertainty (as discussed earlier in section 2.4.1), it can be tempting to resort to model free gradient descent techniques that learn how best to act in certain situations over time.

Reinforcement learning, inspired by neuroscience [Sutton and Barto, 1987, Rescorla and Wagner, 1972], learns the value of performing actions in certain situations according to the rewards received as a result of those actions. Learning these values for every possible action/state is difficult to impossible for most real world applications, so modern implementations typically use some form of value function approximation [Mnih et al., 2013, Tesauro, 1995] to generalise between similar states. This type of approach can perform very well [Gu et al., 2016, Silver et al., 2016] within domains that it has time to practice in, but the lack of explicit world models makes it hard to generalise to new domains, or draw knowledge from an external knowledge base. There have also been attempts to make hierarchical reinforcement learning architectures [Bakker and Schmidhuber, 2004, Parr and Russell, 1998] in order to facilitate learning new situations as they are encountered.

2.5 Reasonable

As mentioned in Section 1.2, it is hoped that future work will combine the concept of relevance with that of reasonableness to achieve something approximating common sense. We believe that both of these will be required for a planning agent to be robust to incomplete or damaged world models. While the focus of this thesis is relevance, here we consider approaches to deciding whether an assumption is reasonable.

A model is a simplified representation of parts of the world that permit predictions and plans to be made about how the world will respond to actions. If the model does not accurately reflect the behaviour and contents of the world, this can lead to plans being found that work in the model, but are impossible to enact, or even meaningless in reality. For this reason, it is desirable to check whether an assumption is likely to hold before taking actions predicated upon it. There are many possible sources of such evidence: direct observations of the real world (eg [Stone, 1998]), logs of past plans (eg [Sridharan et al., 2017, Cresswell et al., 2013]) or a database of default assumptions that may be made in the absence of specific knowledge [Hanheide et al., 2014]. While both sensing actions and logs of previous plans are valid sources of information, they are limited to selecting from possible states that have already been directly specified.

The field of natural language processing [Raina et al., 2005, Madabushi and Lee, 2016] makes effective use of much larger databases that are procedurally generated from less reliable sources such as natural language text [Regneri et al., 2010] and crowdsourcing (eg ConceptNet [Speer and Havasi, 2012] or COMET [Bosselut et al., 2019]), but use of this approach has been limited in planning. The Framer system [Lindsay et al., 2017] uses natural language to generate PDDL domain models from natural language descriptions of events, but this is effectively another form of learning from past plans (described in natural language), with similar limitations.

Large Language Models (LLM) have seen rapid advancement in recent years, although attempts at establishing benchmarks and systematic testing procedures for reasoning in the context of task planning [Valmeekam et al., 2023, Kambhampati, 2024] and mathematics

[Li et al., 2024, Mirzadeh et al., 2024] suggest that they do not currently have this capability. All of these studies report that the responses of LLMs are chaotic, in the sense that small changes to their inputs can lead to dramatically different outputs. While they are capable of answering some simple reasoning questions, the non-sensical way in which this breaks down for more difficult problems supports the hypothesis that their successes are due to learning the expected answer to a question (ie interpolating a pattern) rather than performing abstract reasoning.

Chapter 3

The Relevance Score

In this chapter, we formally describe the concept of relevance as it applies to task planning, and the notation used to describe it. Informally, our aim is to quantify how likely a fact or action is to appear in a partial plan. We build upon the representation of classical planning (see Definition 1), in which a problem is defined by finite sets of facts and deterministic actions (with preconditions and effects), an initial state and a set of goals. To do so, we must first define how the planning problem, and partial plans will be represented. We begin by formalising a representation of a classical planning problem as a tree, rooted at the goal(s). Then, we describe our notation for Lowest Common Ancestors (LCAs), and an algorithm for efficiently computing the sets of them that we will need later for calculating the relevance score. Partial plans will be sampled by a Hypothetical Random Regressor traversing the tree by following a procedure that we outline.

The rest of the chapter is concerned with defining notation and procedures for calculating probabilities. First, we consider the probability of the HRR visiting a specific node, which we refer to as a choices counter ξ. We call this the choices counter because it's inverse increases proportionately to the number of options available every time a choice is made by the HRR. Then, we define the relevance score Ξ, which is the probability that the HRR will visit any node with a particular label. Finally, we show how the sets of LCAs found near the start can be used to efficiently calculate the relevance score.

3.1 Backtracking tree

A tree is a standard representation of a classical planning problem (and changes to the world model). A tree consists of nodes $\underline{n}$ and edges e. A node $\underline{n} = \langle l, E \rangle$ consists of a label $label(\underline{n}) = l \in F \cup A$, which references a fact f or action a, and a set of edges E. An edge $e = (\underline{n} \rightarrow \underline{c})$ links a parent node $\underline{n}$ to a child node $\underline{c}$.

Given an edge $e = (\underline{n} \rightarrow \underline{c})$, the function $parent(\underline{c})$ yields $\underline{n}$. The function $children(\underline{n})$ yields a set of nodes such that for each node $\underline{c}$, $parent(\underline{c}) = \underline{n}$. The root of a tree is the only node with no parent, i.e., $parent(\underline{root}) = None$.

In seeking to quantify relevance, we choose this representation of a tree as it models how each fact or action relates to the goal, without necessarily relating them to the initial state. Backchaining has been used to identify landmarks since their initial definition [Porteous et al., 2001], although more efficient methods have since been introduced [Keyder et al., 2010].

In order to represent a planning problem with multiple goals, we modify all domains by adding action $achieveGoal$ that has a single positive effect $eff^{+}(achieveGoal) = \{success\}$, and the problem's goals as preconditions $pre^{+}(achieveGoal) = G$. Then, $success$ is used as the goal for computing heuristics, allowing all goals to be considered jointly.

Definition 16 (Tree: Backtracking tree, T_{Π})**.** A delete-relaxed planning problem $\Pi = \langle F, A, I, G \rangle$ (see Definitions 1 and 5 in Chapter 2) defines a tree T_{Π}. Node $\underline{root}$ is the root of T_{Π} and has $label(\underline{root}) = success$. An action node $\underline{a}$ (i.e., a node whose label is an action), has children whose labels are its preconditions $f \in pre(label(\underline{a}))$. A fact node $\underline{f}$ has children whose labels are actions a such that $label(\underline{f}) \in eff(a)$.

The function $L(l)$ maps a label l to all nodes in the tree that have that label:

$$L(l) = \{\forall \underline{n} : label(\underline{n}) = l\}$$

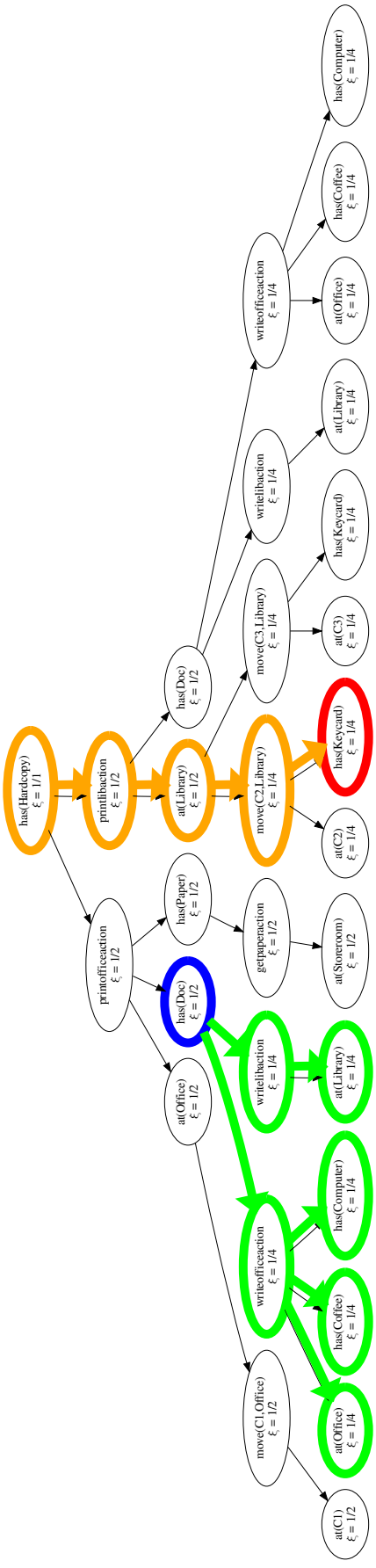


Figure 3.1: **Example problem as a tree** The top 5 layers (3 fact, 2 action) of $\overline{T_\Pi}$ are shown. The **path** of the **red** node, and **descendents** of the **blue** node are marked in the colours indicated. Choices counter values $\xi(\underline{n})$ are shown for each node.

Definition 17 (Tree: Path of a node, $T_{\Pi}^{path}(\underline{n})$)**.** The path of a node is the sequence of alternating fact and action nodes that is generated by adding the parent of the current node until the root is reached:

$$T_{\Pi}^{path}(\underline{n}) = [\underline{n}, parent(\underline{n}), ..., \underline{root}]$$

Definition 18 (Tree: Descendents of a node, $T_{\Pi}^{desc}(\underline{n})$)**.** The descendants of a node $\underline{n}$ are all nodes that have $\underline{n}$ in their path:

$$T_{\Pi}^{desc}(\underline{n}) = \{\forall \underline{d} : \underline{n} \in T_{\Pi}^{path}(\underline{d})\}$$

Actions are excluded from the children of a fact node $\underline{f}$ if any preconditions of that action appear as labels on any node in $path(\underline{f})$. This prevents cycles, which would represent unachievable requirements.

Figure 3.1 shows the top few layers of the tree representation of the example problem shown in Figure 1.1, with illustrations of $T_{\Pi}^{path}(\underline{n})$ and $T_{\Pi}^{desc}(\underline{n})$ applied to arbitrary nodes.

A complete tree could be generated by performing Algorithm 3 with the following modification. If $\rho = 0$ (line 3), then the tree will be explored until there are no nodes left to add - ie it is complete. In Section 3.5, we discuss why this is an unnecessarily expensive approach, and how we choose a more sensible value for ρ.

A planning problem defines a single tree, but multiple planning problems can define the same tree. This could happen if actions are added to a problem, that do not have any effects corresponding to nodes in the tree of the original problem. Similarly, the addition or removal of facts that are not labels for any nodes would not change the tree.

3.2 Lowest common ancestors

Traversing the whole of a tree with a large number of nodes is costly. Section 3.7 will describe methods to calculate the relevance score by visiting a small subset of nodes that can be identified once and reused each time a relevance score is calculated. This will rely on the identification of nodes at which paths diverge.

Definition 19 (**L**owest **C**ommon **A**ncestor (LCA))**.** The Lowest Common Ancestor (LCA) of two nodes $\underline{n}$ and $\underline{m}$ is the lowest node in the tree which is an ancestor of both nodes:

$$LCA(\underline{n}, \underline{m}) = \underset{\underline{i} \in T_{\Pi}^{path}(\underline{n}) \cap T_{\Pi}^{path}(\underline{m})}{argmax} (|T_{\Pi}^{path}(\underline{i})|)$$

This computation is associative and can generalize to any number of nodes.

The backtracking tree consists of two types of nodes, those with a fact label, and those with an action label. Because the HRR (which will be defined in Section 3.3) behaves differently at each of these, it will be useful to distinguish which of these is the LCA for groups of nodes. For the purposes of calculating $\Xi(l)$, these definitions will apply to sets of nodes with the label being evaluated, $K = L(l)$. Figure 3.2 illustrates the concept of LCAs, and identifies the $aLCAs$ and $fLCAs$ for the red or green nodes.

Definition 20 (**a**ction **L**owest **C**ommon **A**ncestors ($aLCAs(K)$))**.** $aLCAs$ are any action nodes that are the LCA of any subset of $K \cap T_{\Pi}^{desc}(\underline{f})$

Definition 21 (nodes with **f**acts for **L**owest **C**ommon **A**ncestors ($fLCAs(K)$))**.** $fLCAs(K)$ is the subset of nodes in K whose paths diverge at fact nodes below node $\underline{f}$:

$$fLCAs(K) = \left\{ \forall \underline{n} \in K : LCA(\underline{l}_i, \underline{l}_j) \in F \cap T_{\Pi}^{desc}(\underline{f}) \right\}$$

Much research has gone into finding the LCA of a pair of nodes in a tree, or more generally a Directed Acyclic Graph (DAG) [Djidjev et al., 1991, Czumaj et al., 2007, Eckhardt et al., 2007]. When applied to DAGs, the lack of a single root as the ancestor of all nodes means that

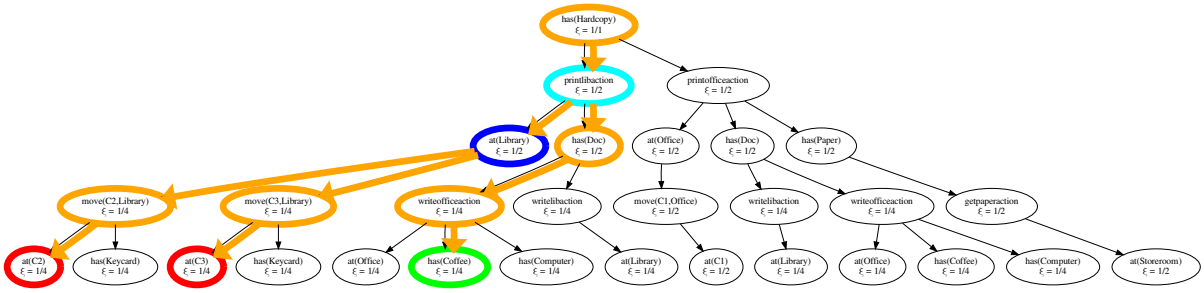


Figure 3.2: **Examples of LCAs** The top 5 layers (3 fact, 2 action) of $\overline{T_\Pi}$ are shown.
The **paths** of the two **red** nodes meet at the **blue** node, which is a fact. The **paths** of the **blue** node and the **green** node meet at the **cyan** node, which is an action.
This means that among the 3 **red** or **green** nodes, there is 1 **aLCA** that is a direct descendent of the root.
This **aLCA** has 2 children. The leftmost of which has no $aLCAs$, and its $fLCAs$ are the two **red** nodes. The rightmost of which has no aLCAs, and just the **green** node as an $fLCA$.

pairs of nodes can have multiple LCAs. Solutions can therefore either be exhaustive (find all LCAs) or representative (find at least one proven LCA). This is an extension that complicates the problem in a way that does not help us here.

The problem of finding LCAs in trees is often translated into the task of calculating Range Minimum Querys (RMQs) [Bender and Farach-Colton, 2000, Bender et al., 2005, Fischer and Heun, 2006], the procedure and proof of equivalence of which is given in [Bender and Farach-Colton, 2000].

Definition 22 (**R**ange **M**inimum **Q**uerys (RMQs) task)**.** Given an array A of length n, and indices $0 \geq i \geq j \geq n$, find the index of the smallest element of the subarray $A[i...j]$.

Both tasks are approached by preprocessing the input (tree or array) in order to make individual queries faster. The complexity of the fastest method [Fischer and Heun, 2006] (based on a solution to RMQ [Berkman and Vishkin, 1993]) costs $O(h)$ to pre-process the tree, where h is the height of the tree, and then $O(1)$ for each query on a pair of nodes. This approach would thus cost $O(h + n^2)$ to find the LCA for each pair of nodes, where n is the number of nodes whose LCAs we want to find.

Instead of this established procedure, we can take advantage of the fact that many LCAs will be shared by multiple pairs; there are at most hn unique LCAs. This procedure is described in Algorithm 1. We first sort the nodes by their paths (line 1; with complexity $O(hn \log(n))$).

Algorithm 1 - Find aLCAs Given a partially explored tree $\overline{T_{\Pi}}$, and a list of nodes in that tree with a given label $TargetNodes = L(l)$, both as found by Algorithm 3, this algorithm yields a list of $aLCAs(L(l))$, and a structure linking some members of $fLCAs(L(l))$ and $aLCAs(L(l))$ to the root of the tree. The implementation details of registering $aLCAs$ and $fLCAs$ is ommited for clarity. Line 1 sorts nodes alphabetically, treating a path as a string, and a node as a character.

```
1: Let ListOfPaths = sorted(TargetNodes)
2: Let j = 0
3: while Some nodes unregistered do
4:    for all i ∈ [0 : len(ListOfPaths) − 1] do
5:       if ListOfPaths[i, j] ≠ ListOfPaths[i + 1, j] then
6:          if i and i + 1 have been registered as linked then
7:             continue
8:          end if
9:          if j is a fact layer then
10:            Register ListOfPaths[i, j] and ListOfPaths[i + 1, j] as linked by an aLCA
11:         else
12:            Register ListOfPaths[i, j] and ListOfPaths[i + 1, j] as linked by a fLCA
13:         end if
14:      end if
15:   end for
16:   j + +
17: end while
```

The order does not matter, just that nodes with the same path up to a certain distance from the goal are in a contiguous group, so nodes are treated as symbols in a sequence (i.e., the path starting at the goal) and sorted alphabetically by their labels. We then traverse along the sorted paths (line 16; variable j tracks the distance traversed along the paths from the root) with worst case complexity $O(hn)$), checking for differences between nodes in adjacent paths (line 5). When a (new, as checked by line 6) difference in adjacent paths is detected, it is registered as linked by an LCA. Whether j is odd or even determines whether this LCA is an fact or action LCA (lines 9-13). This algorithm yields an ordered list of action LCAs ($aLCAs$), and sets of the $fLCAs$ and $aLCAs$ that are direct descendents of the root (ie have no other $aLCAs$ between them and the root). Sets of $aLCAs$ are computed once for each fact, and then filtered by σ when the state aware relevance score $\Xi_{\sigma}(l)$ (Definition 28) needs to be calculated.

3.3 Hypothetical Random Regressor

The aim of the *relevance score* is to estimate how frequently a fact must become true in some distribution of partial plans. We use the behaviour of a **H**ypothetical **R**andom **R**egressor (HRR) to define this distribution. The intention of defining the behaviour of this HRR is to establish a framework for analysing the tree. No implementation of the HRR or Algorithm 2 is necessary.

Definition 23 (Tree: Sub-tree, S_Π)**.** A sub-tree S_Π of T_Π, consists of some subset of nodes in T_Π and their paths, chosen according to Algorithm 2.

Applying the sequence of actions represented by the path of each leaf-node within S_Π will result in the satisfaction of the goal. Thus, S_Π may be considered a partial plan under delete-relaxation conditions. A high probability of being sampled by such an HRR indicates that a fact is highly relevant to achieving the goal. Facts that appear in all partial plans that could be sampled must be in all plans, and so are landmarks.

Algorithm 2 - An HRR sampling sub-tree S_Π from T_Π Lines 4 - 16 add required sub-goals and actions that require them until either a sub-goal is TRUE in the state, or there is no action available that could achieve it. Lines 6 - 9 choose one action with uniform probability from those that achieve the required fact. Lines 10 - 14 require all preconditions of an action.

```
1: Let S_Π ← {root}
2: Let frontier be a stack
3: frontier.push(root)
4: while frontier is not empty do
5:    n ← frontier.pop()
6:    if label(n) ∈ F then
7:       Let m ← choose(children(n))
8:       frontier.push(m)
9:       Add m to S_Π
10:   else
11:      for m ∈ children(n) do
12:         frontier.push(m)
13:         Add m to S_Π
14:      end for
15:   end if
16:   Remove n from frontier
17: end while
```

3.4 Choices counter

Each time the HRR reaches a fact node, it makes a choice as to which branch to explore. If multiple choices are available (ie more than one action could achieve the fact), then each branch represents a possible, but distinct partial plan. The probability of a node being sampled by the HRR can be found by tracking how many choices must be made in a certain way to reach it from the goal. This information will be used in section 3.7 to calculate a composite probability for how likely any node with a certain label will be sampled by the HRR.

Definition 24 (Choices counter, $\xi(\underline{n})$)**.** Let the choices counter $\xi(\underline{n})$ be the probability of a node $\underline{n}$ being sampled by the HRR:

$$\xi(\underline{n}) = P(\underline{n} \in S_{\Pi})$$

$\xi(\underline{n})$ depends on the number of alternative action choices that could have been made instead of those that reach that node. The tree's root will always be sampled:

$$\xi(\underline{root}) = 1$$

Between an action and its preconditions, the HRR has no choices to make, so ξ is passed down unchanged:

$$\xi(\underline{f}) = \xi(parent(\underline{f}))$$

The HRR chooses one action that could supply a fact from the set of actions that make up $children(f)$:

$$\xi(\underline{a}) = \frac{\xi(parent(\underline{a}))}{|children(parent(\underline{a}))|} \tag{3.1}$$

where $\underline{root}$, $\underline{f}$, $\underline{a}$ are root, fact, and action nodes respectively. The choices counter is shown on the faces of nodes in Figures in this chapter. Figure 3.1 is larger to make reading the labels and choices counters easier.

