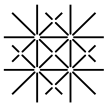

University of Basel

Landmark-based Meta Best-first Search

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
http://ai.cs.unibas.ch/

Examiner: Malte Helmert
Supervisor: Gabriele Röger

Samuel Hugger
s.hugger@unibas.ch
2014-057-442

27.06.2017

Abstract

This thesis deals with the algorithm presented in the paper "Landmark-based Meta Best-First Search Algorithm: First Parallelization Attempt and Evaluation" by Simon Vernhes, Guillaume Infantes and Vincent Vidal. [1]
Their idea was to reconsider the approach to landmarks as a tool in automated planning, but in a markedly different way than previous work had done.
Their result is a meta-search algorithm which explores landmark orderings to find a series of subproblems that reliably lead to an effective solution. Any complete planner may be used to solve the subproblems.
While the referenced paper also deals with an attempt to effectively parallelize the Landmark-based Meta Best-First Search Algorithm (in short referred to as LMBFS), this thesis is concerned mainly with the sequential implementation and evaluation of the LMBFS algorithm in the Fast Downward planning system [2].

Table of Contents

1 Introduction

Landmark-based methods have been a very popular means for problem solving in automated planning. Landmarks represent steps that are requirements for every solution. More formally, landmarks are facts that have to be true at some point of every solution plan.
The two main ways in which landmarks have been used in the past are landmark-based heuristics and landmark-based meta-search. While the approach of using landmarks as part of heuristic functions has been proven to be successful, such as the LM-count heuristic [3] or the LM-cut heuristic [4], the approach of splitting the original task into subtasks which aim to achieve individual landmarks has been lagging behind, suffering from it's incompleteness and lack of flexibility in landmark orderings.
The paper "Landmark-based Meta Best-First Search Algorithm" [1] presents an exploration of the second approach that is not only complete, but also considerably more flexible and thus applicable to a broader range of planning problems.

2
Body of the Thesis

2.1 STRIPS Planning Model

The *STRIPS planning model* is represented by states to describe the world and actions to change the world. A *state* consists of a set of *atoms*, each of which can be either *true* or *false*. An *action* a is a tuple $\langle pre(a), add(a), del(a)\rangle$, where *pre(a)* represents the *preconditions* of a, *add(a)* contains the atoms that are made true by a, and *del(a)* contains the atoms that are made false by a.

A *planning task* is defined as a tuple $\Pi = \langle A, O, I, G\rangle$. A is a finite set of atoms describing the world, O is a finite set of actions which manipulate atoms, $I \subseteq A$ represents the *initial state* and $G \subseteq A$ represents the *goal condition*.

An action a is *applicable* to a state s if and only if $pre(a) \subseteq s$ and the resulting state $s' = (s \setminus del(a)) \cup add(a)$. A *plan* is a sequence of actions $\langle a_1, \ldots, a_n\rangle$ such that for a sequence of states $\langle s_0, \ldots, s_n\rangle$, 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 $pre(a_i)) \subseteq s_{i-1}$. A *solution plan* is a plan for which $s_0 = I$ and $G \subseteq s_n$. $S(\Pi)$ denotes the set of all solution plans of Π. The symbol $\circ$ represents the concatination of two plans, i.e. $\langle a_1, \ldots, a_i\rangle \circ \langle a_j, \ldots, a_n\rangle = \langle a_1, \ldots, a_i, a_j, \ldots, a_n\rangle$.

2.2 Landmark Graph

Landmarks are facts that have to be true at some point of every solution plan. Because of this, we can assume that the achievement of landmarks generally brings us closer to the solution of the task.

While landmark-based heuristics such as the LM-count heuristic [3] do not care about the order of landmark achievement, the approach of LMBFS is firmly based on ordering the landmarks in a way that is benefitial to the effectiveness of the algorithm. The landmark graph represents the space in which we perform the meta-search.

The LMBFS algorithm uses the following Definitions on landmarks and the landmark graph:

Definition 1. (Causal landmark). [1]
Given a planning task $\Pi = \langle A, O, I, G \rangle$, an atom l is a *causal landmark* for Π if either $l \in G$ or $\forall \rho \in S(\Pi), \exists \alpha \in \rho : l \in pre(a)$.

This means that a causal landmark l is either part of the solution state, or it is a precondition of at least one action in every solution plan - either way, it is necessary to achieve all causal landmarks if we are interested in finding a solution.

Definition 2. (Precedence relation $<_{\mathcal{L}}$). [1]
$<_{\mathcal{L}}$ can be defined on a set of landmarks $\mathcal{L}$. For two landmarks $(l, l') \in \mathcal{L}^2$, $l <_{\mathcal{L}} l'$ if l becomes true before l' becomes true during the execution of every solution plan.

Roughly speaking, the precedence relation orders a set of landmarks according to their order of achievement in $S(\Pi)$.

Definition 3. (Landmark graph Γ). [1]
Given a set of landmarks $\mathcal{L}$ and a precedence relation $<_{\mathcal{L}}$, we define $\Gamma = (\mathcal{V}, \mathcal{E})$, the corresponding directed *landmark graph* where the set of vertices $\mathcal{V} = \mathcal{L}$ and the set of edges $\mathcal{E}$ is the transitive reduction of the graph $(\mathcal{V}, \{(l, l') \in \mathcal{L}^2 \mid l <_{\mathcal{L}} l'\})$.

