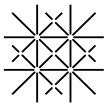


University of Basel

LM-Cut Heuristic With Different Precondition Choice Functions

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence Research Group
https://ai.dmi.unibas.ch/

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. David Jakob Speck

Aeneas Meier
aeneas.meier@stud.unibas.ch
14-052-401

Hand-In-Date: 27. November 2025

Acknowledgments

First I'd like to say thanks to my supervisor Dr. David Speck for his friendly support, his supportive way to answer questions, and his patience throughout the process of this work. He really was a great help, we had a lot of good discussions and he always provided a good-willing atmosphere. I'd like to thank Prof. Dr. Malte Helmert for the opportunity to write my thesis in this group, his spontaneous agreement on this thesis and his help in answering questions. I'd further like to thank Dr. Gabi Roeger for helping me out when my supervisors were at holidays. I also thank the whole Artificial Intelligence Research Group for their support - I always felt welcome and was able to ask questions at any time.
Last and not least I'd like to thanks my friends and family. They would have supported me a lot, if I ever told them any details about what I was actually doing in this work. As I was kind enough not to bother them too much, I remain thanking them for their fictious patience and support in other ways.
Calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing center at University of Basel. Thanks to them for their work in the background.
So Long, and Thanks for All the Fish. And yes - also the LaTeX-template.

Abstract

In this work, we evaluate the precondition choice function (PCF) for the $LMCut$ heuristic. $LMCut$ is an admissible heuristic that provides high-quality estimates for search algorithms like A^*. It achieves this by iteratively searching for landmarks in delete-free planning tasks, identifying operators in a so-called cut set, and then finding paths from the initial states to the goal. The PCF is a core functionality within this algorithm that maps all operators to a single precondition in order to identify landmarks in given problems. Historically, h^{max} is used to make this selection; however, it was never researched whether this is a good choice. In this work, we replaced h^{max} with random choice functions and its additive counterpart h^{add}. We demonstrate that h^{max} appears to be a good choice by collecting evidence from International Planning Competition (IPC) problems. We further provide a theoretical explanation for why this is the case.

Additionally, we examined tie-breaking strategies among equivalently strong candidates, as the current implementation leaves this choice open. We designed and evaluated a set of different algorithms. We found that these can improve $LMCut$ in terms of node expansions, yielding better estimates for some problems. We also provide an exemplary explanation of why tie-breaking can improve the quality of heuristic values.

Table of Contents

1 Introduction

In optimal planning, there is a broad variety of heuristics that provide strong goal-distance estimates. Combining A^* with an admissible heuristic is a prominent approach for finding optimal solutions for tasks within the space of world states. Using pattern databases or maximizing over multiple admissible heuristics can be a suitable way to find high-quality solutions (for example: Franco et al. [5], Seipp et al. [13]). Among these heuristics the landmark heuristics have evidently been shown to guide an effective search in many domains. One such heuristic is the *LMCut* heuristic [8], that provides qualitatively high estimates of the optimal delete-free relaxation heuristic h^+. As *LMCut* does not require any precomputation it is a popular choice for optimal planning.

The core idea of the *LMCut* heuristic is based on iteratively finding disjunctive action landmarks, i.e., actions that have to be applied in any valid plan to reach the goal state. It does this by iteratively creating a simplified version of the delete-free relaxation of the planning task. This so-called *justification graph* can be constructed by applying a *precondition choice function* (PCF) that maps an operator to a single precondition of that operator. After doing so, the algorithm defines a goal zone (which is the set of all propositions that have a zero-cost path to the goal) and then identifies the operators that are relevant for reaching this zone (i.e., the cut set).

The choice of this precondition can have a significant impact on the quality of the heuristic estimate. The original *LMCut* heuristic uses a PCF that selects the precondition that has the highest h^{max} value among the operator's preconditions. The intuition behind this choice is that selecting the most costly precondition is the minimum we have to pay to apply the operator. Due to its low computational cost and very high competitiveness, especially in satisfiability problems, using h^{max} for choosing preconditions seems to be a reasonable choice and has proven to be a successful approach in practice. However, it has never been evaluated whether there are alternative approaches that could be suitable as well.

