Advances in Cognitive Systems 11th (2024) X-X Submitted April/2024; published June/2024

Relevance Score: A Landmark-Like Heuristic for Planning

Oliver Kim OLIVER.KIM@CANTAB.NET
University of Birmingham, UK

Mohan Sridharan M.SRIDHARAN@ED.AC.UK
University of Edinburgh, UK

Abstract

Humans use heuristics to identify and use information relevant to the tasks at hand. For a planning problem, the notion of landmarks refers to facts or actions that must appear in all solutions of the problem. They are identified by analysing a reduced version of the problem. Many methods have been developed to identify such landmarks and to use them to heuristically guide the search for a solution to the planning problem. However, non-trivial landmarks may not exist in many practical problems. In this paper, we define a *relevance score* to identify facts or actions that appear in most but not all plans to achieve any given goal. We describe an approach to compute and use this relevance score in the search for a plan. We experimentally compare the performance of our approach with 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.

1. Introduction

The use of heuristics to guide search is a key feature of cognitive systems research (Langley (2017)), and computer science more broadly (Müller-Merbach (1981)). Heuristics are particularly useful in task planning (Shleyfman et al. (2015)), due to its combinatorial nature. The concepts of relevance, attention, or saliency have also been studied extensively in neuroscience. Research into attention aims to understand how the context of a task or situation filters the sensory inputs of an animal (Treisman & Gelade (1980)), and has inspired computer vision applications (Mnih et al. (2014)). There is also evidence that attention applies to multiple layers of the decision making process beyond sensory filtering (Grossberg (2019)). Applying attention to a planning problem allows appropriate techniques to be applied to different parts of the problem (Sridharan et al. (2019)). Saliency is a closely related concept that identifies parts of an input likely to contain useful information (Kayser et al. (2005); Walther & Koch (2006)).

One successful heuristic for task planning is a count of the *landmarks* that remain to be reached from a given state (Zhu & Givan (2003); Hoffmann et al. (2004); Richter & Westphal (2010); Keyder et al. (2010)). A landmark is a fact, action, or logical formula over these, that must be present in all valid solutions of the problem. They are identified from a reduced version of the planning prob-

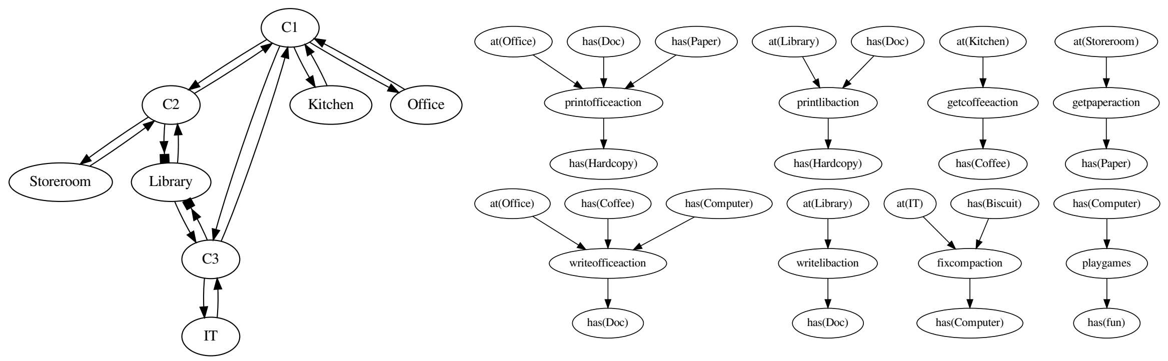

Figure 1: *Example planning problem*: An example planning domain in which a student has to submit their assignment. Facts F and move-actions A are implied by the description. Left: *Example map* connections between rooms or corridors ($C[1, 2, 3]$) are shown as arrows, representing a move action with 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 $Library$ provides keycard access (square at end of arrows). The corresponding $move(?from, Library)$ actions have the additional precondition $has(Keycard)$. Right: *Example actions* each directed graph depicts an action (middle row), with preconditions (top row) and effects (bottom row) in a model planning problem.

lem, such as the delete relaxation. This leads to a preference for actions (in plans) that either are a landmark, or achieve one. While complex logical formulae may serve as landmarks, it can be challenging to compute and use them in a heuristic. Additionally, the task of computing all landmarks for a planning problem is known to be PSPACE-complete (Porteous et al. (2001)). Systems that use landmarks thus identify a subset of single fact landmarks, but the cost of computing landmarks is still significant.

