Landmarks in Planning

Using landmarks as Intermediary Goals or as a Pseudo-Heuristic

Bart van Maris

Supervisor(s): Sebastijan Dumančić, Issa 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: Bart van Maris
Final project course: CSE3000 Research Project
Thesis committee: Sebastijan Dumančić, Issa Hanou, Luís Miranda da Cruz

Abstract

Algorithmic planners occasionally waste effort and thus computing time trying to solve certain tasks, as they often lack the human ability to recognize essential paths. These essential paths ,termed landmarks, are vital for optimizing planning processes. This study revisits landmark-based planning methods introduced by Richter, Helmert, and Westphal in their 2008 paper, adapting and implementing them within a different framework, SymbolicPlanners, using the Julia programming language. The primary research question explores the performance of using landmarks as intermediary goals and pseudo-heuristics in the SymbolicPlanner framework. Sub-questions delve into the effectiveness of specific planning strategies, such as $A*$ Planner with $GoalCount$ and $HAdd$ heuristics, as well as planners utilizing landmarks. Evaluation over diverse domains reveals that LM^{Local} and $LM^{Local}Smart$ outperform the basic $GoalCount$ heuristic and are on par with the $HAdd$ heuristic. LM^{Count}, despite solving fewer instances, exhibits speed improvements over $GoalCount$ in the instances that they both solve. Discussion highlights limitations, such as the non-exhaustive interference check in $LM^{Local}Smart$ and limiting factors in the SymbolicPlanner framework.

1 Introduction

When planning a route through a city we humans can quite quickly determine that we need to take certain roads to get to our destination. However algorithmic planners don't have the ability to recognize these required roads, which for us is quite easy since we can quickly spot choke points and other obstacles. Moreover it would significantly speed up their planning process if they did know they needed those roads. In the context of generalized planning problems these required roads are called landmarks. Giving planning algorithms access to knowledge about these landmarks could prevent wasted effort.

Richter, Helmert and Westphal [2008][10] propose two methods for using landmarks and one method for extracting landmarks. In their work they find that the usage of landmarks in solving planning problems increase the performance of the planners that are used to solve them [10]. Their methods where implemented in a package called FastDownward.

In this paper we aim to reproduce their methods for using landmarks in a different framework and programming language, namely SymbolicPlanners and Julia respectively. Then we would also like to asses the performance of these algorithm to see if the findings of Richter et al. match what we are able to produce. Thus the main research question is as follows:

”What is the performance of using landmarks as intermediary goals and using landmarks as pseudo-heuristics in the SymbolicPlanner framework?”

This main research question can be further divided into the following sub-questions:

- What is the performance of the $A*$ Planner using a Goal-Counting heuristic in the SymbolicPlanner framework?
- What is the performance of the $A*$ Planner using a Additive heuristic in the SymbolicPlanner framework?
- What is the performance of a planner using landmarks as intermediary goals in the SymbolicPlanner framework?
- What is the performance of the $A*$ Planner using a landmark pseudo-heuristic in the SymbolicPlanner framework?
- How do each of the before mentioned methods compare?

In the next section the background and definitions in this paper will be further elaborated on. After this background information a short section on related work follows. Then a detailed description of the implemented algorithms and assessment criteria are contained in the methodology section. This is then followed by the results, conclusion and discussion.

2 Background

In this paper we will be talking about planning, planners and landmarks. Planning in this context is finding a sequence of actions that take you from the starting state to a goal state. Think of finding a route on a map or arranging blocks in a certain way. These different contexts in which a plan can be made are called domains and a possible situation is called a state. These domains and states can be expressed in PDDL, a programming language used in the International Planning Competition to standardize these planning problems [3]. This language describes problems in two parts. Firstly, a domain description containing its name, requirements, an object-type hierarchy, constant objects, predicates and actions. Secondly a problem description containing its name, the domain-name, all objects, initial conditions and a goal state.

For a planner to solve a problem represented in PDDL it needs some way of approximating the distance to the goal from some state. The most state of the art solution is to use some form of heuristic search [1]. A heuristic in this case is a function that contributes some value to a state. These values are then used by a planner to determine a plan. There exists quite a few heuristic search based planners and different heuristics. In this paper the heuristic based planner we use to compare to is the $A*$ planner. Furthermore some of the methods we implement will use a planner, in this case we also use the $A*$ planner. As described in the research questions, we intend to use as heuristics the Goal-Count and Add heuristics. The Goal-Count heuristic simply counts the number of goals that are satisfied. The Add heuristic uses the sum of the costs of the conditions an action depends upon to determine the heuristic value.