In this thesis, we investigate the effect of different precondition choice functions on the quality of the *LMCut* heuristic. We do this by replacing h^{max} with alternative PCFs such as a random choice of preconditions and its additive counterpart h^{add}. Furthermore, we applied tie-breaking rules to the h^{max} PCF, as the current implementation is not deterministic in the case of multiple preconditions sharing the highest h^{max} value. Lauer and Fickert [10] also experimented with tie-breaking in the context of a performance optimization of $LMCut$ called *Quick Cutting*. We investigated their ideas, designed new strategies, and developed novel approaches for the regular algorithm.

2
Background

In this chapter, we first introduce the basic concepts that are used in this work. We assume that basic knowledge about algorithms, logic and graph theory is given. We then introduce the STRIPS planning formalism and the relevant heuristics used in this work.

2.1 State Space

A state space (or transition system) is a 6-tuple $S = \langle S, A, cost, T, s_I, S_G \rangle$ with

- a finite set of states S,
- a finite set of actions A,
- action costs $cost : A \to \mathbb{R}_0^+$,
- a transition relation $T \subseteq S \times A \times S$ that is deterministic in $\langle s, a \rangle$,
- an initial state $s_I \in S$,
- a set of goal states $S_G \subseteq S$.

A triple $\langle s, a, s' \rangle \in T$ is called a transition from state s to state s' using the action a. It can either be written as $s \xrightarrow{a} s'$ or $s \to s'$ (if a does not matter). The phrase deterministic in $\langle s, a \rangle$ means that for each state s and action a there is only a single state s'. The state space can be depicted as a directed, edge-labeled graph, where the vertices represent states and the edges the transitions.
We further introduce the following terminology, that will be used in this work. We say that an action a is applicable in state s if there exists a state s' such that $\langle s, a, s' \rangle \in T$. We call a state s a predecessor of s' if there is an action a such that $\langle s, a, s' \rangle \in T$. Subsequently, s' is called a successor of s. A sequence of states $\pi = \langle s_0, s_1, ..., s_n \rangle$ is called a path from s_0 to s_n if for all $0 \leq i < n$ there is an action a_i such that $\langle s_i, a_i, s_{i+1} \rangle \in T$. The length of a path is defined as the number of transitions (i.e. n). Consecutively, the cost of that path is defined as the sum over the cost of all actions in that path $cost(\pi) = \sum_{i=0}^{n-1} cost(a_i)$. A path is called a solution iff $s_0 = s_I$ and $s_n \in S_G$. The solution is called optimal if there is no other solution with lower cost (i.e. $cost(\pi) \leq cost(\pi')$ for all other solutions π')

2.2 Search Algorithms

2.2.1 A^*

The A^* algorithm is a graph search algorithm, that can find the least-cost path in that graph given a start and a goal node [6]. It uses a combination of Dijkstra's algorithm and greedy best-first search using the actual cost of the current path g and a heuristic estimate h of the cost to the goal s_G.

2.2.2 Formal Definition

Let

$$G = (V, E)$$

be a weighted graph, where V is the set of vertices and $E \subseteq V \times V$ is the set of directed edges. Let $c(u, v) > 0$ denote the cost (or weight) of edge $(u, v) \in E$.
We are given:

- A start node $s_I \in V$,
- A goal node $s_G \in V$,
- A heuristic function $h : V \to \mathbb{R}_0^+$ estimating the cost from node n to the goal s_G.

2.2.3 Core Functions

For each node $n \in V$:

$$g(n) = \text{actual cost of the shortest known path from } s_I \text{ to } n,$$
$$h(n) = \text{heuristic estimate of the cost from } n \text{ to } s_G,$$
$$f(n) = g(n) + h(n).$$

2.2.4 Optimality Conditions

Without reopening, the algorithm is *complete* and *optimal* if the heuristic h satisfies:

1. **Admissibility:**
$$h(n) \leq h^*(n) \quad \forall n \in V,$$
where $h^*(n)$ is the true minimal cost from n to s_G.