Figure 1 shows an example planning problem in which a student has to overcome obstacles to print and submit their assignment, e.g., their office computer is unreliable, their printer frequently runs out of paper, the environment is distracting, and IT services will provide support only if they are given their favorite snack! Figure 2 depicts two plans applicable to an initial state in this domain. A known limitation of methods that use landmarks to guide planning is that such landmarks must exist for the corresponding planning problem. In many complex domains with multiple routes to the goal, the only simple (single-fact) landmarks may be *trivial*, i.e., those in the goal or initial state. Other plans exist in the example domain, e.g. nothing prevents the student from wasting time before completing the assignment. Other than trivial landmarks, the only fact true in both plans in Figure 2 is $has(Doc)$. This is required by all actions ($printlibaction$, $printofficeaction$) that can achieve the goal fact $has(Hardcopy)$, and so is a landmark. Suppose $has(Computer)$ were removed from the initial state. The only action that could achieve it is $fixcompaction$, which has preconditions $[at(IT), has(Biscuit)]$. $at(IT)$ is reachable, but no action achieves $has(Biscuit)$. This makes the bottom strategy from Figure 2 inapplicable. Removing a fact from the initial state thus had the unintuitive effect of creating new landmarks. Because all plans must now follow the top strategy, they must achieve $at(library)$ at some point. Our approach avoids this limitation, and considers $at(library)$ to be relevant for either initial state. It does so using a *relevance score*, which considers the value of achieving facts (or actions) that appear in many but not necessarily all plans.

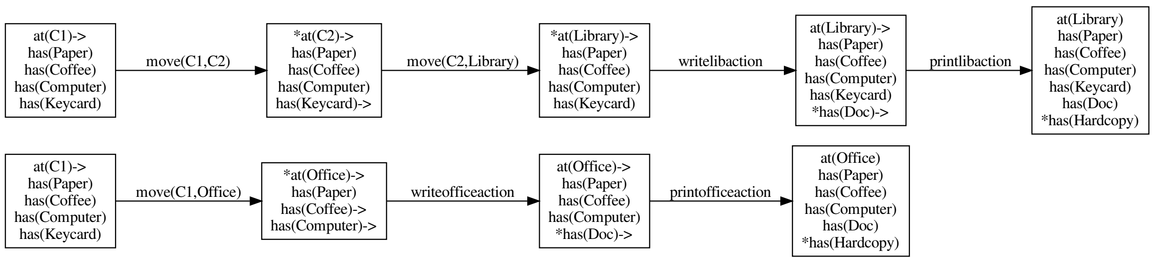

Figure 2: *Example plans to illustrate landmarks*: Two plans applicable to the same initial state (far left). Boxes show states, defined by the facts that are true in them. Actions are represented by the arrows between states. In each new state, facts that were made true by the preceding action are preceded with a *. Facts that are preconditions of the next action are appended with $->$.

The **key contribution** of this paper is to define and describe an approach for computing a novel *Relevance Score* heuristic (h_{Ξ}). This heuristic evaluates facts or actions according to how often they appear in plans under *delete relaxation*, a reduced version of the planning problem (defined later). As the baseline planner, we use LAMA (Richter & Westphal (2010)), which uses a combination of landmark counting and the Fast Forward heuristic (Hoffmann & Nebel (2001)). It has been considered state of the art at planning competitions for over a decade. For ease of comparison with the baseline LAMA planner, we limit our focus to facts as the *landmark-like* entities. We use individual and combinations of benchmark planning problems to demonstrate experimentally that 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.

2. Related Work

Landmarks were originally (Porteous et al. (2001); Zhu & Givan (2003); Hoffmann et al. (2004)) found using a relaxed planning graph (RPG Hoffmann & Nebel (2001)), which performs reachability analysis under delete relaxation conditions, and checks that the landmarks found and their order are sound. Since then, other ways to compute landmarks have been explored. One example is the translation of Relaxed Planning Graphs (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 of the original planning problem or by compiling groups of facts into a domain description that maintains the information of negative preconditions and effects without containing those features. Landmarks have also been used in several fields related to task planning, such as goal recognition (Pereira et al. (2020)) and contingent planning (Maliah et al. (2018)).

3. Definitions and Background Knowledge

We will now formally define the concepts and notation necessary to define the relevance score.