The foundation for this paper is the work by Richter, Helmert and Wesphal [10]. Thus the formal definition for a planning task, landmarks and orderings between facts or landmarks will be kept the same as in the work by Richter et al [10] but will be repeated here for clarity.

Definition 1 ***SAS⁺ planning task***
An SAS⁺ planning task[2] is a tuple $\Pi = \langle \nu, O, s_0, s_* \rangle$ where:

- ν is a finite set of state variables, each with a finite domain D_v. A fact is a pair $\langle v, d \rangle$ (also written as $v \mapsto d$), where $v \in \nu$ and $d \in D_v$. A partial variable assignment s is a set of facts, each with a different variable. (We use set notation such as $\langle v, d \rangle \in s$ and function notation such as $s(v) = d$ interchangeably.) A state is a partial variable assignment defined on all variables ν.
- O is a set of operators, where an operator is a pair ⟨pre, eff⟩ of partial variable assignments.
- s_0 is a sate called the initial state.
- s_* is a partial variable assignment called the goal.

An operator $o = \langle pre, eff \rangle \in O$ is applicable in state s iff $pre \subseteq s$. In that case, it can be applied to s, which produces the state s' with $s'(v)$ =eff(v) where eff(v) is defined and $s'(v) = s(v)$ otherwise. We write $s[o]$ for s'. For operator sequences $\tau = \langle o_1, ..., o_n \rangle$, we write $s[\tau]$ for $s[o_1]...s[o_n]$ (only defined if each operator is applicable in the respective state). The operator sequence τ is a plan iff $s_* \subseteq s_0[\tau]$.

This SAS⁺ representation can be generated from a PDDL representation of a planning task automatically[4].

Definition 2 ***Landmark***
Let $\Pi = \langle \nu, O, s_0, s_* \rangle$ be am SAS⁺ planning task, let $\pi = \langle o_1, ..., o_n \rangle$ be an operator sequence applicable in s_0, and let $i \in 0, ..., n$.

- A fact F is true at time i in π iff $F \in s_0[\langle o_1, ..., o_i \rangle]$.
- A fact F is added at time i in π iff F is true at time i in π, but not at time $i - 1$ (Facts in s_0 are considered added at time 0).
- A fact F is first added at time i in π iff F is true at time i in π, but not at any time $j < i$.
- A fact F is a landmark of Π iff in each plan for Π, it is true at some time.

In s_0 and s_* all facts are landmarks by the above definition (consider $i = 0$ and $i = n$ respectively). To be able to use landmarks as goals or as a heuristics the ordering of those landmarks needs to be known. The following definition follows from Hoffmann et al. [5][7].

Definition 3 ***orderings between facts***
Let A and B be facts of an SAS⁺ planning task Π.

- There is a natural ordering between A and B, written $A \rightarrow B$, iff in each operator sequence where B is true at time i, A is true at some time $j < i$.
- There is a necessary ordering between A and B, written $A \rightarrow_n B$, iff in each operator sequence where B is added at time i, A is true at time $i - 1$.
- There is a greedy-necessary ordering between A and B, written $A \rightarrow_{gn} B$, iff in each operator sequence where B is first added at time i, A is true at time $i - 1$.

Note that a necessary ordering is always also a greedy-necessary ordering.

2.1 Definition of algorithms

Since we intend to reproduce the methods of Richter et al. [10], namely their procedures of LM^{Count} and LM^{Local}, we will give the formal description of those algorithms here. In the methodology section of this paper we will go into detail on changes that had to be made due to the limits of the chosen framework and programming language.

LM Local: Using landmarks as Intermediary goals
LM^{Local} is a procedure which exploits landmarks by using them to divide the larger planning task into smaller sub-tasks. This is done by employing landmarks as intermediary goals in the planner. First a landmark graph is generated using some landmark extraction method. From this directed graph source nodes are extracted. These sources are nodes that have no edges pointing to them. However due to cycles in the graph it can occur that there are no more sources to be found. So before extracting sources all possible cycles must be removed from the graph. This is done by removing edges from the graph. Then the found sources are given to a planner as a disjunctive goal. Once one of those goals is completed the landmark associated with that goal is removed from the landmark graph and the new sources are added to the disjunctive goal. The planner is then told to continue solving for those goals. Once the landmark graph is empty the planner is tasked with solving for the original goal. This is because even though the original goal should be a landmark it might be that it was needed multiple times to complete the full task, thus the planner still has to solve for the original goal.

