# The Landmark-based Meta Best-First Search Algorithm for Classical Planning.

S. Vernhes, G. Infantes, V. Vidal

*STAIRS 2012*
*K. Kersting and M. Toussaint (Eds.)*

*doi:10.3233/978-1-61499-096-3-336*

# The Landmark-based Meta Best-First Search Algorithm for Classical Planning

Simon VERNHES, Guillaume INFANTES and Vincent VIDAL
*<firstname.lastname@onera.fr> — Onera Toulouse, France*

**Abstract.** In this paper, we revisit the idea of splitting a planning problem into subproblems hopefully easier to solve with the help of landmark analysis. This technique initially proposed in the first approaches related to landmarks in classical planning has been outperformed by landmark-based heuristics and has not been paid much attention over the last years. We believe that it is still a promising research direction, particularly for devising distributed search algorithms that could explore different landmark orderings in parallel. To this end, we propose a new method for problem splitting based on landmarks, which has three advantages over the original technique: it is complete (if a solution exists, the algorithm finds it), it uses the precedence relations over the landmarks in a more flexible way (the orderings are explored by way of a best-first search algorithm), and finally it can be easily performed in parallel (by e.g. following the hash-based distribution principle). We lay in this paper the foundations of a meta best-first search algorithm, which explores the landmark orderings and can use any embedded planner to solve each subproblem. It opens up avenues for future research: among them are new heuristics for guiding the meta search towards the most promising orderings, different policies for expanding nodes of the meta search, influence of the embedded subplanner, and parallelization strategies of the meta search.

**Keywords.** Artificial Intelligence, automated planning, landmarks, search algorithms

## Introduction

Automated Planning in Artificial Intelligence [1] is a general problem solving framework which aims at finding solutions to combinatorial problems formulated with concepts such as actions, states of the world, and goals. For more than 50 years, research in Automated Planning has provided mathematical models, description languages and algorithms to solve this kind of problems. We focus in this paper on Classical Planning, which is one of the simplest model but has seen spectacular improvements in algorithm efficiency during the last decade.

Landmark-based analysis are actually among the most popular tools to build efficient planning systems, either optimal or suboptimal. Landmarks are facts that must be true at some point during the execution of any solution plan, and can be approximated, as well as an ordering between them, in polynomial time [2,3].

Landmarks have been used in two main ways. The most successful one is the definition of heuristic functions to guide a best-first search algorithm, such as the landmark-counting heuristic used in the LAMA suboptimal planner [4] or the LM-Cut heuristic for optimal cost-based planning [5]. An anterior method proposed in [2] was to divide the initial planning problem into successive subproblems whose goals were disjunctions of landmarks to be reached in turn by any kind of embedded planner. This method was not as efficient as using landmark-based heuristics: among the most prominent problems were its incompleteness and its lack of flexibility with respect to an initial ordering of the landmarks.

We aim in this paper to revisit this last method, with two objectives in mind: (1) to devise a complete algorithm for subproblem splitting based on landmarks, and (2) to devise an algorithm that could be easily parallelized in order to benefit from the computational power offered by actual parallel architectures. The algorithm we present in the following has reached these goals, although its performance in a sequential setting is generally worse than that of the subplanner it embeds to solve the successive subproblems (actually, YAHSP [6,7]). Its parallelization is also not studied in this paper, but as it is based on a best-first search algorithm, this would be easily made with the hash-based distribution principle previously used in [8,9].

Roughly speaking, our method consists in performing a best-first search algorithm in the space of landmark orderings, in which node expansion implies the search of a subproblem by an embedded planner. This search algorithm is performed at a meta level, the low level being the search made by the embedded planner that can itself use a best-first search algorithm, such as in YAHSP. After giving some background about classical planning and landmark computation, we define the basic components later used to describe the landmark-based meta best-first search algorithm. We propose several heuristics to guide the meta search, and experimentally evaluate their influence on the planner efficiency. We finally conclude and draw some future works.

## 1. Background on Classical Planning

### *1.1. STRIPS*

The basic STRIPS [10] model of planning can be defined as follows. A *state* of the world is represented by a set of ground atoms. A *ground action* $a$ built from a set of atoms $A$ is a tuple $\langle pre(a), add(a), del(a)\rangle$ where $pre(a) \subseteq A$, $add(a) \subseteq A$ and $del(a) \subseteq A$ represent the preconditions, add effects and del effects of $a$ respectively.

