Re-evaluating the Full Landmark Extraction Algorithm

A Performance Analysis of FULL

Noah Tjoen

Supervisor(s): Dr. S. Dumančić, I.K. Hanou

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: Noah Tjoen
Final project course: CSE3000 Research Project
Thesis committee: Dr. S. Dumančić, I.K. Hanou, L. Miranda da Cruz

Abstract

Landmarks are propositions or actions that must be true at some point in every valid solution plan [16]. Using landmarks, planners can develop solutions more efficiently. Different algorithms exist to extract landmarks from a planning problem. The one used in this study is FULL [13], a landmark extraction algorithm by Marzal et al. from 2011. In this research, the performance of the FULL algorithm is analysed by comparing the total number of landmarks found to two other landmark extraction algorithms, namely forward propagation by Zhu and Givan [22] and backward propagation by Porteous et al. [16]. The original FULL algorithm is slightly modified, by removing orderings and disjunctive landmark extraction. FULL is implemented using Julia and was run on five different domains from the International Planning Competitions. All of these domains are logical and 15 problems were randomly selected from them. FULL managed to extract more landmarks in two out of the five domains, Grid and Logistics, compared to the two aforementioned algorithms. In the three other domains, FULL matched the number of landmarks found by the best out of the two. The two domains where FULL performed well, were both transportation domains and this is where FULL's performance excels. Runtime was not an issue when extracting landmarks in four of the five domains. Freecell consistently exceeded the timeout put in place, likely due to a bug. Furthermore, a higher number of landmarks is also a desired outcome due to its use in planners, either as heuristics or intermediary goals.

1 Introduction

Planning is a notoriously complex task and appears in almost everything we do. Think of train schedules, conferences or even a personal daily planner. As it is such a common task, it is important to find a general approach that facilitates the process.

Over the last decades, many different planners and planning algorithms were created. Some of these algorithms are driven by the use of landmarks. Landmarks are propositions or actions that must be true at some point in every valid solution plan [16]. This work by Porteous et al. also laid the foundation for later algorithms, most notably the works of Hoffman et al.. [7], Zhu and Givan [22], Keyder et al. [9], and Richter et al. [17].

Marzal et al. [13] came up with a novel technique in 2011 by combining different steps from the earlier mentioned works to create the FULL algorithm, a full landmark extraction algorithm. Full extraction is defined here by finding the most amount of landmarks possible while using the most prevalent extraction methods from the time of publication. This paper compares the FULL algorithm to the individual algorithms, on which FULL is based, on 12 domains from the International Planning Competitions (IPC). The FULL algorithm matched the best total number of landmarks in most domains and even found a new best in five domains when compared to the number of landmarks found by other landmark extraction and generation algorithms.

Since then, new domains and ways to use landmarks have been proposed. Some examples are different ways to use landmarks as heuristics in an A* planner and updated domains from the latest IPCs. It is therefore wise to reconsider the performance of the FULL algorithm, by re-implementing and re-evaluating it. Thus, this research aims to answer the following question: *How does the performance of the full landmark extraction algorithm, in terms of the number of landmarks identified, compare to other landmark algorithms across different domains?*

This question raises further subquestions:

- *Is a higher number of landmarks always wanted?*
- *Which domains are suited for testing the performance and why?*

These questions and the main research question will be covered in Section 4. Before that, in Sections 2 & 3, further background on planning and its notations and related work will be explained respectively. Section 5 shows the results, while Section 6 will elaborate on the data-gathering process. Lastly, in Sections 7 & 8, the conclusion of the study and the suggestions for future research.

2 Background

Before delving into the main content, it is important to clarify certain concepts that will be used throughout this paper.

2.1 Planning Notation

Planning Domain Definition Language (PDDL) [5] is a standardised way of describing a planning task. A domain in PDDL is characterised by three main components:

- Objects: The objects considered in the problem
- Predicates: A list of predicates which describe information on the state of an object
- Actions: A list of actions. An action is a triple of the following:
  - Parameters: Parameters needed to perform the action
  - Preconditions: Predicates needed to perform the action
  - Effects: effects on the rest of the domain whenever the action is performed

To provide an example, let's consider the following example domain called Blocksworld [20]. The aim in this domain is to stack blocks on top of each other such that they end up in a given goal configuration. Figure 1 shows a Blocksworld domain with three blocks with the start state on the left and the goal state on the right. While this example is simple, the problem sets of Blocksworld can get quite complex, with many more stacks and blocks.