LM Count: Using landmarks as a pseudo-heuristic
The formal description of the simple landmark pseudo-heuristic as described by Richter et al. [10] goes as follows: The heuristic value h of a state s is based on the number of landmarks by the following estimation: $h := n - m + k$. Where:

- n is the total number of extracted landmarks
- m is the total number of *accepted* landmarks in state s
- k is the total number of *accepted* landmarks in state s that are *required again*.

A landmark l is *accepted* in state s iff l is true in state s or l has been true in some successor state of s. A landmark l is *required again* in state s iff l is not true in state s and l has some greedy-necessary predecessor that is not *accepted*. We shall refer to this pseudo-heuristic as LM^{Count} from now on.

It should be noted that LM^{Count} is not a true heuristic, hence the reason that we refer to it as a pseudo-heuristic. LM^{Count} is not a true heuristic because its definition depends on the way the state was reached during search. Nevertheless, it can be used as a heuristic in best-first search algorithms.

3 Related Work

As this paper aims to reproduce the work of Richter, Helmert and Westphal[10], it is closely related to it, in this paper they define the methods that we also intend to use. Other notable pieces are firstly that of Richter and Westphal further elaborated on the usage of landmarks in their LAMA planner [8]. In that work, Richter and Westphal provide pseudo code for using landmarks as a pseudo heuristic and combine it with other heuristics. Secondly, Richter's own thesis on landmark based heuristics [9] contains the previous two works by Richter and adds more detail. Thirdly the work of Pereira, Oren and Meneguzzy provides multiple alternative methods to use landmarks as heuristics [6]. In this work they provide additional methods for computing achieved landmarks, finding landmark uniqueness and a different heuristic. These methods can be combined with this paper's results to improve overall performance of a planner using landmarks. Lastly the work of Segovia-Aguas, Celorrio, Sebastia and Jonsson take the method of using landmarks as a heuristic described Landmarks Revisited and place it in the context of Generalized Planning[11]. This adds important context for when the methods described in this paper can be used in the setting of generalized planning, since now the implementation works only for SAS^+ planning tasks.

4 Methodology

This section of the paper will start with a detailed description of the implemented algorithms. The last subsection will go in detail on assessment criteria and experimentation setup.

4.1 LM Local

The implementation of LM^{Local} differs slightly from the description given in the Background section. This is due to limitations in the SymbolicPlanner framework. Unfortunately the planners in the SymbolicPlanner framework do not support disjunctive goals. To overcome this issue two versions of LM^{Local} where implemented. One simple implementation that we will refer to as LM^{Local} and one version that tries to improve on this simple implementation called $LM^{Local}Smart$.

LM^{Local} is implemented as follows: Instead of providing the planner with a disjunctive goal we instead create a copy of the current planner. This copy is then used to solve for one of the source landmarks. We then take the shortest solution between the solutions produces for each source. The used landmark for that solution is removed from the landmark graph. The planner that produced this solution is used as the basis for the copy in the next step of the algorithm. Then we start again with procuring sources from the landmark graph and making a copy of our planner. The pseudo code for LM^{Local} can be seen in Algorithm: 1

$LM^{Local}Smart$ adds to the implementation of LM^{Local} by trying to combine sources into larger goals to try to minimize the amount of copies of planners that have to be made. This is done by pre computing all possible combinations of landmarks and checking if they are achievable together. Then instead of using the source as a goal, larger goals are constructed based on the current sources. Then for those larger goals or goal groups the copies of planners are made and are told to solve that sub task. Again the shortest of those solutions is taken. Each of the landmarks associated with the solved goal group is then removed from the landmark graph. This repeats until the landmark graph is empty. The pseudo code for $LM^{Local}Smart$ can be seen in Algorithm: 2

Algorithm 1: LM Local

```
Data: $planner$
Input: $lm\_graph$, $domain$, $state$, $goal\_state$
$solution$
while $lm\_graph$ not empty do
    $shortest\_sol, used\_lm, used\_planner$
    for $lm \in$ get_sources($lm\_graph$) do
        $copy\_planner \leftarrow planner$
        $sub\_sol \leftarrow$
          $copy\_planner$.search(domain, state, lm.state)
        if $sub\_sol$ is shorter than $shortest\_sol$ then
            $shortest\_sol \leftarrow sub\_sol$
            $used\_lm \leftarrow lm$
            $used\_planner \leftarrow copy\_planner$
        end
    end
    remove $used\_lm$ from $lm\_graph$
    $planner \leftarrow used\_planner$
    $solution \leftarrow shortest\_sol$
end
$solution \leftarrow$
  $planner$.search(domain, state, goal_state)
return $solution$
```