We build the landmark graph following the precedence relations between landmarks, so that following the landmark graph already yields an ordering of landmarks that is helpful with respect to achieving the goal.

Definition 4. (Relatives of a landmark l). [1]
According to the landmark graph Γ, we denote $Pa_{\Gamma}(l)$ the set of *parents* of l, $Ch_{\Gamma}(l)$ the set of *children* of l, and $\mathcal{P}_{\Gamma}(l)$ the set of *ancestors* of l.

Definition 5. (Root landmark set). [1]
Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a landmark graph. We define $roots(\Gamma) = \{l \in \mathcal{V} \mid Pa_{\Gamma}(l) = \emptyset\}$.

Root landmarks of a landmark graph Γ are landmarks that have no parents. Because Γ is built following the precedence relations, this means that the root landmarks of Γ are likely to appear early in every solution plan, which makes them a good starting point for a meta-search on the landmark graph.

Definition 6. (Landmark subgraph). [1]
Let $\Gamma = (\mathcal{V}, \mathcal{E})$ be a landmark graph and $\mathcal{A}$ be a set of landmarks. The *landmark subgraph* is defined as follows: $\Gamma \backslash \mathcal{A} = (\mathcal{V} \backslash \mathcal{A}, \{(v, v') \in \mathcal{E} \mid v \notin \mathcal{A} \wedge v' \notin \mathcal{A}\})$.
$\mathcal{A} \in \mathcal{L}$ is the set of achieved landmarks.

This subgraph is obtained by removing from the landmark graph all vertices associated to achieved landmarks, as well as all edges that are incident to at least one such vertex.
In other words, we remove all achieved landmarks and connections to them from the landmark graph.

This is a transformation of the landmark graph that we will apply regularly in order to reduce the size of the landmark graph. It is important to note that not all landmarks that have been added to the set of achieved landmarks have actually been achieved. This is because the final algorithm will terminate only if the landmark graph is empty, as no more metanodes will be generated at that point. However, we do not always want to achieve all landmarks - sometimes it is benefitial to skip single landmarks, especially if their achievement would make more important steps of a solution plan impossible. We skip landmarks by adding them to the set of achieved landmarks.

2.3 Metanode and Associated Planning Task

Definition 7. (Metanode). [1]
A *metanode* is a tuple $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ where:

- s is a state of the planning task Π
- h is a heuristic evaluation of the node
- $\mathcal{A}$ is a set of landmarks ($\mathcal{A} \subseteq \mathcal{L}$)
- l is a landmark ($l \in \mathcal{L}$)
- ρ is a plan yielding the state s from the initial state I.

In a metanode, s is the state we are currently in, ρ is how we got to that state, $\mathcal{A}$ is the set of already achieved landmarks, and l is the landmark we wish to achieve in the planning task associated with this metanode.

Definition 8. (Metanode-associated planning task). [1]
The *planning task associated to a metanode* $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ is defined as $\Pi_m = \langle \mathcal{A}, ops_\Gamma(l, \mathcal{A}), s, \{l\} \rangle$.
$ops_\Gamma(l, \mathcal{A})$ is a subset of O defined as follows:
Let $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ be a metanode. $ops_\Gamma(l, \mathcal{A}) = \{a \in O \mid (l \in add(a)) \vee (add(a) \cap roots(\Gamma \backslash \mathcal{A}) = \emptyset)\}$.

In such a planning task, we consider s the initial state, and the achievement of $\{l\}$ the goal. $\mathcal{A}$ is the set of atoms of Π, and $ops_\Gamma(l, \mathcal{A})$ is a subset of the actions O of Π. The actions that remain allowed are all actions that either achieve the landmark l, or that do not achieve any root landmarks. This action restriction focuses the search on l.
In this algorithm, a metanode-associated planning task will usually represent only a part of the original planning task, which we will refer to as subtask.

2.4 Expansion of Metanodes

At the core of the LMBFS algorithm are the metanode generation methods - they work as successor functions that determine the way the algorithm moves through the landmark

graph, and thus are very much make-or-break for the efficiency and usefulness of this algorithm.
As the planning task associated to a metanode m focuses on reaching a landmark l, we also call m the metanode associated to the landmark l.
All metanode generation methods take a metanode m as input and add new metanodes to the open list.

Definition 9. (nextLM metanode generation). [1]
Let $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ be a metanode. If Π_m has a solution ρ', then *nextLM(m)* $= \{\langle s', h', \mathcal{A} \cup \{l\}, l', (\rho \circ \rho')\rangle \mid l' \in \mathit{roots}(\Gamma \backslash (\mathcal{A} \cup \{l\}))\}$. If Π_m has no solution, then *nextLM(m)* $= \emptyset$.

nextLM is straight-forward: it tries to achieve the landmark l. If that is possible, it adds l to the set of achieved landmarks, updates the plan accordingly and generates metanodes corresponding to all root landmarks of the landmark graph. This means that the search stays focused on root landmarks. If l cannot be reached, *nextLM* generates no metanodes. By using only *nextLM* in our algorithm, we follow the landmark graph as closely as possible. But with this approach the algorithm is incomplete, as it is possible to run into dead ends in situations where achievement of one landmark renders the achievement of another landmark impossible. Therefore three other successor functions have been defined alongside *nextLM*, which allow to skip landmarks in various ways. This addition makes the algorithm complete.