3.1 Classical Planning Problem

Classical task planning is the problem of finding a sequence of actions that go from an initial state to a state that satisfies some goals. We use notation based on Keyder et al. (2010).

Definition 1 (Classical planning problem, Π)**.** A classical planning problem is defined by the tuple:

$$\Pi = \langle F, A, I, G \rangle \tag{1}$$

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 time. 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: $Applicable(a, \sigma) \iff \sigma \models pre^+(a) \cap \neg pre^-(a)$. When an action is applied to a state in which it is applicable, the resultant state is given by: $Result(a, \sigma) \models \sigma \cup eff^+(a) \setminus eff^-(a)$. 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 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., $Result(\pi^\Pi, I) \models G$.

3.2 Delete Relaxation

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

$$\Pi = \langle F, A, I, G \rangle \rightarrow \Pi^+ = \langle F, A', I, G \rangle \tag{2}$$

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

3.3 Landmarks

Definition 3 (Landmark)**.** A landmark is a propositional formula of facts that is true at some point in the execution of all valid plans. Here, we only make use of single fact landmarks.

Definition 4 (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 5 (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*.

Definition 6 (Landmark counting heuristic, h_{LC})**.** The landmark counting heuristic (Richter & 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:

$$h_{LC}(\sigma_i) = |LC(\sigma_i)| \tag{3}$$

$$LC(\sigma_i) = (LC(\sigma_{i-1}) \setminus \mathit{Accepted}(\sigma_i)) \cup \mathit{RequiredAgain}(\sigma_i) \tag{4}$$

3.4 Backtracking Tree

A tree is a standard representation of a planning problem. A tree consists of nodes $\underline{n}$ and edges e. A node $\underline{n} = \langle l, E\rangle$ consists of a label $\mathit{label}(\underline{n}) = l \in F \cup A$, 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)).

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

Definition 7 (Tree: Backtracking tree, T_{Π})**.** A delete-relaxed planning problem Π, defines a tree T_{Π}. Node $\underline{root}$ is the root of T_{Π} and has $\mathit{label}(\underline{root}) = \mathit{success}$. An action node $\underline{a}$, has children whose labels are its preconditions $\{\mathit{label}(\underline{c}) \forall \underline{c} \in \mathit{children}(\underline{a})\} = pre(\mathit{label}(\underline{a}))$. A fact node $\underline{f}$ has children whose labels are actions a such that $\mathit{label}(\underline{f}) \in eff(a)$.

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

$$L(l) = \{\forall \underline{n} : \mathit{label}(\underline{n}) = l\} \tag{5}$$

Definition 8 (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:

$$T_{\Pi}^{path}(\underline{n}) = [\underline{n}, parent(\underline{n}), ..., \underline{root}] \tag{6}$$

Definition 9 (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:

$$T_{\Pi}^{desc}(\underline{n}) = \{\forall \underline{d} : \underline{n} \in T_{\Pi}^{path}(\underline{d})\} \tag{7}$$

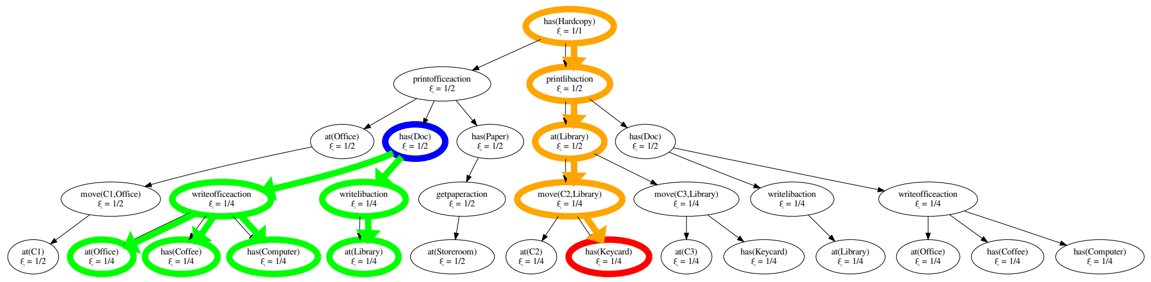

Figure 3: *Example problem as a tree*: The top 5 layers (3 fact, 2 action) of $\overline{T_\Pi}$ are shown. The **path** of the **red** node, and **descendents** of the **blue** node are marked in the colours indicated. Choices counter values $\xi(\underline{n})$ are shown for each node (described in Section 3.7).