4.2 LM Count: Using landmarks for a pseudo-heuristic

The implementation of LM^{Count} comes in two parts. The pseudo-heuristic itself and a landmark status manager:

The landmark status manager is used to keep track of which landmarks are accepted and which are required again. This is done by keeping a set of landmark IDs for each state, for both the $future$ and $past$ landmarks. Landmarks in the $future$ set are required again and landmarks in the $past$ set are accepted. New visited states always have an empty $future$ set and a $past$ set with all landmarks in them. In the initial state all landmarks that are not true in that state are removed from the $past$ set and added to the $future$ set. Also if a landmark that is true in the current state has a parent, that is not true in the current state we add that landmark to the $future$ set. When computing a new value for a state the landmark status manager will update the $past$ and $future$ sets based on the sets of the previous state. It checks the following items:

- If a landmark was previously in the $future$ and it is not true in this state. The landmark remains in the $future$ set.
- If a landmark was previously not in the $past$ and it is not true in this state. The landmark should not be in the $past$ set.
- If a landmark was true in the previous state but is not true in the current state it is needed again. Thus add it to the $future$ set.
- Always add all landmarks that are true in the goal state to the $future$ set.
- If a landmark has a child with a Greedy Necessary ordering that is not in the $past$ and the landmark is not true in the current state. Add this landmark to the $future$ set.

Algorithm 2: LM Local Smart

```
Data: $planner$
Input: $lm\_graph$, $domain$, $state$, $goal\_state$
$compatibilty\_matrix$
for $i = 0; i <$ length($lm\_graph$)$; i++$ do
    for $j = i + 1; j <$ length($lm\_graph$)$; j++$ do
        $compatibility\_matrix[i][j] \leftarrow$
          interferes(lm_graph[i], lm_graph[j])
    end
end
$solution$
while $lm\_graph$ not empty do
    $sources \leftarrow$ get_sources($lm\_graph$)
    $goal\_groups \leftarrow$
      construct_goal_groups(sources, compatibility_matrix)
    $shortest\_sol, used\_planner$
    for $goal \in goal\_groups$ do
        $copy\_planner \leftarrow planner$
        $sub\_sol \leftarrow$
          $copy\_planner$.search(domain, state, goal)
        if $sub\_sol$ is shorter than $shortest\_sol$ then
            $shortest\_sol \leftarrow sub\_sol$
            $used\_planner \leftarrow copy\_planner$
        end
    end
    $planner \leftarrow used\_planner$
    $solution \leftarrow shortest\_sol$
    for $lm \in sources$ do
        if $lm$ is completed in solution then
            remove $lm$ from $lm\_graph$
        end
    end
end
$solution \leftarrow$
  $planner$.search(domain, state, goal_state)
return $solution$
```

- If a landmark has a parent with a Reasonable ordering that is not in the $past$. Add this landmark to the $future$ set.

The pseudo-heuristic itself only does the computation necessary to obtain the heuristic value. It does this by progressing the landmark status manager based on the current and previous states. Then it gets the $future$ and $past$ sets. Finally it computes $h := n - m + k$. Where:

- n is the total number of extracted landmarks
- m is the total number of *accepted* landmarks in state s. The size of the $past$ set.
- k is the total number of *accepted* landmarks in state s that are *required again*. The size of the intersection between $future$ and $past$.

The pseudo code of the landmark status manager can be seen in Algorithm: 3 and 4. The pseudo code for LM^{Count} can be seen in Algorithm: 5.

Algorithm 3: Landmark Status Manager Initial State

```
Data: $lm\_graph = LandmarkGraph$
Input: $state$
$past\_set \leftarrow$ get_past_set(state)
$future\_set \leftarrow$ get_future_set(state) for
 $lm \in lm\_graph$ do
    if landmark_is_true_in_state(lm, state) then
        for $parent \in lm.parents$ do
            if
             !landmark_is_true_in_state(parent, state)
             then
                add $lm.id$ to $future\_set$
            end
        end
    else
        remove $lm.id$ from $past\_set$
        add $lm.id$ to $future\_set$
    end
end
```

5 Experimentation

In this section of the paper we will touch on the exact assessment criteria for the research question and on the results the experimentation has produced.