Definition 10. (deleteLM metanode generation). [1]
Let $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ be a metanode. *nextLM(m)* $= \{\langle s, h', \mathcal{A} \cup \{l\}, l', \rho\rangle \mid l' \in \mathit{roots}(\Gamma \backslash (\mathcal{A} \cup \{l\}))\}$.

deleteLM removes a landmark from the landmark graph and focuses the search on the other root landmarks.

Definition 11. (cutParents metanode generation). [1]
Let $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ be a metanode. If Π_m has a solution ρ', then *cutParents(m)* $= \{\langle s', h', \mathcal{A} \cup \mathcal{P}_\Gamma(l'), l', (\rho \circ \rho')\rangle \mid l' \in \mathit{Ch}_\Gamma(l)\}$ where s' is the state obtained by applying ρ' to s. If Π_m has no solution, then *cutParents(m)* $= \emptyset$.

cutParents first checks if l can be achieved and if it cannot, *cutParents* generates no metanodes. However if l can be reached, *cutParents* does not only achieve l, but it also marks all of it's ancestor nodes as achieved, and generates a new metanode for every child of l. Note that the ancestors are not actually achieved - they are added to the set of achieved landmarks just so that they are ignored. This focuses the search on the children of l, skipping all ancestor nodes of l.

Definition 12. (restartCutParents metanode generation). [1]
Let $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ be a metanode. *restartCutParents(m)* $= \{\langle I, h', \mathcal{A} \cup \mathcal{P}_\Gamma(l'), l', \emptyset\rangle \mid l' \in \mathit{Ch}_\Gamma(l)\}$ where I is the initial state of the original planning task.

restartCutParents is similar to *cutParents*, with the only difference being that the metanodes created by *restartCutParents* start at the initial state of the original planning task. The differences to the original planning task are that all ancestors of the metanode l will be ignored, and that the search now focuses on the achievement of deeper landmarks, namely the children of l.

2.5 The LMBFS Algorithm

LMBFS is a meta-search algorithm that operates on metanodes. Metanodes are associated to landmark nodes of the landmark graph, and represent subtasks with the goal of achieving their associated landmark. The metanodes are chosen in a best-first approach based on a heuristic evaluation of the metanode. The subtasks can be solved using any embedded planner as subplanner.

Algorithm 1: LMBFS [1]

Input : STRIPS planning task $\Pi = \langle A, O, I, G \rangle$, landmark graph Γ, metanode successor function *succ*

Output: solution plan or $\perp$

```
1  open ← ∅;
2  closed ← ∅;
3  ∀l ∈ roots(Γ) : add metanode ⟨I, h, ∅, l, ∅⟩ to open;
4  while open ≠ ∅ do
5      m ← arg min_{⟨s,h,A,l,ρ⟩ ∈ open} h;
6      open ← open \{m};
7      if m ∉ closed then
8          closed ← closed ∪ {m};
9          ρ' ← subplanner(Π_m);
10         if ρ' ≠ ⊥ then
11             s' ← result of executing ρ' in s;
12             if G ⊂ s' then
13                 return ρ ∘ ρ';
14             open ← open ∪ succ(m);
15 return ⊥
```

The algorithm starts by adding the metanodes corresponding to the root landmarks of Γ to the open list. Then the algorithm iterates until either the open list is empty or a solution plan to the planning task has been found, with each iteration including the best-first selection of a metanode m from the open list and the solution of the subtask associated to the metanode m. If the subtask has a solution, m is expanded by adding it's *successors* to the open list.

succ refers to the *successor function*, which uses the metanode generation methods introduced in section 2.4 to generate new metanodes. Two successor functions have been implemented [1]:

- *succDel(m) = nextLM(m) ∪ deleteLM(m)*
- *succDel(m) = nextLM(m) ∪ cutParents(m) ∪ restartCutParents(m)*

Theorem 1. *[1]*
The LMBFS algorithm using succCut or succDel as successor function is sound and complete if the subplanner is sound and complete.

2.6 Heuristic functions

There are two places in which heuristic functions are used in LMBFS - in the meta-search for the metanode open list, and in the subplanner (if applicable). This paragraph focuses on the first place.
As Vernhes et al. [1] showed, the heuristic functions h^{add} [5] and h^{ff} [6] are of good use in LMBFS, but $h^{\mathcal{L}_{left}}$, which was inspired by the LM-count heuristic [3] can be even more effective.

Definition 13. ($h^{\mathcal{L}_{left}}$) [1]
For a metanode $m = \langle s, h, \mathcal{A}, l, \rho \rangle$ and an associated landmark graph $\Gamma = (\mathcal{V}, \mathcal{E})$, the heuristic $h^{\mathcal{L}_{left}}(m) = |\mathcal{V} \backslash \mathcal{A}|$.

This heuristic counts the landmarks that have not yet been added to the set of achieved landmarks $\mathcal{A}$.