Figure 3 illustrates these concepts on the example we introduced earlier. 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 $T_\Pi^{path}(\underline{f})$. This prevents cycles, which would represent unachievable requirements.

As with the search space for planning domains, T_Π can be very large, so we partially explore it to yield $\overline{T_\Pi}$. We select a node $\underline{n}$ from a frontier with probability proportional to $\xi(\underline{n})$ before recursively adding a child (chosen uniformly from $children(\underline{n})$) until a node is reached with no children. Siblings of nodes explored in this way are also added to the frontier. This is procedure is repeated until $\frac{\sum_{\forall \underline{n} \in \texttt{frontier}} \xi(\underline{n})}{\sum_{\forall \underline{n} \in \overline{T_\Pi}} \xi(\underline{n})} \leq \rho = 0.2$. Values of $\Xi(l)$ calculated on $\overline{T_\Pi}$, 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 $\underline{root}$, causing this lower bound to converge quickly upwards as the region of the tree close to the $\underline{root}$ is explored.

3.5 Non-Deterministic Agent (NDA)

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 hypothetical **n**on-**d**eterministic **a**gent (NDA) to define this distribution.

Definition 10 (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 by an NDA according to Algorithm 1.

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 NDA 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 (if any exist), and so are landmarks.

3.6 Lowest Common Ancestors (LCAs)

Traversing the whole of a tree with a large number of nodes is costly. We will describe methods to calculate the relevance score by visiting a small subset of nodes that can be identified once, and

Algorithm 1 - An NDA 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 that achieves the required fact. Lines 10 - 14 require all preconditions of an action.

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

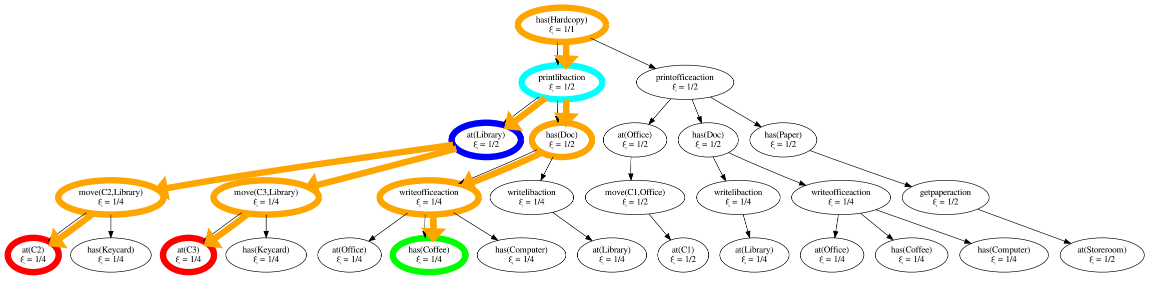

Figure 4: *Examples of LCAs*: The top 5 layers (3 fact, 2 action) of $\overline{T_\Pi}$ are shown. The **paths** of the two **red** nodes meet at the **blue** node, which is a fact. The **paths** of the **blue** node and the **green** node meet at the **cyan** node, which is an action. This means that 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 left-most of which has no $aLCAs$, and its $fLCAs$ are the two **red** nodes. The right-most of which has no aLCAs, and just the **green** node as an $fLCA$.

reused each time a relevance score is calculated. This will rely on the identification of nodes at which paths diverge.

Definition 11 (**L**owest **C**ommon **A**ncestor (LCA))**.** The LCA of two nodes $\underline{n}$ and $\underline{m}$ is the lowest node in the tree which is an ancestor of both nodes. This computation is associative and can

Algorithm 2 - Find aLCAs Given a partially explored tree $\overline{T_{\Pi}}$, and a list of nodes in that tree with a given label $TargetNodes = L(l)$, 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 $\underline{root}$ of the tree. The implementation details of registering $aLCAs$ and $fLCAs$ is ommited for clarity.

```
1: Let $ListOfPaths = sorted(TargetNodes)$
2: Let $j = 0$
3: while Some nodes unregistered do
4:   for all $i \in [0 : len(ListOfPaths) - 1]$ do
5:     if $ListOfPaths[i, j] \neq ListOfPaths[i + 1, j]$ 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
```

generalize to any number of nodes.

$$LCA(\underline{n}, \underline{m}) = \underset{\underline{i} \in T_{\Pi}^{path}(\underline{n}) \cap T_{\Pi}^{path}(\underline{m})}{argmax} (|T_{\Pi}^{path}(\underline{i})|) \tag{8}$$

The backtracking tree consists of two types of node, those with a fact label, and those with an action label. Because the NDA 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 4 illustrates the concept of LCAs, and identifies the $aLCAs$ and $fLCAs$ for the red and green nodes.

