Reproducing the concept of ordered landmarks in planning

The effect of ordered landmarks on plan length in forward search

Paul Tervoort[1]

Supervisor(s): Sebastijan Dumančić[1]**, Issa Hanou**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 28, 2024

Name of the student: Paul Benjamin Tervoort

Final project course: CSE3000 Research Project
Thesis committee: Sebastijan Dumančić, Issa Hanou, Luís Miranda da Cruz

Abstract

A lot of research has been conducted to make the task of plan generation more efficient. One idea to do so is the use of landmarks, which are sub-goals that must be true in every solution to the problem. The approximation of landmarks has a lower complexity than solving the task itself, and they can be used to guide the planner in the right direction. In previous work, ideas to order landmarks are proposed and compared to algorithms that do not use them. We verify if this comparison is fair by testing both algorithms implemented in the same language and framework. In our experiment not many problem instances finish in time, but those that do are in line with previous experiments in that on average planners using landmarks produce longer solutions than planners that do not use them.

1 Introduction

Planning algorithms aim to generate an ordered list of actions to get from the initial state to some target state. This type of algorithm is useful in applications where a significant amount of steps need to be performed to reach a goal, for example when a robotic arm has to reach certain positions [1]. The problem of planning is PSPACE-hard [2], so research in this field has to rely on heuristics to invent algorithms that give acceptable results in a realistic time frame.

One promising heuristic to reduce the running time of a planning algorithm is that of ordered landmarks [3]. A landmark of a planning problem is a fact that needs to be true at some point in all valid solutions for the problem [4]. A list of landmarks can be ordered, which means that for each landmark to be true, all other landmarks earlier in the list must have been true before reaching it. If a landmark is reached earlier than landmarks ordered before it, the planner must revert this achievement before the goal can be reached. Porteous et al. [2001] implemented a planning algorithm that exploits ordered landmarks and is compared against previous work [4], but differences in the implementation details of different algorithms can lead to a bias in the test results.

In this paper we answer the question *How do ordered landmarks affect the solution length of forward search planning in the SymbolicPlanners framework?* This question is important to answer because there are use cases where a shorter plan is more important than a shorter execution time. If this question is answered then better-informed decisions can be made about whether to use landmarks in a planner or not. To answer our research question we answer the following sub-questions:

- *Can the solution lengths for planning problems in previous work be repeated using a different implementation?*
- *How does the solution length of an implementation using ordered landmarks differ from an implementation using the same framework but without using landmarks?*

Planners using landmarks from previous work give longer solutions [4] while giving shorter plans in other work [5]. We challenge the hypothesis that the results in [4] correctly reflect the differences between the compared algorithms. Our research question is answered by verifying their findings. The alternative hypothesis is that the different results are caused by different implementations of their approach and the compared algorithm. We take the alternative hypothesis if obtaining the solution lengths from our experiment has a probability of less than 10% under the assumption that the plan lengths in [4] are correct.

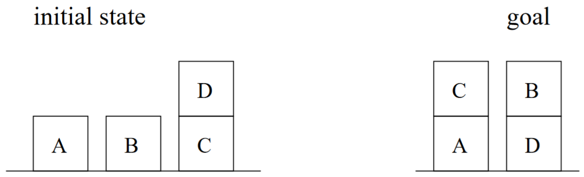

Figure 1: An example Blocksworld task [4, fig. 1].

In section 2 specific terminology for the paper is explained and section 3 highlights the most closely related previous work. Section 4 sets out the methodology for the research. The results of the experiments are listed in section 5. In section 6 we defend that the choices made during the research are ethical and justified and we reflect on the results of the research in section 7. In conclusion, we discuss open questions and future work in section 8.

2 Background

In this section, we will introduce important concepts and definitions that are used in this paper. In other sections in the paper is assumed that these are understood by the reader.

A planning domain abstractly models a type of environment. This includes possible object types, global or object-related properties and actions that can modify these properties. Figure 1 visualizes an example problem in the *Blocksworld* domain [6].