A *planning problem* can be defined as a tuple $\Pi = \langle A, O, I, G\rangle$, where $A$ is a finite set of *atoms*, $O$ is a finite set of ground actions built from $A$, $I \subseteq A$ represents the *initial state*, and $G \subseteq A$ represents the *goal* of the problem. The *application* of an action $a$ to a state $s$ is possible if and only if $pre(a) \subseteq s$ and the resulting state is $s' = (s \setminus del(a)) \cup add(a)$. A *solution plan* is a sequence of actions $\langle a_1, \ldots, a_n\rangle$ such that for $s_0 = I$ and for all $i \in \{1, \ldots, n\}$, the intermediate states $s_i = (s_{i-1} \setminus del(a_i)) \cup add(a_i)$ are such that $pre(a_i) \subseteq s_{i-1}$ and $G \subseteq s_n$. $S(\Pi)$ denotes the set of all solution plans of the planning problem $\Pi$.

We denote $\circ$ the concatenation of two plan, i.e. $\langle a_1, \ldots, a_i \rangle \circ \langle a_j, \ldots, a_k \rangle = \langle a_1, \ldots, a_i, a_j, \ldots, a_k \rangle$.

### *1.2. Landmarks*

All landmark definitions state that *landmarks* are facts that must be true at some point during the execution of any solution plan [3,2]. In this section, we will summarize some types of landmarks, some techniques for finding ordered landmarks and some approaches to exploit them [11].

**Definition 1** (Landmark [2]). *Given a planning problem* $\Pi = \langle A, O, I, G \rangle$*, an atom* $l$ *is a* landmark *for* $\Pi$ *if* $(\forall P \in S(\Pi))(\exists a \in P)\ l \in add(a)$

**Definition 2** (Causal landmark [12]). *Given a planning problem* $\Pi = \langle A, O, I, G \rangle$*, an atom* $l$ *is a* causal landmark *for* $\Pi$ *if either* $l \in G$ *or* $(\forall P \in S(\Pi))(\exists a \in P)\ \ l \in pre(a)$.

Definition 1 and 2 have some subtle differences. The causal landmark definition gives landmarks that are only useful to achieve the goal, whereas the definition 1 gives landmarks which are true at some point in all solution plans even if they are not useful to achieve the goal. For example, in a simple problem with an empty initial state $I = \emptyset$, a problem goal $G = \{\alpha\}$ and only one action with no precondition and two produced atom $\alpha$ and $\beta$, then both $\alpha$ and $\beta$ are landmarks according to definition 1, while only $\alpha$ is a causal landmark. In other words, from a goal point of view, definition 1 can produce irrelevant landmarks.

#### *1.2.1. Landmark Graph*

**Definition 3** (Precedence relation $\leq_{\mathcal{L}}$). *A precedence relation* $\leq_{\mathcal{L}}$ *can be defined on a set of landmarks* $\mathcal{L}$*. It means that* $(\forall (l, l') \in \mathcal{L}^2)$ *if* $l \leq_{\mathcal{L}} l'$ *then* $l$ *should be obtained earlier than* $l'$ *in every solution plan.*

**Definition 4** (Landmark graph $\Gamma$). *Given a set of landmarks* $\mathcal{L}$ *and a precedence relation* $\leq_{\mathcal{L}}$*, let us define* $\Gamma = (\mathcal{V}, \mathcal{E})$*, the corresponding landmark directed graph, where* $\mathcal{V} = \mathcal{L}$ *is the set of vertices and* $\mathcal{E} = \{(l, l') \in \mathcal{L}^2 \mid l \leq_{\mathcal{L}} l'\}$ *the set of edges.*

*We denote* $Pa_\Gamma(l)$ *the set of parents of* $l$ *in the graph* $\Gamma = (\mathcal{V}, \mathcal{E})$*, i.e.* $Pa_\Gamma(l) = \{l' \in \mathcal{V} | (l', l) \in \mathcal{E}\}$*. We also denote* $\mathcal{P}_\Gamma(l)$ *or* $\mathcal{P}(l)$ *when non-ambiguous the set of landmarks in the transitive closure of* $Pa_\Gamma(l)$*, that is the set of parents of* $l$ *and the set of parents of these parents and so on.*

An example of a landmark graph is given in figure 1 (vertices with grey background are atoms in the goal $G$).

We now introduce the following (non-standard) definition that we will heavily rely on for our contribution. First, we denote *root landmarks* of the landmark graph $\Gamma = (\mathcal{V}, \mathcal{E})$ as all vertices in the graph $\Gamma$ with no parents:

**Definition 5** (Root landmark set). $roots(\Gamma) = \{l \in \mathcal{V} \mid Pa_\Gamma(l) = \emptyset\}$

We now define the subgraph $\Gamma \setminus F$ where $\Gamma = (\mathcal{V}, \mathcal{E})$ is a landmark graph and $F$ is a set of landmarks.