2. **Consistency:**
$$h(n) \leq c(n, v) + h(v) \quad \forall (n, v) \in E.$$

With reopening, A^* remains optimal if h is admissible. As $LMCut$ is admissible but not consistent, this is what was used here.

2.2.5 Algorithm

The A^* algorithm works by maintaining two data structures: an *open list* containing nodes to be expanded, and a *closed list* of processed nodes. Starting with the initial state s_I,

the algorithm repeatedly selects the node with the lowest f-value from the open list (using tie-breaking if multiple nodes have the same f-value). It expands this node by generating its successors and updates cost estimates. For each successor, if it has not been visited or a better path is found (lower g-value), the successor is added to the open list with updated cost information (there might be duplicates with differing g and h values). The search terminates when a goal state is selected for expansion. Note that some implementations may differentiate between *selection termination* (stopping when a goal is selected for expansion) and *expansion termination* (stopping after fully expanding the goal node). The result is optimal given the heuristic matches the criteria mentioned in the section before.

2.3 Classical Planning

In this work we consider classical planning tasks (i.e. state spaces) that can be described in a formal language called a *planning formalism*. There are a number of different formalisms that can be used to describe planning tasks. This work will work with the STRIPS formalism that is a widely used, very simple formalism based on binary state variables.

2.3.1 STRIPS

The STRIPS classical planning formalism was introduced by Fikes and Nilsson [4] and consists of propositional states that are connected by operators. A STRIPS classical planning task is defined as a 4-tuple $\Pi = \langle P, I, G, O \rangle$, with:

- P: a finite set of propositional state variables (or propositions),
- $I \subseteq P$: the initial state,
- $G \subseteq P$: the set of goal states,
- O is a finite set of operators. Each operator $o \in O$ is defined as follows:
  - $\text{pre}(o) \subseteq P$: the preconditions of o,
  - $\text{add}(o) \subseteq P$: the add effects of o,
  - $\text{del}(o) \subseteq P$: the delete effects of o,
  - $\text{cost}(o) \in \mathbb{N}_0$: the cost of o.

Propositional state variables are binary variables that can either be $true$ or $false$. Here we represent a state s as a subset of all state variables $S \subseteq P$ that are true in s. As an extension of this "traditional" STRIPS formalism this definition considers additional costs for the operators.
Starting from an initial state, we can apply an operator if all propositions in its precondition set $\text{pre}(o)$ are $true$ in the current state. Applying an operator changes the values of all propositions in the $\text{add}(o)$ set to $true$ and those in the $\text{del}(o)$ set to $false$.

Formally, a STRIPS task $\Pi = \langle P, I, G, O \rangle$ induces the state space $S(\Pi) = \langle S, A, cost, T, s_I, S_G \rangle$ as follows:

- Set of states $S = 2^P$ (the power set of P)
- Actions $A = O$ (the operators as defined in Π)
- Action costs *cost* as defined in Π
- Transitions $s \xrightarrow{o} s'$ for states $s, s' \in S$ and operator $o \in O$ iff
  - $\text{pre}(o) \subseteq s$ (preconditions are satisfied), and
  - $s' = (s \setminus \text{del}(o)) \cup \text{add}(o)$ (effects are applied)
- Initial state $s_I = I$
- Goal states $s \in S_G$ for state s iff $G \subseteq s$ (all goals are reached)

2.3.1.1 Relaxed Planning Tasks

For estimating costs of planning tasks we consider a simplified version of the original task where disadvantageous operator effects are ignored. This is called a *relaxed* or *delete-free* planning task Π^+. In the STRIPS formalism this means that we ignore the delete effects of operators. The intuition is that delete effects can only remove propositions from the current state, so they decrease the number of applicable operators.
A delete-free STRIPS task is defined as a STRIPS task where all operators have no delete effects, i.e., $\text{del}(o) = \emptyset$ for all $o \in O$. A proposition p is reachable if there exists a path from the initial state s_I to some state s with $p \in s$. All propositions that are reachable in Π are also reachable in Π^+.