A planning problem describes a desired goal state which has to be reached starting from an initial state. The objects which exist in the problem instance are also defined. A planning domain must be connected to infer the behavior of these objects and what actions are available to reach the goal state. A planning problem is formally defined in [7]:

Definition 1. (Planning problem)
A planning problem *is a 4-tuple* $\Pi = \langle F, A, I, G \rangle$ *where*

- F *is a finite set of* propositional state variables,
- A *is a finite set of* actions, *each with associated* reconditions $pre(a) \subseteq F$, add effects $add(a) \subseteq F$, delete effects $del(a) \subseteq F$ *and* cost $cost(a) \in \mathbb{R}_0^+ + 0$,
- $I \subseteq F$ *is the* initial state, *and*
- $G \subseteq F$ *is the set of* goals.

[7, p. 335, def. 1]

PDDL [8] is a language to describe planning problems and their domains using predicates. The algorithms in this paper only accept a subset of PDDL called STRIPS [9]. Within this paper, PDDL refers to this subset.

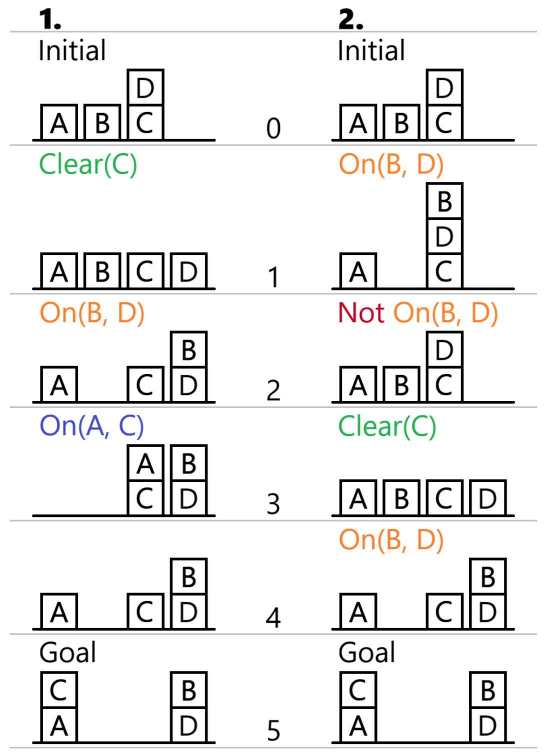

Figure 2: The states in two valid plans for the problem in Figure 1. States where one of the blocks is being held are not shown because they are not meaningful for this illustration. An interesting new predicate is shown for each step. Proposition $On(A, C)$ is not a landmark. Landmark $Clear(C)$ is reasonably ordered before landmark $On(B, D)$.

Forward search is a family of planning algorithms [10]. These algorithms model a planning problem as a graph and search this graph using the A* algorithm [11] until a node is reached which satisfies the goal of the problem. The difference in algorithms within this family is the use of different heuristics guide A* [10].

A landmark for a planning problem is a proposition that must become true in all valid solutions for the problem [4]. For the example problem in Figure 1, the proposition $Clear(C)$ is considered a landmark because C must be clear to allow it to be picked up and placed on A. All propositions that are true in the goal state are trivial landmarks [4]. Figure 2 illustrates two solutions for the problem in Figure 1 and shows where landmark $Clear(C)$ and trivial landmark $On(B, D)$ become true. Solution 2 shows that proposition $On(A, C)$ is not a landmark because this solution reaches the goal without a state in which proposition $On(A, C)$ is true.

A landmark A is reasonably ordered before a landmark B if B must be false to make A true in all plans. In Figure 2 landmark $Clear(C)$ is reasonably ordered before $On(B, D)$. In solution 2 landmark $On(B, D)$ is reached early but is made false again in step 2 in the process of reaching landmark $Clear(C)$ which adds unnecessary steps. This illustrates how ordering landmarks can help to make more efficient plans.