2.7 Lazy Metanode Generation

The *deleteLM* metanode generator can lead to a very hight amount of metanodes for some tasks, which results in a considerably increased search time on these tasks. This problem does not affect succCut, as succCut does not use the *deleteLM* metanode generator.
To solve this problem, Vernhes et al. [1] introduced an alternative strategy for LMBFS regarding the open list that involves de-prioritizing *deleteLM* via a secondary open list: While there are metanodes in the primary open list, only *nextLM* is used for metanode generation. Every metanode that is pushed into the closed list is also pushed into the secondary open list. Only when the primary open list is empty do we generate new metanodes with *deleteLM* by popping a metanode from the secondary open list and applying *deleteLM* to it.
This means that, instead of using *nextLM* and *deleteLM* equally often, we use only *nextLM* until the primary open list is empty.

2.8 Evaluation

2.8.1 Test Environment

The tested implementation of LMBFS was done in the Fast Downward planning system [2], with a modified version of Fast Downward's EagerGreedy search as subplanner. For land-

mark generation the method of Zhu and Givan [**?**] was used. This implementation does not support axioms or conditional effects and lacks a few features that could positively impact the algorithm's runtime and general effectiveness.
These features include the transitive reduction of the landmark graph introduced in 3, the lazy metanode generation introduced in 2.7, and the usage of deferred heuristic evaluation [7]. This does not impact the algorithm's completeness in principle, but it very well might do so in practice as there might be planning tasks that terminate within the time and memory limits with these optimizations, but not without them.
This implementation uses as heuristic for the meta-search $h^{\mathcal{L}_{left}}$. As for the sub-search, we refrain from using landmark-based heuristics in the spirit of Vernhes et al. [1]and use the popular heuristic h^{ff} [6]. The aforementioned EagerGreedy search was used with the heuristic h^{ff} [6] in the experiments as comparison. In instances where LMBFS is compared with EagerGreedy, LMBFS was used with the succCut successor function, as it is the better performing successor function in this implementation due to the lack of Lazy metanode generation.
The experiments were executed on the Maia Cluster at the University of Basel, consisting of machines with Intel Xeon E5-2660 CPUs running at 2.2 GHz. In order to run the experiments, downward-lab [8] was used.

2.8.2 LMBFS(succCut) vs LMBFS(succDel)

In Figure 2.1, we can see that LMBFS expands fewer nodes for a considerable amount of tasks using succCut. As we can see in the table further below, this results in a 146% increase from succCut expansions to succDel expansions in total.

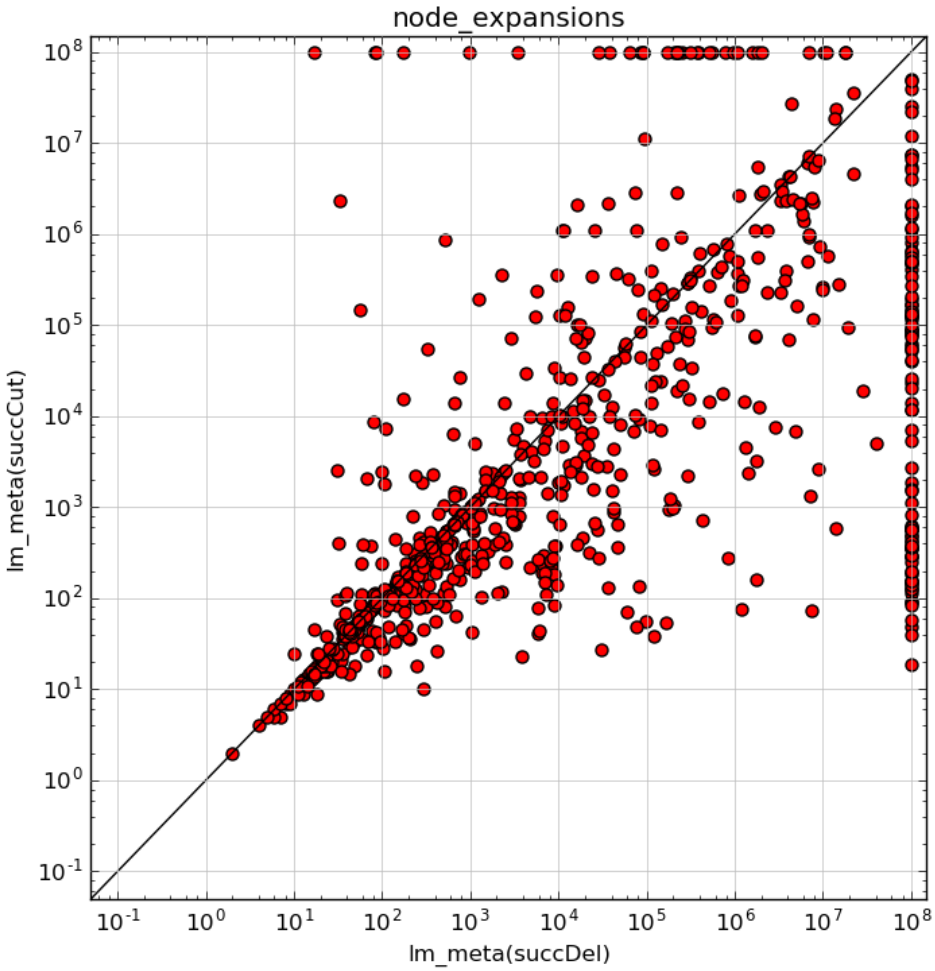

Figure 2.1: Node expansions - LMBFS(succCut) vs LMBFS(succDel)

In Figure 2.2 we can see the length of the solution plans that both LMBFS implementations produce. While there are instances where only one implementation cannot produce a plan, the two implementations are very similar for most tasks and domains.