2.3.1.2 i-g-Form

Next we introduce the i-g-form, which is a special case of delete-free STRIPS tasks. This normal form summarizes goal and initial states into a single variable each, which is useful for later simplification. A delete-free STRIPS task $\Pi^+ = \langle P, I, \gamma, O \rangle$ is in i-g-form if:

- P contains propositions i and g,
- initially exactly i is true: $I(v) = T$ iff $v = i$,
- g is the only goal proposition: $\gamma = \{g\}$,
- every operator has at least one precondition.

Any delete-free STRIPS task $\Pi = \langle P, I, O, \gamma \rangle$ can be transformed into an analogous task in i-g-form as follows:

1. If i or g are already in P, rename them everywhere.
2. Add i and g to P.
3. Add an artificial operator $\langle \{i\}, \{p \in P \mid p \in I\}, \emptyset, 0 \rangle$ that makes all initial propositions true.

4. Add an artificial operator $\langle \gamma, \{g\}, \emptyset, 0 \rangle$ that achieves the artificial goal proposition.
5. Replace all operator preconditions $\top$ (true) with $\{i\}$.
6. Set the initial state to $\{i\}$ and the goal to $\{g\}$.

The i-g-form was used in the implementation and the theoretical background. Note that in some illustrative examples we refrain from using the i-g-form in order to reduce them to the essential points.

2.3.2 h^{max} and h^{add}

Among the simplest heuristics for relaxed planning tasks are h^{max} and h^{add} [2]. Both heuristics are based on the idea of estimating the cost of reaching a certain proposition from the initial state. They are computed in a layer-based approach starting with the initial state and annotating each proposition with the cost of reaching it. These delete relaxation heuristics compute their estimates on a delete-free task, like the later introduced $LMCut$ algorithm does.

2.3.2.1 Relaxed Planning Graph

The algorithm runs on a graph with two layers types, here called *proposition layers* P_i and *operator layers* O_i:

- Proposition layer P_0 contains the proposition vertex p_0 for all $p \in I$.
- Operator layer O_{i+1} contains the operator vertex o_{i+1} for operator o if P_i contains the vertex p_i for all $p \in \text{pre}(o)$.
- Variable layer P_{i+1} contains the variable vertex p_{i+1} if the previous variable layer contains p_i, or the previous action layer contains o_{i+1} with $p \in \text{add}(o)$.

A goal vertex g exists if $p_n \in P_n$ for all $p \in G$, where n is the last layer. The graph can be constructed for an arbitrary number of layers but the algorithm stops when the layers stabilize, i.e:

$$P_{i+1} = P_i \quad \text{and} \quad O_{i+1} = O_i.$$

The directed edges are labeled in the following way

- From p_i to o_{i+1} if $p \in \text{pre}(o)$ (precondition edges)
- From o_i to p_i if $p \in \text{add}(o)$ (effect edges)
- From p_i to p_{i+1} (no-op edges)
- From p_n to g if $p \in G$ (goal edges)

2.3.2.2 Algorithm

The algorithm starts with the initial state propositions in the proposition layer P^0 in which each proposition has cost 0. It then applies all applicable operators in operator layer O^i to reach a new variable layer P^{i+1}. Thus, we start with applying all operators in operator layer O^0 to reach new propositions in the proposition layer P^1, which are the initial propositions plus the effects of all operators $o \in O^1$. The algorithm continues to do so until it stabilizes after a bounded number of layers. The h value of each operator vertex o_i is calculated by adding the h value of all preconditions together (h^{add}) respectively taking the max over them (h^{max}) and adding the operator's cost on top. The h values of a proposition vertex is calculated by taking the minimum h value among all incoming edges. The single value $h(p, s)$ is an estimate of the cost of reaching a proposition p from state s in the relaxed planning task Π^+. For h^{add} this is:

$$h(p,s) = \begin{cases} 0, & \text{if } p \in s, \\ \min\limits_{o \in O_p} \left\{ \text{cost}(o) + \sum_{q \in \text{pre}(o)} h(q,s) \right\}, & \text{otherwise,} \end{cases}$$

where O_p is the set of operators having p in their add list. So the final estimate of h^{add} is then the sum over all heuristic value among preconditions of the goal state γ:

$$h^{add}(\gamma) = \sum_{p \in G} h(p, \gamma)$$

In a similar way, h^{max} can be calculated by replacing the sum with taking the maximum value among preconditions as mentioned before:

$$h(p,s) = \begin{cases} 0, & \text{if } p \in s, \\ \min\limits_{o \in O_p} \left\{ \text{cost}(o) + max_{q \in \text{pre}(o)} h(q,s) \right\}, & \text{otherwise,} \end{cases}$$

with a final estimate $h^{max}(\gamma)$ for the goal state γ:

$$h^{max}(\gamma) = max_{p \in G} h(p, \gamma)$$

2.3.3 LM-Cut

The LM-cut heuristic [8] is an admissible, but not consistent heuristic based on the iterative computation of landmarks. It uses a *precondition choice function* (PCF) to construct a simplified problem representation called a *justification graph*, from which it identifies disjunctive action landmarks.

2.3.3.1 Cut Landmarks

A *precondition choice function* $PCF : O \to P$ for a delete-free STRIPS task $\Pi^+ = \langle P, I, \gamma, O \rangle$ in i-g form maps each operator to one of its preconditions, i.e., $PCF(o) \in \text{pre}(o)$ for all $o \in O$. This mapping is used to construct a *justification graph* $J = \langle V, E \rangle$, which is a directed, edge-labeled graph where the vertices are the propositional states from P and E contains an edge $PCF(o) \xrightarrow{o} a$ for each operator $o \in O$ and each add effect $a \in \text{add}(o)$.

Within this simplified problem representation, a *cut* is defined as a subset C of edges such that all paths from i to g contain an edge from C. The set of edge labels (i.e., operators $\{o \mid \langle p, o, p' \rangle \in C\}$) forms a *disjunctive action landmark*, meaning that every plan contains at least one operator from every landmark [3]. Since the justification graph is a simpler representation of the original problem and cuts are landmarks for this representation, the cuts are also landmarks for the original problem.

2.3.3.2 Algorithm

The LM-Cut heuristic can be calculated iteratively. After initializing $h^{LMCut} = 0$, the heuristic value is calculated by proceeding as follows:

1. Calculate h^{max} for all propositions [2]. If $h^{max}(g) = \infty$, return ∞ (unsolvable). If $h^{max}(g) = 0$, stop and return the computed value.
2. Compute a justification graph J based on the mapping of the PCF of each operator to one of its preconditions with maximal h^{max} value.
3. Determine the goal zone V_g of J consisting of all nodes having a zero-cost path to g.
4. Compute the cut L that contains the labels of all edges $\langle p, o, p' \rangle$ such that $p \notin V_g$ and $p' \in V_g$ and p is reachable from i without passing through a node in V_g.
5. Increase h^{LMCut} by $cost(L)$, i.e., the minimum cost of all operators in L.
6. Reduce the cost of all operators $o \in L$ by $cost(L)$.

2.4 Note on Terminology

We try to be consitent with terminology. We try to use the word *action* in the context of state spaces around A^* and the word *operator* in terms of STRIPS formalism. In literature, sometimes these terminologies are mixed. Subsequently, the words *predecessor* and *successor*, here used for describing relations between node in a state space, should not be confused with *precondition* and *effect*, which are again STRIPS-related nomenclature. The *atom*, *variable*, *fact* or *proposition* describe the same thing and can be used synonymously. Here we use the word *proposition*.