3 Related Work

In this section related research and software are mentioned. We explain how these are connected to our research and what value they add, or how they approach the topic differently.

We verify the experiment from [4] in our research. This paper introduces reasonable landmark orders as a type of landmark order. This paper proposes a method for not only ordering the goal terms in a problem but also landmarks that are not in the goal state. In their experiment, the new approach is often faster than their reference implementation, but the plans have more steps on average.

SymbolicPlanners is a framework that contains implementations of different planning algorithms [12]. The algorithms in this framework take as input planning problems in PDDL format and the framework contains test problems that can be used as benchmarks. SymbolicPlanners is implemented in the Julia language [13].

Different methods to use landmarks to help with planning are explored in [5]. One of these methods which uses landmarks as intermediary goals is also proposed in [4]. [5] also mentions a technique to use landmarks to compute a heuristic for a forward search planner [10]. The algorithms in this paper are implemented in the SymbolicPlanners framework [12][14], and we will use those implementations for our experiment.

Fast Downward is an implemented planner program for PDDL instances [15]. This planner is open-source and uses concepts from [4]. We use this planner for inspiration during the implementation of our landmark extraction algorithm. Fast Downward contains multiple planning algorithms and heuristics. A notable difference compared to other forward search [10] planners is that they internally represent propositions as facts that are explicitly true or false [15].

Fast-Forward is a heuristic intended for forward search planning [16]. This heuristic uses a Relaxed Planning Graph (RPG) [17] to estimate which actions are most likely to cause the goal state. A planner using Fast-Forward represents previous work for the experiment in [4]. In our research, we will also compare the experiment results to a planner which uses this heuristic. Both Fast-Forward and a planner that can use this heuristic are implemented in SymbolicPlanners [12].

Zhu and Givan (2003) extract landmarks by propagating a relaxed planning graph. Their method can extract more landmarks than the landmark exploration in [4]. Methods to extract information other than landmarks using planning graph propagation are also mentioned in this paper [18]. Some techniques for landmark extraction from this work are also found in the part of Fast Downward which we used as inspiration for our implementation.

4 Methodology

We explain the steps of our experiment. This section should provide enough details to reproduce the experiment. To answer the research question we first answer the two mentioned sub-questions.

To answer the first sub-question, we create an implementation of an algorithm that extracts ordered landmarks from a problem and uses this information for plan generation. Then we run the algorithm on the problem instances used in [4]. These are found in the *experiments* branch of Symbolic-Planners [12] and other repositories [19][20][21]. We compare the lengths of the plans generated by our algorithm with the lengths in [4], both relative to a planner using the Fast-Forward heuristic [16]. A t-test [22] will be conducted to compare both results sets. We assume that both sets are samples from the same general difference in plan length between planners with- and without using landmarks. If the probability that this assumption is true is $\leq 10\%$ we accept the alternative hypothesis. If not then the plan lengths in [4] are reproduced successfully.

If sub-question one is answered positively, the answer to the second sub-question is in line with previous work. To answer this question we compare the average solution lengths of our and previous experiments, both in general and per planning domain. For this comparison, the data obtained for answering sub-question one is used.

We implement a planning algorithm within the SymbolicPlanners framework which uses landmarks as described in [4]. Common steps in planning from the framework are reused in this implementation. This minimizes the effect of differences in implementation when the new algorithm is compared against a forward search [10] algorithm with Fast-Forward heuristic from SymbolicPlanners [16][12].

For algorithm implementation, the programming language that we used is Julia [13]. Julia is chosen because it is fast for an interpreted language. It can pre-compile parts of the code before execution. SymbolicPlanners is implemented in Julia, and to make our code compatible with this framework it helps to use the same language.