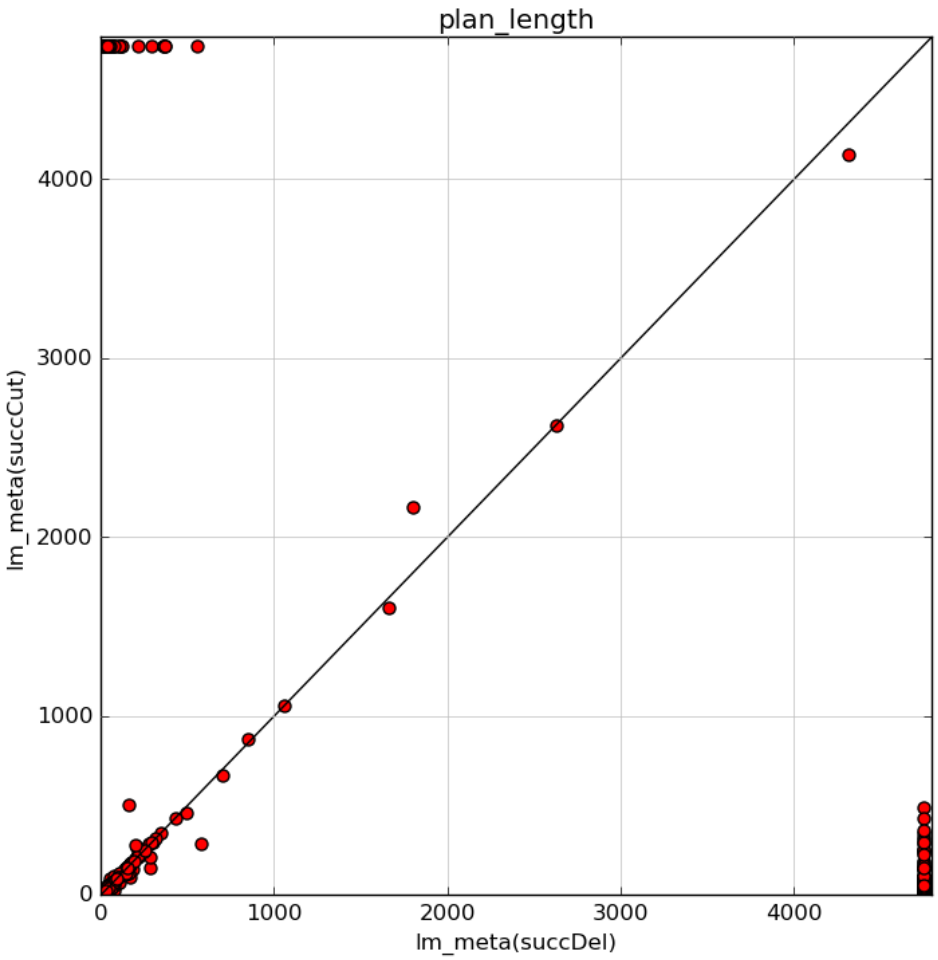

Figure 2.2: Plan length - LMBFS(succCut) vs LMBFS(succDel)

Figure 2.3 shows a comparison between the two LMBFS alternatives in search time. Again, succDel performs significantly worse, with the geometric mean across all domains and tasks being 125% higher than for succCut.