3

LMCut Heuristic with Different Precondition Choice Functions

This work explores the effect of different precondition choice functions on the LMCut heuristic. The aim is to investigate whether the choice of precondition can lead to higher, more precise heuristic values, thereby improving the overall performance of the LMCut heuristic. The primary focus was on replacing the default PCF, which selects the precondition with the highest h^{max} value, with alternative PCFs. Additionally, a variety of tie-breaking strategies were explored to determine if they could enhance the performance of the default PCF. In the following sections, these alternative PCFs and the additional tie-breaking strategies are described in detail.

3.1 Precondition Choice Functions

Replacing h^{max} as the PCF in LMCut can be done in principle with any function that maps an operator to one of its preconditions. However, not all functions are meaningfully suited for this purpose. For instance, a function that always returns the same precondition would not be a good strategy, as it would essentially ignore the other preconditions, leading to less informative heuristic values. Therefore, we focused first on a random precondition choice strategy to get an impression of whether diversifying the precondition choice can lead to better heuristic values at all. Then we explored the use of h^{add} as a PCF, as another heuristic that shares similarities with h^{max} and is a suitable candidate for this purpose.

3.1.0.1 Terminology

In the following, the $LMCut$ algorithm will be denoted as $LMCut^{PCF}$, where the superscript describes the applied precondition choice function. The heuristic value will be denoted as $h^{\text{LMCut}}_{\text{PCF}}(I)$. The PCF itself will be denoted as $PCF^{heuristic}$, where the superscript denotes the heuristic based on which the operator is mapped on its precondition (as described in the following sections; e.g. PCF^{hadd}). Furthermore the tie-breaking criterion will be denoted in the subscript for the PCF and $LMCut$, e.g. PCF_{p-min}. If left empty, we relate to the original implementation, i.e. h^{max} based PCF and no tie-breaking specified. In *Fast*

Downfast tie-breaking favors the precondition that first pops from the priority queue, which is further referenced as *FD tie-breaking*.

3.1.0.2 Note on the Termination Criterion

Formally, replacing the PCF only means that we change the PCF mappings in the second step of the algorithm (Section 2.3.3.2). This alone would not change the termination criterion, which states that the h^{max} value of the artificial goal g proposition must be zero (further denoted as *terminating on* h^{max}). This would mean that we must maintain the h^{max} stack, while implementing the other PCF logics aside. So, in this study we primarily focused on an implementation that consequently replaced the termination criterion.

3.1.1 Random Choice PCF

Replacing the current PCF with a random strategy is a straightforward approach to diversify the precondition selection. The intuition behind this strategy was to explore if diversifying the precondition choice can lead to better heuristic values at all just by chance. There were different possibilities of indroducting randomness into the PCF, like a fully random selection, excluding some preconditions from random choice (for example those within the goal zone) or switching the termination criterion. Thus, we implemented a series of different approaches and made basic evaluations. Among them the two most interesting random strategies will be explained here in more detail:

3.1.1.1 PCF^{rd}: Completely Random Choice

In the first strategy, the precondition was chosen uniformly at random among all preconditions of the selected operator. The selection of preconditions was repeated in each iteration of the LMCut algorithm, leading to a potentially different precondition being selected in each round and therefore a varying justification graph and cut set. In this random implementation we changed the termination logic, such that it terminates if there is a zero-cost path from i to g in the current justification graph, after reducing the costs of all operators in the cut set.

3.1.1.2 $LMCut^{rd-max}$: Random Choice With h^{max} as Termination Criterion

Another variation of the completely random strategy worth mentioning, is to keep the h^{max} based termination criterion but select randomly among preconditions. This can be done either by a fully random choice as before. Alternatively you can avoid all candidates that allready have a h^{max} value of 0, except if there are none, which is when we switch back to fully random choice among all candidates (PCF^{rd-max}). Here, we implemented the latter with the idea to see if we can get better choices just by luck. Except now we want to maintain higher heuristic estimates and therefore decrease the number of node expansions.