3.5 Exploration

For practical problems, T_Π is potentially very large, prohibiting its full representation. We aim to represent enough of it to calculate a lower bound on calculations performed on it. To this end, we establish a Monte-Carlo sampling procedure to explore part of T_Π.

Definition 25 (Tree: Partially explored tree $\overline{T_\Pi}$)**.** We use Algorithm 3 to sample $\overline{T_\Pi}$ from T_Π. This performs a series of depth first dives from nodes that are chosen with probability proportionate to the value of their choices counter(line 4). This choice of which node to dive from was designed to favour exploration of parts of the tree that are more likely to be visited by the HRR. The exploration threshold ρ (line 3) is an estimate of the ratio of how much information is present in the frontier compared to the explored tree.

We found that the performance of h_Ξ (see Section 4.1) as a heuristic is not sensitive to small changes in the value of ρ. Lower values of ρ cause the exploration phase (and to a lesser extent subsequent search phases) to take longer, at an increasing rate. Below about $\rho < 0.22$, we observed little to no improvement in performance on experiments of the type reported in Chapter 4. Unless otherwise stated, $\rho = 0.2$ is used, as further optimisation was deemed to be unnecessary. The condition on line 3 of Algorithm 3 ensures that the sampling does not terminate early due to the volatility of its terms when small. We used a value of $minexp = 100000$, as it was found to consistently prevent the problem occuring, without making a significant difference to minimum run time.

Note that this sampling procedure differs from the sampling performed by the HRR. Both sampling procedures select actions that satisfy a frontier fact at random, but the HRR explores all facts that are preconditions for these actions, whereas the sampling procedure in Algorithm 3 selects a single precondition. This ensures that the T_Π is sampled more diversely without too

much focus on a few samples close to the root that branch widely, leading to their siblings being ignored.

Algorithm 3 - Back-jumping depth-first tree search Lines 3 - 10 start by exploring a minimum of *minexp* (as discussed in Section 3.5) nodes, and continue until the ratio of the sum of $\xi(\underline{n})$ in the frontier to that in the explored tree is less than ρ. Line 4 selects a node $\underline{n}$ from which to perform the next depth-first dive with probability proportional to $\xi(\underline{n})$. Lines 5 - 9 perform a depth-first dive, adding the children of nodes explored to $\overline{T_\Pi}$, but choosing one to explore at each depth.

1: Let $frontier \leftarrow \{goal\}$
2: Let $\overline{T_\Pi} \leftarrow \{goal\}$
3: **while** $|\overline{T_\Pi}| < minexp$ OR $sumxi(frontier)/sumxi(\overline{T_\Pi}) > \rho$ **do**
4: Let $\underline{n} \leftarrow choose(frontier)$
5: **while** $\underline{n}$ has children **do**
6: Remove $\underline{n}$ from $frontier$
7: Add $children(\underline{n})$ to $frontier$
8: Add $children(\underline{n})$ to $\overline{T_\Pi}$
9: Let $\underline{n} \leftarrow choose(children(\underline{n}))$
10: **end while**
11: **end while**

function $sumxi(\mathtt{K})$:
return $\sum_{\forall \underline{n} \in \mathtt{K}} \xi(\underline{n})$

function $choose(options)$:
return $\underline{m_i} \sim P(\underline{m_i}) = \xi(\underline{m}) / \sum_{options} \xi(m_i)$

3.6 Relevance score

This section defines a score that describes how relevant a fact or action is to a goal. That fact or action is represented by the label l. The tree T_Π represents information about how labels are connected to that goal. This relevance score will be used in Chapters 4 and 5 to solve the problems of task planning and diagnosis.

Definition 26 (Relevance score, $\Xi(l)$)**.** Let the relevance score $\Xi(l)$ represent the probability that a node with label l would be sampled by the HRR:

$$\Xi(l) = P\left(\exists \underline{l} \in S_\Pi | label(\underline{l}) = l\right)$$

Definition 27 (Local relevance score, $\Xi(l, \underline{n})$)**.** Let the local relevance score $\Xi(l, \underline{n})$ represent the probability that a node with label l would be sampled by the HRR, given that node $\underline{n}$ (and thus $S_\Pi^{path}(\underline{n})$) has been sampled.

$$\Xi(l, \underline{n}) = P\left(\exists \underline{l} \in S_\Pi^{desc}(\underline{n}) | label(\underline{l}) = l\right) \tag{3.2}$$

Definition 28 (State aware relevance score, $\Xi_\sigma(l), \Xi_\sigma(l, \underline{n})$)**.** In any particular state σ, all facts in the state $f \in \sigma$ are true and do not need to be achieved by the planner. The relevance score applied to a state $\Xi_\sigma(l)$ represents the probability that a node with label l would be sampled by the HRR, if it stops at nodes that are true in state σ (because they do not need an action to supply them). Calculations that can be performed on $\overline{T_\Pi}$ can be made state aware by performing them on on $\overline{T_\Pi / \sigma}$ instead:

$$\overline{T_\Pi / \sigma} = \overline{T_\Pi} \Big/ \bigcup_{\substack{\forall \underline{f} \in L(f) \\ \forall f \in \sigma}} T_\Pi^{desc}(\underline{f})$$

The effect of this truncation is illustrated in Figure 3.1, where the green nodes are the descendents of the blue node, and so would be ignored in calculations performed on a state for which only the blue node had a label corresponding to a fact in the state. Note that the illustrated truncation is applied to the blue node, not the state $has(Doc)$, for which another node exists.

Values of $\Xi(l)$ calculated on a partially explored tree $\overline{T_\Pi}$, are a lower bound on those that would be calculated on the full tree T_Π. Nodes for which $\xi(\underline{n})$ is small contribute less to $\Xi(l)$,

and are found further from the root, causing this lower bound to converge quickly upwards as the region of the tree close to the root is explored.

3.7 Calculating $\Xi(l)$

This section uses the tools defined earlier in this chapter to derive the equations needed to calculate $\Xi(l)$. Algorithm 4 shows the application of these equations to this goal, and Section 3.7.2 illustrates the application of this to calculate the relevance score for a label on an example tree.

3.7.1 Derivation

Recalling Equation 3.2, consider that if $label(\underline{n})$ is a fact, any one of its children could be sampled by the HRR, and may have a node with label l among its descendants:

`if` $label(n) \in F$:

$$\Xi(l, \underline{n}) = \sum_{\underline{c} \in children(\underline{n})} P\left(\underline{c} \in S_{\Pi}^{desc}(\underline{n})\right) \times \Xi(l, \underline{c})$$

The HRR chooses one child of a fact node with uniform probability, which implies:

$$\Xi(l, \underline{n}) = \sum_{\underline{c} \in children(\underline{n})} \frac{\Xi(l, \underline{c})}{|children(\underline{n})|} \tag{3.3}$$

If $label(\underline{n})$ is an action, then all its children will be sampled. $\Xi(l, \underline{n})$ is thus 1 minus the probability that none of its' sampled descendants have label l:

`if` $label(n) \in A$:

$$\Xi(l, \underline{n}) = 1 - \prod_{\underline{c} \in children(\underline{n})} (1 - \Xi(l, \underline{c})) \tag{3.4}$$

Equation 3.2 allows us consider the descendants of a node independently of other branches arising from its ancestors. This can be applied recursively to calculate the local relevance score of the tree's root: $\Xi(l, \underline{root}) = \Xi(l)$.

We now consider how we can ignore the local relevance score $\Xi(l, \underline{n})$ for any nodes without descendants that have $label(\underline{n}) = l$. If a node has label l, then the HRR sampling that node has sampled a node with label l:

$$label(\underline{n}) = l \qquad \implies \Xi(l, \underline{n}) = 1 \tag{3.5}$$

If none of the descendents of a node have label l, then the HRR will not sample a node with $label = l$ in its descendents:

$$T_{\Pi}^{desc}(\underline{n}) \cap L(l) = \emptyset \qquad \implies \Xi(l, \underline{n}) = 0$$

Consider a node $\underline{n}$ and one of its descendants $\underline{d}$ such that all nodes with label l that are a descendant of one are also a descendant of the other:

$T_{\Pi}^{desc}(\underline{n}) \cap L(l) = T_{\Pi}^{desc}(\underline{d}) \cap L(l)$

$$\begin{aligned} \implies \qquad \Xi(l, \underline{n}) =& P\left(\underline{d} \in S_{\Pi}^{desc}(\underline{n})\right) && \times \Xi(l, \underline{d}) \\ =& P\left(\underline{d} \in S_{\Pi} | \underline{n} \in S_{\Pi}\right) && \times \Xi(l, \underline{d}) \\ =& \frac{\xi(\underline{d})}{\xi(\underline{n})} && \times \Xi(l, \underline{d}) \end{aligned} \tag{3.6}$$

Equation 3.6 describes the relationship between the local relevance scores of such a pair of nodes.

Now, consider Equation 3.3 when each child $\underline{c}$ of a fact node $\underline{f}$ has a single descendant $\underline{d}$ with $label(\underline{d}) = l$:

$$\Xi(l, \underline{f}) = \sum_{\underline{c} \in children(\underline{f})} \frac{\Xi(l, \underline{c})}{\left|children(\underline{f})\right|}$$

Using Equation 3.1, this can be written as:

$$= \sum_{\underline{c} \in children(\underline{f})} \frac{\Xi(l, \underline{c})}{\frac{\xi(\underline{f})}{\xi(\underline{c})}}$$
$$= \sum_{\underline{c} \in children(\underline{f})} \frac{\Xi(l, \underline{c}) \times \xi(\underline{c})}{\xi(\underline{f})}$$

Next, using Equation 3.6, this can be rewritten as:

$$= \sum_{\substack{\underline{c} \in children(\underline{f}) \\ \underline{l} \in T_{\Pi}^{desc}(\underline{c}) \cap L(l)}} \frac{\frac{\xi(\underline{l})}{\xi(\underline{c})} \times \Xi(l, \underline{l}) \times \xi(\underline{c})}{\xi(\underline{f})}$$
$$= \sum_{\underline{l} \in T_{\Pi}^{desc}(\underline{f}) \cap L(l)} \frac{\xi(\underline{l})}{\xi(\underline{f})}$$
$$= \frac{1}{\xi(\underline{f})} \sum_{\underline{l} \in T_{\Pi}^{desc}(\underline{f}) \cap L(l)} \xi(\underline{l}) \tag{3.7}$$

If all descendants of a node $\underline{d} \in T_{\Pi}^{desc}(\underline{n})$ are such that either $label(\underline{d}) = l$, or fact nodes for which Equation 3.7 applies, this process can be repeated, causing further $\frac{1}{\xi(\underline{c})}$ terms to cancel. If some nodes with $label(\underline{d_i}) = l$ have an $LCA(\underline{d_1}, \underline{d_2}) = \underline{a}$ that is an action (i.e., an $aLCA$), then Equation 3.6 does not apply between between $\underline{c_i}$ and $\underline{d_i}$. It will apply

between $\underline{a_i}$ and $\underline{c_i}$, but $\Xi(l,\ \underline{a})$ must be computed according to Equation 3.4. These can be combined to give:

$$\begin{aligned}\Xi(l,\underline{f}) =& \frac{1}{\xi(\underline{f})} \sum_{\underline{l} \in fLCAs(T_{\Pi}^{desc}(\underline{f}))} \xi(\underline{l}) + \sum_{\underline{a} \in aLCAs(T_{\Pi}^{desc}(\underline{f}))} \frac{\xi(\underline{a})}{\xi(\underline{f})} \times \Xi(l,\underline{a}) \\ =& \frac{1}{\xi(\underline{f})} \left(\sum_{\underline{l} \in fLCAs(T_{\Pi}^{desc}(\underline{f}))} \xi(\underline{l}) + \sum_{\underline{a} \in aLCAs(T_{\Pi}^{desc}(\underline{f}))} \xi(\underline{a}) \times \Xi(l,\underline{a}) \right) \end{aligned} \tag{3.8}$$

Equation 3.8 allows the local relevance score $\Xi(l,\underline{f})$, for any fact node to be found from its $fLCAs(T_{\Pi}^{desc}(\underline{f}))$ and $aLCAs(T_{\Pi}^{desc}(\underline{f}))$. As the root of the tree is a fact node, $\Xi(l)$ can be found by recursively applying equation 3.8 to fact nodes (starting at the root), and resolving the $aLCAs$ with equation 3.4 applied to their children (which are fact nodes resolved by equation 3.8 etc). These calculations may be simplified by sorting the list of $aLCAs$ by their depth in the tree, and resolving them from the bottom up.

The complete procedure for calculating $\Xi(l)$ is outlined in Algorithm 4. This sorts the list of $aLCAs$ (line 1), before resolving them deepest first. Doing so in this order ensures that any $aLCAs$ will have already been calculated by the time they are used themselves for a calculation. $aLCAs$ are resolved by first calculating their children (line 4) according to Equation 3.8, and then combining these (line 6) according to Equation 3.4. Once all $aLCAs(L(l))$ have been calculated, $\Xi(l)$ can be found (line 8) by applying Equation 3.8 to the root.

3.7.2 Example calculation

We will now walk through an example of this calculation on the example problem introduced in Figure 1.1 Figure 3.3 highlights the information needed to calculate $\Xi(at(Library))$, if the tree is explored to the degree shown. We refer to nodes in the colour they are assigned in Figure 3.3. 3 nodes have $label(\mathbf{n}) = at(Library)$, and Algorithm 1 registers that the root node **has(Hardcopy)**, has one **aLCA** with two children, [**leftchild**, **rightchild**], and

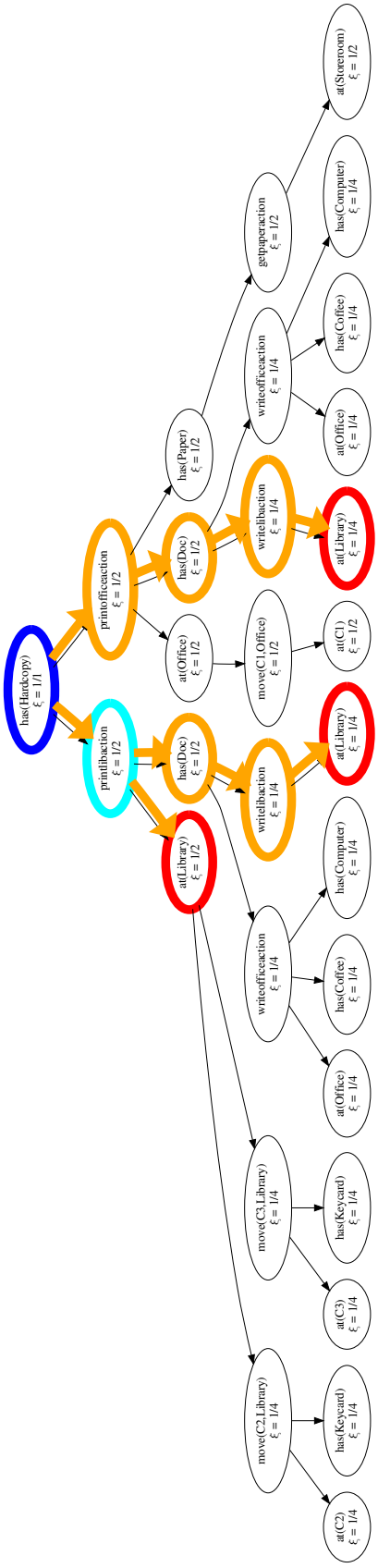


Figure 3.3: **Example tree for calculating** $\Xi(at(Library))$ The top 5 layers (3 fact, 2 action) of $\overline{T_\Pi}$ are shown. Nodes with the label $at(Library)$ are highlighted **red**. Their **paths** merge at **fact LCAs** and **action LCAs**. The values of $\xi(l)$ are shown for each node.

Algorithm 4 - Calculating the relevance score To calculate $\Xi(l)$ for label l, this algorithm makes use of the list of $aLCAs$ stored for l, and the $fLCAs$ and $aLCAs$ registered as direct descendents of $root$ - all found by 1. Line 1 sorts $aLCAs$ according to depth, with deepest first, which ensures that $aLCAs$ are only used after they have been resolved.

1: Let $ListOfaLCAs \leftarrow sorted(aLCAs(l))$
2: **for all** $aLCA \in ListOfaLCAs$ **do**
3: **for all** $child \in aLCA.children$ **do**
4: Resolve Equation 3.8

$$\Xi(l, \underline{child}) = \frac{1}{\xi(\underline{child})} \left(\sum_{\underline{l} \in fLCAs(T_{\Pi}^{desc}(\underline{child}))} \xi(\underline{l}) + \sum_{\underline{a} \in aLCAs(T_{\Pi}^{desc}(\underline{child}))} \xi(\underline{a}) \times \Xi(l, \underline{a}) \right)$$

5: **end for**
6: Resolve Equation 3.4
$\Xi(l, \underline{aLCA}) = 1 - \prod_{\underline{c} \in children(\underline{aLCA})} (1 - \Xi(l, \underline{c}))$
7: **end for**
8: Resolve Equation 3.8

$$\Xi(l) = \Xi(l, \underline{root}) = \frac{1}{\xi(\underline{root})} \left(\sum_{\underline{l} \in fLCAs(T_{\Pi}^{desc}(\underline{root}))} \xi(\underline{l}) + \sum_{\underline{a} \in aLCAs(T_{\Pi}^{desc}(\underline{root}))} \xi(\underline{a}) \times \Xi(l, \underline{a}) \right)$$

one $fLCA = [\underline{\textbf{rightmost}}]$ (Algorithm 4 line 1). Next we resolve the children of the **aLCA** (Algorithm 4 line 4):

$$\Xi(at(library), \underline{\textbf{leftchild}}) = 1 \qquad \texttt{Equation 3.5}$$

$$\begin{aligned} \Xi(at(library), \underline{\textbf{rightchild}}) &= \frac{1}{\xi(\underline{\textbf{rightchild}})} \left(\sum_{\underline{l} \in [\underline{\textbf{rightchild.descendent}}]} \xi(\underline{l}) + 0 \right) \\ &= \frac{\xi(\underline{\textbf{rightchild.descendent}})}{\xi(\underline{\textbf{rightchild}})} \qquad \texttt{Equation 3.8} \\ &= \frac{\frac{1}{4}}{\frac{1}{2}} = \frac{1}{2} \end{aligned}$$

Here we used Equation 3.5 as a simpler special case of Equation 3.8, which could have been used to reach the same result. The **aLCA** can then be resolved (Algorithm 4 line 6):

$$\Xi(at(library), \underline{\textbf{aLCA}}) = 1 - \prod_{\underline{c} \in children(\underline{\textbf{aLCA}})} (1 - \Xi(at(library), \underline{c})) \quad \texttt{Equation 3.4}$$

$$= 1 - \begin{pmatrix} (1-\Xi(at(library), \underline{\textbf{leftchild}})) \\ \times (1-\Xi(at(library), \underline{\textbf{rightchild}})) \end{pmatrix}$$

$$= 1 - (1-1)(1-\frac{1}{2}) = 1$$

Finally (Algorithm 4 line 8) the the $root$, **has(Hardcopy)** is resolved:

$$\Xi(at(library), \underline{\textbf{has(Hardcopy)}})) = \frac{1}{\xi(\underline{\textbf{has(Hardcopy)}})} \left(\sum_{\underline{l} \in [\underline{\textbf{rightmost}}]} \xi(\underline{l}) + \sum_{\underline{a} \in \underline{\textbf{aLCA}}} \xi(\underline{a}) \times \Xi(l, \underline{a}) \right)$$

$$= \frac{1}{1} \left(\frac{1}{4} + \frac{1}{2} \times 1 \right) = \frac{3}{4}$$

Thus the fact $\Xi(at(library))$ has a relevance score of $\frac{3}{4}$. The reason for this can be seen by considering Figure 3.3. In order to achieve $has(Hardcopy)$, the student must either $printlibaction$ or $printofficeaction$. If $printlibaction$ is chosen, then $at(library)$ is a pre-condition, so this branch necessarily visits a node with that label. If $printofficeaction$ is chosen, then there are 2 ways its precondition of $has(Doc)$ could be met: $writelibaction$ or $writeofficeaction$. As with printing, one of these branches requires a visit to the library, the other does not.

If $\overline{T_\Pi}$ had been explored further, more nodes with $label(\underline{\textbf{n}}) = at(Library)$ would be found. These would represent partial plans where the student goes to the library by a different route before either writing or printing there. Inclusion of these nodes would slightly increase $\Xi(at(Library), \underline{\textbf{printofficeaction}})$ by adding descendents to it, making it an $aLCA(L(at(library)))$.

3.8 Discussion

We have defined the behaviour of an HRR that samples partial plans by randomly selecting actions that support the goal, and the preconditions of other actions it has taken. This behaviour was analysed by considering the probability that facts or actions are a part of these partial plans, which we define as the relevance score $\Xi(l)$. We found that representing the space explored by the HRR as a tree, rooted at a node representing the goal, allowed us to evaluate parts of that space independently before combining that information to find $\Xi(l)$.