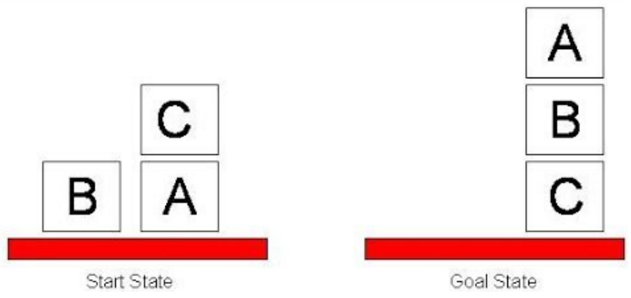

Figure 1: An example of a Blocksworld problem [19].

Another planning notation commonly used in previous literature is the Stanford Research Institute Problem Solver (STRIPS) [4], a predecessor and subset of PDDL. It shares many similarities with PDDL, as actions and a set of goals and initial states are also used. The main difference is in the actions itself. As mentioned earlier in this section, PDDL defines actions as a triple of parameters, preconditions, and effects. Instead of using one set for all of the effects, STRIPS uses two sets: an addition set to represent positive effects, and a deletion set to represent negative effects. In the Blocksworld example, a negative effect of picking up a block would be that it is no longer on the table, while a positive effect would be that the block would be in the hand. Apart from this, PDDL and STRIPS operate similarly and can be used interchangeably. As we will see in Section 3, most previous methods consider relaxed STRIPS, which removes the delete list entirely, allowing for easier computation of the planning graph.

2.2 Planning Graphs

A planning graph is a graph used to represent a planning problem [1]. In practice, either PDDL or STRIPS is used to fill a planning graph with nodes. The initial state of the planning problem is used as the lowest level in the graph. Every subsequent layer is connected to the previous layer through the means of edges. These edges represent the actions required to go to the states/nodes in the next layer. Only actions that are possible, i.e. their preconditions are met at the current level, can be used to transition to the next level. The final level of a planning graph consists of the goal state of the problem.

An apparent limitation of this method is the termination of the process. When creating a graph, a goal state might not be expanded. This could be due to the graph looping on itself, creating cycles, or by parts of the graph being disconnected from the rest [2]. Both of these problems can lead to a high runtime. In practice, a timeout is used to prevent exponential runtime.

While state expansion of STRIPS and PDDL are identical, PDDL provides a more generalised representation of the planning problem than STRIPS [5]. To account for more generalisation, most instances in STRIPS are translated into a relaxed version. Most often the relaxation removes the delete list, leaving only the ‘positive’ effects of taking an action. Another relaxation which is used for both PDDL and STRIPS is to allow all actions to be taken from a state, disregarding the preconditions necessary to take the action. For this paper, this will not be considered, as none of the subroutines of FULL require this relaxation. However, both of the aforementioned relaxations create a new problem. It is now uncertain if the actions can be taken in a non-relaxed planning. This means a verification of the original planning problem is required.

2.3 Heuristics and Search

Landmarks are used to guide the search in a planning algorithm to make smarter choices in the planning graph. This is frequently done by heuristics in combination with a path-finding algorithm. The most prevalent of these path-finding algorithms is A* due to its efficiency and guarantee of finding an optimal solution, given that a consistent heuristic is provided [18]. A* works the same as Dijkstra’s shortest path algorithm but changes the order of node expansion. While Dijksta’s algorithm greedily expands nodes, taking the lowest cost edge first, A* uses a heuristic function.

Heuristics are functions to guide a search towards the goal. A common heuristic used in A* is to use the greedy function of Dijkstra, $g(n)$, with the total cost of reaching the goal after taking that edge, $h(n)$, leading to the heuristic function $cost(n) = g(n) + h(n)$. This cost is then used to determine which node to expand first in the A* algorithm.

Many different heuristic functions are possible, but not all are suitable. To find an optimal solution in a graph, a consistent and admissible heuristic has to be used. A heuristic is consistent if it never overestimates the growth of the path cost, while admissibility means that the true cost is never overestimated [21]. Consistency implies admissibility, but the contrary is not true. However, most admissible heuristics are also consistent. The earlier-mentioned function is an example of a consistent and admissible heuristic.