3.1.2 PCF^{hadd}: Choosing Preconditions Based on h^{add}

In the next and more important step, we replaced the current PCF with the one that selects the precondition with the highest h^{add} value among all preconditions. For this algorithm in principal there are two variants that again differ in the termination criterion. In the first would only change the PCF without modifying the termination criterion that is based on h^{max}. In the second variant we also updated the termination criterion, such that it validates if the h^{add} value of the goal state is zero. This makes sense, as if $h^{max}(g) = 0$ also $h^{add}(g) = 0$ must be true. This way, the necessity for maintaining two different heuristics can be removed, which makes the algorithm more efficient, while still having a meaningful termination criterion. So in this work, we only considered the latter.

3.1.3 Tie-Breaking

Besides replacing the entire PCF, another strategy is to keep using h^{max} as the PCF, but additionally use a tie-breaking mechanism for selecting a precondition among candidates with the same highest h^{max} value. As exemplarily explained in Figure 3.1 in theory, tie-breaking choices can have an effect on the final heuristic value of h^{LMCut}. This is as the choice of the precondition influences the operators in the cut set, thereby affecting the h^{max} values and the justification graphs in the resulting round.
Again, for tie-breaking there are numerous strategies that can be applied as there are many criteria that can be considered for such a choice as demostrated before by Lauer and Fickert [10]. In this work we implemented a broad variety of different approaches, made some basic evaluations to see if they are promising candidates and selected the most successful ones. These strategies are explained in the following:

3.1.3.1 PCF_{e-max}: Favor Effects of Many Operators

One of the most obvious choices for tie-breaking is considering the influence a precondition has on other operators by considering its own effects or preconditions. For instance, we can prefer preconditions that are the effect of the largest number of operators. Doing so will lead to larger cut sets, thus $LMCut$ will potentially underestimate the true cost as operators that should be considered can be reduced to zero cost without being applied. Again, in Figure 3.1 we can see this behavior in a small example, where this approach leads to a heuristic value of 1, while the true cost would be 2. So, although this should not be a promising candidate, we left it for demonstration purposes.

3.1.3.2 PCF_{e-min}: Favor Effects of Few Operators

The better choice would be to choose a precondition that is the effect of the fewest number of operators. This should reduce the number of operators in the cut set, thereby forcing them to be applied and thus yielding a more realistic estimate.

3.1.3.3 PCF_{p-min}: Favor Rarer Preconditions

Vice versa, instead of choosing preconditions that are the effect of other operators, we could choose those that are a precondition of less other operators instead. So this approach consecutively favors the operator having the lowest number of preconditions among candidates.