Because [4] does not contain pseudo-code for landmark extraction or subroutines used in it, we used Fast Downward [15] as inspiration for the implementation of ordered landmark extraction. This planner program implements the ideas from [4], is open-source and has a public license[1]. For using extracted landmarks in a planner we used the heuristic and planner from [14]. These implementations are developed together with our landmark extraction algorithm and they can be used together. *Our implementation* will refer to our landmark extraction implementation used together with a planner from [14].

The algorithm implementation for our experiment can be found on GitHub[2]. This branch contains the code that produced the results in this paper together with the raw experiment result files. Also, the pseudo-code for our ordered landmark extraction algorithm is in algorithm 1.

5 Experimentation Results

In this section, the results of our experiment are explained and interpreted. We explain how the data in the tables and charts are obtained from the raw experiment results.

Our planner implementation is executed with all test problems listed in [4, Fig. 5]. For each instance, landmarks are

[1]https://www.gnu.org/licenses/gpl-3.0.en.html

[2]https://github.com/PaulTervoort/SymbolicPlanners.jl-landmarks/tree/experiment-reproducing-the-concept-of-ordered-landmarks-in-planning

Algorithm 1: Ordered landmark generation

```
Data: $(F, A, I, G)$
$P \leftarrow$ build_planning_graph($F, A, I, G$);
$N \leftarrow \langle\rangle$;
$E \leftarrow \langle\rangle$;
$Q \leftarrow \langle\rangle$;
foreach $f \in I$ do
    enqueue($Q, f$);
end
while $\neg$empty($Q$) do
    $n \leftarrow$ dequeue($Q$);
    foreach $p \in$ prerequisites_in_graph($n, P$) do
        if $\neg$reach_without_prop($p, G$) then
            enqueue($Q, p$);
            insert($N, p$);
            insert($E, (p, n)$);
    end
end
foreach $n1 \in N$ do
    if true_in_state($n1, G$) then
        foreach $n2 \in N$ do
            if interferes($n1, n2$) then
                insert($E, (n1, n2)$);
        end
    else
        foreach $n2 \in$ interesting_nodes($n1$) do
            if interferes($n1, n2$) then
                insert($E, (n1, n2)$);
        end
end
Result: $(N, E)$
```

Domain	Task	[4, Fig. 5]	Table 2
Blocksworld	bw-large-a	1.33	1.50
Blocksworld	bw-large-b	0.80	1.44
Blocksworld	bw-large-c	-	1.07
Blocksworld	bw-large-d	0.86	1.83
Logistics	prob-4-0	-	1.45
Logistics	prob-6-0	-	1.00
Logistics	prob-8-0	-	1.00
Logistics	prob-10-0	-	1.02
Logistics	prob-12-0	-	1.21
Logistics	prob-38-0	1.28	-
Logistics	prob-39-0	1.20	-
Logistics	prob-40-0	1.26	-
Logistics	prob-41-0	1.25	-
Tireworld	fixit-1	1.00	1.00
Tireworld	fixit-2	-	1.07
Tireworld	fixit-3	-	1.05
Tireworld	fixit-10	1.15	-

Table 1: Relative solution lengths algorithm using landmarks compared to Fast-Forward. This is FF-v1.0+L/FF-v1.0 in [4, Fig. 5] and LM+FF/FP+FF in Table 2.