3 Related Work

This section will cover work which is similar or which inspirations are taken from. The papers considered in this section are chronologically ordered, as the later articles use the discoveries from the earlier papers.

3.1 Landmark Extraction Using Backward Propagation

As mentioned in the introduction, the paper by Porteous et al. [16] from 2001 laid the foundation for the use of landmarks in planning. They first define what landmarks are and provide extensive mathematical proofs to support their definitions. They then describe a method to extract landmarks from a relaxed planning graph (RPG). This is done by first creating the RPG from which a backward graph traversal from the goals is done. Doing so extracts landmarks as it progresses with the traversal. As mentioned in Section 2.2, when extracting landmarks from an RPG, they need to be verified. In this approach, it is done by checking the following proposition:

Proposition 1 *Given a Planning task $P = (O, I, G)$ and a fact L. Define $P_L = (O_L, I, G)$ as follows:*

$$O_L := \{(pre(o), add(o), \emptyset) \mid (pre(o), add(o), del(o)) \in O, L \notin add(o)\}$$

If P_L is unsolvable, then L is a landmark in P.

This proposition changes the set of actions to a relaxed set, where the delete lists are emptied for actions that do not have fact L in their addition list.

Both the verification and landmarks generation are subroutines of FULL. A critical change FULL makes is immediately verifying the landmark candidates for the generation of the RPG. This results in creating a graph of landmarks instead, which can be used in a later step of FULL.

3.2 Landmark Extraction Using Forward Propagation

While previous work focuses on the backward traversal of planning graphs to find landmarks, the paper by Zhu and Givan from 2003 takes a different approach [22]. By using forward propagation of labels on a regular planning graph, they were able to find a set of landmarks. As these landmarks were extracted from the original planning graph, unlike the work of Porteous et al., no verification of landmarks is needed. In short, this method starts on the original state of the planning problem and sets this as the first level. For each subsequent level, information about the previous levels is propagated through the form of labels. This process is repeated until the goal state has been reached, after which the labels from the goal states are united, creating a set of landmarks.

This way of extracting landmarks is another subroutine used by the full landmark extraction algorithm. In FULL, the relaxed planning graph is used which adds the need for more verification. Compared to the backward propagation of the previous section, forward propagation can find different landmarks due to the nature of the node exploration. Depending on the problem, this can lead to more or less landmarks being found.

3.3 Full Landmark Extraction

Marzal et al. first introduced the FULL landmark extraction algorithm in 2011 [13]. By combining techniques from the two papers mentioned previously, they created a new technique which outperformed the individual algorithms considered in the paper in terms of landmarks extracted. This algorithm consists of five steps which are as follows:

1. Extract landmarks from a relaxed planning graph [16]
2. Use forward propagation on a relaxed planning graph to find more landmarks [22]
3. Verify all previously found landmarks using Proposition 1
4. Use verified landmarks to find disjunctive landmarks [15]
5. Compute dependency ordering between landmarks, i.e. landmark L happens before landmark L'

The last step of this algorithm is only applied to landmarks found using Zhu and Givan's method as the method from Porteous et al. already includes this in its landmark extraction.

The algorithm by Marzal et al. is the main inspiration for this paper. However, there are a few differences. The first of which is the removal of Step 4. This is due to this paper's availability, as neither the author nor the institution where it was originally published has it in possession. Orderings are mainly used for running a planner with landmarks. As this paper is only concerned with finding the total number of landmarks, the dependency orderings in Step 5 add redundant computational complexity and will thus not be considered.

4 Methodology

Before delving into the findings, we need to fully define the algorithm used and explain all the domains, out of which landmarks will be extracted.

4.1 The Full Landmark Extraction Algorithm

As briefly described in Section 3.3, the full landmark extraction algorithm used here consists of four steps, as opposed to the five-step procedure used in the reference paper. Each of these steps contains a subroutine of the algorithm.

The algorithm starts by creating a relaxed planning graph and extracting possible landmarks from it. As mentioned in Section 3.1, this is done by creating a planning graph, after which backwards propagation on this graph is used to extract landmarks. These landmarks are then immediately verified using Proposition 1 and are stored in a graph, constructing a graph of landmarks and the orderings between them.

This is then followed by extracting landmark candidates using forward propagation of labels, described by Zhu and Givan [22]. These labels contain information about the previous layers of the planning graph, like all previously visited nodes and all actions taken. By using a relaxed planning graph for this step instead of a regular planning graph, the algorithm has fewer options to consider when expanding nodes, theoretically leading to a faster runtime.