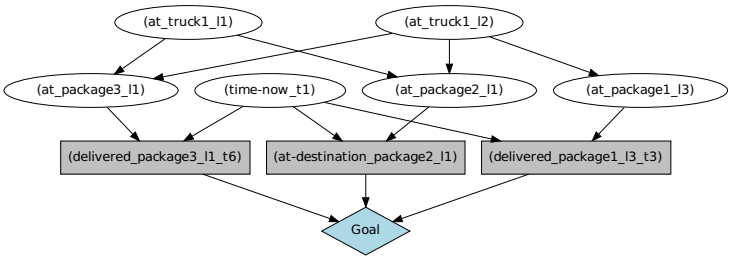



**Figure 1.** An example of a landmark graph (problem 1 in the Trucks domain of the 5$^{\text{th}}$ IPC)

**Definition 6** (Landmark subgraph)**.** $\Gamma \setminus F = (\mathcal{V} \setminus F,\ \{(v, v') \in \mathcal{E} \mid v \notin F \wedge v' \notin F\})$

$\Gamma \setminus F$ is the subgraph of $\Gamma$ build from $\Gamma$ by removing vertices associated to landmarks in $F$ and corresponding edges.

### *1.2.2. Landmark Graph Generation*

All methods proposed to produce such landmark graphs for landmarks [2] and causal landmarks [3] are based on a Relaxed Planning Graph (RPG) of $\Pi$.
Let us define $\Pi^+$, the relaxed problem of $\Pi$, which is obtained by removing the delete effects of each action of $\Pi$. The RPG is the planning graph [13] of $\Pi^+$ until the goal is achieved or until a fixed point is reached (no more atoms are added). More specifically, the RPG is generated layer by layer. First, an atom layer $\lambda_1$ which is the set of all initial atoms is computed. From the first layer, an action layer $\lambda_2$, with all actions $a$ where $pre(a) \subseteq \lambda_1$ is generated. Then, another atom layer is computed: $\lambda_3 = \lambda_1 \cup \bigcup_{a \in \lambda_2} add(a)$, and so on by interleaving action and atom layers until the goal or a fixed point is reached.

By using a forward propagation technique in a pre-computed RPG [3], we can compute a sound and complete causal landmark graph. Let $\Delta_{\lambda_i}(f)$ (respectively $\Delta_{\lambda_i}(a)$) be a set of atoms for each atom $f$ (respectively action $a$) of the layer $\lambda_i$ called label of atom $f$ at layer $\lambda_i$ (which will contain the causal landmarks for the current atom). For the first layer $\lambda_1$, the label corresponds to its atoms: $(\forall f \in \lambda_1)\ \ \Delta_{\lambda_1}(f) = \{\ f\ \}$. Then, for each other layers:

$$\begin{cases} \text{action layer:} & (\forall i, \text{ even})(\forall a \in \lambda_i)\ \ \Delta_{\lambda_i}(a) = \bigcup_{f \in \lambda_i - 1} \Delta_{\lambda_i - 1}(f) \\ \text{atom layer:} & (\forall i, \text{ odd} > 1)(\forall f \in \lambda_i)\ \ \Delta_{\lambda_i}(f) = f \cup \bigcap_{a \in \lambda_i - 1} \Delta_{\lambda_i - 1}(a) \end{cases}$$

The union of all labels of the goal atoms at the last layer are the sound and complete causal landmarks (when the RPG is computed until a fixed point). By nature (propagation through the RPG), this method gives an *acyclic* landmark graph.

Finding all landmarks from definition 1 and ordering them is harder: it has been proven to be PSPACE-Complete [2]. Thus practical methods for finding landmarks are incomplete and unsound but various relaxed versions of these landmarks and various ways to order them have been discussed in [2].

### *1.3. Related Work on Using Landmarks*

Previous approaches used landmarks in two different ways. One approach is computing heuristics. For example, the LAMA heuristic [4] estimates the remaining number of landmarks to reach for a state $s$ and a plan $\rho$: $h^{lama}(s,\rho) = |\mathcal{L} \setminus (Accepted(s,\rho) \setminus ReqAgain(s,\rho))|$, where $\mathcal{L}$ is the set of all the landmarks, $Accepted(s,\rho)$ is the set of landmarks already reached at state $s$ through the plan $\rho$, and $ReqAgain(s,\rho)$ is the set of required again landmarks (already accepted landmarks, but required again to reach another landmark).