		FP+FF	FP+LM	LM+FF
Domain	Task	Steps	Steps	Steps
Blocksworld	bw-large-a	12	12	18
Blocksworld	bw-large-b	18	-	26
Blocksworld	bw-large-c	28	-	30
Blocksworld	bw-large-d	36	-	66
Grid	prob01	14	-	-
Grid	prob02	-	-	-
Grid	prob03	-	-	-
Grid	prob04	-	-	-
Grid	prob05	-	-	-
Logistics	prob-38-0	-	-	-
Logistics	prob-39-0	-	-	-
Logistics	prob-40-0	-	-	-
Logistics	prob-41-0	-	-	-
Tireworld	fixit-1	19	19	19
Tireworld	fixit-10	-	-	120
Tireworld	fixit-20	-	-	-
Tireworld	fixit-30	-	-	-
Freecell	prob-7-1	-	-	-
Freecell	prob-7-2	-	-	-
Freecell	prob-7-3	-	-	-
Freecell	prob-8-1	-	-	-
Freecell	prob-8-2	-	-	-
Freecell	prob-8-3	-	-	-
Freecell	prob-9-1	-	-	-
Freecell	prob-9-2	-	-	-
Freecell	prob-9-3	-	-	-
Freecell	prob-10-1	-	-	-
Freecell	prob-10-2	-	-	-
Freecell	prob-10-3	-	-	-
Freecell	prob-11-1	-	-	-
Freecell	prob-11-2	-	-	-
Freecell	prob-11-3	-	-	-
Extra				
Logistics	prob-4-0	20	20	29
Logistics	prob-6-0	25	25	25
Logistics	prob-8-0	31	-	31
Logistics	prob-10-0	45	-	46
Logistics	prob-12-0	42	-	51
Logistics	prob-14-0	-	-	-
Tireworld	fixit-2	30	30	32
Tireworld	fixit-3	41	-	43
Tireworld	fixit-4	-	-	54
Tireworld	fixit-5	-	-	65
Freecell	prob-2-1	-	-	-
Freecell	prob-2-2	-	-	-
Freecell	prob-2-3	-	-	-

Table 2: Comparison between a forward planner using the FastForward heuristic (FP+FF), a forward planner using an ordered landmarks heuristic (FP+LM) and a planner which uses landmarks as sub-goals and solves these using FP+FF (LM+FF). The extra instances are smaller problems from the same dataset which have a higher probability to terminate.

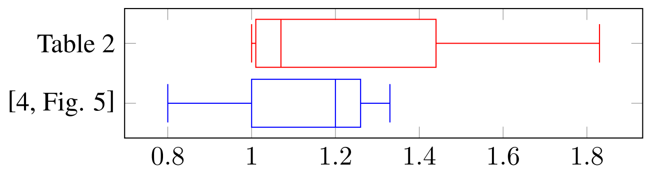

Figure 3: Distribution of the lengths of solutions from the algorithm using landmarks relative to Fast-Forward.

Set	Size	Mean	Std. Dev.
A	9	1.13	0.19
B	12	1.22	0.27

Table 3: Statistics of sets A and B, where A is [4, Fig. 5] from Table 1 and B is Table 2.

extracted after which all algorithms are run up to four times. All tasks are performed by an AMD Ryzen 7 5800H™ Processor clocked at 4.45GHz and are stopped if running longer than 10 minutes or consuming more than 7Gb of memory. The resulting solution lengths are collected in Table 2 and the complete data is included with the experiment code.

Not many executions finish within the time and memory constraints, while in [4] most problems finish within ten seconds and a peak of under seven minutes. A reason for this difference might be that SymbolicPlanners is less efficient than the implementations in [4], given that the baseline algorithm is also slower. Our algorithm which uses a heuristic based on landmarks finishes only for a few small problems and is therefore not useful for comparison. In the $Grid$ and $Freecell$ domains, no problem has more than one planner finishing and they are therefore also not used for comparison.

Both other algorithms in our experiment finish on some larger problems but not enough for comparison, so we added some smaller instances to the experiment from the same datasets as the larger problems used in [4] for those domains that have more instances available. Because for comparison the data is normalized such that Fast-Forward has a length of 1.0, the extra results are useful to model general differences in solution length. The solution lengths can be found in Table 2 for our experiment and in [4, Fig. 5] for the reference values. The normalized lengths of the problems which are used for the comparison are in Table 1. Figure 3 visualizes the distribution of this data.

Let the relative lengths from the experiment of [4] be set A, and let the relative lengths from our experiment be set B. The size, mean and standard deviation from these sets are calculated and are listed in Table 3. With these values, Welch's t-test [22] gives $t_w = 0.93$ and $df_w = 18$. For 18 degrees of freedom and 10% significance level $t = 1.73$ [23]. Because $t_w \leq t$, it is not probable that A and B are samples from different output sets.