This representation, and the analysis we perform on it makes no reference to the initial state. This is an important property, as it allows the analysis of problems where the initial state is either unknown, or known to be incomplete. A comparison can be made between the relevance score, and landmarks. The relevance score quantifies how likely a fact (or action) is to be included in a partial plan, whereas landmarks query whether a fact (or set of facts) is present in all plans. The existence of landmarks relies on an initial state that permits a complete plan, whereas the relevance score does not.

3.9 Future work

We presented methods for calculating the relevance score efficently, in a way that allows the reuse of many calculation results for answering similar queries. However, this representation of the space as a tree can be very large, even for small planning problems. Because of this, the exploration stage (which can be tuned to suit the resource constraints by adjusting ρ) consumes a significant amount of both computation time and space. The calculation of relevance scores is also computationally expensive, considering that it must be performed many times in some of the applications in which we use it. As shown in Section 4.4, particularly Table 4.4, using a heuristic based on the relevance score (h_{Ξ}, see Definition 29) takes about 100 times longer to evaluate less than half as many states as using the landmark counting heuristic (h_{LC}, see Definition 11).

The computational cost of exploring the tree could be reduced by observing that many parts of the tree will be exact, or close replicas of other parts. For example, when a fact or action appears in multiple places in the tree, the sub-trees beneath them will only differ where they involve facts or actions in their path. We anticipate that the (near) repetition of subtrees like this could be leveraged to find a more compact representation using symbolic computation to explore identical sections of the tree simultaneously.

The representation suggested above would also allow us to perform calculations once for each set of repeated subtrees, reducing the number of calculations that must be performed. Further improvement could be achieved by seeking additional ways to reuse intermediate steps in the calculation. For example by caching local relevance scores and adjusting them according to the state being evaluated rather than recalculating them each time. Consider the situation that Algorithm 4 has already been used to calculate the state aware relevance score $\Xi_{\sigma_0}(l)$ and is then used to calculate $\Xi_{\sigma_1}(l)$ (same label, different state). The calculations on lines 4, 6, and 8 (Equations 3.4 and 3.8) only need to be performed again for nodes whose same-labelled descendents are in one of $\overline{T_\Pi/\sigma_0}$ and $\overline{T_\Pi/\sigma_1}$ but not the other. For states that are similar (as would be expected for states that are evaluated consecutively by a forward chaining planner, like that used in Chapter 4), this could allow the reuse of most calculation results.

Chapter 4

The Relevance Score as a Heuristic

There is a close relationship between the relevance score and landmarks. Equations 3.3 and 3.4 from the previous chapter, were inspired by [Keyder et al., 2010]'s treatment of actions as AND nodes, and facts as OR nodes for the identification of landmarks. When a tree is fully explored (i.e. $\overline{T_\Pi} = T_\Pi$), landmarks will be present in all partial plans that could be sampled by the HRR[1], and therefore have a relevance score $\Xi(l) = 1$. Facts that do not appear in any valid plans will have a relevance score $\Xi(l) = 0$, and facts that appear in some, but not all plans will have a value between 0 and 1, with those most relevant to the goal having higher scores than those that appear in only a few partial plans. The relevance score may be viewed as a continuous measure of the relevance of a fact, that expands upon the binary classification of a fact as either a landmark (ie absolutely relevant) or not a landmark (ie not necessarily relevant). Because of this conceptual similarity, and the success of landmarks as a planning heuristic, it was hypothesised that the relevance score could be successfully employed as a heuristic to guide a classical planner.

Real world problems can often be solved in many different ways. While one might expect this to make finding one of those solutions easier than if there were fewer available, it can cause problems for landmark counting heuristics. As shown in the illustrative example in Section 2.2.2, the existence of alternative paths to the goal can mean that non-trivial landmarks

[1] The exception to this is that facts present in the initial state are landmarks because they are already true at the start of all plans, but may not contribute the plan at all.

are few or non-existent, preventing the information captured by landmarks being used to guide the search. The relevance score captures similar information to landmarks, but as a more general, continuous measure, is more robust to this problem. The presence of alternative paths to the goal will reduce the relevance score of facts that would otherwise have been landmarks, but not to the point of being irrelevant. There will still be an informative heuristic gradient to follow, that captures the same kind of information as landmarks.

4.1 Heuristic h_Ξ

As discussed further in Section 2.2.2, landmarks are used as a heuristic by counting how many need to be achieved to reach the goal from the state being evaluated. The relevance score heuristic attempts to do something similar to this, by evaluating the sum of the relevance scores for all facts in the state being evaluated.

Definition 29 (The relevance score heuristic $h_\Xi(\sigma)$)**.** The relevance score heuristic for a state $h_\Xi(\sigma)$, is defined as the sum over the state aware relevance scores of all facts:

$$h_\Xi(\sigma) = \sum_{l \in F} \Xi_\sigma(l) \tag{4.1}$$

Facts in a state $f_\sigma \in \sigma$, are always excluded from $\overline{T_\Pi/\sigma}$, and so $\Xi(f_\sigma) = 0$, which may be interpreted as them being no longer relevant as they have already been achieved. Facts that were only needed to achieve facts in the current state will be represented by nodes that are truncated, and so be considered irrelevant (ie $\Xi_\sigma(l) = 0$). States that are closer to the goal will have the relevance scores for all facts calculated on a tree that is truncated at facts that are closer to the root, leading to higher values.

Recall that the relevance score is calculated under the delete relaxation, so does not consider that facts may become untrue, and then need to be achieved again, potentially multiple times in the way modern landmark counting methods do. However, if a fact that was relevant before it was achieved becomes untrue again, the planner may still regain the reduction in

h_Ξ in future. This apparent loss of progress when facts become untrue will be offset by the attainment of other facts if doing so moves the planner closer to the goal.

4.2 Evaluation

In order to evaluate the performance of h_Ξ as a heuristic to guide a classical planner towards a plan, a program to calculate Equation 4.1 (using Algorithms 1, 3, and 4) was implemented in C++, using elements of the LAMA architecture [Richter and Westphal, 2010, github: LAMA,] to read and access planning problems/domains specified in PDDL [McDermott et al., 1998]. All problems were parsed using LAMA's *translate* and *preprocess* scripts, the outputs of which were then read by subclasses of LAMA's *search* module. All code used to implement and test h_Ξ has been made available[2]. In order to be used by LAMA, this code must receive states σ in the state variable/value format used by LAMA, and so is only directly compatible with other code that is derived from LAMA. See Section 5.3.1 for a discussion of related code that is independent of LAMA. The currently available version of LAMA appears to compile and run on Ubuntu≤ 18 but no higher. Further details of this code can be found in Section 6.1.

4.2.1 Implementation details

The LAMA planner can perform one of two search strategies: Best First Search (**BFS**) or weighted A* (**wA***) [Pohl, 1970]. Multiple heuristics can be used at the same time, by alternating between queues kept according to each, updating both each time a state is evaluated. Normally, these would be the Fast Forward (h_{FF} , see Definition 6) [Hoffmann and Nebel, 2001] and landmark counting (h_{LC}, see Definition 11) heuristics.

LAMA can make use of preferred operators [Helmert, 2006] that encode information about the order in which landmarks need to be achieved to further guide the search for a plan. In order to isolate the effect of using different heuristics, preferred operators were not used.

[2] `https://bitbucket.org/Oliver_Kim/relevanceheuristic/`

The relevance score heuristic is a sum of probabilities, $h_\Xi(\sigma) \to \mathbb{R}$. The LAMA planning system, on the other hand, requires heuristics to return an integer value $h_{LC}(\sigma), h_{FF}(\sigma) \to \mathbb{Z}$. To allow for use and comparison with LAMA, $h_\Xi(\sigma)$ is multiplied by 10000 to prevent loss of precision before being rounded to the nearest integer. When weighted A* search is used, the cost to reach the state is also multiplied by 10000 to prevent it being ignored.

All experiments were performed on a computer running Ubuntu 18.04.6, with 65 831 912B RAM, and 24 AMD Ryzen 9 3900XT (3.8GHz) processors. Up to 6 experiments were run concurrently, leaving a minimum of approximately 18GB RAM and 18 processors for system and control processes.

4.2.2 Hypotheses

We experimentally evaluated the following hypotheses:

H1 The relevance score heuristic h_Ξ is slower than the landmark counting heuristic h_{LC} at solving standard planning problems, but is able to find a plan most of the time.

H2 The relevance score heuristic h_Ξ substantially improves the ability to find plans compared to the landmark counting heuristic h_{LC} in domains without non-trivial landmarks.

4.2.3 Measures of success

Each search attempt is evaluated by 3 measures. Measure **M1** is whether or not a plan is found. Heuristics are used because exhaustive search of the entire planning space is prohibitively expensive, in terms of both memory and time. Because of this, a practical meaning of failure to find a plan is that it exceeds the computational resources available to it. All experiments were performed with an 8GB RAM limit, and 2h time limit, with failure reported if either of these is reached. When one of these was reached, it was typically the RAM limit in under 1h. The main program components that use a significant amount of RAM are: the tree explored to calculate the relevance score (only for h_Ξ; typically under 500MB, even for

large problems); and LAMA's representation of its search history (which grows linearly with time once the search begins; affects all heuristics).

Two properties of a successful search attempt were measured and compared:

M2 How expensive is finding a plan

M3 How good is the solution that was found

Measure **M2** was evaluated by recording how many times the heuristic was calculated for a state before a plan was found. LAMA makes use of deferred heuristic evaluation, only calculating a heuristic when a state is expanded, not when it is generated by applying an action. Because of this, the number of states expanded is used as the representative measure of search cost, rather than the number of states generated. The BFS search strategy prioritises finding a plan quickly without concern for plan quality, so was used to evaluate this measure. Evaluating fewer states before finding a plan is considered preferable.

Measure **M3** was evaluated by the length of the plan found. The wA* search strategy balances the competing priorities of finding a solution quickly, and solution quality. It does so according to the weight w, in the equation it uses to assign a cost c, to a state, $c = w \times h + g$ (h is the heuristic, g is the length of the shortest path to that state). LAMA's default starting value of $w = 10$ was used. Shorter plans are considered preferable.

Failure to find a plan makes other measures meaningless, so we gave **M2** and **M3** an infinitely high value (ie the worst possible value).

4.3 Experimental design

4.3.1 Standard problems

Hypothesis **H1** was tested on the 675 problems defined in the examples folder of the HSP2 repository [Bonet and Geffner, 2001a,github: hsp2,]. This is a collection of standard problems defined in PDDL, that have been used as benchmarks for IPC competitions, consisting of:

blocks	35	driverlog	20	elevators	30
freecell	80	grid	5	logistics00	28
logistics98	35	mprime	35	openstacks	1
parcprinter-strips	30	pegsolitaire	30	pipesworld-notankage	50
pipesworld-tankage	50	rovers	40	satellite	36
scananalyzer	30	sokoban	30	tpp	30
transport	30	woodworking-strips	30	zenotravel	20

Table 4.1: **Standard problems counts.**

Measures **M1** and **M2** were evaluated by running LAMA employing a BFS strategy and either h_{Ξ} or h_{LC}. Measure **M3** was evaluated by running LAMA employing a wA* search strategy and either h_{Ξ} or h_{LC}. Each of these configurations was also tested with h_{FF} as a second heuristic, using LAMA's multi-queue capability.

4.3.2 Problems without non-trivial landmarks

In order to evaluate **H2**, we generated new PDDL specifications for problems that contain no landmarks other than facts in the goal and initial state. This was achieved by merging a pair of problems, Π_1, Π_2 that were solved on all 3 attempts by both heuristics (h_{LC}, h_{Ξ}), in such a way as to prevent them from interacting. This ensures that each new problem can be solved by at least 2 plans that have no overlap between the facts or actions involved in them.

To generate $merged(\Pi_1, \Pi_2)$, all domain or problem specific elements (types, constants, predicates, actions, objects, see Section 2.1.1) in each problem were prepended with a label unique to that problem. This ensured that no element shared a name with elements in the

problem they are being merged with. If a problem did not use types, a new type consisting of just its unique label was applied to all constants and objects within it. The new *merged* problem is then comprised of all labeled elements from the two source problemss, with the following modifications:

- Two new actions are defined:

  $a_1 : pre(a_1) = G_1; eff(a_1) = \{winning\}$

  $a_2 : pre(a_2) = G_2; eff(a_2) = \{winning\}.$

- The merged problem has a single goal:

  $G_{merged} = \{winning\}$

A total of 500 problems were generated with this procedure by randomly selecting a pair of problems from the pool of those solved individually by both h_Ξ and h_{LC} with a BFS strategy. The script used for this is unable to handle PDDL that, while valid, does not follow certain conventions. Whenever this occurred, a new pair was randomly selected for experimental evaluation. The problems generated in this way were also attempted by the same set of planner configurations as the standard problems, and measured according to the same criteria.

4.4 Results

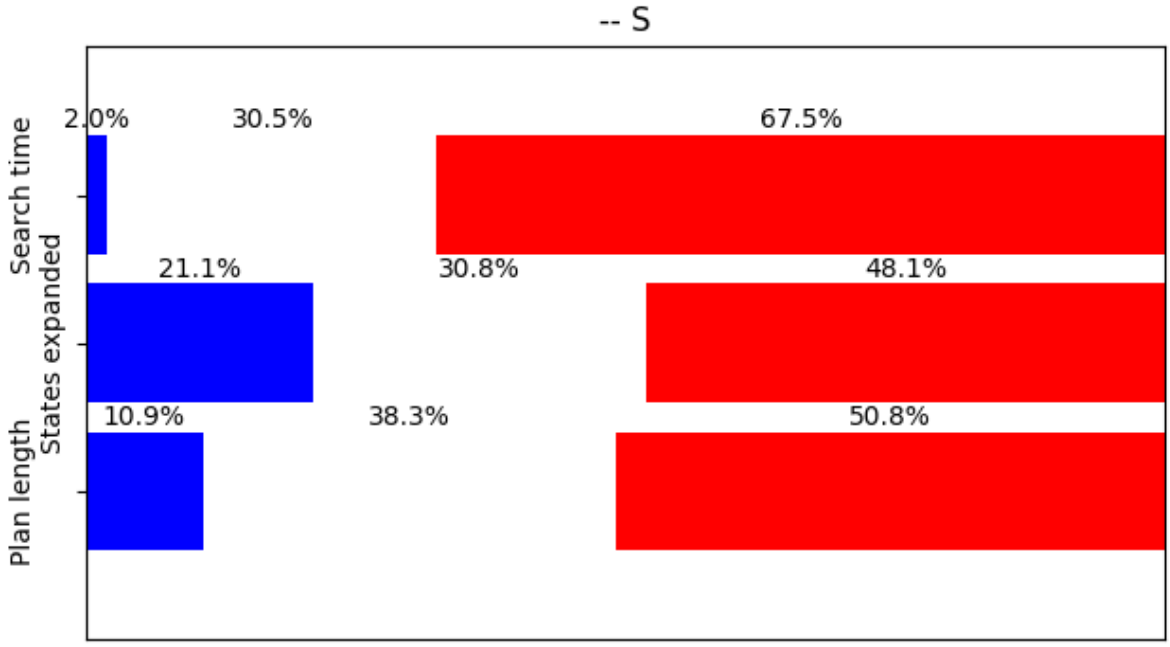
We used each planner configuration ($[(h_\Xi), (h_{LC}), (h_\Xi, h_{FF}), (h_{LC}, h_{FF})] \times [BFS, wA^*]$) described in Section 4.2.3 to attempt to solve all problems described in Section 4.3 (standard IPC problems or landmark-free merged problems) 3 times. An additional configuration $(h_\Xi, h_{LC}, h_{FF})BFS$ was also tested once, and only its success rate for finding a plan is reported. The results are presented here.

4.4.1 Standard Problems

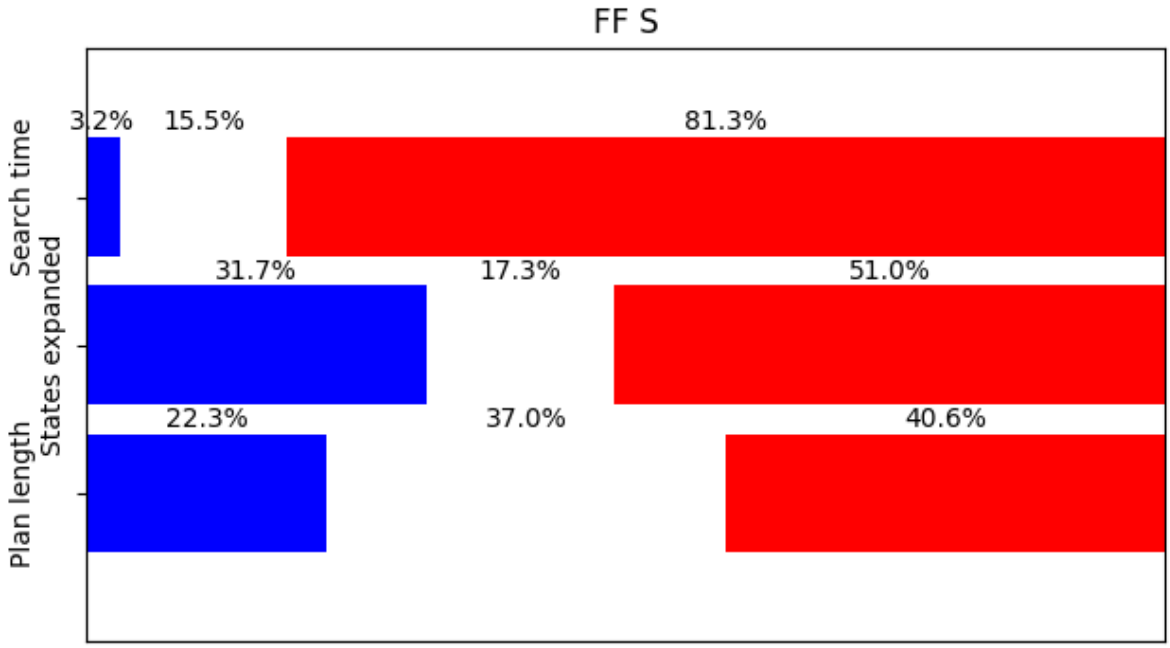
	Problems solved by h_{LC}	Problems solved by h_Ξ	Problems solved by both	Problems solved by neither
As the only heuristic	70.22%	53.04%	51.41%	28.15%
Alongside h_{FF}	*84.94%	73.93%	71.75%	12.89%
All 3 combined	74.81%			

Table 4.2: **Success rates of heuristics on standard problems** Percentages of standard problems solved by each heuristic either alone or with h_{FF}, using BFS. Each of the 675 problems defined in HSP2-examples was tested 3 times by each planner configuration, apart from all 3 combined (h_{FF}, h_{LC}, h_Ξ), which was tested 1 time. The best heuristic for this type of problem is marked with a *.

Table 4.2 shows the success rates of each heuristic on standard problems. h_{LC} was able to solve more problems than h_Ξ. Both heuristics solved more problems when paired with h_{FF} than either did alone. Tables 4.3 and 4.6 report this data in more detail, breaking it down by domain. Figure 4.1 shows how often each heuristic did better or worse on paired standard problems. Tables 4.4 and 4.7 show the mean costs (number of states expanded and time taken) of finding a plan, broken down by domain. Tables 4.5 and 4.8 show the mean plan length found using each heuristic, broken down by domain. As the only heuristic, h_{LC} finds plans faster than h_Ξ using BFS, and shorter plans using wA* in the majority of trials. When paired with h_{FF}, this difference is smaller, but still significant.



(a) Standard problems, heuristic tested alone



(b) Standard problems, heuristic tested in conjunction with h_{FF}

Figure 4.1: **Heuristic results on standard problems** These charts show the fraction of trials where the performance of h_Ξ was better, equal to, or worse than h_{LC}, according to the metrics shown on the x-axis. Each of the 675 problems was attempted 3 times per planner configuration, in paired trials.