Another approach is to split a planning problem into subproblems. Disjunctive Search Control (DSC) [2] is a search control algorithm based on the landmark graph. It runs a subplanner on the problem $\Pi$ whose goal is the disjunction of the leafs of the landmark graph or $G$. If the subplanner finds a valid plan, then the found landmark is removed from the landmark graph and the algorithm iterates (the reached state is used as the new initial state) until the landmark graph is empty. Finally, the subplanner is called a last time with $G$ as goal.

## 2. The Landmark-based Meta Best-First Search (LMBFS) Algorithm

Our approach is based on the DSC idea [2] which splits a general STRIPS problem into subproblems using a landmark graph. This choice is motivated because we think that DSC could be enhanced by using a more flexible exploitation of the landmark graph. LMBFS performs a best-first search algorithm in the space of landmarks ordering.

In the following, the landmark graph is generated using causal landmarks. Thus, LMBFS relies on the acyclicity and soundness of the landmark graph.

### *2.1. Metanode and Associated Planning Problem*

Given a planning problem $\Pi = \langle A, O, I, G\rangle$, a corresponding landmark set $\mathcal{L}$ and a set of landmarks $F$, we define a metanode as the following:

**Definition 7** (Metanode)**.** *A* metanode *is a tuple* $m = \langle s, h, F, l, \rho\rangle$ *where:*

- $s$ *is a state of the planning problem* $\Pi$*;*
- $h$ *is a heuristic evaluation of the node;*
- $F$ *is a set of landmarks* ($F \subseteq \mathcal{L}$)*;*
- $l$ *is a landmark* ($l \in \mathcal{L}$)*;*
- $\rho$ *is a solution plan from the initial state* $I$ *to the state* $s$*.*

We now define the action restriction associated to a landmark subgraph:

**Definition 8** (Landmark subgraph action restriction)**.** *For a problem* $\Pi$ *and a metanode* $m = \langle s, h, F, l, \rho\rangle$*, we define* $ops_\Gamma(l, F) = \{a \in O \mid l \in add(a) \vee add(a) \cap roots(\Gamma \setminus F) = \emptyset\}$.

In other words, $ops_\Gamma(l, F)$ is the set of ground actions which does not produce any root landmark of the subgraph $\Gamma \setminus F$ except $l$. We can see here that $F$ is used as a set of forbidden landmarks.

Finally, a metanode $m$ defines a planning (sub-)problem in the following way:

**Definition 9** (Metanode-associated planning problem)**.** *The planning problem associated to a metanode* $m = \langle s, h, F, l, \rho \rangle$ *is* $\Pi(m) = \langle A, ops_\Gamma(l, F), s, l \rangle$

We consider the planning problem where $s$ is the initial state, $A$ is the set of ground atoms of the initial problem $\Pi$, $l$ is the goal. The set of ground actions $ops(l, F)$ is a subset of $O$ computed using the landmark graph, used to forbid some actions. The restriction of the possible actions of the subproblem is motivated by the fact that for a given metanode, we want to be able to force the search to achieve a given landmark $l$ and not any other one. The generation of subproblems and particularly action restriction is delegated to the generation of metanodes itself.

### *2.2. Expansion of Metanodes*

There are several ways to generate sons of a metanode. Let us recall that a metanode $m = \langle s, h, F, l, \rho \rangle$ defines a problem starting from $s$ and focusing on achievement of landmark $l$ by forbidding achievement of any landmark in $F$.

#### *2.2.1. First Approach*

The first version tries to follow the landmark graph $\Gamma$ as close as possible. The idea is when the goal landmark of the metanode can be reached, to generate sons that will try to reach one of the remaining root landmarks in the landmark graph $\Gamma$.

We thus define the *nextLM* operator as:

**Definition 10** (Next landmarks metanode generation)**.** $\text{nextLM}(\langle s, h, F, l, \rho \rangle) = \{\langle s', h', F \cup \{l\}, l', (\rho \circ \rho') \rangle \mid \rho' \neq \bot \wedge l' \in roots(\Gamma \setminus (F \cup \{l\}))\}$ *where, if a solution plan from* $s$ *to* $l$ *exists (if not, nothing is generated):*

- $\rho'$ *the solution plan from* $s$ *to* $l$*;*
- $s'$ *the state obtained by applying* $\rho'$ *to* $s$*;*
- $h'$ *is the heuristic evaluation of the new metanode, discussed in section 2.4.*

In other words, in a metanode $m$, we try to reach the landmark $l$; and if there is a plan, we generate metanodes by looking at next landmarks in the landmark graph. The achieved landmark becomes forbidden, and the partial plan is updated accordingly. But, even if the landmark graph $\Gamma$ is sound and complete, using only this nextLM operator for metanode generation makes the algorithm incomplete, as shown in following counter-example.