In [4, Fig. 5] the average solution length of the planner using landmarks is 13% longer compared to the planner that does not use landmarks. In our experiment, this difference is 22%. From the compared domains, *Blocksworld* has the greatest relative difference in average solution length between the two experiments as can be seen in Figure 4. Instance *bw-*

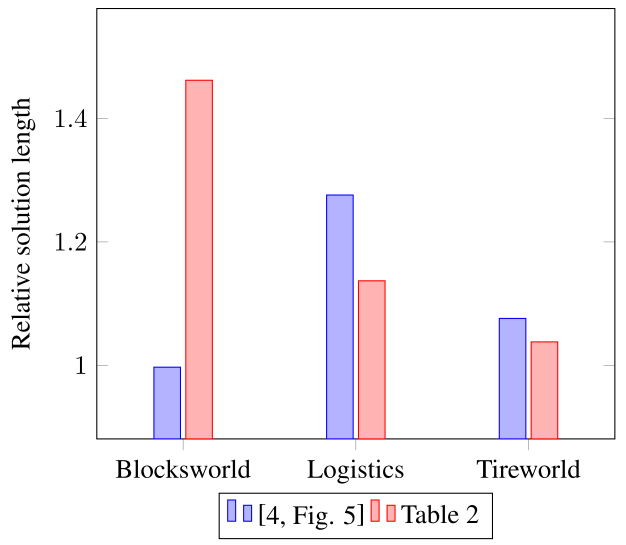

Figure 4: Average length of solutions found by the algorithm using landmarks relative to Fast-Forward. Averages are per domain.

large-d influences this difference significantly while the differences between experiments are under 15% for the other compared domains.

6 Responsible Research

An important aspect of research is the verification of other results through reproducing them because it keeps science reliable. This is also stated in the Netherlands Code of Conduct for Research Integrity [24, 3.4] which is supported by TU Delft. To facilitate this in the field of computer science, it is important to make the code that was used for an experiment available to other researchers or provide a sufficiently detailed pseudo-code that describes all subroutines that can influence the results of the experiment. For paper [4], of which we try to reproduce the results, neither can be found. The names of the problem instances used are listed in the paper and can be found on the internet, but it is not possible to guarantee that they are the same instances.

To fill in these uncertainties for the experiment we had to make assumptions, which possibly do not reflect the choices of the author. SymbolicPlanners has an Apache 2.0[3] license, and our code for this research which is based on it inherits the licence. The code and experiment files of our research are publicly available, so the results can be verified. Also, other researchers can use them to verify whether our assumptions are correct and are in line with the assumptions in [4].

7 Conclusions and Future Work

Forward search is a type of planning algorithm that uses the A*-algorithm to search for valid plans based on a heuristic. Ordered landmarks are information about a planning problem that can be used to guide this algorithm. We have researched the question *How do ordered landmarks affect the solution length of forward search planning in the SymbolicPlanners framework?*

To answer this question we first found an answer to the question *Can the solution lengths for planning problems in previous work be repeated using a different implementation?* We have implemented the ideas of Porteous et al. [4] and compared the plan lengths of this implementation to their results. We obtained marginally different plan lengths using our implementation, but we can conclude that these relative differences are not significant.

We have established that we can repeat the results from previous work with our implementation and we use this implementation to answer the question *How does the solution length of an implementation using ordered landmarks differ from an implementation using the same framework but without using landmarks?* We compared the plan lengths produced by our implementation to a forward search planner with the Fast-Forward [16] heuristic, which are both implemented in the SymbolicPlanners framework. Our implementation using landmarks generated on average 22% longer plans, so we conclude that the idea from Porteous et al. [4] implemented in SymbolicPlanners negatively impacts solution length. This answers the main question of our research.