Definition 12 (**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 13 (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 $\underline{f}$:

$$fLCAs(K) = \left\{ \forall \underline{n} \in K : LCA(\underline{l}_i, \underline{l}_j) \in F \cap T_{\Pi}^{desc}(\underline{f}) \right\} \tag{9}$$

Much research has gone into finding the LCA of a pair of nodes in a tree. The problem has been translated into that of calculating Range Minimum Querys (Fischer & Heun (2006)). The complexity of this method is $O(h)$ 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 LCAs 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. We first sort the nodes by their paths (with complexity $O(hn \log(n))$). Traversing along the sorted paths (worst case complexity $O(hn)$), checking for differences between nodes in paths (and whether they are action or fact nodes) that are adjacent in the sort order yields an ordered list of action LCAs ($aLCAs$), and sets of the $fLCAs$ and $aLCAs$ that are direct descendents of the $\underline{root}$. Sets of $aLCAs$ are computed once for each fact, and then filtered by σ when $h_\Xi(\sigma)$ needs to be calculated.

3.7 Choices Counter

Each time the NDA reaches a fact node, it makes a choice as to which branch to explore. If multiple choices are available, then each branch represents a possible, but distinct partial plan. The probability of a node being sampled by the NDA can be found by tracking how many choices must be made in a certain way to reach it from the goal.

Definition 14 (Choices counter, $\xi(\underline{n})$)**.** Let the choices counter $\xi(\underline{n})$ be the probability of a node $\underline{n}$ being sampled by the NDA:

$$\xi(\underline{n}) = P(\underline{n} \in S_\Pi) \tag{10}$$

$\xi(\underline{n})$ 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 NDA has no choices to make, so ξ is passed down unchanged: $\xi(\underline{f}) = \xi(parent(\underline{f}))$. The NDA chooses one action that could supply a fact from the set of actions that make up $children(f)$:

$$\xi(\underline{a}) = \frac{\xi(parent(\underline{a}))}{|children(parent(\underline{a}))|} \tag{11}$$

The choices counter is shown on the faces of nodes in Figures 3 and 4.

3.8 Relevance Score

This section defines a score that describes how relevant a fact or action is to a goal.

Definition 15 (Relevance score, $\Xi(l)$)**.** Let the <u>relevance score</u> $\Xi(l)$ represent the probability that a node with label l would be sampled by the NDA:

$$\Xi(l) = P\left(\exists \underline{l} \in S_\Pi : label(\underline{l}) = l\right) \tag{12}$$

Definition 16 (Local relevance score, $\Xi(l, \underline{n})$)**.** Let the <u>local relevance score</u> $\Xi(l, \underline{n})$ represent the probability that a node with label l would be sampled by the NDA, given that node $\underline{n}$ (and thus $S_\Pi^{path}(\underline{n})$) has been sampled.

$$\Xi(l, \underline{n}) = P\left(\exists \underline{l} \in S_\Pi^{desc}(\underline{n}) : label(\underline{l}) = l | \underline{n} \in S_\Pi\right) \tag{13}$$

Definition 17 (State aware relevance score $\Xi_\sigma(l)$, $\Xi_\sigma(l, \underline{n})$)**.** Facts in a state are true and do not need to be achieved by planner. The state aware relevance score $\Xi_\sigma(l)$ does not consider parts of $\overline{T_\Pi}$ that achieve facts in σ .The relevance score applied to a state $\Xi_\sigma(l)$ represents the probability that a node with label l is sampled by the NDA if it stops at nodes that are true in state σ. Calculations that can be performed on $\overline{T_\Pi}$ can be made state aware by performing them on $\overline{T_\Pi/\sigma}$ instead:

$$\overline{T_\Pi/\sigma} = \overline{T_\Pi} \Bigg/ \bigcup_{\substack{\forall \underline{f} \in L(f) \\ \forall f \in \sigma}} T_\Pi^{desc}(\underline{f}) \tag{14}$$

Figure 3 illustrates the effect of this, where the green nodes are the descendents of the blue node, and so would be excluded from calculations performed on a state consisting of the blue node.