However, this means that verification of these landmarks is necessary, hence the next step of the algorithm. Using Proposition 1, found landmarks from Step 2 can be verified.

Lastly, the landmarks found in the first step and the verified landmarks from the previous step are merged to create a final set of landmarks, which will be used to measure the performance of FULL.

This leaves us with the following definition of FULL:

1. Extract landmarks from a relaxed planning graph using backward propagation [16]
2. Use forward propagation on a relaxed planning graph to find more landmarks [22]
3. Verify landmarks found in Step 2 using Proposition 1.
4. Merge verified landmarks from Step 3 with landmarks found in Step 1

4.2 Testing Domains

As the algorithm used is now defined, it is also important to decide the planning domains used. In general, there are two different types of planning domains: logical and numerical.

Logical domains consist of actions and states which can only take boolean values. The Blocksworld domain from Section 2 is a good example of a logical domain. The position of the blocks can either be true if the block is at a certain position or false if it is not. Within logical domains, a division can also be made into transportation and non-transportation domains [6]. Transportation domains involve the transportation of some object, i.e. packages, trucks or people, while non-transportation domains, like Blocksworld, do not have this requirement.

Numerical domains are different to logical domains, as they can also take numerical values as well as boolean values. An example is the Zeno-Travel domain [11]. In this domain, passengers need to be transported between multiple cities on planes, which have different speeds, fuel levels and passenger counts. This makes these domains much more complex and allows for the generation of a wider spread of landmarks.

For this study, only logical domains will be used due to the nature of the implementation of the algorithm and its landmarks. The limitations of the implementation will be further discussed in Section 8.

To continue with the descriptions of domains used for this study, we will consider the Logistics domain [3]. This domain consists of packages that need to be delivered within and between cities. The packages can move between cities through the means of planes, and within cities via trucks. This domain is also a logical domain. Some landmarks to think of for this domain are a package being on a plane or a truck being at a certain airport.

The next domain to consider is called Freecell [11]. Freecell is a variant of the popular card game Solitaire. The goal of this game is to move all cards into piles of the four suits in ascending order, starting with Ace. Cards are dealt face up in eight columns, from which they can be moved to the other columns, the corresponding suit pile or one of four free cells. Cards can only be stacked on each other if the card on top is of a different suit and lower cost than the card below, i.e. a seven of spades can be stacked on top of an eight of hearts. An example of a landmark found in the Freecell domain is that a card has to be at a free cell spot in the final plan.

The penultimate domain is named Miconic [3]. In this domain, an elevator can move between floors and transport passengers to their selected floors. There are no restrictions on the number of people in the elevator at once, however, there are variations of this domain where this not is the case. It is unnecessary to consider these variations as the original domain is complex when many passengers and floors are present.

The last domain to be used is called Grid [14]. This domain consists of locations on a NxN grid. A robot can move one grid square at a time, either horizontally or vertically. Some squares are locked and the robot can only move to it by unlocking it. The key must have the same shape as the lock for it to be unlocked. The keys must be picked up and can themselves be in locked locations.

5 Experimentation & Results

This section gives a visualisation of all the results found during the study. Firstly, the performance criteria and problem selection are explained, after which the final results are shown and analysed.

5.1 Performance Criteria

As the research question states, FULL was tested on the number of landmarks found. This method was chosen because it is consistent regardless of the implementation of the algorithm and hardware specifications. However, the runtime was still meaningful to consider. While a higher number of landmarks may be found using FULL, the runtime could also increase substantially, leading to an inefficient extraction.

This algorithm was implemented in Julia. Julia is one of the fastest programming languages among different applications [8], making runtime less of a limiting factor. This is particularly convenient for larger problem sets, where the graph computation and traversal can take polynomial to exponential runtime.

It was still required for a timeout to be put in place, as certain problems regarding runtime can arise, mentioned in Section 2.2. For this study, a 10-minute timeout for each problem was used. This bound is theoretically tight enough for all problems to finish. However, the results may vary on machine specifications. Less memory or lower CPU clock speeds could lead to more timeouts.