Let us consider the example in Figure 2 where circles are atoms, squares are actions, arrows mean consumption or production of an atom and dashed arrows mean deletion of an atom. The initial state is $\{a, f, d\}$ and the goal set is $\{c\}$. As we can see, $g$ and $c$ are landmarks, and $g$ has to be reached before $c$. If we only have metanodes generated by nextLM, then the first metanode will have the landmark $g$ as a goal. The subplanner can give the simple plan $\langle \alpha \rangle$ (which is valid and optimal for this subproblem). Only one metanode will be added to the open list for the state $\{f, g\}$ and $\{c\}$ as a goal, which is an impossible problem. Then, the loop stops (no more metanode to explore).

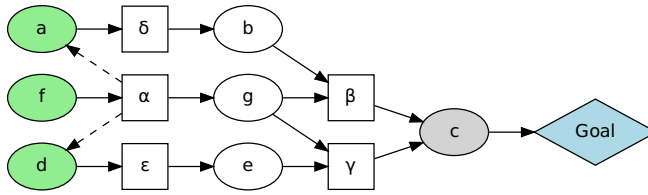



**Figure 2.** Planning Graph of "open metanode" problem

*2.2.2. Cut-parents Metanode Generation*

We then introduce other metanode generators, in order to take into account only a subpart of the landmark graph $\Gamma$.

**Definition 11** (Cut-parents metanode generation)**.** cutParent$(\langle s, h, F, l, \rho\rangle) = \{\langle s', h', F \cup \mathcal{P}(l'), l', (\rho \circ \rho')\rangle \mid \rho' \neq \bot \wedge l' \in roots(\Gamma \setminus (F \cup \{l\}))\}$ *where, if a solution plan from s to l exists (if not, nothing is generated):*

- $\rho'$ *is the solution plan from s to l;*
- $s'$ *is the state obtained by applying* $\rho'$ *to s;*
- $\mathcal{P}(l')$ *denotes the set of landmarks in the transitive closure of* $Pa_\Gamma(l')$*;*
- $h'$ *is the heuristic evaluation of the new metanode, discussed in section 2.4.*

A variant is the restartCutParent metanode generation, defined as :

**Definition 12** (Restart cut-parents metanode generation)**.** restartCutParent$(\langle s, h, F, l, \rho\rangle) = \{\langle I, h', F \cup \mathcal{P}(l'), l', \emptyset\rangle \mid l' \in roots(\Gamma \setminus (F \cup \{l\}))\}$ *where:*

- $I$ *is the initial state of the original planning problem;*
- $\mathcal{P}(l')$ *denotes the transitive closure of* $Pa_\Gamma(l')$*;*
- $h'$ *is the heuristic evaluation of the new metanode, and will be discussed in section 2.4.*

The idea is that sometimes, a total order constructed on the partial order defined by the landmark graph is too restrictive, like in the counter example, and one may skip some landmarks and just try to achieve landmarks in the graph in a "depth-first" way, ignoring landmarks that should be achieved before.

*2.2.3. Delete Landmark Metanode Generation*

Finally, we introduce the very generic landmark deletion operator, meaning that the metanode will be generated as if the landmark simply did not exist:

**Definition 13** (Delete landmark metanode generation)**.** deleteLM$(\langle s, h, F, l, \rho\rangle) = \{\langle s, h', F \cup \{l\}, l', \rho\rangle \mid l' \in roots(\Gamma \setminus (F \cup l))\}$ *where* $h'$ *is the heuristic evaluation of the new metanode.*

This operator simply "skips" a landmark, and will cause the main search to directly try to achieve a "following" landmark. One can see that applying this last

operator enough times on the first metanode (that has $I$ as initial state) simply "empties" the landmark graph, eventually giving a metanode with the original planning problem. Another important point is that the cut-parents operator is a shortcut for several delete landmark operators, guided by the $Pa_\Gamma$ relation.

### *2.3. Algorithm*

LMBFS (see Algorithm 1) is a best-first search algorithm with deferred heuristic evaluation [14] where nodes are the previously defined metanodes. The heuristic evaluation of the metanodes are not computed upon generation but instead they are inserted into the open list with the heuristic evaluation of their parent.