3.9 Relevance Score Heuristic

Because of the conceptual similarity between them, 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.

Definition 18 (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:

$$h_\Xi(\sigma) = \sum_{l \in F} \Xi_\sigma(l) \tag{15}$$

4. Calculating the Relevance Score

Any nodes with no descendents with $label(\underline{n}) = l$ can be ignored: $T_\Pi^{desc}(\underline{n}) \cap L(l) = \emptyset \implies \Xi(l, \underline{n}) = 0$. If a node has label l, then the NDA has already sampled such a node: $label(\underline{n}) = l \implies \Xi(l, \underline{n}) = 1$. Equation 13 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: $\Xi(l, \underline{root}) = \Xi(l)$. The NDA chooses one action to achieve a fact (ie samples one child of a fact node) with uniform probability:

$$\texttt{if } label(n) \in F \implies \Xi(l, \underline{n}) = P\left(\underline{c} \in S_\Pi^{desc}(\underline{n})\right) \times \Xi(l, \underline{c}) \tag{16}$$

$$= \sum_{\underline{c} \in children(\underline{n})} \frac{\Xi(l, \underline{c})}{|children(\underline{n})|} \tag{17}$$

The NDA requires all preconditions of an action (ie samples children of an action node), and so any descendents having $label(n) = l$ would count:

$$\texttt{if } label(n) \in A \implies \Xi(l, \underline{n}) = 1 - \prod_{\underline{c} \in children(\underline{n})} (1 - \Xi(l, \underline{c})) \tag{18}$$

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:

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

$$\Xi(l,\underline{n}) = P\left(\underline{d} \in S_\Pi^{desc}(\underline{n})\right) \times \Xi(l,\underline{d}) = P\left(\underline{d} \in S_\Pi | \underline{n} \in S_\Pi\right) \times \Xi(l,\underline{d}) = \frac{\xi(\underline{d})}{\xi(\underline{n})} \times \Xi(l,\underline{d}) \quad (19)$$

Now, consider a scenario where each child $\underline{c}$ of a fact node $\underline{f}$ has a single descendant $\underline{d}$ with $label(\underline{d}) = l$. By combining Equations 11 and 17 , we get:

$$\Xi(l,\underline{f}) = \sum_{\underline{c} \in children(\underline{f})} \frac{\Xi(l,\underline{c})}{|children(\underline{f})|} = \sum_{\underline{c} \in children(\underline{f})} \frac{\Xi(l,\underline{c})}{\frac{\xi(\underline{f})}{\xi(\underline{c})}} = \sum_{\underline{c} \in children(\underline{f})} \frac{\Xi(l,\underline{c}) \times \xi(\underline{c})}{\xi(\underline{f})} \quad (20)$$

Next, using Equation 19, this can be rewritten as:

$$\sum_{\substack{\underline{c} \in children(\underline{f}) \\ \underline{l} \models T_\Pi^{desc}(\underline{c}) \cap L(l)}} \frac{\frac{\xi(\underline{l})}{\xi(\underline{c})} \times \Xi(l,\underline{l}) \times \xi(\underline{c})}{\xi(\underline{f})} = \sum_{\underline{l} \in T_\Pi^{desc}(\underline{f}) \cap L(l)} \frac{\xi(\underline{l})}{\xi(\underline{f})} = \frac{1}{\xi(\underline{f})} \sum_{\underline{l} \in T_\Pi^{desc}(\underline{f}) \cap L(l)} \xi(\underline{l}) \quad (21)$$

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 21 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 19 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 18. These can be combined to give:

$$\Xi(l,\underline{f}) = \frac{1}{\xi(\underline{f})} \sum_{\underline{l} \in fLCAs(T_\Pi^{desc}(\underline{f}))} \xi(\underline{l}) + \sum_{\underline{a} \in aLCAs(T_\Pi^{desc}(\underline{f}))} \frac{\xi(\underline{a})}{\xi(\underline{f})} \times \Xi(l,\underline{a}) \quad (22)$$

$$= \frac{1}{\xi(\underline{f})} \left(\sum_{\underline{l} \in fLCAs(T_\Pi^{desc}(\underline{f}))} \xi(\underline{l}) + \sum_{\underline{a} \in aLCAs(T_\Pi^{desc}(\underline{f}))} \xi(\underline{a}) \times \Xi(l,\underline{a}) \right) \quad (23)$$

Equation 23 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 23 to fact nodes (starting at the root), and resolving the $aLCAs$ with Equation 18 applied to their children (which are fact nodes resolved by Equation 23 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.