The results from a study by van Maris showed that a higher number of landmarks is preferable whenever landmarks are used as intermediary goals or pseudo-heuristics in a planner [12]. When more landmarks are available, especially when ordered, the runtime of planners increases significantly. Thus, the more landmarks extracted by FULL, the better.

5.2 Problem instance selection

As mentioned in Section 4.2, the FULL algorithm was run on various domains. Various problem instances were randomly picked within these domains while ensuring complex instances were still considered. Where possible, 15 problems were selected to run FULL on. This depended on the availability of the problem set of the IPC. These results were then compared to other landmark extraction algorithms, namely forward propagation [22] and backward propagation [7], both subroutines of the FULL algorithm. Finally, the different domains were compared to each other, to determine which domains FULL performs better on.

5.3 Blocksworld

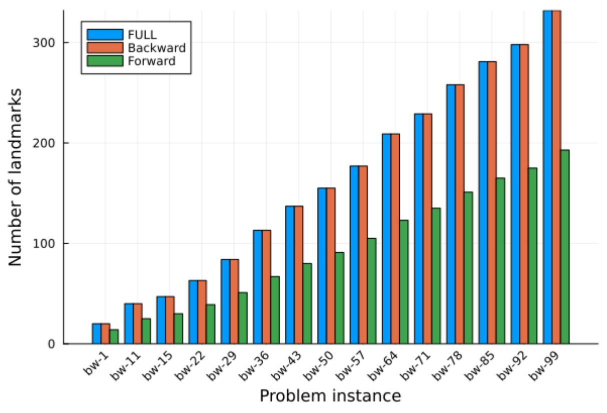

Figure 2: Full landmark extraction plotted against forward and backward propagation of landmarks on various problems from the Blocksworld domain. Problems along the x-axis are sorted by increasing complexity.

The total number of landmarks extracted using FULL (blue), as well as the backward and forward propagation (orange and green respectively) are plotted against the respective problem number in the Blocksworld domain (Figure 2). The problems on the x-axis are displayed in increasing order in terms of complexity. Problem 1 contains only three blocks in a simple start and goal configuration, while Problem 99 has 47 blocks and a hard goal state. Figure 2 shows that FULL extracts the same number of landmarks as backward propagation. Forward propagation finds fewer landmarks. This could be due to the simple nature of the domain, as there is not much information to propagate through the labels.

5.4 Logistics

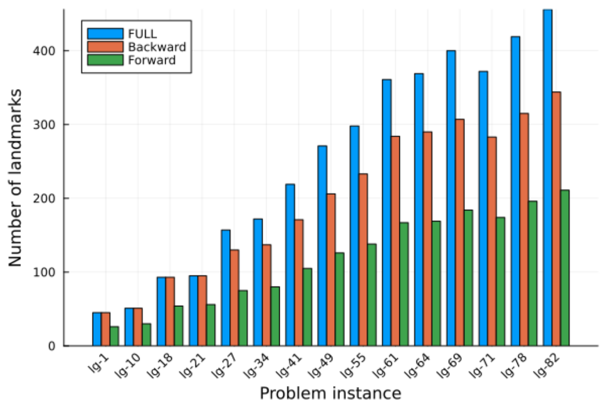

Figure 3: Full landmark extraction plotted against forward and backward propagation of landmarks on various problems from the Logistics domain. The problems on the x-axis are sorted by increasing complexity.

In Figure 3, landmark extraction of the Logistics domain can be seen. The landmarks extracted by FULL (blue), backward propagation (orange), and forward propagation (green) are plotted against the problem instances from the Logistics domain, which are sorted in order of increasing complexity. Figure 3 shows that FULL extracts more landmarks than both other algorithms starting from problem 27. In the instances prior, FULL matches the number of landmarks extracted by backward propagation. The forward propagation consistently performs worse than the other two methods in this domain. However, forward propagation finds different landmarks from backward propagation, indicated by a higher number of landmarks extracted by FULL.

5.5 Freecell

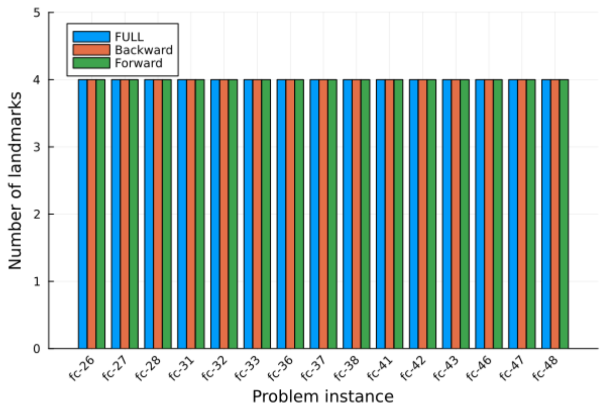

Figure 4: Full landmark extraction plotted against forward and backward propagation of landmarks on various problems from the Freecell domain. Problems along the x-axis are sorted by increasing complexity.