**Algorithm 1:** LMBFS

```
   input : STRIPS problem Π = ⟨A, O, I, G⟩, landmark graph Γ
   output: solution plan
 1 open ← ∅; closed ← ∅;
 2 ∀l ∈ roots(Γ) : add ⟨I, h, ∅, l, ∅⟩ to open;
 3 while open ≠ ∅ do
 4     m ← arg min_{⟨s,h,F,l,ρ⟩∈open} h;
 5     open ← open \ {m};
 6     if m ∉ closed then
 7         closed ← closed ∪ {m};
 8         ρ' ← subplanner(Π(m));
 9         if ρ' ≠⊥ then
10             s' ← result of executing ρ' in s;
11             if G ⊆ s' then                  /* Global goal G found ? */
12                 return ρ ∘ ρ';
               /* Node expansion, see section 2.2                      */
13             open ← open ∪ successors(m);
14 return subplanner(Π);
```

First, the algorithm adds the metanodes associated to each root landmark of $\Gamma$ in the open list. Then, at each iteration of the loop, the algorithm extracts the best metanode $m$ from the open list, and runs a subplanner on the associated subproblem $\Pi(m)$. If the subplanner returns a valid plan, then the metanode $m$ is expanded by adding its *successors* to the open list. Next, the algorithm iterates until the open list is empty or the global goal $G$ has been reached. Eventually, if $G$ has not been reached before the end of the process, LMBFS runs the subplanner a last time on the global problem $\Pi = \langle A, O, I, G\rangle$.

The set successors($m$) (Algorithm 1 line 13) is the set obtained by an operator or the union set of several operators described in section 2.2. In our current implementation, successors($m$) = nextLM($m$)$\cup$restartCutParent($m$) (because we want to use nextLM and be sure to have the completeness; and restartCutParent was the first operator we thought about to do so).

### 2.4. Heuristics for Metanode Selection from the Open List

One way to improve the algorithm effectiveness is to select the most promising metanode to expand from the open list. Two simple approaches have been implemented, yet many variations and new possibilities could be envisaged.

The first one, inspired by the landmark-counting heuristic of LAMA [4], uses the landmark graph $\Gamma$ and counts the remaining landmarks to be reached. The metanode with the least number of remaining landmarks is chosen. This heuristic is not admissible because even if the landmarks are sound, one action can achieve more than one landmark. We will refer to this heuristic as $h^{\mathcal{L}_{left}}$.

**Definition 14** ($h^{\mathcal{L}_{left}}$ for metanodes)**.** *For a metanode $m = \langle s, h, F, l, \rho \rangle$ and an associated landmark graph $\Gamma = (\mathcal{V}, \mathcal{E})$, the heuristic $h^{\mathcal{L}_{left}}$ is defined by $h^{\mathcal{L}_{left}}(m) = |\mathcal{V} \setminus F|$.*

Another approach is to compute a standard heuristic on the starting state of the metanode. We decided to use the well-known non admissible heuristic $h^{add}$, as it is the one employed in our actual subplanner to order states in its open list.

**Definition 15** ($h^{add}$ heuristic [15] for metanodes)**.** *Let us define $h^{add}$ for each possible state and for each atom:*

$$\begin{cases} (\forall s \in 2^A) & h^{add}(s) = \sum_{f \in s} h^{add}(f) \\ (\forall f \in A) & h^{add}(f) = min_{a \in O}\{h^{add}(f), 1 + h(pre(a))\} \end{cases}$$

*For a metanode $m = \langle s, h, F, l, \rho \rangle$, the heuristic $h^{add}$ is $h^{add}(m) = h^{add}(s)$.*

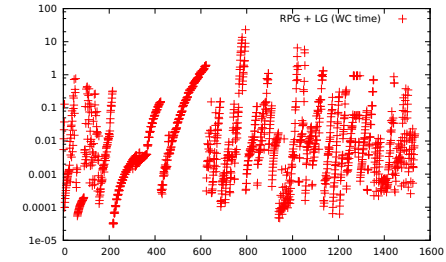
### 2.5. Subplanner Embedded in LMBFS

For subproblem resolution, we chose YAHSP [7] for several reasons.

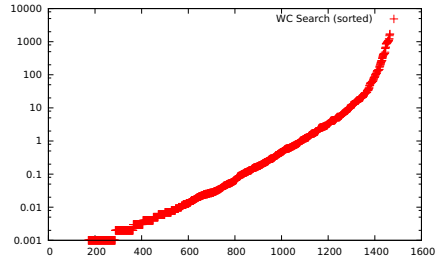
Firstly, because a planner already using landmarks such as LAMA is (hopefully) not useful in our context, because LMBFS tries to navigate from landmark to landmark by forbidding to reach landmarks which are not its current goal. Generally, subproblems contain very few landmarks not discovered by our landmark generation procedure and most of the time, there are none. So a landmark-based subplanner would work blindly in its space search. Besides, the extra landmarks that might be found on the subproblems should be used to feed directly the LMBFS algorithm, thus splitting even more the global problem $\Pi$.