5. Experimental Setup and Results

To evaluate the performance of h_Ξ as a heuristic to guide a classical planner towards a plan, a program to calculate Equation 15 was implemented in C++, using elements of the LAMA architecture (Richter & Westphal (2010)) 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.[1]

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}) and landmark counting (h_{LC}, see Definition 6) heuristics. LAMA can make use of preferred operators that encode information about the order in which landmarks need to be achieved to further guide the search for a plan. To isolate the effect of using different heuristics, we did not use preferred operators.

We experimentally evaluated two 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 planner's ability to find plans compared to the landmark counting heuristic h_{LC} in domains without non-trivial landmarks.

Each search attempt was evaluated using three measures: **M1** measured whether or not a plan was found; **M2** measured the cost of finding the plan; and **M3** measured the quality of the solution. Heuristics are used because exhaustive search of the entire space is expensive in terms of both memory and time. A practical meaning of failure to find a plan is that the available computational resources are exceeded. All experiments were performed with an 8GB RAM limit and a two-hour time limit, with failure reported if either limit was reached. When a limit was reached, it was typically the RAM limit in under one hour. The 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). Failure to find a plan makes other measures meaningless; **M2** and **M3** were assigned an infinitely high (i.e., worst possible) value.

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. As a result, the number of states expanded is used as the representative measure of search cost, rather than the number of states generated. Since the BFS search strategy prioritises finding a plan quickly without concern for plan quality, it 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 finding a solution of high quality. It does so according to the weight w in its computation of the cost assigned to a state: $c = w \times h + g$, where h is the heuristic and 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.

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

5.1 Problems Without Non-Trivial Landmarks

We evaluted **H1** on standard problems, specifically the 674 problems found in the examples folder of the HSP2 repository.[2] To evaluate **H2**, we generated new PDDL specifications for domains 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 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. A total of 500 problems were generated in this way 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 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.

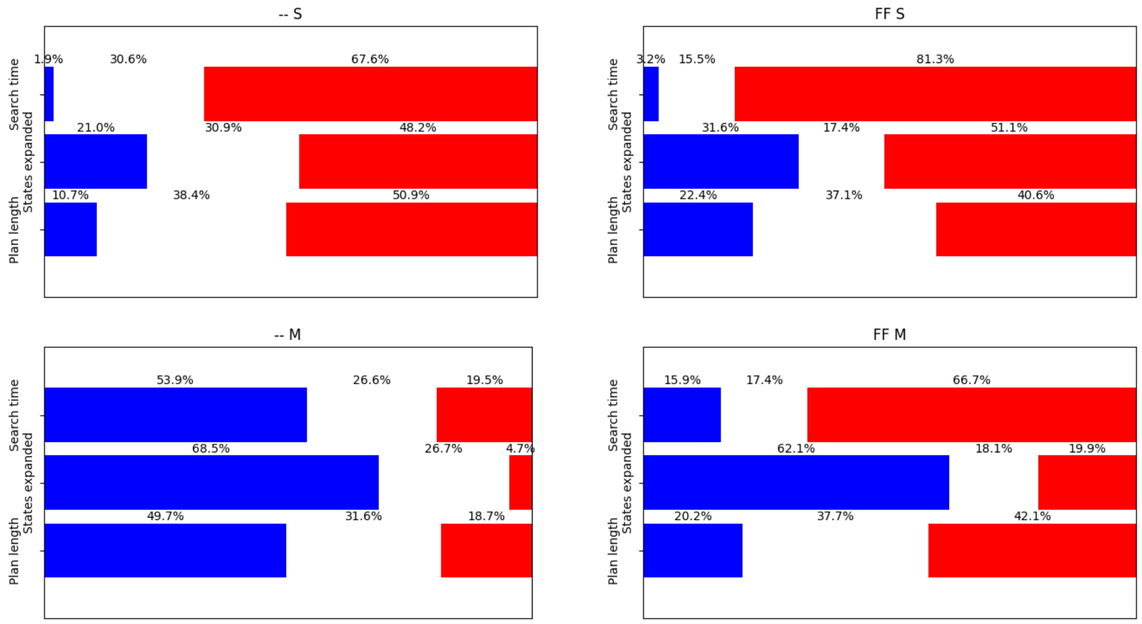

Figure 5: These plots show the fraction of trials where the performance of h_Ξ was better 🟦, equal to ⬜, or worse 🟥 than h_{LC}, according to the metrics shown on the x-axis. *Top*: Standard problems (674 problems). *Bottom*: Landmark free problems (500 problems). *Left*: Single heuristic (h_Ξ or h_{LC}). *Right*: Heuristic (h_Ξ or h_{LC}) paired with h_Ξ.