Figure 4 shows the number of landmarks extracted by FULL (blue), backward propagation (orange), and forward propagation (green) in the Freecell domain. All extraction methods find four landmarks across the different problem sets. This is likely due to a bug in reading the domain by all the algorithms considered. As the Freecell domain always contains four goal states, the landmarks found by the three methods consist of these states. The fact a bug occurred in this domain is also apparent when analysing the runtime of this domain, which can be seen in Section 5.9.

5.6 Miconic

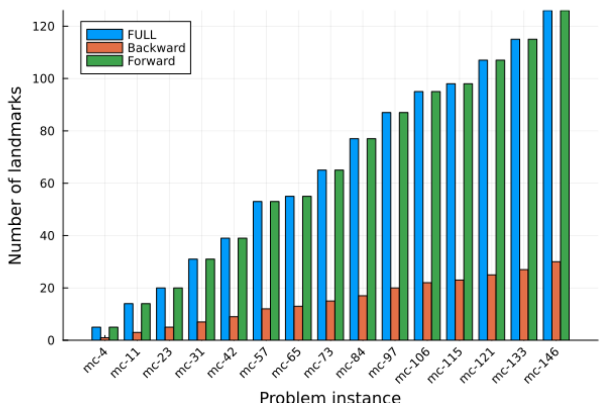

Figure 5: Full landmark extraction, and forward and backward landmarks propagation plotted against each other on various problems from the Miconic domain. Problems along the x-axis are sorted by increasing complexity.

Figure 5 displays the extraction of landmarks by FULL (blue), backward propagation (orange), and forward propagation (green) in the Miconic domain. The number of landmarks found using backward propagation in this domain is substantially lower compared to those found by the other two algorithms. Forward propagation also finds the same landmarks as backward propagation does. This means that the number of landmarks found by FULL is capped by the landmarks found in forward propagation.

5.7 Grid

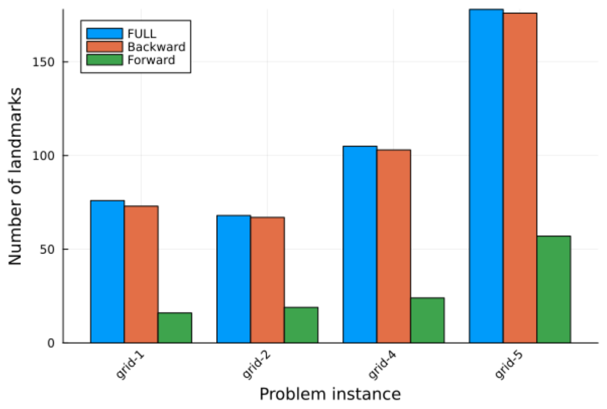

Figure 6: Full landmark extraction plotted against forward and backward propagation of landmarks on various problems from the Grid domain. Problems along the x-axis are sorted by increasing complexity.

In Figure 6, landmark extraction of the Grid domain can be found. The landmarks extracted by FULL are in blue, the ones extracted using backward propagation are in orange, and those found by forward propagation are in green. As mentioned at the beginning of this section, 15 problems were selected per domain where possible. For Grid, only five were available. The graph only shows four of these problems, as Problem 3 timed out within the given time frame of 10 minutes. FULL performed as expected for the other four problems, finding more landmarks than both methods. Backward propagation found more landmarks than forward propagation, but some different landmarks were extracted using the latter. This led to FULL finding more landmarks than backward propagation.