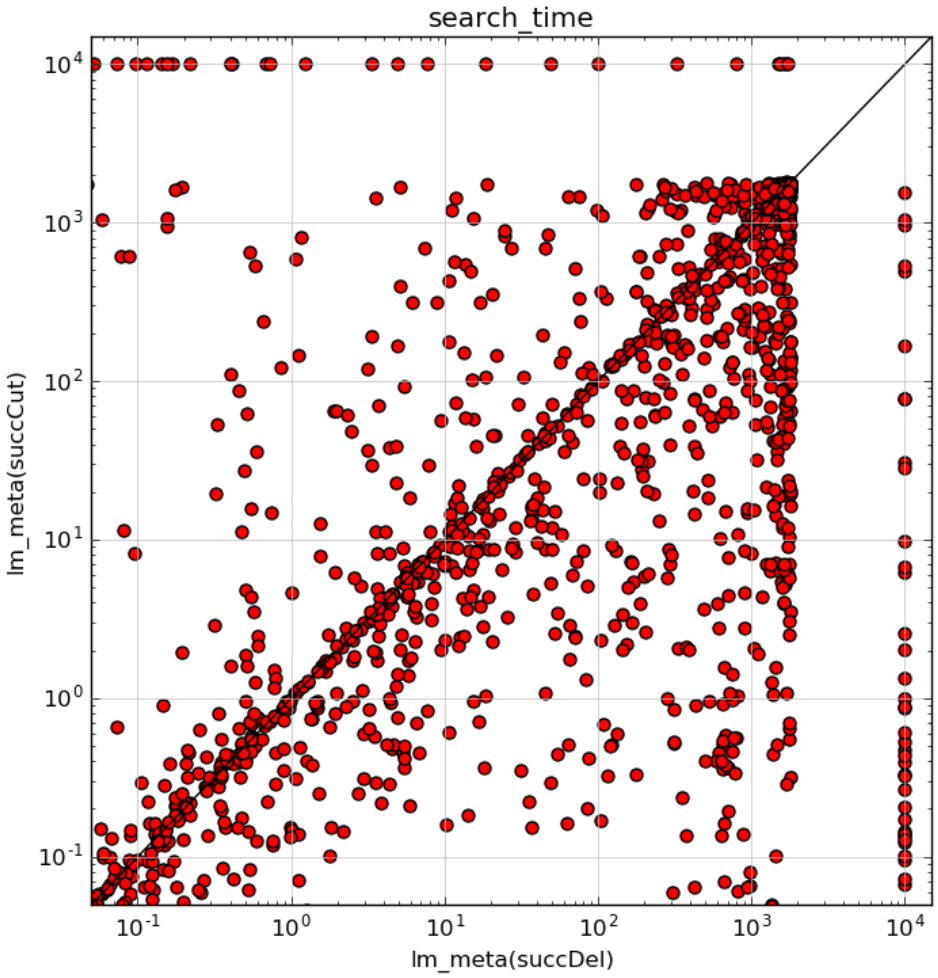

Figure 2.3: Search time - LMBFS(succCut) vs LMBFS(succDel)

As we can see in the table further below, succCut outperforms succDel significantly in evaluated nodes, expanded nodes, generated nodes and search time. In coverage, and especially in cost and plan length, the two LMBFS alternatives are quite similar. This strengthens the suspicion that the main reason for the difference between the two is the lack of Lazy Metanode Generation, which is a lot more detrimental for deleteLM than it is for (restart)cutParents.

2.8.3 LMBFS(succCut) vs EagerGreedy(ff)

In Figure [**?**], we can see a comparison in the number of node expansions between eagerGreedy(ff) and LMBFS(succCut). While the searches perform similarly well in a decent amount of tasks, eagerGreedy clearl outperforms LMBFS, with the mean of all node expansions being a factor of 10 lower for eagerGreedy.

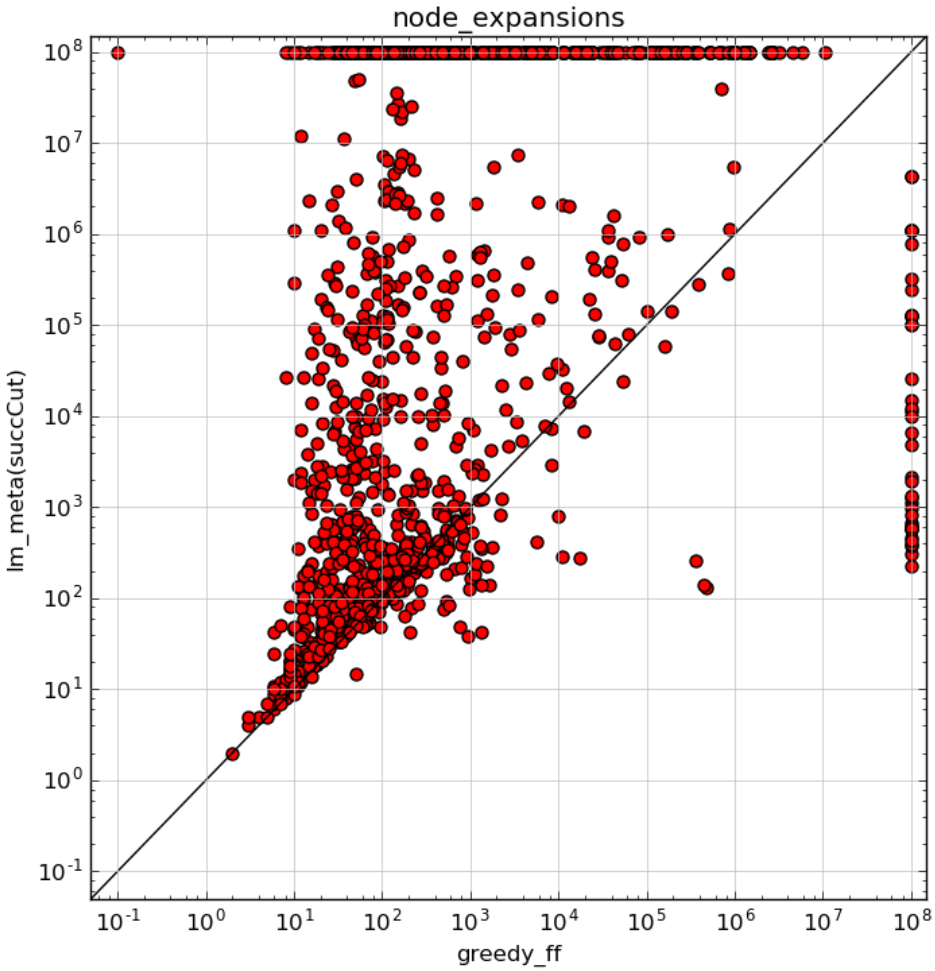

Figure 2.4: Node expansions - LMBFS(succCut) vs eagerGreedy(ff)

In Figure [**?**] we can see that the plan lengths of LMBFS and eagerGreedy are roughly similar for most instances, with only a marginal difference in the sum across all domains and tasks.