5.2 Results: Standard Problems

Table 1 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. Figure 5 (top) shows how often each heuristic did better or worse on paired standard

2. `https://github.com/bonetblai/hsp-planners/tree/master/hsp2-1.0/examples`

Table 1: Success rates of heuristics on standard problems

	Problems solved by h_{LC}	Problems solved by h_{Ξ}	Problems solved by both	Problems solved by neither
As only heuristic	70.33%	52.97%	51.48%	28.19%
Alongside h_{FF}	*84.92%	73.89%	71.71%	12.91%
All 3 combined	74.8%			

Table 2: Success rates of heuristics on landmark-free problems

	Problems solved by h_{LC}	Problems solved by h_{Ξ}	Problems solved by both	Problems solved by neither
As 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%			

problems. 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.

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}.

5.3 Results: Problems Without Non-Trivial Landmarks

Table 2 shows the success rates of each heuristic on landmark-free problems. h_{Ξ} solved far more problems than h_{LC}. Again, both heuristics solved more problems when paired with h_{FF} than alone. Even when h_{LC} was paired with h_{FF}, it solved fewer than h_{Ξ} could as the only heuristic. Figure 5 (bottom) 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 after expanding fewer states than h_{LC} most of the time.

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.

6. Conclusion

Overall, we observe that our new 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, having 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.

Acknowledgements

We would like to thank the reviewers for their detailed comments and suggestions. This work was partly funded by a Priestley scholarship awarded by the University of Birmingham and the University of Melbourne.

References

Fischer, J., & Heun, V. (2006). Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. Combinatorial Pattern Matching (pp. 36–48). Springer Berlin Heidelberg.

Grossberg, S. (2019). The resonant brain: How attentive conscious seeing regulates action sequences that interact with attentive cognitive learning, recognition, and prediction. Attention, Perception, and Psychophysics, 81, 2237–2264.

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

Hoffmann, J., Porteous, J., & Sebastia, L. (2004). Ordered Landmarks in Planning. Journal of Artificial Intelligence Research, 22, 215–278.

Kayser, C., Petkov, C. I., Lippert, M., & Logothetis, N. K. (2005). Mechanisms for allocating auditory attention: An auditory saliency map. Current Biology, 15, 1943–1947.

Keyder, E., Richter, S., & Helmert, M. (2010). Sound and complete landmarks for And/Or graphs. European Conference on Artificial Intelligence, (pp. 335—-340).

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

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

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Artificial Intelligence Planning Systems.

Mnih, V., Heess, N., Graves, A., & Kavukcuoglu, K. (2014). Recurrent Models of Visual Attention. Advances in Neural Information Processing Systems..

Müller-Merbach, H. (1981). Heuristics and their design: a survey. European Journal of Operational Research, 8, 1–23. From `https://www.sciencedirect.com/science/article/pii/0377221781900242`.

Pereira, R. F., Oren, N., & Meneguzzi, F. (2020). Landmark-based approaches for goal recognition as planning. Artificial Intelligence, 279.

Pohl, I. (1970). Heuristic search viewed as path finding in a graph. Artificial Intelligence, 1, 193–204.

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

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

Shleyfman, A., Katz, M., Helmert, M., Sievers, S., & Wehrle, M. (2015). Heuristics and Symmetries in Classical Planning. Association for the Advancement of Artificial Intelligence (AAAI), (pp. 3371–3377). From `http://http//ai.cs.unibas.ch/papers/shleyfman-et-al-aaai2015.pdf`.

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

Treisman, A. M., & Gelade, G. (1980). A Feature-Integration Theory of Attention. Cognitive Psychology, 12, 97 – 136.

Walther, D., & Koch, C. (2006). Modeling attention to salient proto-objects. Neural Networks, 19, 1395–1407.

Zhu, L., & Givan, R. (2003). Landmark Extraction via Planning Graph Propagation. International Conference on Automated Planning Systems Doctoral Consortium, (pp. 156–160).