Domain (# problems)	Problems solved by h_{LC}	Problems solved by h_{Ξ}	Problems solved by both	Problems solved by neither
All problems (675)	**70.22%**	**53.04%**	**51.41%**	**28.15%**
blocks (35)	100.00%	100.00%	100.00%	0.00%
driverlog (20)	81.67%	65.00%	65.00%	18.33%
elevators (30)	100.00%	95.56%	95.56%	0.00%
freecell (80)	98.75%	71.25%	71.25%	1.25%
grid (5)	100.00%	46.67%	46.67%	0.00%
logistics00 (28)	100.00%	100.00%	100.00%	0.00%
logistics98 (35)	36.19%	19.05%	19.05%	63.81%
mprime (35)	48.57%	30.48%	26.67%	47.62%
openstacks (1)	100.00%	100.00%	100.00%	0.00%
parcprinter-strips (30)	62.22%	50.00%	37.78%	25.56%
pegsolitaire (30)	96.67%	87.78%	87.78%	3.33%
pipesworld-notankage (50)	56.00%	32.67%	30.67%	42.00%
pipesworld-tankage (50)	46.00%	20.67%	20.67%	54.00%
rovers (40)	50.83%	25.00%	25.00%	49.17%
satellite (36)	24.07%	30.56%	24.07%	69.44%
scananalyzer (30)	100.00%	70.00%	70.00%	0.00%
sokoban (30)	60.00%	18.89%	13.33%	34.44%
tpp (30)	60.00%	34.44%	34.44%	40.00%
transport (30)	100.00%	100.00%	100.00%	0.00%
woodworking-strips (30)	27.78%	16.67%	16.67%	72.22%
zenotravel (20)	90.00%	73.33%	68.33%	5.00%

Table 4.3: **Success rates of heuristics (alone) on standard problems - by domain** Percentages shown are over 3 repeats for all problems within domain.

Domain	Mean states expanded by h_{LC}	Mean states expanded by h_{Ξ}	Mean search time (s) by h_{LC}	Mean search time (s) by h_{Ξ}
All problems	**16853.88 ±106932.78**	**6361.38 ±20181.40**	**0.45 ±3.71**	**59.13 ±198.48**
blocks	12232.51 ±32755.58	1670.37 ±2683.56	0.42 ±1.40	7.71 ±16.71
driverlog	2783.41 ±4654.41	1870.90 ±2751.34	0.04 ±0.09	46.42 ±97.40
elevators	52052.48 ±257212.79	5929.78 ±8521.32	1.50 ±8.81	37.87 ±65.81
freecell	2722.71 ±12532.37	7277.24 ±12320.35	0.22 ±1.50	107.42 ±238.19
grid	759.29 ±1738.24	26165.29 ±30365.73	0.02 ±0.05	338.71 ±397.85
logistics00	1119.39 ±1978.95	2560.67 ±3258.52	0.03 ±0.07	10.04 ±22.65
logistics98	20512.25 ±59423.41	62783.70 ±69418.13	0.42 ±1.27	193.01 ±328.75
mprime	17982.04 ±23556.39	2436.18 ±3285.24	0.22 ±0.30	46.67 ±72.53
openstacks	33.67 ±0.58	54.00 ±1.00	0.00 ±0.00	0.50 ±0.22
parcprinter-strips	2330.47 ±6943.08	507.09 ±2516.17	0.07 ±0.20	6.65 ±30.46
pegsolitaire	40845.47 ±224398.62	918.34 ±1304.55	1.24 ±8.48	42.02 ±68.19
pipesworld-notankage	36199.07 ±115956.01	5446.54 ±12238.85	1.13 ±4.23	152.21 ±368.25
pipesworld-tankage	30191.19 ±81909.68	3203.84 ±9757.39	0.44 ±1.16	38.21 ±100.08
rovers	414.97 ±526.14	33711.13 ±70287.51	0.01 ±0.01	14.15 ±32.42
satellite	41248.62 ±125152.88	6576.50 ±14030.79	0.97 ±3.34	41.54 ±115.27
scananalyzer	2746.02 ±9337.37	8009.54 ±19980.93	0.05 ±0.20	118.05 ±502.09
sokoban	1559.92 ±943.24	277.75 ±127.70	0.02 ±0.01	14.60 ±7.98
tpp	8341.52 ±17331.77	3616.16 ±5465.57	0.13 ±0.28	45.10 ±83.92
transport	3281.04 ±6786.50	4830.73 ±7784.60	0.06 ±0.14	60.57 ±127.69
woodworking-strips	2239.27 ±4805.21	2852.67 ±6313.50	0.08 ±0.16	9.33 ±18.98
zenotravel	59156.83 ±113672.36	4331.78 ±9259.58	0.92 ±1.83	26.45 ±61.96

Table 4.4: **Search costs (alone) for standard problems - by domain** Values are only calculated on trials when both h_{LC} and h_{Ξ} found a plan.

Domain (# problems)	Plan length for h_{LC}	Plan length for h_{Ξ}
All problems	**36.01 ±35.11**	**40.78 ±38.07**
blocks	59.68 ±45.33	67.96 ±50.75
driverlog	26.87 ±16.04	29.62 ±13.61
elevators	28.20 ±8.66	30.86 ±10.22
freecell	43.36 ±25.25	51.20 ±31.91
grid	27.29 ±20.19	47.14 ±14.19
logistics00	42.63 ±23.13	50.39 ±27.98
logistics98	26.50 ±7.42	30.55 ±8.46
mprime	5.82 ±2.51	8.61 ±4.29
openstacks	17.67 ±0.58	18.33 ±0.58
parcprinter-strips	24.12 ±16.13	24.56 ±16.07
pegsolitaire	23.47 ±9.43	25.57 ±10.29
pipesworld-notankage	28.04 ±22.88	26.26 ±17.68
pipesworld-tankage	15.58 ±7.61	16.19 ±7.70
rovers	21.10 ±11.25	22.93 ±12.26
satellite	23.27 ±9.56	26.08 ±12.26
scananalyzer	28.52 ±25.11	31.13 ±24.57
sokoban	145.42 ±171.23	144.00 ±171.91
tpp	39.58 ±37.25	48.55 ±36.89
transport	48.51 ±33.96	56.26 ±37.19
woodworking-strips	10.60 ±4.75	9.53 ±3.93
zenotravel	20.37 ±14.24	23.29 ±15.72

Table 4.5: **Plan length (alone) for standard problems - by domain** Values are only calculated on trials when both h_{LC} and h_{Ξ} found a plan.

Domain (# problems)	Problems solved by h_{LC}	Problems solved by h_{Ξ}	Problems solved by both	Problems solved by neither
All problems (675)	**84.94%**	**73.93%**	**71.75%**	**12.89%**
blocks (35)	100.00%	99.05%	99.05%	0.00%
driverlog (20)	98.33%	88.33%	88.33%	1.67%
elevators (30)	100.00%	100.00%	100.00%	0.00%
freecell (80)	97.92%	95.42%	93.33%	0.00%
grid (5)	100.00%	100.00%	100.00%	0.00%
logistics00 (28)	100.00%	100.00%	100.00%	0.00%
logistics98 (35)	72.38%	54.29%	53.33%	26.67%
mprime (35)	84.76%	87.62%	83.81%	11.43%
openstacks (1)	100.00%	100.00%	100.00%	0.00%
parcprinter-strips (30)	86.67%	82.22%	76.67%	7.78%
pegsolitaire (30)	98.89%	90.00%	88.89%	0.00%
pipesworld-notankage (50)	84.00%	60.00%	60.00%	16.00%
pipesworld-tankage (50)	50.00%	33.33%	32.00%	48.67%
rovers (40)	72.50%	46.67%	46.67%	27.50%
satellite (36)	61.11%	68.52%	53.70%	24.07%
scananalyzer (30)	100.00%	73.33%	73.33%	0.00%
sokoban (30)	93.33%	37.78%	37.78%	6.67%
tpp (30)	68.89%	42.22%	42.22%	31.11%
transport (30)	100.00%	100.00%	100.00%	0.00%
woodworking-strips (30)	63.33%	63.33%	52.22%	25.56%
zenotravel (20)	100.00%	100.00%	100.00%	0.00%

Table 4.6: **Success rates of heuristics (with h_{FF}) on standard problems - by domain** Percentages shown are over 3 repeats for all problems within domain.

Domain	Mean states expanded by h_{LC}	Mean states expanded by h_{Ξ}	Mean search time (s) by h_{LC}	Mean search time (s) by h_{Ξ}
All problems	**11970.17 ±71256.51**	**4142.27 ±13714.79**	**7.12 ±73.23**	**65.81 ±240.12**
blocks	6381.98 ±16632.40	1853.38 ±3578.45	0.39 ±1.16	11.37 ±41.57
driverlog	16829.40 ±61086.95	3572.85 ±8221.71	3.64 ±15.79	265.99 ±873.72
elevators	422.68 ±454.68	503.51 ±671.82	0.02 ±0.02	4.36 ±6.20
freecell	8633.88 ±53236.68	3239.13 ±7794.11	20.17 ±164.49	63.55 ±196.85
grid	1107.73 ±1204.56	10164.40 ±18932.36	0.44 ±0.62	121.85 ±268.81
logistics00	581.88 ±799.48	548.82 ±550.49	0.02 ±0.04	2.28 ±5.24
logistics98	9353.80 ±13410.30	14824.52 ±18458.33	1.01 ±1.86	134.80 ±195.37
mprime	7910.73 ±41515.68	7892.95 ±39894.30	5.07 ±24.91	62.38 ±247.28
openstacks	41.33 ±2.31	56.67 ±3.21	0.00 ±0.00	0.61 ±0.32
parcprinter-strips	169.14 ±275.12	270.29 ±747.23	0.01 ±0.02	5.05 ±9.49
pegsolitaire	3150.40 ±12424.79	928.79 ±1576.80	0.14 ±0.74	50.09 ±103.11
pipesworld-notankage	44035.59 ±169173.10	5480.12 ±11533.35	13.77 ±50.46	123.52 ±295.12
pipesworld-tankage	92163.96 ±254299.77	4044.56 ±7021.61	24.81 ±82.38	117.84 ±304.93
rovers	2970.82 ±11662.34	4196.41 ±12675.69	0.46 ±2.51	31.47 ±151.22
satellite	20206.53 ±32578.53	9055.93 ±17750.41	2.06 ±4.20	47.97 ±93.05
scananalyzer	2001.73 ±3647.21	7401.79 ±16462.91	31.15 ±124.10	68.79 ±122.20
sokoban	1043.65 ±1208.94	688.15 ±689.09	0.05 ±0.06	110.17 ±155.70
tpp	17078.89 ±26946.22	4597.00 ±7993.90	0.64 ±1.12	102.81 ±197.28
transport	3714.30 ±8669.88	5882.19 ±10922.14	0.97 ±2.51	85.98 ±189.06
woodworking-strips	21614.60 ±59856.97	3399.64 ±6999.26	4.08 ±9.65	63.13 ±146.51
zenotravel	7058.00 ±12358.78	3765.10 ±6249.57	2.31 ±4.68	34.72 ±61.64

Table 4.7: **Search costs (with h_{FF}) for standard problems - by domain** Values are only calculated on trials when both h_{LC} and h_{Ξ} found a plan.

Domain (# problems)	Plan length for h_{LC}	Plan length for h_{Ξ}
All problems	**40.56 ±36.67**	**41.73 ±37.30**
blocks	55.52 ±43.12	62.46 ±45.83
driverlog	41.92 ±44.48	43.91 ±44.99
elevators	26.32 ±8.17	25.81 ±8.08
freecell	56.93 ±35.60	58.64 ±36.47
grid	68.47 ±53.29	72.80 ±52.24
logistics00	42.52 ±22.69	42.71 ±22.10
logistics98	51.61 ±30.40	52.34 ±30.73
mprime	6.76 ±1.98	7.16 ±2.46
openstacks	17.67 ±0.58	18.00 ±0.00
parcprinter-strips	39.10 ±23.85	39.06 ±23.84
pegsolitaire	24.11 ±9.78	24.86 ±10.07
pipesworld-notankage	32.46 ±20.40	31.06 ±20.02
pipesworld-tankage	23.06 ±16.29	22.12 ±14.77
rovers	30.00 ±14.24	31.34 ±15.98
satellite	38.67 ±23.03	40.05 ±23.58
scananalyzer	28.85 ±25.49	31.61 ±25.90
sokoban	96.88 ±110.54	96.03 ±109.14
tpp	45.89 ±34.21	41.11 ±28.77
transport	48.86 ±38.04	52.46 ±37.01
woodworking-strips	33.04 ±22.65	31.72 ±21.33
zenotravel	36.77 ±31.58	38.47 ±35.76

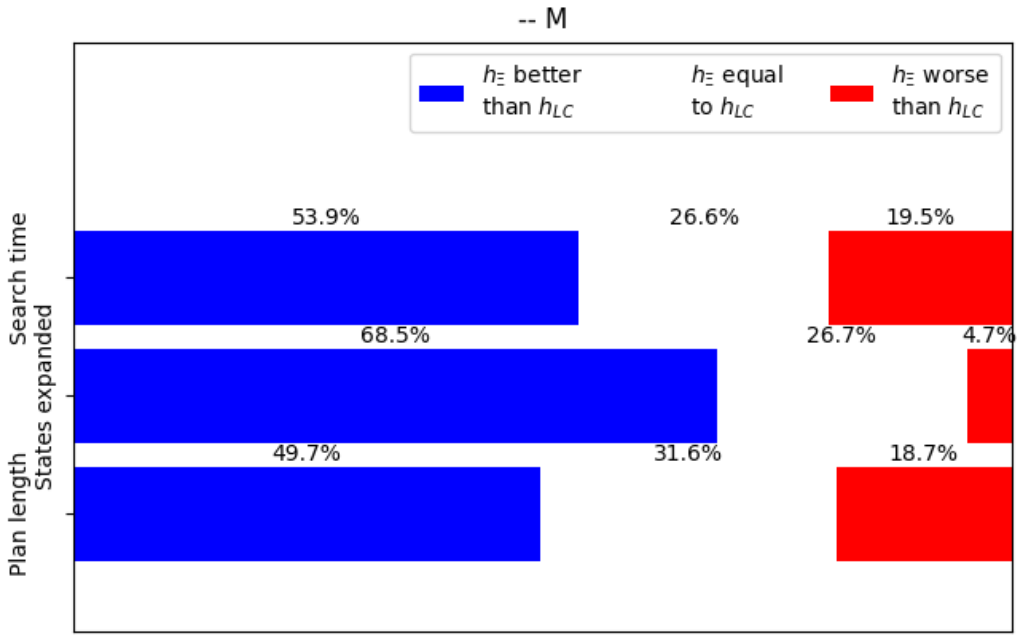
Table 4.8: **Plan length (with h_{FF}) for standard problems - by domain** Values are only calculated on trials when both h_{LC} and h_{Ξ} found a plan.

4.4.2 Problems without non-trivial landmarks

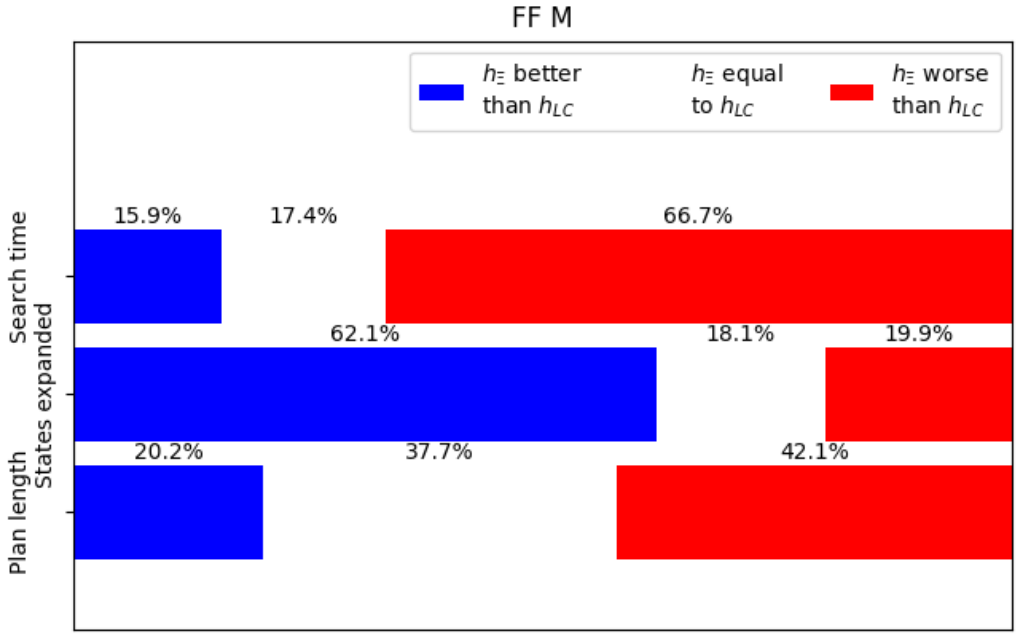
	Problems solved by h_{LC}	Problems solved by h_{Ξ}	Problems solved by both	Problems solved by neither
As the only heuristic	24.80%	71.93%	23.33%	26.60%
Alongside h_{FF}	70.33%	*80.87%	68.60%	17.40%
All 3 combined	79.8%			

Table 4.9: **Success rates of heuristics on landmark-free problems** Percentages of landmark-free problems solved by each heuristic either alone or with h_{FF}, using BFS. Each of the 500 problems generated was tested 3 times by each planner configuration, apart from all 3 combined (h_{FF}, h_{LC}, h_{Ξ}), which was tested 1 time. The best heuristic for this type of problem is marked with a *.

Table 4.9 shows the success rates of each heuristic on landmark-free problems. h_{Ξ} was able to solve far more problems than h_{LC}. Again, both heuristics solved more problems when paired with h_{FF} than alone, but even when h_{LC} was paired with h_{FF}, it solved fewer than h_{Ξ} could as the only heuristic. Figure 4.2 shows how often each heuristic did better or worse on paired landmark-free problems. As the only heuristic, h_{Ξ} finds plans faster than h_{LC} using BFS, and shorter plans using wA* in the majority of trials, which is expected given that h_{LC} failed to find a plan for most of this set of problems. When paired with h_{FF}, h_{Ξ} still finds a plan faster with BFS than h_{LC} most of the time. Because each problem tested here was generated by combining pairs of problems, it no longer makes sense to separate the results by domain.



(a) Landmark-free problems, heuristic tested alone



(b) Landmark-free problems, heuristic tested in conjunction with h_{FF}

Figure 4.2: **Heuristic results on landmark free problems** These charts show the fraction of trials where the performance of h_Ξ was better, equal to, or worse than h_{LC}, according to the metrics shown on the x-axis. Each of the 500 problems was attempted 3 times per planner configuration, in paired trials.

4.5 Discussion

In this chapter, we defined a novel relevance score heuristic h_{Ξ} based on the relevance score $\Xi(l)$ presented in Chapter 3. We then tested its performance as a heuristic with the LAMA planner on both standard IPC problems, and on new problems generated to lack non-trivial landmarks. While capable of solving both kinds of problems, we found that h_{Ξ} performed worse (ability to find a plan, and length of plan if found) than h_{LC} on standard problems. On landmark-free problems however, h_{Ξ} did much better than h_{LC}. We will now explain why we think this was observed, and how to further improve performance.

4.5.1 Standard problems

The relevance score heuristic h_{Ξ} calculated for a fully explored tree ($\overline{T_{\Pi}} = T_{\Pi}$) is the number of facts that are landmarks for plans originating from σ (for which $\Xi_{\sigma}(l) = 1$), added to the relevance calculated for other facts. This additional information, on top of landmarks, seems to impair the planners ability to find a plan within the resource limits imposed (**M1**) compared to h_{LC}. Our explanation for this is that the relevant but not essential facts, for which $\Xi_{\sigma}(l)$ is high but less than 1, guide the planner towards potentially competing plans. This "distraction" causes it to find plans that include elements of other partial plans that it could have found, leading to longer plans **M3**. Exploring more of the available search space causes more resources to be spent expanding states **M2**, which are therefore more likely to run out before a plan is found **M1**.

There is less of a difference in performance between h_{LC} and h_{Ξ} when paired with h_{FF}, but not enough to change which would be preferred on standard problems. Both h_{LC} and h_{Ξ} are assisted by h_{FF}, although all 3 together does worse than h_{LC} with h_{FF}.

While many real world problems involve key elements that must be achieved in order to solve the problem as a whole (ie contain landmarks), many can also be solved in multiple different ways (and so contain few or no landmarks). As can be seen in Tables 4.3, and 4.6, h_{Ξ} outperforms h_{LC} on the `satellite` domain. The `satellite` domain (see Appendix C

for the domain definition, and [github: hsp2,] to see all problems defined in PDDL) involves choosing combinations of instruments and satellites to take images of objects in different regions of the sky. Problems set in this domain sometimes contain a single instrument able to take a particular picture, corresponding to a landmark; but they also often contain multiple instruments that could manage it. They are not interchangable (because the satellites may also carry different instruments competing for the power supply and direction), but represent different, non-overlapping ways to approach the goal. This property favours h_{Ξ} over h_{LC}, and tends to be present more interesting problems in the real world. Real world problems can normally be solved in many different ways, some of which share points of interest - almost landmarks, that will be assigned high relevance scores.

4.5.2 Problems without non-trivial landmarks

For landmark-free problems, h_{LC} can only tell that a partial plan might be good when it finds one of the goal facts. Until then, it searches a flat surface, increasing the distance from the initial state in all directions. This predictably does very poorly by all measures.