5.1 Assessment Criteria

To asses the performance of each of the previously mentioned algorithms, a clear definition of assessment criteria is needed. The base planner we shall compare to is the $A*$ planner with the Goal-Count and Add heuristics. Each planner and heuristic is given three minutes to do three runs over a domain both as a compiled problem and as an interpreted problem. In Julia there is a difference between a compiled and interpreted problem since the language is just-in-time compiled. Thus a compiled problem tends to be solved faster. However, performance of a planner or heuristic can differ between the two

Algorithm 4: Landmark Status Manager Process

```
Data: $lm\_graph = LandmarkGraph$
Input: $state, prev\_state$
$past\_set \leftarrow$ get_past_set(state)
$future_set \leftarrow$ get_future_set(state)
$prev\_past\_set \leftarrow$ get_past_set(prev_state)
$prev\_future\_set \leftarrow$ get_future_set(prev_state)
for $lm \in lm\_graph$ do
    if $lm.id \in prev\_future\_set$ then
        if !landmark_is_true_in_state(lm, state)
         then
            add $lm.id$ to $future\_set$
            if $lm.id \in prev\_past\_set$ then
                remove $lm.id$ from $past\_set$
            end
        else if
         landmark_is_true_in_state(lm, prev_state)
         then
            add $lm.id$ to $future\_set$
        end
    end
    if lm is goal landmark then
        add $lm.id$ to $future\_set$
    end
end
```

Algorithm 5: LM Count heuristic

```
Data: $lm\_graph = LandmarkGraph$
$status\_manager = LandmarkStatusManager$
Input: $state$
$past, future, curr\_true =$
  $status\_manager$.progress(state)
$n =$ size(lm_graph)
$m =$ size(past)
$future\_past =$ intersect(past, future)
$k =$ size(difference(future_past, curr_true))
return $n - m + k$
```

methods. Information stored in the planner is retained between each run. Thus allowing us to see if a planner gets faster if it is asked to solve the same problem again. The total number of solved problems in this time is the first and main performance criteria. The second criteria is the time it took the planner to solve the problem. The domains that will be used as benchmarks are:

- Blocksworld
- FreeCell
- Grid
- Logistics
- Miconic
- Prodogy Blocksworld
- Tireworld

This is a large and diverse set of domains, which will allow us to see in which types of domains the planner and heuristic by Richter et al. will perform the best and worst in. The exact domain specifications and exact problem instances can be found in the GitHub repository [1].

5.2 Results

Running the four planners over all of the proposed domains resulted in a large table [2] of data. The full data table contains information on the following: domain, problem, size of the problem, compiled domain or not, the planner, which run number, number of steps in the solution, time it took to complete (-1 for a timeout), size in bytes of the solution, number of evaluated nodes, number of expanded nodes, number of landmarks in the graph and finally whether or not the solution returned was a correct plan to reach the goal.

To then properly asses our performance criteria we first looked at how many problem instances where solved in each domain. As can be seen in table 1, LM^{Count} performed the worst in this category. While LM^{Local} and $LM^{Local}Smart$ performed similarly to $HAdd$ in this metric.

Secondly we looked at the time it took for a planner to complete the search for a solution. This can be seen in figure 1, figure 2 and in figure 3. First thing to note in figure 1 is that the time scale is logarithmic. Since in very small problem instances it often takes only a fraction of a second to solve the instance it is important to look at the times on this scale.

Second thing to note about both figures is that problem instances that where not completed by a planner are not added to the statistics of this plot. Meaning that both $GoalCount$ and LM^{Count} seem to perform quite well in figure 1. However this is because they simply did not complete the larger problem instances that the other three methods completed. These larger problem instances often take much longer to complete. If it seems like those methods are in the same performance range as $HAdd$, LM^{Local} and $LM^{Local}Smart$, while having only completed small and simple problems

[1]'https://github.com/PaulTervoort/SymbolicPlanners.jl-landmarks' on branch 'dev-bart' in './experiments/logical'

[2]This larger data-set can be found at 'https://github.com/PaulTervoort/SymbolicPlanners.jl-landmarks/tree/dev-bart'

means that they actually perform worse. Figure 2 and figure 3 better show how $GoalCount$ and LM^{Count} only complete the small problem instances.

However there are some specific instances that LM^{Count} is the fastest method to solve a problem [3]. This is most likely due to a small landmark graph that is of high quality. The most likely explanation for its good performance in small instances but quick downfall when the problem becomes larger, lies in that it starts to significantly increase the amount of nodes that are expanded and evaluated.