5.8 Domain comparison

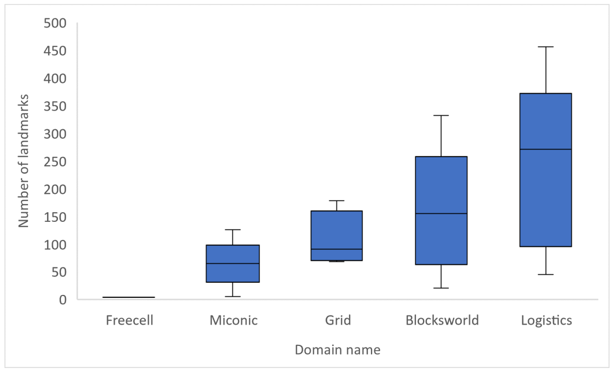

Figure 7: Total number of landmarks extracted by FULL across all considered domains, in increasing order of total landmarks found.

Figure 7 shows the number of landmarks found by FULL. Each box represents the total number of landmarks found across all problem instances for each domain. The line within each boxplot shows the median number of landmarks found across each domain, while the whiskers show the outliers.

The Grid domain has the smallest number of outliers, while Logistics has the most. This could be explained by the number of problems considered for Grid. The median for all domains but Grid lay around the mean, which is desired. This means that the outliers do not influence the overall results and that the data points are distributed normally.

FULL finds the most landmarks in the Logistics domain, although the complexity of this domain is similar to the rest. A possible explanation for this is that Logistics categorises objects differently. Logistics divides its objects into trucks, planes, airports, locations, and packages. The other domains only use a maximum of two different types of objects. From this, we can conclude that FULL extracts more landmarks if more object types are defined in a domain.

5.9 Runtime comparison

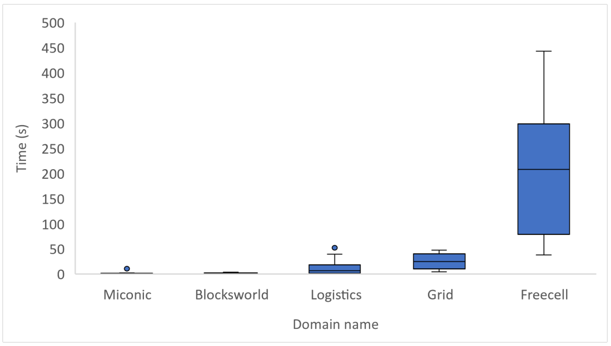

Figure 8: Runtime of FULL on all considered domains, in increasing order of total time taken.

In Figure 8, the runtime of FULL is plotted in seconds across every domain. The runtime for all problems in Freecell was extremely high, relative to the number of landmarks found. This could be explained by a possible bug mentioned in Section 5.5. For the rest of the domains, the runtime was less of an influential factor. No domain had any notable outliers, with a tight spread on each box. The total execution time for even the most complex problems was less than one minute across all domains, meaning a tighter timeout could have been used.

6 Responsible Research

During this research, no personal data was used. All domains and problem sets were taken from the IPC and are publicly available [3][11][14]. 15 problems were chosen for each domain from problem sets of different sizes. When the size of the problem set was less than 15, all problems were selected from the domain. This was only the case for Grid. These problems were chosen using a random number generator to avoid any bias. However, this caused other issues regarding runtime, but this will be further discussed in Section 8. The data shown in the results was generated using self-written code, with no assistance from Large Language Models, like ChatGPT. The code for FULL, backward propagation, and forward propagation can be found on the TU Delft repository. The data visualisation and reporting are also under the Netherlands Code of Conduct for Research Integrity [10]. The same applies to the rest of the methods used in this report, such as citing and adding references to give original authors credit.

7 Conclusion

From the results in the previous section, a few things can be concluded. Firstly, FULL finds more or the same amount of landmarks in all problems than the forward and backward propagation. Hence it is proven that overall FULL performs better and is a more reliable way to extract landmarks as it is never outperformed by the other two methods considered. While more landmarks were expected to be found by FULL than the other two landmark extraction algorithms across all problems, matching them is a good finding. Furthermore, a higher number of landmarks extracted means an improved runtime when landmarks are used in combination with planners as heuristics [12].

Some of the domains used were more suited to highlight the performance of FULL, namely Grid, Logistics, and Miconic. This is due to these domains being transportation domains, according to the definition by Helmert [6], while the other domains are not.

In the future, more non-transportation domains could be considered to see whether the performance of FULL is also limited in these domains. Another possible avenue of exploration is to analyse the quality of the landmarks that are found. While currently only the total number of landmarks is found, the quality of these landmarks varies. Thus, the landmarks found by FULL could be used in combination with various planners to test if the quality matters in future research. An additional recommendation is to analyse the runtime using the landmarks found by FULL.