By contrast, h_{Ξ} is able to climb an informative surface that guides it toward potential plans, allowing it to find a plan more reliably, after less searching. By rewarding the planner for finding facts that are relevant to alternative, but potentially separate plans, it has a tendency to include some actions in the final plan that did not contribute to achieving the goal. We believe this explains why it finds longer plans than h_{LC}, particularly on problems that are known to be solvable by disjoint plans. This is an unintentional byproduct of the way these domains were created that may not be the case on problems generated in other ways.

It is significant that h_{Ξ} alone is able to solve more problems than either configuration that does not include it (ie $[h_{LC}]$ or $[h_{LC}, h_{FF}]$). This demonstrates that h_{Ξ} provides useful information that is not available to the other heuristics.

Despite the work done in Chapter 3 to minimise the calculations that must be done each time a state is evaluated, h_{Ξ} requires more computations, and therefore more time to compute

for each state than h_{LC} or h_{FF}, and probably most other commonly used heuristics. Further optimisation of the code (as proposed in Section 3.9) is planned to address this.

4.5.3 Synergy with h_{FF}

As reported by [Keyder et al., 2010], combining h_{LC} with h_{FF} significantly improves its ability to solve problems. Table 4.9 verifies that this is particularly true in problems containing no non-trivial landmarks, which h_{LC} struggles with alone.

As may be expected, given its similarity to h_{LC}, h_{Ξ} is also able to solve more problems when paired with h_{FF}. For problems with no non-trivial landmarks, h_{Ξ} and h_{FF} together is able to solve more landmark-free problems than any other planner tested.

4.5.4 Conclusion

Overall, the key observation is that the heuristic, based on the relevance score, is able to guide the LAMA planner toward solving a class of planning problems for which landmark counting is ineffective. This comes at the cost of being more expensive to compute, worse performance on standard problems, and a tendency to find longer plans. It can therefore only be recommended on the class of problems for which it is superior; those with few or no non-trivial landmarks. The fact that landmarks are identified before a plan search procedure begins (and the number of non-trivial landmarks is reported by the LAMA architecture), allows for a cheap and simple way to leverage the benefits of both h_{LC} and h_{Ξ}: use h_{LC} (and h_{FF}) on problems with well-defined landmarks, and use h_{Ξ} (and h_{FF}) for problems that only have trivial landmarks.

Much like h_{LC}, h_{Ξ} is not strictly admissible as a heuristic (see Definition 4). This is because both heuristics are linked to the number of facts that remain to be achieved, without considering how many actions might be needed to achieve them. This can be proven by observing that as there is no limit to how many facts may be achieved by a single action, it is possible to design a planning problem that can be solved by 2 actions (ie the initial state has a distance to the goal of 2), the first of which has effects consisting of an arbitrarily large

number (greater than 2) of highly relevant or landmark facts that are preconditions of the second action. If it were solvable with a single action, then all landmarks would be trivial. Despite this theoretical result, it is rare for either heuristic to overestimate the distance to the goal in practice. This is because both heuristics count a subset of the facts that are achieved in a plan. Many actions in a plan will reduce both heuristics by less than 1, allowing the distance to the goal to reduce at a consistently faster rate before reaching 0 (ie maintaining a greater value than either heuristic until the goal state is reached).

4.6 Future work

In future work, we hope to explore these insights further with additional planning problems, with a range of non-trivial landmark counts from few to many. We anticipate this revealing the point at which h_{LC} starts to fail and is overtaken by h_{Ξ}. Finding some real world problems with this property, and doing similar experiments to those described here on them could reveal (or refute) that the performance improvement of h_{Ξ} over h_{LC} is due to an artificial properties of the synthetic domains. We also plan to compare performance to other planning heuristics to better understand what properties of a planning problem influence the performance of h_{Ξ}.

In many problems, a large proportion of facts have a very low $\Xi(l)$, and so contribute very little information to h_{Ξ}. Excluding such facts by introducing a threshold (eg top quartile) would reduce the cost of computing $\Xi(l)$ with a negligible impact on its value. It is also possible that reducing the draw towards less common partial plans (while preserving the ability to recognise relevant but not essential paths) might improve the issue of distraction.

We plan to explore the idea of using the order of nodes in a branch of the tree $\overline{T_{\Pi}}$ to calculate something similar to the orderings that are used by landmarks. This could allow us to reward repeated progress towards the same plan strategy by following the $\overline{T_{\Pi}^{path}}$ of nodes representing facts as they are achieved. We hope that this would alleviate the issue of distraction, and make h_{Ξ} a more focussed heuristic. Orderings are used by landmark based systems to efficiently calculate the landmark count for states that proceed from explored states

with minimal calculations. If we can find a similar application of orderings to the relevance score, this might also help to improve the cost of computing h_{Ξ}.

Chapter 5

The Relevance Score for Diagnosis

The task of diagnosis by abduction (as described further in Section 2.3.3) can be viewed as finding a completion of the initial state by assumption, and the verification of a corresponding plan [Sohrabi et al., 2010]. The main difference between a standard classical planning problem, and a diagnosis problem expressed as a classical planning problem, is the incompleteness of the initial state. Supplying a list of permitted fixes requires much of the problem to have been solved manually by the person encoding the problem. Without these, valid plans may not exist unless additional facts are assumed to be true at the start.

In Chapter 4 we demonstrated that the relevance score is particularly useful for planning in problems that lack non-trivial landmarks. The lack of viable plans precludes the existence of landmarks (trivial or non-trivial), but does not interfere with the calculation of the relevance score. Other heuristics based on distance to the goal would also be undefined for problems where no paths exist. Based on this, it was hypothesised that the relevance score could be useful for finding candidate facts for assumption, without relying on a predefined list.

This chapter defines a method for generating diagnosis problems in PDDL. Then, a method is described for using the relevance score to populate a list of candidate fixes (assumptive actions that make those facts true in the initial state) that are then validated by finding a plan that uses them to achieve the goal.

5.1 Problem formulation

Fixing problems that have been ablated by the removal of facts from the initial state is directly equivalent to some formulations of diagnosis, including the one used here (see Section 2.3.3). In both problem types, a classical planning problem with no classical solution must have its initial state fixed before a plan can be found. We use Definitions 13, 14 and 15 of diagnostic planning problems stated in Section 2.3.3.

The rest of this section will clarify how other forms of diagnosis relate to this formalism. The rest of this chapter will describe a method to identify a list of candidate fixes for a problem, and verify that they permit an explanation. Facts will be scored and ranked according to an assumability score $\tau(f)$, and those with the highest values (the top n) selected as candidates.

5.1.1 Encoding observations as goals

In many diagnosis problems, the time, or relative order in which observations are made is often a factor in identifying the problem. Encoding temporal information in a classical planning problem has been well studied (see Section 2.4.2 and [Ghallab et al., 2004] Chapters 13, 14), with many approaches to choose from.

The approach used in [Sohrabi et al., 2010] (based on [Grastien and Kelareva, 2007]) is to express observations as temporally extended goals (TEGs). TEGs force an ordering on goals by defining an $advance$ action that has a precondition of an early goal, and an effect that is required for the subsequent goal. This does not fundamentally change the properties of a classical planning problem, and is not a property of the problems tested here.

5.1.2 Faulty actions

Some formulations of diagnosis problems allow actions to be *faulty* (eg [Sohrabi et al., 2010]), and is a reasonable interpretation of the faulty components or axioms present in conventional (ie non-planning) diagnosis [Rodler, 2023]. While the relevance score can be calculated for actions as well as facts, the range of alternative/faulty actions that could allow

an explanation if permitted, exist in the space of n-ary combinations of facts. In practice, this is constrained by the real world mechanics of those actions. Encoding this knowledge of how actions may be faulty is another way of providing a list of fixes, which the approach described here is designed to avoid.

A specific subset of faulty actions may be considered - a relaxation of preconditions. The approach described here could be used directly to find this sort of fix, by applying the same scoring approach used for facts to actions. This is conceptually and functionally equivalent to assuming the preconditions that are relaxed, but would require a greater degree of code modification.

5.2 Selecting a fix

In this section, we will describe the criteria by which we wish to select facts for assumption, and the way we quantify how good a candidate appears to be.

5.2.1 Desired properties of selected fixes

The question of which facts are best to assume does not have a single, obviously correct answer. For the ablated problems attempted here, there is a corresponding unablated problem to compare the answer to, but for real world problems, this may not be the case. Even if the entire sequence of events, and the mechanisms behind them are known, it is hard to define causative relations between them [Pearl, 2002]. Chains of causation often have no definitive start, for example, one could say that the student has the Hardcopy because they printed it in the office. This would be correct, but so would saying that they wrote and printed it in the office. Both of these could accurately describe the situation described in Figure 2.2, but the latter is a more complete explanation.

The following are desirable properties of facts to be considered as part of a candidate fix (see Definition 14):

1. Likely to be used in a plan
2. Avoid trivial fixes

Property 1 favours facts that are likely to have a plan that achieves G_{obs}. Without this, α could not be part of an explanation.

Property 2 aims to exclude assumptions that are too close to the goal to be informative. As discussed above, it is preferable to assume a fact that occurs early in a plan, than to assume a consequence of that fact. There will always be a chain of events prior to any assumed fact that explains how it came to be true, and facts further down this chain are considered more explanatory to the observations than their immediate precursors.

5.2.2 The assumability score, $\tau(l)$

The properties described in section 5.2.1 can be quantified, using applications of the relevance score. Each candidate fact is assigned an *assumability score* $\tau_i(l)$, for each property.

Definition 30 (Assumability score - relevance, $\tau_1(l)$)**.** The relevance score of l in the initial state is defined (Definition 26) as the likelihood of l being in a partial plan sampled by the HRR.

$$\tau_1(l) = \Xi_I(l) \tag{5.1}$$

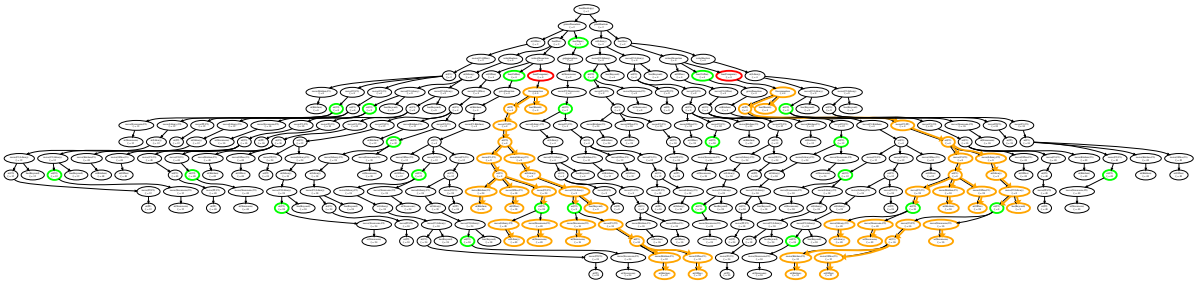
Definition 31 (Assumability score - complexity, $\tau_2(l)$)**.** The relevance score heuristic is used in Chapter 4 as a measure of how close the operand state is to the goal. If applied to the initial state plus l, this provides an estimate of how many relevant facts still need to be achieved after l is assumed.

$$\tau_2(l) = h_\Xi(I \cup l)$$

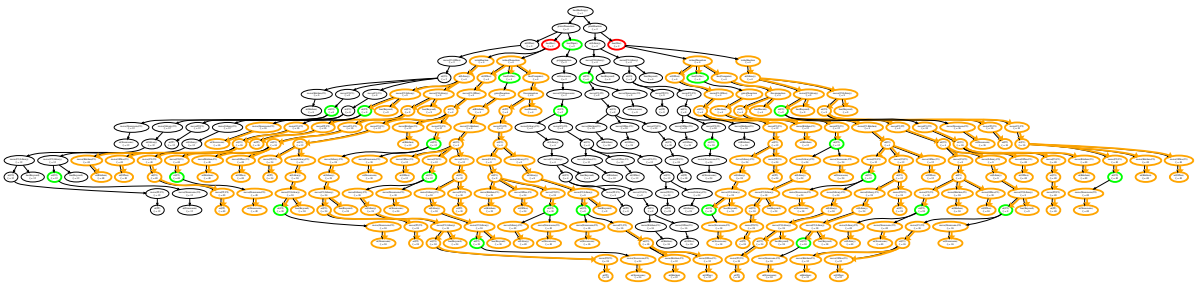
Definition 32 (Assumability score, $\tau(l)$)**.** A value, $\tau(l)$, that combines these appropriately represents how well assuming fact l would satisfy the desired properties of a fix. Using the product of these values favours situations where both are high, without allowing one or the other to compensate for the other being very low.

$$\begin{aligned}\tau(l) &= \tau_1(l)\tau_2(l) \\ &= \Xi_I(l) \times h_\Xi(I \cup l)\end{aligned} \tag{5.2}$$

5.2.3 Worked example



(a) T_Π **with has(Computer) (red) and descendents highlighted**



(b) T_Π **with has(Doc) (red) and descendents highlighted**

Figure 5.1: **Example of candidate fixes** Fully explored tree for the example problem described in Figure 1.1. Nodes with labels that are true in the initial state are highlighted in green.

Figure 5.1 illustrates the effect of choosing different nodes as candidates for fixes. $\tau_1(has(Doc)) = 1$, because all actions that achieve the goal ($printlibaction$ and $printofficeaction$) require $has(Doc)$ as a precondition. $\tau_1(has(Computer)) = 0.5$ because of the two actions that can achieve $has(Doc)$ ($writeofficeaction$ and $writelibaction$), only one requires $has(Computer)$ as a precondition. Assuming $has(Doc)$ is correctly

judged to enable the HRR to sample applicable partial plans more frequently than assuming $has(Computer)$.

$\tau_2(has(Doc))$ will have a low value, because most of the tree is truncated below $has(Doc)$ nodes. Only facts relating to accessing a working printer will contribute to $\tau_2(has(Doc))$. $\tau_2(has(Computer))$ will have a moderate value, higher than $\tau_2(has(Doc))$. This is because the relevance score heuristic, $h_\Xi(I \cup has(Computer))$ is calculated over more of the tree than $h_\Xi(I \cup has(Doc))$. Assuming $has(Computer)$ is correctly judged to lead to longer explanations than assuming $has(Doc)$.

5.2.4 The size of α

Equation 5.2 calculates $\tau(l)$, for a single fact, but a fix, α is defined (Definition 14) as a set of facts.

Definition 33 (Assumability score - relevance, $\tau_1(\alpha)$)**.** The only equation needed to calculate $\tau(l)$ for which l must be a single fact rather than a set of facts, is Equation 5.1. This can be modified accordingly, without changing the interpretation of its value:

$$\tau_1(\alpha) = \sum_{l \in \alpha} \Xi_I(l) \tag{5.3}$$

Definition 34 (Assumability score, $\tau(\alpha)$)**.** The other equation for τ_2 already treats l in a way that is compatible with it being replaced by a set of facts, α.

$$\tau(\alpha) = \sum_{l \in \alpha} \Xi_I(l) \times h_\Xi(I \cup \alpha) \tag{5.4}$$

As noted in [Sohrabi et al., 2010], smaller, or ideally minimal, fixes are generally preferable to larger ones, as well as coming from a combinatorially smaller pool of potential sets. Therefore, it is appropriate to try the best n fixes of size $|\alpha| = m$, as determined by Equation 5.4, before increasing m by 1, until an explanation is found.

5.2.5 Normalisation

There is likely to be a significant disparity between the values of $\tau_1(l)$ and $\tau_2(l)$. $\tau_1(l)$ is a probability, so has range $[0, 1]$, wheras $\tau_2(l)$ is a sum of probabilities, so can have any value in $\mathbb{R}^+$. The range of $\tau_2(l)$ could be normalised to $[0, 1]$ by considering it relative to $h_\Xi(I)$ (which is its upper bound). As this upper bound is constant with respect to α, normalising with respect to it would just multiply all values that are being evaluated by the same constant for a given diagnostic problem. This is not done for the sake of clarity and computational cost.

If $\tau_1(\alpha)$ is to be compared to fixes of different sizes, it should be normalised by dividing by $|\alpha|$. As discussed in Section 5.2.4, it is anticipated that $\tau_1(\alpha)$ will be used to compare candidate sets of equivalent size, making this normalisation unnecessary.

5.3 Evaluation

In order to evaluate the ability of $\tau(l)$ to quantify how appropriate a fix is for a diagnostic planning problem, the code used in Chapter 4 to calculate Ξ was adapted to read such a problem, calculate $\tau(l)$ for all facts represented by labels in $\overline{T_\Pi}$, and use this to select the most promising candidate fixes. This code has been made available[1] (see Section 6.1 for further discussion).

5.3.1 Implementation details

The *translate* and *preprocess* modules of LAMA compile away facts that cannot be reached, and actions that cannot become applicable from the initial state under the delete-relaxation. This means that for a classically unsolvable problem, many parts of the problem that may be essential to an explanation would be unavailable to the program. Because LAMA was considered unsuitable for this type of problem, a version of the relevance-score code used for diagnosis was separated from it, and made to run independently. This also has the benefit of

[1] `https://bitbucket.org/Oliver_Kim/relevancediagnosis/`

allowing it to run on newer operating systems, as LAMA is currently limited to ≤Ubuntu 18 (at least with reasonable attempts at installation), and has not been updated for a few years.

Currently, this system has only been tested for $|\alpha| = 1$, on problems where it is known that it is possible to find an explanation by assuming a single fact. This limitation will be addressed in future work.

Domains and problems specified in PDDL are preprocessed by applying the delete relaxation, and translated into a format that exposes actions that can achieve predicates. The tree $\overline{T_\Pi}$ is explored as described in Algorithm 3, only representing ground actions and predicates as they are encountered during this exploration. $\tau(f)$ is then calculated for all $f \in F$ (those with no nodes in $\overline{T_\Pi}$ can be trivially assigned $\tau(f) = 0$ and not considered) according to Equation 5.2, and the $n = 10$ facts with the highest value of $\tau(f)$ are listed as good candidates for assumption. If multiple candidate facts have the same value of $\tau(f)$, they are chosen at random within their rank up to a total of 10 to be considered candidate fixes (see Definition 14). Any remaining facts that would take the total above 10 are discarded.

Each candidate fix is used to generate a new PDDL problem file by adding it to I, and the resulting problem, $\Pi^*_{diag} = \langle F, A, I \cup \alpha, G_{obs} \rangle$ is attempted by LAMA using BFS and h_{FF}, h_{LC} heuristics, with resource limits of 8GB RAM and 1h. If this yields a plan (explanation), then the fix is considered successful. All experiments were performed on the same hardware as described in Section 4.2.1. The unablated problems are known to contain landmarks, so this is the planner configuration with the best chance of solving them.

5.3.2 Hypotheses

We experimentally evaluated the following hypotheses:

H1 The assumability score $\tau(l)$ can be used to find fixes for diagnostic planning problems that lead to viable explanations.

H2 The assumability score $\tau(l)$ can be used to identify the fact that was ablated from Π to generate Π_{diag}.

5.3.3 Measures of success

Each diagnostic planning problem given to the program was evaluated by 3 measures.

Measure **M1** is how far down the sorted list of candidate assumptions is it necessary to go before finding a fix that leads to an explanation. Better candidates ought to be more likely to permit the existence of a plan, and real world applications would test those with the highest scores first. Lower values of measure **M1** indicate greater support for hypothesis **H1**.

Measure **M2** is the fraction of the candidate list (of 10 proposed fixes) that lead to viable explanations. Higher values of measure **M2** indicate greater support for hypothesis **H1**.

Measure **M3** is the position of the ablated fact that generated the problem in the sorted candidate list. Lower values of measure **M3** indicate greater support for hypothesis **H2**.

5.3.4 Potential problem sets

We were unable to find a suitable set of problems available online to directly test the hypotheses described in Section 5.3.2. Code and problem sets used for [Sohrabi et al., 2010] are not published. Some other diagnosis systems that directly use task planning approaches use a more complex representation of planning domains that incorporate probability (eg [Hanheide et al., 2017]), which the relevance score is not intended to handle directly. Most diagnosis or logical abduction literature (eg [Baral et al., 2000], [Zhou et al., 2023]) represent problems in variations of first order logic that would require significant work to translate into PDDL.

Related fields such as plan recognition often have a different focus, such as the use of real-time motion data [website: planrec,]. There are problem sets available in PDDL (eg. [github: IJCAI,] and [website: Process mining,]) but these still seek to identify different components of the planning problem (ie the goals, G_{obs}). While it seems possible that the relevance score could be used to guide a planner in such systems, this would not make use of its main benefit - that it does not rely on the existence of plans the way landmarks do.

There is a problem set of (potentially) unsolvable planning problems, specified in PDDL available for the Unsolvability track of the International Planning Competition 2016 [website:

unsolve,], for which all problems are available [github: unsolve,]). Closer inspection of the problems revealed that they tend to be unsolvable for one of two reasons:

- They employ a finite resource that is required to make progress. Problems are unsolvable when there is insufficient of this resource available (eg document-transfer [2]). Fixes for this type of domain would mostly be in the form of increasing this resource above the threshold.