Secondly, because the successive subproblems solved during metanode expansion should, and generally are, easy to solve with very few lookaheads computed in YAHSP. Moreover, directly embedded in the form of a C library, YAHSP does not require any preprocessing when faced with a new subproblem extracted from a global planning problem. It can thus generally answer very fast. It has also already been embedded with some success in another planner based on evolutionary algorithms [16].

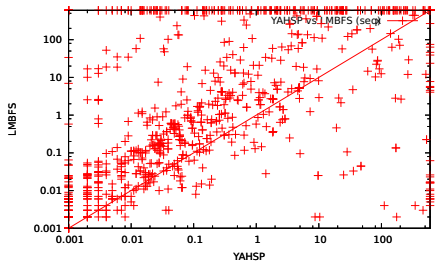
Thirdly, because a parallel version of YAHSP already exists [9], which uses the hash-based distribution principle we intend to employ in future works for parallelizing LMBFS. The evaluation of this parallelization will then be more thorough thanks to a comparison of both approaches.



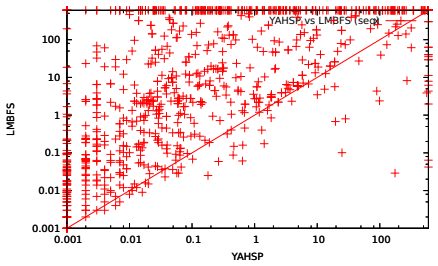

(a) Landmark graph generation WC time




(b) WC search time




(c) WC resolution time for YAHSP and LMBFS with $h^{\mathcal{L}_{left}}$




(d) WC resolution time for YAHSP and LMBFS with $h^{add}$

**Figure 3.** Experimental results

## 3. Experimental Evaluation

We conducted a set of experiments with 1794 benchmarks from the 1st to the 7th International Planning Competition (IPC) within a 30 minutes CPU time limit. The experiments were all run on an Intel X5670 processor (using only one core as it is a sequential algorithm) running at 2.93Ghz with 24GB of RAM. In the next figures, each plot will represent an IPC problem.

On a subset of these planning tasks (from the 3rd to the 7th IPC), YAHSP, the subplanner used by LMBFS, resolve 1026 out of 1163 problems (88.2%) within a 10 minutes CPU time limit.

### *3.1. Efficiency of Landmark Graph Generation*

As we can see in figure 3(a), the computation time is less than one second for most problems. It takes longer for large problems like the nontemporal STRIPS airport problem (4th IPC) because the size of the computed RPG is high (128 layers for the biggest problem). LMBFS is designed to be a suboptimal algorithm (i.e. it not necessarily outputs the optimal solution but answers as fast as possible). So as it is now, computing the landmark graph on the initial state is acceptable. But it cannot be processed for each metanode during search (for example, to enhance the value of a heuristic).

### *3.2. Efficiency of LMBFS with the $h^{\mathcal{L}_{left}}$ Heuristic*

Using the $h^{\mathcal{L}_{left}}$ heuristic, LMBFS solves 1466 out of the 1794 problems (nearly 81.7%) under 30 minutes. Figure 3(b) shows the Wall-Clock (WC) time for all the

problems (sorted out by increasing WC time). Figure 3(c) shows a comparison of the WC time of LMBFS and the subplanner we used (YAHSP [7]) launched on the global problem $\Pi$ (below $y = x$, LMBFS was faster than YAHSP, and above vice versa). As we can see, most of the problems quickly solved by YAHSP (under 0.1s) are solved by LMBFS nearly as fast. The slow down probably comes from the landmark graph generation which induces a non-amortized overhead for small problems. For larger problems, we can see that LMBFS sometimes improves the speed of YAHSP and sometimes finds a solution where YAHSP did not. But it also does worse on a large part of the problems (as we can see on top of the figure). Even if these results are not a real improvement compared to YAHSP itself, we believe it is a good start. Moreover, the $h^{\mathcal{L}_{left}}$ is a really simple and probably not truly informative heuristic. Thus, using an appropriate one might greatly enhance the LMBFS algorithm.