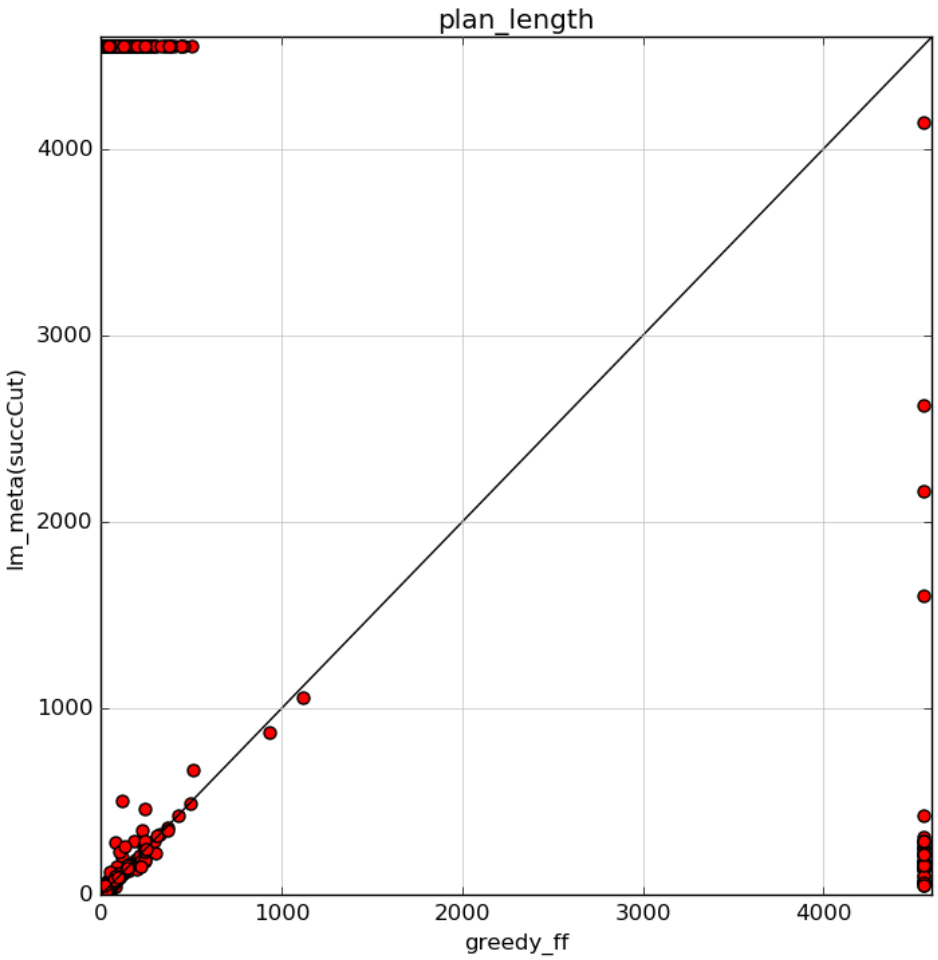

Figure 2.5: Plan length - LMBFS(succCut) vs eagerGreedy(ff)

The comparison in search time is almost as clearly in eagerGreedy's favour as the comparison in expanded states - the geometric mean for search time for eagerGreedy makes up only 16% of the geometric mean of search time for LMBFS.