- They employ a fixed structure that follows a logic based on Euclidian geometry (eg bottleneck[3]). Fixes for this type of domain would involve opening paths that make sense of the underlying topology of the map.

Meaningfully fixing problems in either type of domain would require an understanding of how they work, beyond their direct representation in PDDL. PDDL is a poor representation of quantified or physical spaces for planning. The relevance score does not provide a means to represent or use these types of structural information, or the constraints they imply on what facts can or cannot be changed.

[2] https://github.com/AI-Planning/unsolve-ipc-2016/tree/master/domains/FINAL/document-transfer

[3] https://github.com/AI-Planning/unsolve-ipc-2016/tree/master/domains/FINAL/bottleneck/

Algorithm 5 - Generation of diagnostic planning problems Line 1 takes the components of a planning problem, specified in PDDL. Line 9 chooses a fact from F with uniform probability. Lines 2 and 11 attempt to solve Π_{diag} using BFS with h_{FF} and h_{LC}, restricted by 4GB RAM and 10min. In practice, problems are either solved or quickly detected to be unsolvable by LAMA's preprocessing. Up to $limit = 10$ attempts (checked on line 6)were made to choose a fact, whose ablation prevents the problem being solvable.

1: Let $\Pi_{diag} \leftarrow \langle F, A, I, G_{obs} \rangle$
2: Let $solved \leftarrow attempt(\Pi_{diag})$
3: Let $counter = 0$
4: **while** $solved == TRUE$ **do**
5: $counter++$
6: **if** $counter > limit$ **then**
7: **return** FAILURE
8: **end if**
9: Choose $f \in I$
10: Let $\Pi_{input} \leftarrow \langle F, A, I/f, G_{obs} \rangle$
11: Let $solved \leftarrow attempt(\Pi_{diag})$
12: **end while**
13: **return** Π_{diag}

5.3.5 Problem generation

Considering the difficulties encountered in finding a suitable problem set used in comparable work, it was decided that generating such a problem set would be appropriate. Based on the observation that diagnosis is equivalent to planning with an incomplete initial state, diagnosis planning problems were generated by ablating classical planning problems according to Algorithm 5. Resource limits of 4GB RAM and 10min were applied to each attempt at solving the ablated problem before it was designated unsolved. In many cases, LAMA identified ablated problems as unsolvable without needing to start heuristic search.

This ablation procedure was applied to problems defined in the examples folder of the HSP2 repository [github: hsp2,] [Bonet and Geffner, 2001a] (summarised in Table 4.1) that were solved by the LAMA planner using BFS and both $[h_{FF} + h_{LC}]$ and $[h_{FF} + h_{\Xi}]$ combinations of heuristics in under 10min. Seven domains (freecell, openstacks, parcprinter-strips, pegsolitaire, pipesworld-notankage, pipesworld-tankage, sokoban) contain syntax that is not yet supported by the new parser, and so were not used. 291 problems satisfied this criteria and were successfully ablated. The number of problems successfully ablated in this

manner within each domain is shown in Table 5.1. The code used to generate diagnosis problems has been published along with the solver (see Section 6.1)

5.4 Results

The system described in Section 5.3.1 was applied to each of the ablated problems generated according to the procedure described in Algorithm 5. Of the 291 ablated problems, a set of candidate fixes was generated for 257 (88.32%), of which 191 (65.64%) led to viable explanations. For 49 (16.84%) problems, the fact that was originally ablated was recovered as one of the candidate fixes. Table 5.1 and Figure 5.2 show a breakdown of these results according to domain.

Domain (# problems)	Problems generated candidate fixes	Problems generated fixes	Problems regenerated ablation
All domains (291)	**257 (88.32%)**	**191 (65.64%)**	**49 (16.84%)**
blocks (35)	35 (100.00%)	12 (34.29%)	7 (20.00%)
driverlog (15)	15 (100.00%)	15 (100.00%)	6 (40.00%)
elevators (30)	30 (100.00%)	22 (73.33%)	2 (6.67%)
grid (4)	4 (100.00%)	4 (100.00%)	0 (0.00%)
logistics00 (28)	28 (100.00%)	21 (75.00%)	5 (17.86%)
logistics98 (19)	19 (100.00%)	14 (73.68%)	1 (5.26%)
mprime (29)	6 (20.69%)	6 (20.69%)	0 (0.00%)
rovers (18)	16 (88.89%)	14 (77.78%)	4 (22.22%)
satellite (20)	20 (100.00%)	19 (95.00%)	4 (20.00%)
scananalyzer (21)	13 (61.90%)	9 (42.86%)	9 (42.86%)
tpp (11)	11 (100.00%)	9 (81.82%)	6 (54.55%)
transport (26)	26 (100.00%)	24 (92.31%)	1 (3.85%)
woodworking-strips (16)	15 (93.75%)	5 (31.25%)	1 (6.25%)
zenotravel (19)	19 (100.00%)	17 (89.47%)	3 (15.79%)

Table 5.1: **Diagnosis results split by domain**

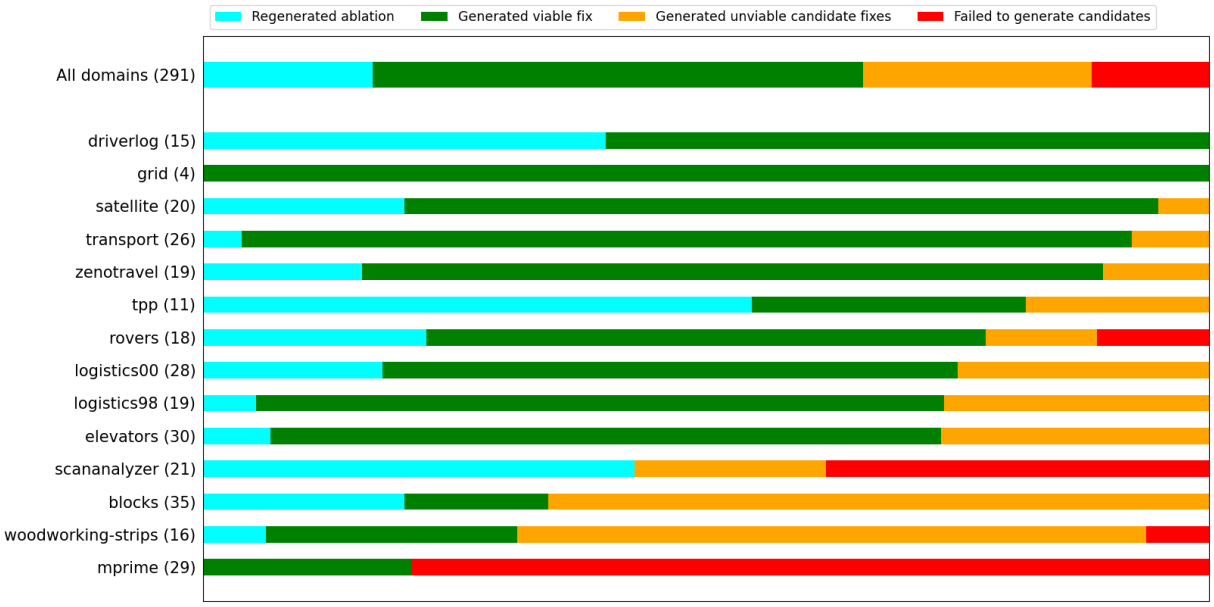


Figure 5.2: **Diagnosis results split by domain** Sorted by the fraction that generated a viable fix (ties broken by fraction that regenerated the ablated fact). Categories are dependent on each other from left to right as plotted, and include all categories to their left. I.e. if a problem regenerated the ablated fact, then it must also have generated a viable fix (the unablated problem is known to be solvable), which in turn implies that candidate fixes were generated.

5.4.1 Finding explanations

Of the 291 ablated problems, 191 (65.64%) generated fixes that led to a viable explanation. (see Definition 15). As discussed in Section 5.2.1, multiple fixes could be considered valid. In order to give structure to the range of potentially valid fixes generated, we assigned each candidate fact l a rank according to its value of $\tau(l)$. Each rank may be assigned to more or fewer than one fix in cases of ties. Figure 5.3 shows how much of the list must be traversed before finding a viable fix. Figure 5.4 shows how likely a fix was to be viable at each position (or rank) in the candidate list.

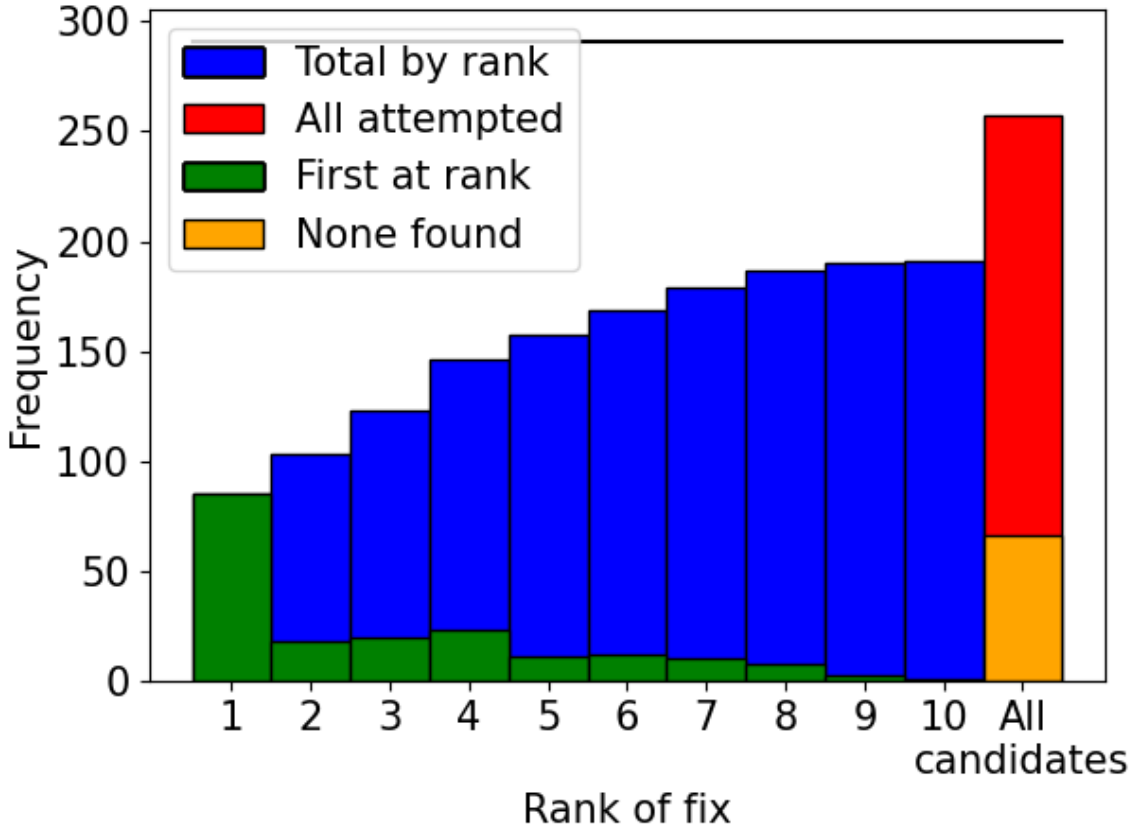


Figure 5.3: **Diagnosis results M1 - First viable fix** Histogram showing how often a viable fix would have been found by a rank, if the candidate list were traversed in order. If the first viable fix for a given problem is found at a rank, it is shown in green. If a viable fix would already have been found by the time that rank was tried, it is shown in blue. The red bar shows the total number of problems for which candidate fixes were generated, and the orange bar below it shows how many of those included no viable fixes. The black line shows the total number of problems attempted.

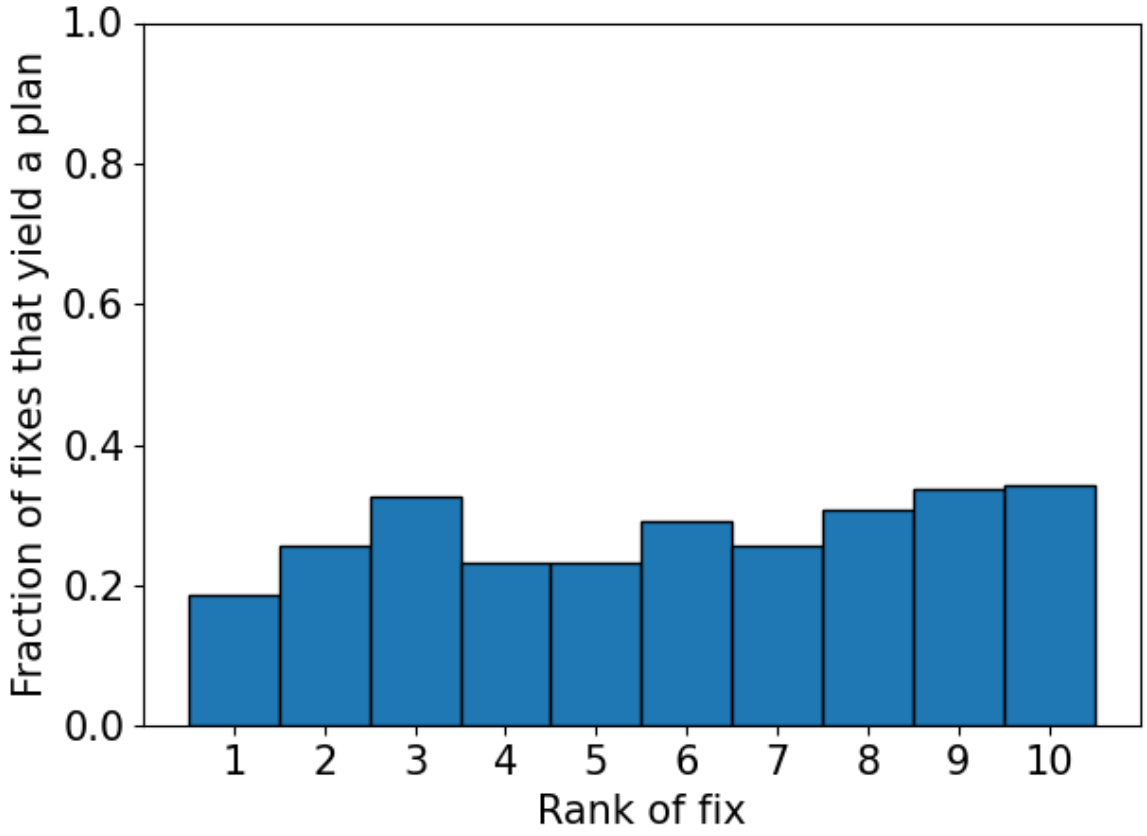


Figure 5.4: **Diagnosis results M2 - Average viability of fixes** Bar chart showing the fraction of fixes for which a plan/explanation can be found at each rank. The probability of a fix at any rank being viable was 0.248

5.4.2 Finding the fix that was ablated

Of the sets of candidate fixes generated, 49 (16.84%) contained the fact that had been ablated when the problem was created. Figure 5.5 shows where the ablated fact was ranked if it was found.

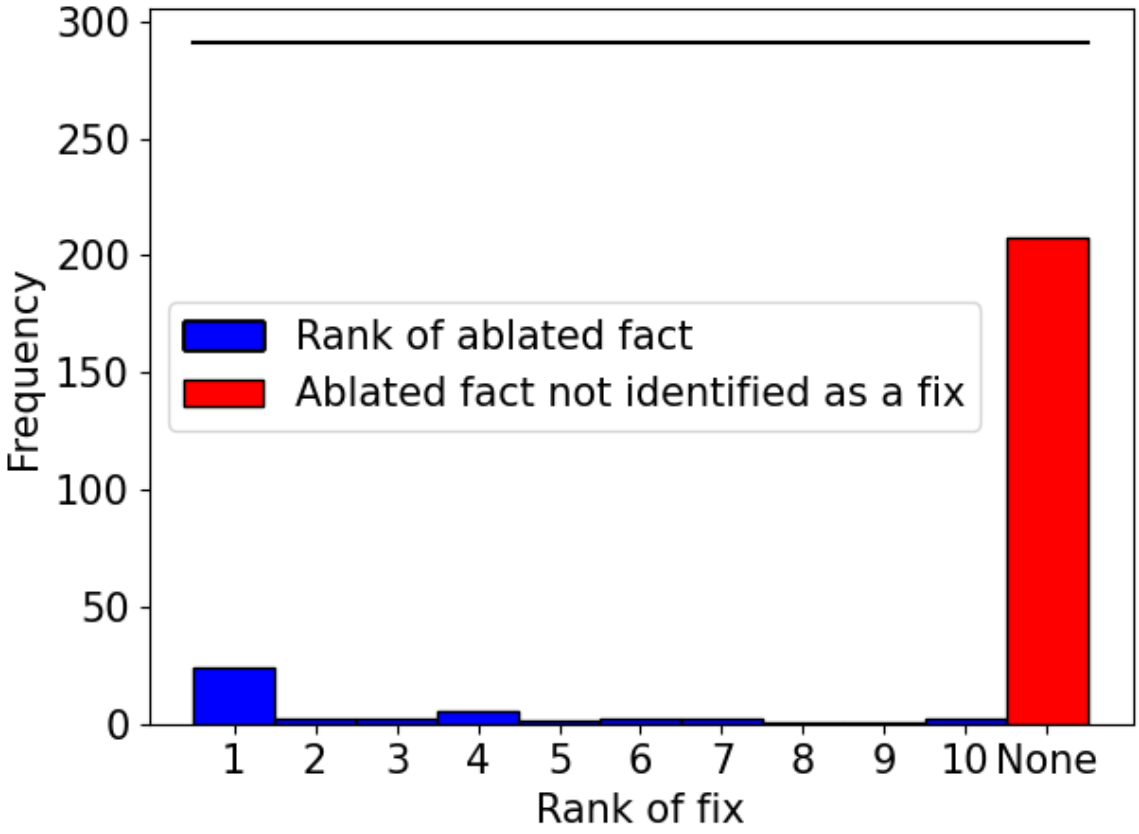


Figure 5.5: **Diagnosis results M3 - Rank of ablated fact** Histogram showing the number of times a rank was given to the ablated fact. The red bar counts instances where the ablated fact was not identified as a candidate fix for that problem. The black line shows the total number of problems attempted.

5.5 Discussion

In this chapter, we presented a novel method for repairing planning domains that had been damaged by the removal of facts from the initial state, leaving them with no solution. This used the Relevance Score $\Xi(l)$ presented in Chapter 3 to evaluate facts, and define an Assumability Score $\tau(l)$ to decide which facts (or in principal, combinations of facts) are most likely to enable a plan to be found. This yielded fixes for at least some problems in every domain tested, with significant variation in performance between domains.

5.5.1 Finding explanations

As Figure 5.3 shows, when a list of candidate fixes was generated for a problem, the highest rank would contain a fix more often than lower ranks. Part of the reason for this (and the reason that this is consistent with Figure 5.4 not showing fixes of rank 1 as being more likely than other ranks to lead to an explanation) is that 658 fixes were assigned rank 1 for the 257 problems that generated a list of candidates. This indicates that finding fixes with the same value of $\tau(f)$ was common. The high rate of the candidate list containing a viable fix (65.54%) leads us to accept **H1** .

The fraction of fixes that were determined to be viable (**M2**, see Figure 5.4) is fairly low, and further investigation yielded an explanation for this. Several fixes that looked like they should be viable failed to enable the generation of a plan. This was determined to be a result of the SAS representation used by LAMA, which generates implicit preconditions for actions that are not specified in the PDDL, corresponding to the state variables altered by an effect.

An example of this is shown in Figure 5.6. The action `(put-down a)` should be applicable in the initial state, as its only precondition, `(holding a)` is true. However, in LAMA's SAS representation, a variable will exist, for which `(handempty)` is a possible value, and which must hold a different value in the preceeding state for `(put-down a)` to be considered applicable. In effect, this creates additional preconditions for each action, making it inapplicable in any state in which some of its effects are already true.

```
(define (problem blocks-5-1)
(:domain blocks)
(:objects a d c e b )
(:init
    (clear b)
    (clear e)
    (clear c)
    (ontable d)
    (ontable e)
    (ontable c)
    (on b a)
    (handempty)
    (holding a)
  )
(:goal (and
    (on d c)
    (on c b)
    (on b a)
    (on a e)
    ))
)
```

```
(:action put-down
    :parameters (?x)
    :precondition (holding ?x)
    :effect
    (and (not (holding ?x))
        (clear ?x)
        (handempty)
        (ontable ?x)))
```

Figure 5.6: **Inconsistent blocks fix. Left**: A problem definition generated by ablating `(on a d)` from probBLOCKS-5-1.pddl, and attempting a fix by assuming `(holding a)`. **Right**: The `put-down` action from the blocks domain. The original problem and complete blocks domain are shown in Appendix B.