In our experiment, the algorithm did not solve many problems within time constraints, but the Fast-Forward planner from SymbolicPlanners to which we compared our implementation also did not. Porteous et al. [4] also compared their algorithm to Fast-Forward and there it did finish within time constraints. Further research can answer how the SymbolicPlanners framework affects the running time of planners that use this framework. In the future performance issues with our implementation can be solved to collect more results for comparison.

8 Discussion

We attempted to reproduce the results in [4] with our experiment but their method is not properly reproducible. We had to make some assumptions which can influence the results. Fast-Downward [15] uses ideas that are discovered after [4] was published. Some subroutines in our implementation that are inspired by Fast-Downward may include newer ideas.

We have confirmed that the planning algorithm from [4] using landmarks does on average produce longer plans than a forward planner with the Fast-Forward heuristic. It is possible that [5] presents more optimistic results, or their idea works better for generating shorter plans.

References

[1] S. M. LaValle, *PLANNING ALGORITHMS*. Cambridge University Press, 2006.

[2] K. Solovey, “Complexity of planning,” *ArXiv*, vol. abs/2003.03632, 2020.

[3] J. Contact and L. Sebastia, “Extracting landmarks and ordering them for planning,” 07 2001.

[4] J. Porteous, L. Sebastia, and J. Hoffmann, “On the extraction, ordering, and usage of landmarks in planning,”

[3]https://www.apache.org/licenses/LICENSE-2.0

in *ECP-01. Sixth European Conference on Planning, Toledo, Spain*, pp. 37–48, 2001.

[5] S. Richter, M. Helmert, and M. Westphal, “Landmarks revisited,” in *AAAI Conference on Artificial Intelligence*, 2008.

[6] J. Slaney and S. Thiébaux, “Blocks world revisited,” *Artificial Intelligence*, vol. 125, no. 1, pp. 119–153, 2001.

[7] E. Keyder, S. Richter, and M. Helmert, “Sound and complete landmarks for and/or graphs,” in *Proceedings of the 2010 Conference on ECAI 2010: 19th European onference on Artificial Intelligence*, (NLD), p. 335–340, IOS Press, 2010.

[8] M. Helmert, *An Introduction to PDDL*, 2014.

[9] R. E. Fikes and N. J. Nilsson, “Strips: A new approach to the application of theorem proving to problem solving,” *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, 1971.

[10] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, “Backward-forward search for manipulation planning,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Sept. 2015.

[11] B. Anderson, *A* Search*, 2007.

[12] T. Zhi-Xuan, “SymbolicPlanners.jl,” Feb. 2023.

[13] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *SIAM Review*, vol. 59, pp. 65–98, Sept. 2017.

[14] B. van Maris, “Landmarks in planning: Using landmarks as intermediary goals or as a pseudo-heuristic,” 2024.

[15] M. Helmert, “The fast downward planning system,” *J. Artif. Int. Res.*, vol. 26, p. 191–246, jul 2006.

[16] J. Hoffmann, “Ff: The fast-forward planning system,” in *AAAI Vol. 22 No. 3: Fall 2001*, vol. 22, p. 57, Sep. 2001.

[17] E. Delisle, *Variable Elimination Example*, 2014.

[18] L. Zhu and R. Givan, “Landmark extraction via planning graph propagation,” 2003.

[19] U. of Potsdam, “pddl-instances,” 2016.

[20] H. Kautz and B. Selman, “Blackbox,” 2018.

[21] M. A. Christian Muise, Rowan Monk, “classical-domains,” 2015.

[22] Z. Lu and K.-H. Yuan, *Welch’s t test*, pp. 1620–1623. Thousand Oaks, CA: Sage, 01 2010.

[23] B. B. Gerstman, “t-table,” tech. rep., San José State University, 2007.

[24] K. N. N. T. federatie; Vereniging Hogescholen; VSNU, *Nederlandse gedragscode wetenschappelijke integriteit*. DANS, 2018.