LM^{Count} is also the only planner that occasionally experienced worse times in subsequent runs over the same problem instance. This can be explained by the fact that it is a pseudo-heuristic, meaning that the way it reached a state matters. Therefore when it is searching for a solution again it can not use its prior knowledge since it might have reached a state differently. Thus it would have a different Heuristic value.

When comparing LM^{Local} and $LM^{Local}Smart$ to $HAdd$ we see that in the medium to small problem instance sizes they are faster in the compiled domains. In the interpreted domains of those same instances they are a little bit slower. Once problem instances started becoming much larger both LM^{Local} and $LM^{Local}Smart$ started performing worse than $HAdd$. This is likely due to the larger computational strain these two methods experience when dealing with the large landmark graphs from these problems.

While LM^{Local} was able to solve more problem instances than $LM^{Local}Smart$, it was slower in almost all of the instances that they both solved. Meaning the upfront cost of computing the larger goals is worth it in terms of time spent. However, instances that $LM^{Local}Smart$ was not able to solve was most likely because that version could still produce goals that are impossible to solve. This is because the interferes method that determines whether or not landmarks can be achieved together is not exhaustive. This seemingly became more of a issue in problem instances with bigger landmark graphs. In those larger landmark graphs there is a higher chance that a mistake is made in the compatibility matrix.

Table 1: Number of problems solved by each planner per domain. The number in parentheses after the domain name is the number of problem instances in that domain.

	GoalCount	HAdd	LM Count	LM Local Smart	LM Local
Blocksworld (75)	18	38	15	31	34
Blocksworld_Prodigy (4)	2	4	1	2	4
Freecell (15)	0	0	0	0	0
Grid (5)	0	1	0	0	0
Tyreworld (2)	1	1	1	1	1
Miconic (77)	54	77	53	76	77

6 Responsible Research

To reproduce the work done in this paper one only needs to have their own computer with access to the internet. The code

[3]Blocksworld: prob01, prob02 and prob03 as compiled domains for instance

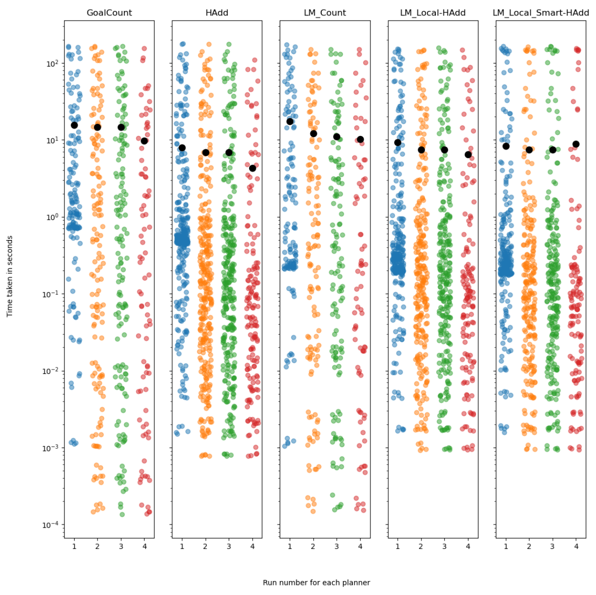


Figure 1: Plot of all times of completed problems per run. The time scale is logarithmic. The number indicates the run iteration over the same problem. The large black point in each run is the mean of that planner in that run

used to generate the results described in the paper will be available for use by anyone [4]. The one difference between the work in this paper and the reproduction by someone else will be the exact machine the code is ran on, thus possibly leading to slight discrepancies in exact times. However the relative relations of those times should stay roughly the same. The specifications of the computer the experiments where ran on in this paper are as follows:

- Processor: Intel(R) Core(TM) i5-6600K CPU @ 3.50GHz
- RAM: 16GB of DDDR4
- GPU: NVIDIA GeForce GTX 1060 6GB

The domains and their problems are all open-source and have all been previously used in the International Planning Competition. These problems are chosen and designed by the IPC to be free of bias as possible and in some cases they have no relation to the real world preventing such issues.