Of course, it should not be possible for both `(handempty)` and `(holding a)` to be true at the same time, but this is not enforced by the PDDL specification. Intuitively, assuming that the missing block a is anywhere, should be considered a viable fix, but it needs further manipulation of the initial state to maintain logical consistency. The domains blocks and woodworking-strips are both vulnerable to this issue, which explains them having a particularly low proportion of fixes found to be viable. This illustrates the importance of applying logical reasoning and domain knowledge to constrain or select fixes, after a list of candidates has been generated.

5.5.2 Finding the fix that was ablated

Figure 5.5 shows that when the ablated fact is included in the list of candidate fixes, it is normally ranked first. However, overall, it is not ranked in the top 10 fixes most of the time.

This suggests that there is a bimodal distribution of problems where it either works well or poorly. As a result, we are forced to reject **H2** for the assumability score in its current form.

To explain this, we looked at the values for $\tau_1(l)$, and $\tau_2(l)$ that were calculated for facts in problems where the ablated fact was not found. It was noted that while the full range of $\tau_1(l)$ was represented (ie values ranged from 0 to 1), the same was not true for $\tau_2(l)$. The lowest value of $\tau_2(l)$ was frequently not much less than the highest value in the same problem. This meant that the differences in $\tau(l)$ were mostly determined by $\tau_1(l)$, which is not sufficient by itself to determine whether a fact is a good candidate for assumption.

5.6 Future work

While we have shown that the Assumability score can be used to diagnose and fix some ablated problems, there are several ways in which it could be improved:

1. Some additional work is needed to allow this software to robustly support a wider range of PDDL syntax. As a minimum, it should be able to work with the problems listed in Section 5.3.5.

2. In Section 5.5.1, we identified a reason for many candidate fixes being classfied as unviable, despite superficially looking like they should work. While extensive reasoning about how reasonable it is to make a fix is outside the scope of this work (see Sections 1.2 and 2.5), a simple check could allow a large amount of the failed candidate fixes to become viable. If a fact is only achievable by actions that include a specific negative effect, then the fix should also involve removing that fact I. We limit this to concurrent negative effects, because adding extra facts to I will be handled explicitly when we extend the problem to $|\alpha| > 1$.

3. We plan to introduce a third term $\tau_3(\alpha)$ to the Assumability score that quantifies whether α enables new regions of the search space to be reached. If α can be reached from I, then (in the delete relaxation) assuming it would not extend the set of facts that could have

been reached. This could be achieved by using reachability analysis (see Definition 7) to determine whether facts in α are reachable from the initial state. An alternative approach being considered is to attempt to generate a plan with α as the goal (ie solve $\Pi_\alpha = \langle F, A, I, \alpha \rangle$. Either would provide a binary answer to whether α could be reached from I.

4. In Section 5.4.2, we noted that $\tau_2(\alpha)$ is limited in its ability to differentiate how close an assumption is to the goal. We plan to see whether using a heuristic based on the distance to the goal such as $\tau_2(\alpha) = h_{FF}(I \cup \alpha)$ might perform better.

5. Once the previous issues have been addressed, we plan to test the system on problems that have had more than 1 fact ablated($|\alpha| > 1$).

6. As discussed in Section 2.3.3, [Göbelbecker et al., 2010] describes solving a similar task. While they report attempting a limited number of problems, a comparison between their approach and that taken here is appropriate, and planned. As discussed in Section 2.3.3, fair comparison to other diagnosis systems is made difficult by their reliance on providing lists of allowable fixes.

Chapter 6

Discussion

In this thesis, we have identified a limitation of the concept of landmarks - that they do not exist in all planning problems, or at all in diagnosis problems. This prevents them from being used as a heuristic to guide the search for a plan (for an identifiable class of classical planning problems) or fix (for diagnosis problems). Based on this observation, we defined the concept of relevance, that expands on the concept of a landmark from a binary assessment of whether a (set of) fact(s) must become true in all plans, to a continuous score of how often partial plans are found that contain that (set of) fact(s). This was presented, along with an algorithm for calculating the relevance score, without reference to an initial state, or reliance on a problem being solvable.

We then used this relevance score to define a heuristic to guide a search algorithm towards a plan in a classical planning domain. This was shown to outperform other heuristics (whether alone, or paired with h_{FF}) on a class of planning problems that lacks non-trivial landmarks. A procedure for generating such planning problems was also detailed.

We identified a limitation of previous approaches to solving diagnostic problems with planning techniques - the necessity of providing the system with a list of candidate fixes. In response to this we used the relevance score to define and test a method for generating such a list, without relying on externally provided information.

6.1 Publications

- The work in Chapters 3 and 4 was accepted for presentation at ACS 2024. A formatted version of this manuscript is available on arXiv [Kim and Sridharan, 2024].

- The code used to demonstrate and evaluate the use of the relevance score as a heuristic (Chapters 3 and 4) is available on Bitbucket:

  `https://bitbucket.org/Oliver_Kim/relevanceheuristic/`

  This software, mostly written in C++, provides extentions to LAMA [Richter and Westphal, 2010, github: LAMA,] that enable it to use h_{Ξ} in conjunction with its existing heuristics. As mentioned in Section 4.2, LAMA and therefore our code that relies on it can only be used on Ubuntu≤ 18. In order to be compatible with LAMA, we define a subclass of LAMA's heuristic that implements a variety of methods required to function. These must take arguments corresponding to states as they are represented in LAMA (state variable assignments), although we use a set of ground predicates internally. As long as the representation used by a planner can be translated into a list of ground predicates, any planner compatible with C++ should be able to integrate the code needed to calculate h_{Ξ}. Various Python and Bash scripts for running, evaluating and reporting experiments are also included, as well as code for generating landmark-free problems (Section 4.3.2).

- The code used to demonstrate and evalute the use of the relevance score for diagnosis (Chapters 3 and 5) is available on Bitbucket:

  `https://bitbucket.org/Oliver_Kim/relevancediagnosis/`

  This software, also mostly written in C++ (with additional control and reporting scripts in Python and bash), generates and solves ablated planning problems (Definition 13) in PDDL as described in Section 5.3.5. It reads and grounds PDDL files directly, allowing it to run as a stand-alone program, without relying on LAMA and the OS limitations that come with it. This separation was primarily implemented to gain control over which facts and actions are grounded, as LAMA only grounded facts and actions that are used in

delete-relaxed plans (if any exist). We ground facts and actions as they are encountered during exploration of the tree (Algorithm 3). The core code used for calculating $\Xi(l)$ is mostly unchanged from that in the other repository, and is used to calculate $\tau(\alpha)$ for $|\alpha| = 1$.

6.2 Future work

In this section, we will summarise the limitations to the relevance score, and its uses in classical planning and diagnosis. We will also consider how these might be overcome and better understood.

6.2.1 Calculating $\Xi(l)$ more efficiently

Despite our attempts to minimise and reuse as many of the calculations for the relevance score as possible, it remains a computationally expensive value to calculate. In Section 3.9 we speculate that employing a symbolic computation approach to reuse parts of the tree that are equivalent would reduce the cost of both exploring $\overline{T_\Pi}$, and using it to calculate $\Xi(l)$. We also describe how maintianing a cache of local relevance scores $\Xi(l, \underline{n})$ could reduce the number of times expensive calculations need to be performed.

6.2.2 Improving the performance of h_Ξ

While h_Ξ was shown to work as a heuristic, and to be the best choice for a certain class of classical planning problems, its performance on standard problems leaves room for improvement. In Section 4.6, we consider the likely benefits of employing a threshold to ignore low value facts when calculating h_Ξ. We hypothesise that as well as reducing the time spent calculating it, it may also help to focus the planner on the most promising paths. We also propose extracting additional information from T_Π in the form of partial orderings. These could enable the planner to separate divergent partial plans and better direct it towards a complete plan.

6.2.3 Further evaluation of h_{Ξ} for classical planning

In Section 4.6 we consider the benefits of further testing of h_{Ξ} on a wider range of planning problems, particularly those resembling real life tasks. We also recognise that it would be appropriate to test h_{Ξ} against other heuristics, and consider the code modifications necessary to do this.

6.2.4 Improving the diagnosis system

Section 5.6 identified a number of ways in which the presented diagnosis system could be improved. Supporting a wider, defined set of PDDL syntax will make it more robust, and allow its use on more problems. Components of the assumability score $\tau(\alpha)$ can be improved (the proximity of a fact to the goal) or added (how much a fact extends the reachable search space) to better represent the properties that we aim to estimate.

Performing extensive reasoning about what assumptions are reasonable in the domain is outside the scope of this thesis. However, we identified a simple way to ensure that an assumption is not made incompatible with it's problem by interfering with stated or implied negative preconditions.

6.2.5 Further evaluation of $\tau(\alpha)$ for diagnosis

As recognised in Section 5.6, the the assumability score is defined for fixes of any size $\alpha \geq 1$, but has only been tested on $\alpha = 1$. This will be addressed in future work. Since becoming aware of [Göbelbecker et al., 2010], we hope to test both on a shared dataset and compare their capabilities.

6.3 Conclusion

The relevance score successfully quantifies how relevant a piece of information (fact) is to a task. It has the useful property that, while it can take account of information in the initial state, it does not rely on it. We demonstrate that it can be applied to both classical planning, and diagnosis, extending the capabilities of both. We have presented some of this work at an international conference (ACS), and have plans for further publications including some of the future work. In a commitment to transparency and reproducability, all code has been made available.

Appendix A

Motivating domain in PDDL

```
(define (domain student)
    (:requirements :strips :types)
    (:types location object - concept)
    (:predicates
        (at ?loc - location)
        (has ?item - object)
        (connected ?loc1 ?loc2 - location)
        )
    (:constants
        Library - location
        Hardcopy Doc Paper Keycard Computer Coffee Biscuit Fun - object
        )
    (:action move
        :parameters (?from ?to - location)
        :precondition (and (at ?from) (connected ?from ?to) (not (at Library)))
        :effect (and (at ?to) (not (at ?from))))
    (:action movetolibrary
        :parameters (?from - location)
        :precondition (and (at ?from) (connected ?from Library) (has Keycard))
        :effect (and (at Library) (not (at ?from))))
```

Figure A.1: **Motivating domain in PDDL** The domain part of a PDDL representation of the planning domain outlined informally in Figure 1.1. (Continued on next page)

```
    (:action writeofficeaction
        :parameters ()
        :precondition (and (at Office) (has Coffee) (has Computer))
        :effect (and (has Doc)))
    (:action writelibaction
        :parameters ()
        :precondition (and (at Library))
        :effect (and (has Doc)))
    (:action printofficeaction
        :parameters ()
        :precondition (and (at Office) (has Doc) (has Paper))
        :effect (and (has Hardcopy)))
    (:action printlibaction
        :parameters ()
        :precondition (and (at Library) (has Doc))
        :effect (and (has Hardcopy)))
    (:action fixcompaction
        :parameters ()
        :precondition (and (at IT) (has Biscuit))
        :effect (and (has Computer)))
    (:action getcoffeeaction
        :parameters ()
        :precondition (and (at Kitchen))
        :effect (and (has Coffee)))
    (:action getpaperaction
        :parameters ()
        :precondition (and (at Storeroom))
        :effect (and (has Paper)))
    (:action playgames
        :parameters ()
        :precondition (and (has Computer))
        :effect (and (has Fun)))
)
```

Figure A.1: (Continued)

```
(define (problem demo)
(:domain student)
(:objects Storeroom IT Kitchen Office C1 C2 C3 - location)
(:init
    (connected C1 C2)
    (connected C2 C1)
    (connected C1 C3)
    (connected C3 C1)
    (connected C1 Kitchen)
    (connected Kitchen C1)
    (connected C1 Office)
    (connected Office C1)
    (connected C2 Storeroom)
    (connected Storeroom C2)
    (connected C2 Library)
    (connected Library C2)
    (connected C3 Library)
    (connected Library C3)
    (connected C3 IT)
    (connected IT C3)
    (at C1)
    (has Paper)
    (has Coffee)
    (has Computer)
    (has Keycard)
    )
(:goal (and
    (has Hardcopy)
    ))
)
```

Figure A.2: **Motivating problem in PDDL** PDDL representation of the planning problem for which plans are shown in Figure 2.1. Variations of this, with different (`has ?item`) initial conditions are used throughout the thesis.

Appendix B

Blocks domain in PDDL

```
(define (domain blocks)
  (:requirements :strips)
  (:predicates (on ?x ?y)       (ontable ?x)      (clear ?x)
               (handempty)      (holding ?x) )
  (:action pick-up
     :parameters (?x)
     :precondition (and (clear ?x)      (ontable ?x)      (handempty) )
     :effect (and (holding ?x)
       (not (ontable ?x))     (not (clear ?x))    (not (handempty)) ) )
  (:action put-down
     :parameters (?x)
     :precondition (holding ?x)
     :effect (and (clear ?x)  (handempty)    (ontable ?x)
       (not (holding ?x)) ) )
  (:action stack
     :parameters (?x ?y)
     :precondition (and (holding ?x)    (clear ?y))
     :effect (and (clear ?x)  (handempty)    (on ?x ?y)
       (not (holding ?x))     (not (clear ?y)) ) )
  (:action unstack
     :parameters (?x ?y)
     :precondition (and (on ?x ?y) (clear ?x)     (handempty) )
     :effect (and (holding ?x)     (clear ?y)
       (not (clear ?x))       (not (handempty))   (not (on ?x ?y)) ) )
)
```

Figure B.1: **Blocks domain in PDDL** Blocks domain from `https://github.com/bonetblai/hsp-planners/tree/master/hsp2-1.0/examples` [Bonet and Geffner, 2001a]. Edited to remove comments and improve layout.

```
(define (problem blocks-5-1)
(:domain blocks)
(:objects a d c e b )
(:init
    (clear b)
    (clear e)
    (clear c)
    (ontable d)
    (ontable e)
    (ontable c)
    (on b a)
    (on a d)
    (handempty)
    )
(:goal (and
    (on d c)
    (on c b)
    (on b a)
    (on a e)
    ))
)
```

Figure B.2: **Blocks problem in PDDL** Blocks problem probBLOCKS-5-1.pddl from `https://github.com/bonetblai/hsp-planners/tree/master/hsp2-1.0/examples` [Bonet and Geffner, 2001a]. Edited to remove comments and improve layout.

Appendix C

Satelite domain in PDDL

```
(define (domain satellite)
  (:requirements :equality :strips)
  (:predicates
        (on_board ?i ?s)   (supports ?i ?m)   (pointing ?s ?d)
        (power_on ?i)      (calibrated ?i)    (have_image ?d ?m)
        (satellite ?x)     (direction ?x)     (instrument ?x)
        (power_avail ?s)   (mode ?x) )        (calibration_target ?i ?d)
(:action turn_to
  :parameters ( ?s ?d_new ?d_prev)
  :precondition (and (satellite ?s)     (direction ?d_new)
        (direction ?d_prev)  (pointing ?s ?d_prev))
  :effect (and (pointing ?s ?d_new) (not (pointing ?s ?d_prev)) )
  )
(:action switch_on
  :parameters ( ?i ?s)
  :precondition (and (instrument ?i) (satellite ?s)
        (on_board ?i ?s) (power_avail ?s) )
  :effect (and (power_on ?i) (not (calibrated ?i)) (not (power_avail ?s)) )
  )
(:action switch_off
  :parameters ( ?i ?s)
  :precondition (and (instrument ?i) (satellite ?s)  (on_board ?i ?s)
        (power_on ?i) )
  :effect (and (power_avail ?s) (not (power_on ?i)) )
  )
(:action calibrate
  :parameters ( ?s ?i ?d)
  :precondition (and (satellite ?s) (instrument ?i) (direction ?d)
        (on_board ?i ?s) (calibration_target ?i ?d) (pointing ?s ?d)
        (power_on ?i) )
  :effect (calibrated ?i)
  )
(:action take_image
  :parameters ( ?s ?d ?i ?m)
  :precondition(and (satellite ?s) (direction ?d) (instrument ?i) (mode ?m)
        (calibrated ?i) (on_board ?i ?s) (supports ?i ?m) (power_on ?i)
        (pointing ?s ?d) (power_on ?i) )
  :effect (have_image ?d ?m)
  )
)
```

Figure C.1: **Satellite domain in PDDL** Satellite domain from `https://github.com/bonetblai/hsp-planners/tree/master/hsp2-1.0/examples` [Bonet and Geffner, 2001a]. Edited to remove comments and improve layout.

List of Definitions

List of Algorithms

List of Tables

List of Figures

Bibliography

[Alcazar et al., 2013] Alcazar, V., Borrajo, D., Fernandez, S., Fuentetaja, R., Alcázar, V., Borrajo, D., Fernández, S., and Fuentetaja, R. (2013). Revisiting regression in planning. IJCAI International Joint Conference on Artificial Intelligence, pages 2254–2260.

[Asai, 2019] Asai, M. (2019). Unsupervised grounding of plannable first-order logic representation from images. Proceedings International Conference on Automated Planning and Scheduling, ICAPS, pages 583–591.

[Baier et al., 2014] Baier, J. A., Mombourquette, B., and McIlraith, S. A. (2014). Diagnostic Problem Solving via Planning with Ontic and Epistemic Goals. Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning Diagnostic, pages 388–397.

[Bakker and Schmidhuber, 2004] Bakker, B. and Schmidhuber, J. (2004). Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. Proceedings of the 8-th Conference on Intelligent Autonomous Systems, pages 438–445.

[Baral et al., 2000] Baral, C., McIlraith, S., and Son, T. C. (2000). Formulating Diagnostic Reasoning Using an Action Language with Narratives and Sensing. KR, pages 311–322.

[Bender and Farach-Colton, 2000] Bender, M. A. and Farach-Colton, M. (2000). The LCA problem revisited. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 1776 LNCS:88–94.

[Bender et al., 2005] Bender, M. A., Farach-Colton, M., Pemmasani, G., Skiena, S., and Sumazin, P. (2005). Lowest common ancestors in trees and directed acyclic graphs. Journal of Algorithms, 57(2):75–94.

[Berkman and Vishkin, 1993] Berkman, O. and Vishkin, U. (1993). Recursive Star-Tree Parallel Data Structure. Society for Industrial and Applied Mathematics Journal computing, 22(2):221–242.

[Blum et al., 1997] Blum, A. L., Furst ', M. L., and Furst, M. L. (1997). Artificial Intelligence Fast planning through planning graph analysis *. Artificial Intelligence, 90(96):28–300.

[Bochman, 2007] Bochman, A. (2007). A causal theory of abduction. Journal of Logic and Computation.

[Bonet and Geffner, 2001a] Bonet, B. and Geffner, H. (2001a). Heuristic search planer 2.0. AI Magazine, 22(3):77–80.

[Bonet and Geffner, 2001b] Bonet, B. and Geffner, H. (2001b). Planning as heuristic search. Artificial Intelligence, 129(1-2):5–33.

[Bosselut et al., 2019] Bosselut, A., Rashkin, H., Sap, M., Malaviya, C., Celikyilmaz, A., and Choi, Y. (2019). COMET: Commonsense Transformers for Automatic Knowledge Graph Construction. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4762–4779.

[Brafman and Hoffmann, 2004] Brafman, R. I. and Hoffmann, J. (2004). Conformant planning via heuristic forward search: A new approach. Proceedings of the 14th International Conference on Automated Planning and Scheduling, ICAPS 2004, 170:355–364.

[Brázdil et al., 2016] Brázdil, T., Chmelík, M., Novotný, P., Chatterjee, K., and Gupta, A. (2016). Stochastic Shortest Path with Energy Constraints in POMDPs. Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), pages 1465–1466.

[Brenner, 2003] Brenner, M. (2003). A Multiagent Planning Language. International Conference on Automated Planning and Scheduling, 102(15).

[Büchner et al., 2021] Büchner, C., Keller, T., and Helmert, M. (2021). Exploiting Cyclic Dependencies in Landmark Heuristics. Proceedings International Conference on Automated Planning and Scheduling, ICAPS, 2021-Augus:65–73.

[Chopra and Zhang, 2001] Chopra, S. and Zhang, D. (2001). Postdiction problems in dynamic logic. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2256:119–129.

[Cimatti and Roveri, 2000] Cimatti, A. and Roveri, M. (2000). Forward Conformant Planning via Symbolic Model Checking. AAAI, pages 21–34.

[Cresswell et al., 2013] Cresswell, S., McCluskey, T., and West, M. (2013). Acquiring planning domain models using LOCM. The Knowledge Engineering Review, 28(2):195–213.

[Czumaj et al., 2007] Czumaj, A., Kowaluk, M., and Lingas, A. (2007). Faster algorithms for finding lowest common ancestors in directed acyclic graphs. Theoretical Computer Science, 380(1-2):37–46.