8 Discussion

Three of the five domains used, Blocksworld, Freecell and Logistics, were evaluated in the original paper by Marzal et al. [13]. From these three domains, Blocksworld and Logistics showed similar results. Both studies showed an increase or match in terms of the number of landmarks found compared to the other algorithms considered.

For the Freecell domain, the version of FULL by Marzal et al. discovers a very high number of landmarks compared to the version implemented for this study. However, the paper also states that the other sub-algorithms find hundreds of landmarks. This was not the case during this study. As mentioned in Section 5.5, four landmarks were found over all the problem instances. This could likely be attributed to the poor translation from the PDDL file to the input used for FULL, as only the goal states were found as landmarks.

As mentioned in Section 4.2, the way landmarks are implemented limits the use of numerical domains. Currently, landmarks are represented as a fact pair. This pair has an action or predicate from the planning graph alongside a boolean value, indicating whether the landmark is true in a given state. Since boolean values are used instead of numerical values, using numerical domains is impossible. Reimplementing landmarks using terms would be an alternative and allow numerical domains to be used.

The runtime results displayed in Section 5.9, may vary depending on the hardware used. However, the ratio between the domains will stay the same as the complexity of the domains can not change. Only one problem timed out, which was Grid 3. This could be due to a deadlock occurring by the robot using the keys in the wrong order.

Lastly, tighter runtime bounds could have been used. As seen in Section 5.9, the runtime only exceeded a minute on the bugged Freecell domain. Tighter bounds would have resulted in a faster data-gathering process, which could have allowed for more problem instances or different domains to have been considered.

Acknowledgements

I would like to express my appreciation to my responsible professor, Dr. S. Dumančić, and my supervisor, I.K. Hanou, for their efforts during this study, for providing meaningful feedback during the weekly meetings and their openness to answering questions whenever necessary. I would also like to thank my research group members, B. van Maris, P. Tervoort., K. Yang, and P. Hengst, for their teamwork and their work ethic during this project, pushing each other to achieve better results.

References

[1] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1):281–300, 1997.

[2] B Bollobás. *Modern Graph Theory*. Springer New York, 1998.

[3] Bacchus F. Competition domains, 2000.

[4] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, December 1971.

[5] M Ghallab, C Knoblock, D Wilkins, A Barrett, D Christianson, M Friedman, C Kwok, K Golden, S Penberthy, D Smith, Y Sun, and D Weld. Pddl - the planning domain definition language. 08 1998.

[6] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003.

[7] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, November 2004.

[8] Julialang. Julia micro-benchmarks, 2023.

[9] 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 Conference on Artificial Intelligence*, page 335–340, NLD, 2010. IOS Press.

[10] KNAW, NFU, NWO, TO2-Federatie, Vereniging Hogescholen, and VSNU. Nederlandse gedragscode wetenschappelijke integriteit, 2018.

[11] D Long and M Fox. Competition domains, 2002.

[12] B Maris van. Using landmarks as intermediary goals or as pseudo-heuristics. Bachelor's thesis, Delft University of Technology, 2024.

[13] E. Marzal, L. Sebastia, and E. Onaindia. Full extraction of landmarks in propositional planning tasks. In *AI*IA 2011: Artificial Intelligence Around Man and Beyond*, pages 383–388. Springer Berlin Heidelberg, 2011.

[14] D McDermott. Competition domains, 2000.

[15] J. Porteous and S. Cresswell. Extending landmarks analysis to reason about resources and repetition. November 2002. Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG '02) ; Conference date: 21-11-2002 Through 22-11-2002.

[16] J. Porteous, L Sebastia, and J Hoffmann. On the extraction, ordering, and usage of landmarks in planning. *Proc. European Conf. on Planning*, July 2001.

[17] S Richter, M Helmert, and M Westphal. Landmarks revisited. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, page 975–982. AAAI Press, 2008.

[18] S Russell and P Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, February 2010.

[19] B Supervisor, H Fiorino, and C Martin. Interactions in planning systems.

[20] G. J. Sussman. A computational model of skill acquisition, August 1973.

[21] N. Yorke-Smith. Algorithms for np-hard problems, part i – heuristic search, lecture 2: Greedy best-first search. Lecture Slides.

[22] L Zhu and R Givan. Landmark extraction via planning graph propagation. June 2003.