7 Conclusion

Using these results we can now accurately asses the performance of using landmarks as intermediary goals or as a pseudo-heuristic. Both of the LM^{Local} planners perform better then the extremely basic $GoalCount$ heuristic. LM^{Count} does however solve less problem instances than $GoalCount$ but it is significantly faster than it in all cases where it does

[4]https://github.com/PaulTervoort/SymbolicPlanners.jl, specifically on the branch 'dev-bart'

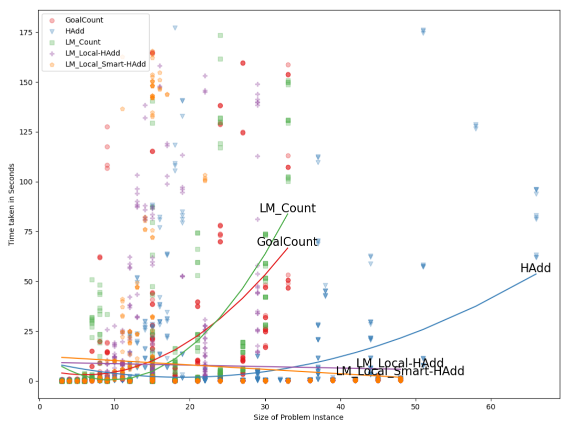


Figure 2: Scatter plot of all times of completed problems versus the problem size. The fitted line indicates the trend of that specific planner. This plot contains all domains.

solve the problem. LM^{Count} is even faster than $HAdd$ in some problem instances. The reason LM^{Count} seems to struggle with larger problem instances is that it very quickly starts to evaluate and expand a very large amount of nodes compared to the other planners.

LM^{Local} and $LM^{Local}Smart$ are not quite as fast as $HAdd$ but are also not significantly slower. In some instances they are even faster than $HAdd$, it is quite possible that given more landmarks and more orderings those two methods could even perform better.

Regarding all of the above conclusions it is important to note that performance of all of our methods greatly depends on the number of landmarks and of their quality. Mistakes in the landmark graph or a lack of nodes in the graph will make LM^{Count} practically useless and LM^{Local} and $LM^{Local}Smart$ will pre-compute quite a few things that are then not used, like the compatibility matrix for $LM^{Local}Smart$, since the landmark graph is almost immediately empty. Since the landmark graph will be very small.

In general our findings are in line to that of Richter et al. [10]. However in their work they do suggest to combine LM^{Count} with other heuristics. Since this was not implemented in this paper we can unfortunately not speak on that topic. Differences that where found can be explained by the difference in framework and programming language.

8 Discussion

Due to some limitations of the SymbolicPlanner framework there are some uncertainties with the implementations of the methods described in this paper. Firstly LM^{Count} is a pseudo-heuristic that needs access to the state it came from to get the heuristic value of that state. However the way that that previous state is currently gotten might not be correct, thus possibly leading to incorrect h-value estimations. Secondly the unfortunate fact that the planners in the SymbolicPlanner framework do not support disjunctive goals led to LM^{Local} and $LM^{Local}Smart$ need to make copies of their internal planner. These need to be deep-copies. Making a deep-copy, especially of a planner that already has done quite a bit of searching, is computationally expensive. Thus hindering performance of those methods. Besides that the interference check for $LM^{Local}Smart$ being non exhaustive and thus sometimes allowing landmarks to be grouped even though they cant be completed together also causes that method to get stuck. Both of these issues could be solved by adding support for disjunctive goals to the SymbolicPlanner framework. Due to time limitations and some mistakes when running the code the last 27 and 24 problem instances of the Blocksworld and Logistics domains where not tested. However all of those problem instance were bigger than the ones before those. Since none of our planners completed a problem instance that was slightly smaller than the problems that are left it can be reasonably concluded that the planners would not have been able to complete these final problems. So in the context of our research question, not having these final results has little to no effect on the conclusion. Unfortunately also due to the lack of time we were unable to run the last 70 problem instances of the Miconic domain. This is specifically very unfortunate since both $HAdd$, LM^{Local} and $LM^{Local}Smart$ were still completing those problems. This means that in that domain we have incomplete data. However we expect a similar trend as with Blocksworld. Finally, an issue with landmark generation for the Freecell domain meant that in that domain only the goal was a landmark. Meaning that our methods don't have any extra information to work with compared to a base planner.

9 Future work

There are five possible continuations on this paper. Firstly improving LM^{Count} and by extension the planner implementation in SymbolicPlanners by adding proper support for passing the previous state to the pseudo-heuristic. Secondly adding support for disjunctive goals to planners in the SymbolicPlanner framework. This would allow for a redo of LM^{Local} that should greatly improve its performance. Thirdly assessing the effect of using different landmark extraction methods to see how exactly the quality and size of a landmark graph affects the methods described in this paper. Fourthly combining LM^{Count} with other heuristics as described by Richter and Westphal in their paper on the LAMA planner[8]. Finally seeing what the effect on performance is of using different planners as the internal planner of LM^{Local} and $LM^{Local}Smart$. Even just changing the heuristic of the current $A*$ planner could be interesting.