[Davis-Mendelow et al., 2013] Davis-Mendelow, S., Baier, J. A., and McIlraith, S. A. (2013). Assumption-Based Planning: Generating Plans and Explanations under Incomplete Knowledge. AAAI.

[de Kleer et al., 1992] de Kleer, J., Mackworth, A. K., and Reiter, R. (1992). Characterizing diagnoses and systems. Artificial Intelligence, 56(2-3):197–222.

[Dechter et al., 1991] Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal Constraint Networks. Artificial Intelligence, 49:61–95.

[Dechter and Pearl, 1985] Dechter, R. and Pearl, J. (1985). Generalized Best-First Search Strategies and the Optimality of A. Journal of the ACM (JACM), 32(3):505–536.

[Denecker and Kakas, 2002] Denecker, M. and Kakas, A. (2002). Abduction in Logic Programming. Computational Logic: Logic Programming and Beyond, pages 402–436.

[Djidjev et al., 1991] Djidjev, H. N., Pantziou, G. E., and Zaroliagis, C. D. (1991). Computing shortest paths and distances in planar graphs. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 510 LNCS(3075):327–338.

[Eckhardt et al., 2007] Eckhardt, S., Mühling, A. M., and Nowak, J. (2007). Fast lowest common ancestor computations in dags. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 4698 LNCS, pages 705–716.

[Edelkamp and Hoffmann, 2004] Edelkamp, S. and Hoffmann, J. (2004). PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical Report 195, University of Freiburg,.

[Elkhatib et al., 2004] Elkhatib, O., Pontelli, E., and Son, T. C. (2004). ASP-PROLOG: A System for Reasoning about Answer Set Programs in Prolog. PADL.

[Eppe et al., 2013] Eppe, M., Bhatt, M., and Dylla, F. (2013). Approximate epistemic planning with postdiction as answer-set programming. Logic Programming and Nonmonotonic Reasoning, 8148 LNAI:290–303.

[Fang et al., 2014] Fang, C., Yu, P., and Williams, B. C. (2014). Chance-Constrained Probabilistic Simple Temporal Problems. Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, pages 2264–2270.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). Strips: a New Approach To the Application of Theorem Proving To Problem Solving. Artificial Intelligence, 2:189–208.

[Fischer and Heun, 2006] Fischer, J. and Heun, V. (2006). Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In Combinatorial pattern matching, volume 4009, pages 36–48. Springer Berlin Heidelberg.

[Gelfond and Kahl, 2014] Gelfond, M. and Kahl, Y. (2014). Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press.

[Ghallab et al., 2004] Ghallab, M., Nau, D., and Traverso, P. (2004). Automated Planning: Theory and Practice. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science.

[github: hsp2,] github: hsp2. `https://github.com/bonetblai/hsp-planners/tree/master/hsp2-1.0/examples` (accessed 12/03/24).

[github: IJCAI,] github: IJCAI. `https://github.com/shirin888/planrecogasplanning-ijcai16-benchmarks/tree/master` (accessed 12/03/24).

[github: LAMA,] github: LAMA. `https://github.com/rock-planning/planning-lama.git` (accessed 15/03/24).

[github: unsolve,] github: unsolve. `https://github.com/AI-Planning/unsolve-ipc-2016/tree/master` (accessed 12/03/24).

[Göbelbecker et al., 2010] Göbelbecker, M., Keller, T., Eyerich, P., Brenner, M., and Nebel, B. (2010). Coming up with good excuses: What to do when no plan can be found. Proceedings of the 20th International Conference on Automated Planning and Scheduling, (Icaps):81–88.

[Grastien and Kelareva, 2007] Grastien, A. and Kelareva, E. (2007). Modeling and Solving Diagnosis of Discrete-Event Systems via Satisfiability. International Workshop on Principles of Diagnosis.

[Gu et al., 2016] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2016). Deep Reinforcement Learning for Robotic Manipulation. arXiv e-prints arXiv:1610.00633v1.

[Hanheide et al., 2014] Hanheide, M., Göbelbecker, M., Horn, G., Pronobis, A., Sjöö, K., Aydemir, A., Jensfelt, P., Gretton, C., Dearden, R., Janicek, M., Zender, H., Kruijff, G.-J., Hawes, N., and Wyatt, J. (2014). Dora - A Retaskable Mobile Robot that Self-understands. Artificial Intelligence.

[Hanheide et al., 2017] Hanheide, M., Göbelbecker, M., Horn, G. S., Pronobis, A., Sjöö, K., Aydemir, A., Jensfelt, P., Gretton, C., Dearden, R., Janicek, M., Zender, H., Kruijff, G.-J., Hawes, N., and Wyatt, J. L. (2017). Robot task planning and explanation in open and uncertain worlds. Artificial Intelligence, 247:119–150.

[Haslum, 2009] Haslum, P. (2009). h^m(P) = h^1(P^m): Alternative Characterisations of the Generalisation From h^{max} To h^m. Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling, pages 354–357.

[Haslum and Geffner, 2000] Haslum, P. and Geffner, H. (2000). Heuristics for Optimal Planning. Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling, pages 140–149.

[He et al., 2010] He, R., Brunskill, E., and Roy, N. (2010). PUMA: Planning under Uncertainty with Macro-Actions. AAAI.

[He et al., 2011] He, R., Brunskill, E., and Roy, N. (2011). Efficient planning under uncertainty with macro-actions. Journal of Artificial Intelligence Research, 40:523–570.

[Helmert, 2006] Helmert, M. (2006). The fast downward planning system. Journal of Artificial Intelligence Research, 26:191–246.

[Hoffmann and Brafman, 2005] Hoffmann, J. and Brafman, R. I. (2005). Contingent Planning via Heuristic Forward Search with Implicit Belief States org Hoffmann. International Conference on Automated Planning and Scheduling.

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research, 14:253–302.

[Hoffmann et al., 2004] Hoffmann, J., Porteous, J., and Sebastia, L. (2004). Ordered Landmarks in Planning. Journal of Artificial Intelligence Research, 22:215–278.

[Holler et al., 2020] Holler, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., and Alford, R. (2020). HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. Proceedings of the AAAI conference on Artificial Intelligence, 34(06):9883–9891.

[Hu et al., 2011] Hu, Y., Giacomo, G. D., Universit, S. A., and Ariosto, V. (2011). Generalized Planning : Synthesizing Plans that Work for Multiple Environments. IJCAI International Joint Conference on Artificial Intelligence, pages 918–923.

[Hyafil and Bacchus, 2003] Hyafil, N. and Bacchus, F. (2003). Conformant Probabilistic Planning via CSPs. ICAPS, pages 205–214.

[Kaelbling et al., 1998] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101:99–134.

[Kambhampati, 2024] Kambhampati, S. (2024). Can large language models reason and plan? Annals of the New York Academy of Sciences, 1534(1):15–18.

[Karpas and Domshlak, 2009] Karpas, E. and Domshlak, C. (2009). Cost-optimal planning with landmarks. IJCAI International Joint Conference on Artificial Intelligence, pages 1728–1733.

[Katz et al., 2017] Katz, M., Moshkovich, D., Lipovetzky, N., and Tüisov, A. (2017). Adapting novelty to classical planning as heuristic search. Proceedings International Conference on Automated Planning and Scheduling, ICAPS, (Icaps):172–180.

[Kearns et al., 2000] Kearns, M., Mansour, Y., and Ng, A. (2000). Approximate planning in Large POMDPs via reusable trajectories. NIPS.

[Keyder et al., 2010] Keyder, E., Richter, S., and Helmert, M. (2010). Sound and complete landmarks for And/Or graphs. ECAI, pages 335—-340.

[Kim and Sridharan, 2024] Kim, O. and Sridharan, M. (2024). Relevance Score: A Landmark-Like Heuristic for Planning. arXiv preprint arXiv:2403.07510v1.

[Kovacs, 2012] Kovacs, D. L. (2012). A Multi-Agent Extension of PDDL3.1. International Conference on Automated Planning and Scheduling, pages 19–27.

[Kurniawati et al., 2011] Kurniawati, H., Du, Y., Hsu, D., and Lee, W. S. (2011). Motion planning under uncertainty for robotic tasks with long time horizons. IJRR, 30(3):308–323.

[Kurniawati et al., 2008] Kurniawati, H., Hsu, D., and Lee, W. S. (2008). SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. Proceedings of Robotics: Science and Systems.

[Lacerda et al., 2015] Lacerda, B., Parker, D., and Hawes, N. (2015). Optimal policy generation for partially satisfiable co-safe LTL specifications. IJCAI International Joint Conference on Artificial Intelligence, 2015-Janua:1587–1593.

[Langley, 2017] Langley, P. (2017). Heuristics and Cognitive Systems. Advances in Cognitive Systems, 5:3–12.

[Li et al., 2024] Li, Q., Cui, L., Zhao, X., Kong, L., and Bi, W. (2024). GSM-Plus: A Comprehensive Benchmark for Evaluating the Robustness of LLMs as Mathematical Problem Solvers. Proceedings ofthe 62nd Annual Meeting ofthe Association for Computational Linguistics, 1:2961–2984.

[Li and Epstein, 2010] Li, X. and Epstein, S. L. (2010). Learning cluster-based structure to solve constraint satisfaction problems. Annals of Mathematics and Artificial Intelligence, 60(1):91–117.

[Lindsay et al., 2017] Lindsay, A., Read, J., Ferreira, J. F., Hayton, T., Porteous, J., and Gregory, P. (2017). Framer: Planning models from natural language action descriptions. AAAI.

[Lipovetzky and Geffner, 2012] Lipovetzky, N. and Geffner, H. (2012). Width and serialization of classical planning problems. Frontiers in Artificial Intelligence and Applications, 242:540–545.

[Lipovetzky and Geffner, 2017a] Lipovetzky, N. and Geffner, H. (2017a). A polynomial planning algorithm that beats lama and FF. Proceedings International Conference on Automated Planning and Scheduling, ICAPS, (Icaps):195–199.

[Lipovetzky and Geffner, 2017b] Lipovetzky, N. and Geffner, H. (2017b). Best-First Width Search : Exploration and Exploitation in Classical Planning. Proceedings of the 31th Conference on Artificial Intelligence (AAAI 2017).

[Long and Gerevini, 2006] Long, D. and Gerevini, A. (2006). Plan constraints and preferences in PDDL3. International Conference on Automated Planning and Scheduling, pages 1–12.

[Ma and Pineau, 2015] Ma, H. and Pineau, J. (2015). Information Gathering and Reward Exploitation of Subgoals for POMDPs. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pages 3320–3326.

[Madabushi and Lee, 2016] Madabushi, H. T. and Lee, M. (2016). High Accuracy Rule-based Question Classification using Question Syntax and Semantics. Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 1220–1230.

[Maliah et al., 2018] Maliah, S., Shani, G., and Brafman, R. I. (2018). Landmark-based heuristic online contingent planning. Autonomous Agents and Multi-Agent Systems, 32(5):602–634.

[Mansouri and Pecora, 2016] Mansouri, M. and Pecora, F. (2016). A robot sets a table: a case for hybrid reasoning with different types of knowledge. Journal of Experimental and Theoretical Artificial Intelligence, 28:801–821.

[Maria Fox and Derek Long, 2003] Maria Fox and Derek Long (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research, 20:61–124.

[McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. AIPS.

[McIlraith, 1998] McIlraith, S. A. (1998). Logic based abductive inference. Technical report, Knowledge Systems Laboratory Stanford.

[Mirzadeh et al., 2024] Mirzadeh, I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., and Farajtabar, M. (2024). GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. arXiv preprint arXiv:2410.05229.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. NIPS Deep Learning Workshop.

[Mundhenk, 2000] Mundhenk, M. (2000). The Complexity of Planning with Partially-Observable Markov Decision Processes. Technical report.

[Parr and Russell, 1998] Parr, R. and Russell, S. (1998). Reinforcement learning with hierarchies of machines. Advances in neural information processing systems, pages 1043–1049.

[Pearl, 2002] Pearl, J. (2002). Reasoning with Cause and Effect. IJCAI, pages 95–111.

[Pereira, 2020] Pereira, R. F. (2020). Goal Recognition over Imperfect Domain Models. PhD thesis, PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL.

[Pereira et al., 2017] Pereira, R. F., Oren, N., and Meneguzzi, F. (2017). Landmark-Based Heuristics for Goal Recognition. Proceedings of the 31th Conference on Artificial Intelligence, pages 3622–3628.

[Pereira et al., 2020] Pereira, R. F., Oren, N., and Meneguzzi, F. (2020). Landmark-based approaches for goal recognition as planning. Artificial Intelligence, 279.

[Pineau et al., 2003] Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. IJCAI, pages 1025–1030.

[Pohl, 1970] Pohl, I. (1970). Heuristic search viewed as path finding in a graph. Artificial Intelligence, 1(3-4):193–204.

[Pollock, 1998] Pollock, J. L. (1998). The logical foundations of goal-regression planning in autonomous agents. Artificial Intelligence, 106(2):267–334.

[Porteous et al., 2001] Porteous, J., Sebastia, L., and Hoffmann, J. (2001). On the Extraction, Ordering, and Usage of Landmarks in Planning. Proceedings of the Sixth European Conference on Planning.

[Raina et al., 2005] Raina, R., Ng, A., and Manning, C. D. (2005). Robust Textual Inference Via Learning and Abductive Reasoning. AAAI, pages 1099–1105.

[Rao et al., 2016] Rao, D., Jiang, Z., Wen, Y., and Li, J. (2016). Performance of the RDDL planners. 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS), pages 115–118.

[Regneri et al., 2010] Regneri, M., Koller, A., and Pinkal, M. (2010). Learning Script Knowledge with Web Experiments. Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 979–988.

[Reiter, 1987] Reiter, R. (1987). A theory of diagnosis from first principles. Artificial Intelligence, (1987):57–95.

[Rescorla and Wagner, 1972] Rescorla, R. A. and Wagner, A. R. (1972). A theory of Pavlovian conditioning. In Black, A. H. and Prokasy, W. F., editors, Classical conditioning II current research and theory, chapter 3, pages 64–99. Appleton-Century-Crofts.

[Richter et al., 2008] Richter, S., Helmert, M., and Westphal, M. (2008). Landmarks revisited. Proceedings of the National Conference on Artificial Intelligence, 2:975–982.

[Richter and Westphal, 2010] Richter, S. and Westphal, M. (2010). The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. Journal of Artificial Intelligence Research, 39:127–177.

[Rodler, 2023] Rodler, P. (2023). How Should I Compute My Candidates? A Taxonomy and Classification of Diagnosis Computation Algorithms. Frontiers in Artificial Intelligence and Applications, 372:1986–1993.

[Russell and Norvig, 2016] Russell, S. and Norvig, P. (2016). Artificial Intelligence: A Modern Approach. Always learning. Pearson.

[Sacerdoti, 1974] Sacerdoti, E. D. (1974). Hillclimbing in a Hierarchy of Abstraction Spaces. Artificial Intelligence, (5):115–135.

[Sampath et al., 1995] Sampath, M., Sinnamohideen, K., Lafortune, S., and Teneketzis, D. (1995). Diagnosability of Discrete-Event Systems. IEEE Transactions on Automatic Control, 40(9):1555–1575.

[Schwering et al., 2015] Schwering, C., Lakemeyer, G., and Pagnucco, M. (2015). Belief revision and progression of knowledge bases in the epistemic situation calculus. IJCAI.

[Segovia-Aguas et al., 2019] Segovia-Aguas, J., Jiménez, S., and Jonsson, A. (2019). Computing programs for generalized planning using a classical planner. Artificial Intelligence, 272:52–85.

[Segovia-Aguas et al., 2022] Segovia-Aguas, J., Jiménez Celorrio, S., Sebastiá, L., and Jonsson, A. (2022). Scaling-Up Generalized Planning as Heuristic Search with Landmarks. Proceedings of the International Symposium on Combinatorial Search, 15(1):171–179.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587):484–489.

[Smith and Simmons, 2005] Smith, T. and Simmons, R. (2005). Point-Based POMDP Algorithms : Improved Analysis and Implementation. Uncertainty in Artificial Intelligence, pages 542–549.

[Sohrabi et al., 2010] Sohrabi, S., Baier, J. A., and McIlraith, S. A. (2010). Diagnosis as Planning Revisited. Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning, pages 26–36.

[Speer and Havasi, 2012] Speer, R. and Havasi, C. (2012). Representing General Relational Knowledge in ConceptNet 5. Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), pages 3679—-3686.

[Sridharan et al., 2019] Sridharan, M., Gelfond, M., Zhang, S., and Wyatt, J. (2019). REBA : A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. Journal of Artificial Intelligence Research, 65:87–180.

[Sridharan et al., 2017] Sridharan, M., Meadows, B., and Gomez, R. (2017). What can I not do? Towards an Architecture for Reasoning about and Learning Affordances. International Conference on Automated Planning and Scheduling, pages 461–469.

[Srivastava et al., 2011] Srivastava, S., Immerman, N., and Zilberstein, S. (2011). A new representation and associated algorithms for generalized planning. Artificial Intelligence, 175:615–647.

[Stone, 1998] Stone, M. (1998). Abductive planning with sensing. AAAI.

[Sutton and Barto, 1987] Sutton, R. S. and Barto, A. G. (1987). A temporal-difference model of classical conditioning.

[Teichteil-Königsbuch et al., 2010] Teichteil-Königsbuch, F., Kuter, U., and Infantes, G. (2010). Incremental plan aggregation for generating policies in MDPs. Autonomous Agents and Multiagent Systems, pages 1231–1238.

[Tesauro, 1995] Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. Commun. ACM, 38(3):58–68.

[Tsamardinos, 2002] Tsamardinos, I. (2002). A probabilistic approach to robust execution of temporal plans with uncertainty. Methods and Applications of Artificial Intelligence, pages 751–751.

[Valmeekam et al., 2023] Valmeekam, K., Olmo, A., Marquez, M., Sreedharan, S., and Kambhampati, S. (2023). PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. Advances in Neural Information Processing Systems, 37(NeurIPS 2023).

[Vered et al., 2018] Vered, M., Pereira, R. F., Magnaguagno, M. C., Kaminka, G. A., and Meneguzzi, F. (2018). Towards Online Goal Recognition Combining Goal Mirroring and Landmarks. International Conference on Autonomous Agents and Multiagent Systems, pages 2112–2114.

[Vidal and Fargier, 1999] Vidal, T. and Fargier, H. (1999). Handling contingency in temporal constraint networks: from consistency to controllabilities. Journal of Experimental and Theoretical Artificial Intelligence, 11(1):23–45.

[Wang and Williams, 2015] Wang, A. J. and Williams, B. C. (2015). Chance-constrained Scheduling via Conflict-directed Risk Allocation. Proceedings of the 29th Conference on Artificial Intelligence (AAAI 2015), pages 3620–3627.

[Wang et al., 2014] Wang, Z., Wang, Z., Moll, M., Huang, P.-s., Grady, D., Nasrabadi, N., Huang, T., Kavraki, L., and Hasegawa-johnson, M. (2014). Active Planning , Sensing and Recognition Using a Resource-Constrained Discriminant POMDP. IEEE Conference on Computer Vision and Pattern Recognition Workshops Active.

[website: planrec,] website: planrec. `http://www.planrec.org/Resources.html` (accessed 12/03/24).

[website: Process mining,] website: Process mining. `https://melbourne.figshare.com/articles/dataset/Process_mining-based_goal_recognition_a_running_example_in_an_11_by_11_grid/21749570` (accessed 12/03/24).

[website: unsolve,] website: unsolve. `https://unsolve-ipc.eng.unimelb.edu.au/` (accessed 12/03/24).

[Wichlacz et al., 2022] Wichlacz, J., Höller, D., and Hoffmann, J. (2022). Landmark Heuristics for Lifted Classical Planning. IJCAI International Joint Conference on Artificial Intelligence, pages 4665–4671.

[Xing et al., 2006] Xing, Z., Chen, Y., and Zhang, W. (2006). MaxPlan: Optimal Planning by Decomposed Satisfiability and Backward Reduction. Proc. Fifth Int'l Planning Competition, Int'l Conf. on Automated Planning and Scheduling (ICAPS'06).

[Younes and Littman, 2004] Younes, H. and Littman, M. (2004). PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical report, Carnegie Mellon University.

[Zhang et al., 2014] Zhang, S., Sridharan, M., Gelfond, M., and Wyatt, J. (2014). Towards an Architecture for Knowledge Representation and Reasoning in Robotics. Social Robotics, 8755:400–410.

[Zhou et al., 2023] Zhou, H., Ouyang, D., Tian, X., and Zhang, L. (2023). DiagDO: an efficient model based diagnosis approach with multiple observations. Frontiers of Computer Science, 17(6).

[Zhu and Givan, 2003] Zhu, L. and Givan, R. (2003). Landmark Extraction via Planning Graph Propagation. ICAPS Doctoral Consortium, pages 156—-160.