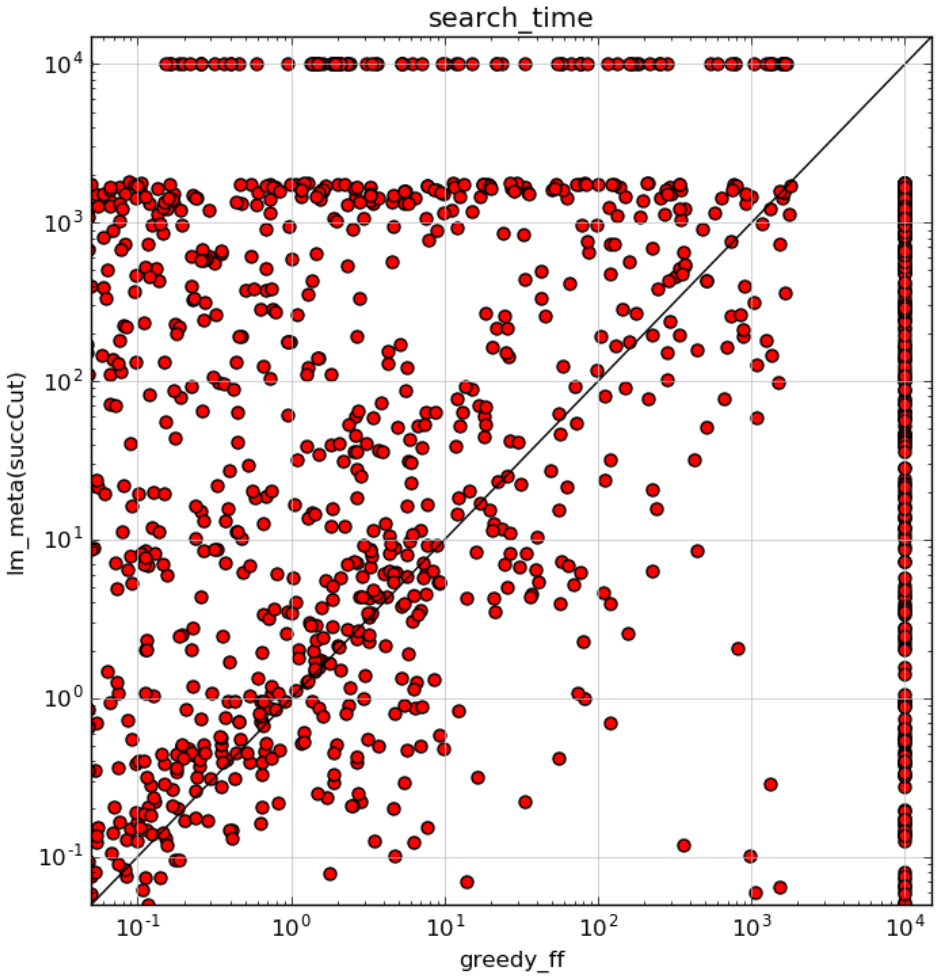

Figure 2.6: Search time - LMBFS(succCut) vs eagerGreedy(ff)

2.8.4 Overview

Overall we can see that this implementation of LMBFS is at least a number of optimizations away from being competitive with eagerGreedy(ff), as can be seen prominently in the halved coverage of LMBFS when compared to eagerGreedy, as well as the roughly 10-times increased evaluated, expanded, generated and search time values. This implementation of LMBFS is competitive, in aggregated statistics, only in summed plan length and plan cost.

Summary	**greedy_ff**	**lm_meta(succCut)**	**lm_meta(succDel)**
Cost - Sum	**5784772.00**	5794109.00	5794379.00
Coverage - Sum	**1438**	878	801
Evaluated - Sum	**30785444**	318829794	927036208
Expansions - Geometric mean	**147.76**	1527.58	3729.37
Generated - Geometric mean	**1567.10**	10317.14	24509.36
Plan length - Sum	**32469**	35848	36605
Search time - Geometric mean	**0.65**	3.97	8.92

This table shows a summary of the experiments, aggregated across all domains.

3
Conclusion

Landmark meta best first search is a rather unexplored and potentially very promising area in problem solving in automated planning. The appeal of meta-searches are their immense potential in flexibility, and therefore in my opinion this is one of the most interesting leads in landmark meta-search is the exploration of a number of new successor generators, as well as attempts to effectively alternate between different successor functions.
While the experiments in this thesis have, due to implementations issues, not been able to back up the findings of Vernhes et al. that LMBFS is competitive with the state of the art of landmark-based methods in automated planning, this is still very much the case and one should not forget that.
I'm looking forward to future work in this area.

Bibliography

[1] Vernhes, S., Infantes, G., and Vidal, V. Landmark-based Meta Best-First Search Algorithm: First Parallelization Attempt and Evaluation. In *In Proceedings of the ICAPS-2013 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP 2013)* (2013).

[2] Helmert, M. The Fast Downward Planning System. In *Journal of Artificial Intelligence Research 26 (2006), 191-246* (2006).

[3] Richter, S., Helmert, M., and Westphal, M. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intellicence (AAAI 2008), pp. 975–982. AAAI Press* (2008).

[4] Helmert, M. and Domshlak, C. Landmarks, Critical Paths and Abstractions: What's the Difference Anyways? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009), pp. 162-169. 2009* (2009).

[5] Bonet, B. and Geffner, H. Planning as heuristic search. In *Artificial Intelligence 129 (2001), 5-33* (2001).

[6] Hoffmann, J. and Nebel, B. The FF Planning System: Fast Plan generation through heuristic search. In *Journal of Artificial Intelligence Research, 14 (2001), 253-302* (2001).

[7] Zhu, L. and Givan, R. Landmark extraction via planning graph propagation. In *Proceedings of the ICAPS'03 Doctoral Consortium, 156-160.* (2003).

[8] Richter, S. and Helmert, M. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009), pp. 273-280. AAAI Press* (2009).

[9] Seipp, J., Pommerening, F., Sievers, S., and Helmert, M. downward-lab 2.0 (2017). URL https://doi.org/10.5281/zenodo.399255.

A
Appendix

I want to thank Gabriele Röger for the weekly advice, and I want to thank Malte Helmert for the chance of this learning opportunity.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Samuel Hugger

Matriculation number — Matrikelnummer
2014-057-442

Title of work — Titel der Arbeit
Landmark-based Meta Best-first Search

Type of work — Typ der Arbeit
Bachelor Thesis

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 27.06.2017

Signature — Unterschrift