### *3.3. LMBFS with $h^{add}$*

Using the $h^{add}$ heuristic, LMBFS solves 1382 out of the 1794 problems (nearly 77%) under 30 minutes. Figure 3(d) shows a comparison of the WC time of LMBFS and YAHSP [7]. Here the results are clearly in favor of YAHSP which outperformed in most of the problems. The $h^{add}$ heuristic is also the one used by YAHSP during its state space search, so it is redundant to use it in our landmark-based metasearch planner. Moreover, using a landmark-based heuristic (eventually in combination with a standard heuristic like $h^{add}$) could be more informative for this kind of search which is based on the landmark graph. One more problem about using the $h^{add}$ heuristic is that it gives the same heuristic value for any son a metanode because the initial states of all sons of a metanode are the same. One way to differentiate these metanodes would be to run $h^{add}$ on the landmark instead of on the global goal $G$.

## 4. Conclusion and Future Works

In this paper have been presented several contributions towards a new landmark-based planning algorithm. First, we propose a sound framework for a (meta)search based on the order of landmarks, given a landmark graph. We formalize the link between so-called metanodes and subproblems of the original planning problem, including restrictions on the allowed actions themselves. We give several operators that allow to explore different orders for using landmarks as subgoals, including skipping some. We also propose a first approach for evaluating heuristic values of such metanodes, or equivalently giving priorities to subproblems. We put everything together in a (deferred) best-first search algorithm, leading to a complete algorithm. Last but not least, we implemented the whole thing and give preliminary results.

From now on, several leads will be followed.

A key point for performance is the heuristic evaluation of metanodes, linked to the operators used for generation. For instance, nextLM-generated nodes are always evaluated before restartCutParent-generated ones, which is not necessarily

good. We believe that in order to have a more informed heuristic, the landmark subgoal has to be used for heuristic evaluation, as for now only the landmark of the parent (more or less the starting state of the node) is used, leading to poorly discriminating heuristic values.

Another point is the operators used. While deleteLM is very general, cutParent can be seen as special case (a shortcut for a given sequence of deleteLM, or said differently, a lookahead in the landmark graph itself ), and other special cases may be very useful.

Another next step will focus on (and indeed is a primary objective of the algorithm design) the modification of the LMBFS algorithm to make it distributed for execution on new parallel architectures. The objective is to integrate ideas of the HDA*[8] algorithm into the LMBFS algorithm. The idea behind HDA* is to distribute the nodes among the processing units based on a hash key computed from planning states (in our case metanodes).

## References


[1] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning, theory and practice*. Morgan-Kaufmann, 2004.

[2] J. Hoffmann, J. Porteous, and L. Sebastia, "Ordered landmarks in planning," *Journal of Artificial Intelligence Research*, vol. 22, pp. 215–278, 2004.

[3] E. Keyder, S. Richter, and M. Helmert, "Sound and complete landmarks for and/or graphs," in *Proc. of Euro. Conf. on Artificial Intelligence (ECAI)*, pp. 335–340, 2010.

[4] S. Richter, M. Helmert, and M. Westphal, "Landmarks revisited," in *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 975–982, 2008.

[5] M. Helmert and C. Domshlak, "Landmarks, critical paths and abstractions: What's the difference anyway?," in *Proc. ICAPS*, 2009.

[6] V. Vidal, "A lookahead strategy for heuristic search planning," in *Proc. ICAPS*, pp. 150–159, 2004.

[7] V. Vidal, "YAHSP2: Keep it simple, stupid," in *Proc. of the 7th International Planning Competition (IPC'11)*, 2011.

[8] A. Kishimoto, A. S. Fukunaga, and A. Botea, "Scalable, parallel best-first search for optimal sequential planning," in *Proc. ICAPS*, 2009.

[9] V. Vidal, S. Vernhes, and G. Infantes, "Parallel AI planning on the SCC," in *Proc. of the 4th Symposium of the Many-core Applications Research Community (MARC)*, 2011.

[10] R. Fikes and N. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1972.

[11] J. Zhao and D. Liu, "Recent advances in landmarks research," in *Progress in Informatics and Computing (PIC)*, vol. 1, pp. 238–241, 2010.

[12] L. Zhu and R. Givan, "Landmark extraction via planning graph propagation," in *ICAPS Doctoral Consortium*, pp. 156–160, 2003.

[13] A. Blum and M. Furst, "Fast planning through planning graph analysis," *Artificial intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.

[14] S. Richter and M. Helmert, "Preferred operators and deferred evaluation in satisficing planning," in *Proc. ICAPS*, pp. 273–280, 2009.

[15] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1, pp. 5–33, 2001.

[16] J. Bibaï, P. Savéant, M. Schoenauer, and V. Vidal, "An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning," in *Proc. ICAPS*, pp. 18–25, 2010.