10 Acknowledgements

This paper would not have been possible without the following people. Firstly Paul Tervoort for providing me with his implementation of landmark extraction. He has his own paper on this specific topic [12]. Secondly the rest of our research group, Pauline Hengst, Noah Tjoen and Ka Fui Yang, for support and insightful discussions on our shared topic of landmarks. Thirdly Issa Hanou and Sebastijan Dumančić, our supervisor and responsible professor for providing feedback

and answering any questions we had during the course of this project. A big thank you to all of them for helping us in making this paper possible.

References

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

[2] Christer Bäckström and Bernhard Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11:625–656, 01 1995.

[3] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3. Technical report, Department of Electronics for Automation, University of Brescia, Italy, 2005.

[4] Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5):503–535, 2009. Advances in Automated Plan Generation.

[5] J. Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *The journal of artificial intelligence research*, 06 2011.

[6] Ramon Pereira, Nir Oren, and Felipe Meneguzzi. Landmark-based heuristics for goal recognition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017.

[7] Julie Porteous, Laura Sebastia, and Jorg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. *Proc. European Conf. on Planning*, 07 2001.

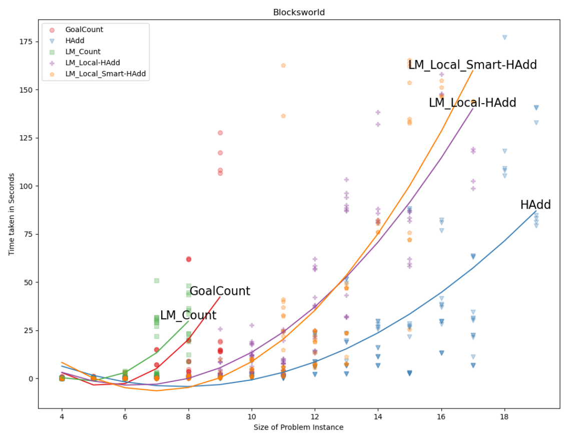
[8] S. Richter and M. Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, September 2010.

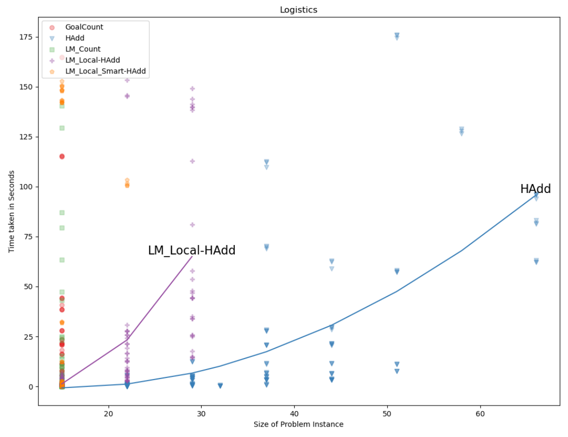
[9] Silvia Richter. Landmark-based heuristics and search control for automated planning. pages 3126–3130, 08 2013.

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

[11] Javier Segovia-Aguas, Sergio Jiménez, Anders Jonsson, and Laura Sebastiá. Scaling-up generalized planning as heuristic search with landmarks, 2022.

[12] Paul Tervoort. Reproducing the concept of ordered landmarks in planning: The effect of ordered landmarks on plan length in forward search, 2024.





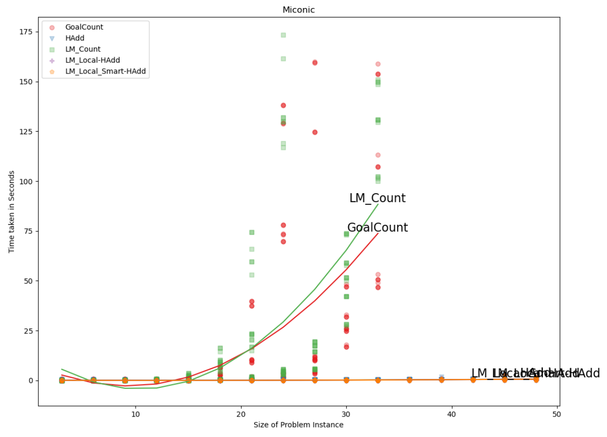


Figure 3: Scatter plots for the Blocksworld, Logistics and Miconic domains. All times of completed problems versus the problem size are featured. The fitted line indicates the trend of that specific planner. The reason not all domains are featured here is because those domains only had 4 or less completed problem instances.