3.1.3.4 PCF_{reach}: Favor Reachable Candidates

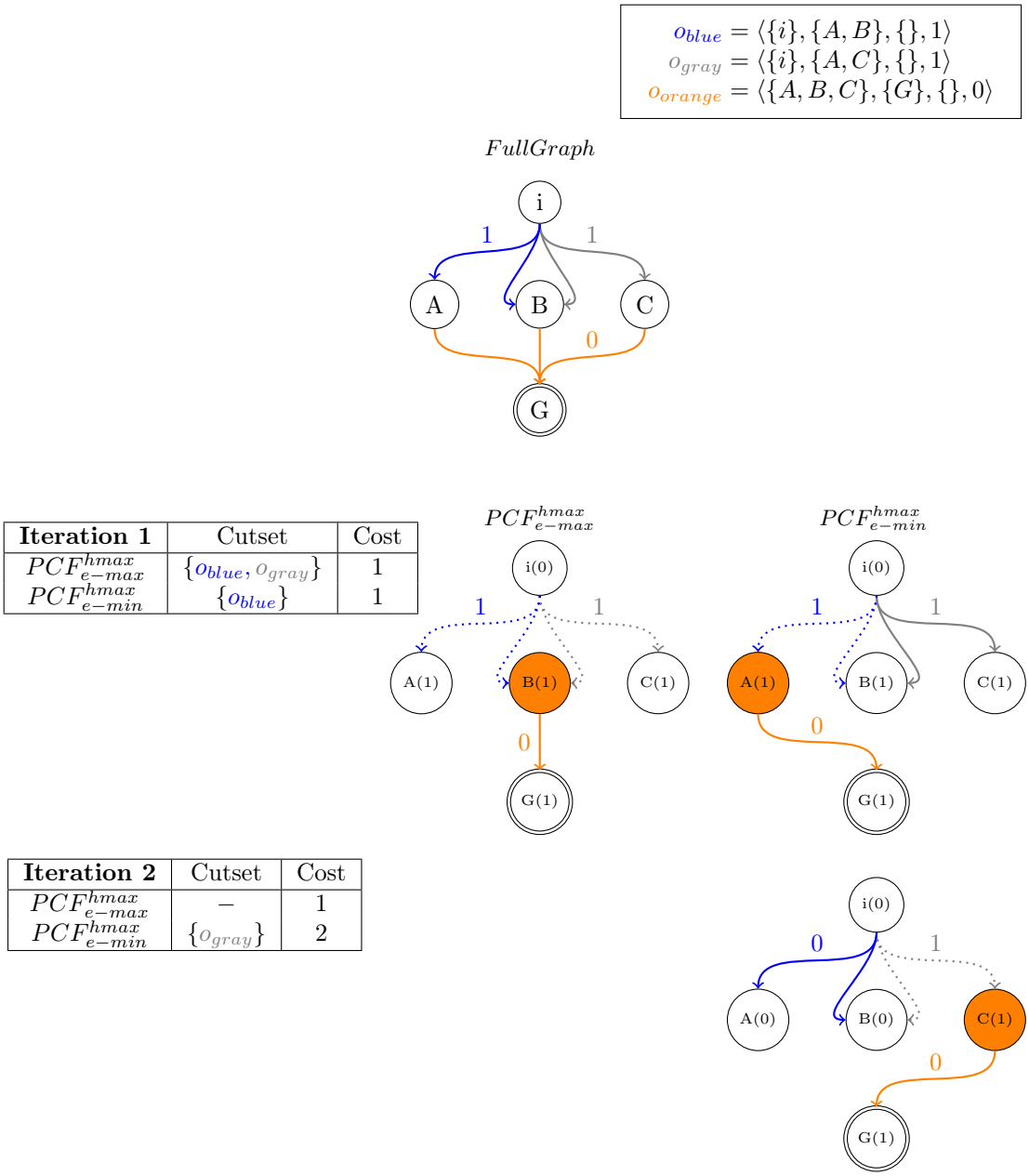
For the next approach we calculated a reachability score for each proposition. The reachability of a proposition essentially is its h^{max} value, when each operator is treated as if it had a cost of 1 (in a delete-free task). This way we get the minimum number of operators that have to be applied to reach that proposition. In tie-breaking situations, we now choose the candidate that is easier to reach. The intuition behind this approach is that we should prefer propositions that are easier to reach, as we might reach them otherwise while further progressing backwards. This way we could skip them although they should have been applied.

3.1.3.5 PCF_{unused}: Favor Unused Preconditions

Another strategy is to keep track of the preconditions that have already been used in previous iterations. Doing so, we can avoid repeatedly preferring the same operator in tie-breaking situations. The intuition behind this approach is that using different preconditions in each round will alter the justification graphs more frequently and therefore force the algorithm to choose a diversified set of operators in cut sets. This way, LMCut should apply more operators, thus run additional rounds that lead to a higher heuristic value.

3.1.3.6 $PCF_{unused-n}$: Favor Less Frequently Used Preconditions

Additionally, we can keep track of the number of times an operator has been chosen as precondition. This should diversify even more in situations where operators are repeatedly selected after decreasing their cost, with the penalty of using slightly more computational resources.



Iteration 1	Cutset	Cost
PCF^{hmax}_{e-max}	$\{o_{blue}, o_{gray}\}$	1
PCF^{hmax}_{e-min}	$\{o_{blue}\}$	1

Iteration 2	Cutset	Cost
PCF^{hmax}_{e-max}	–	1
PCF^{hmax}_{e-min}	$\{o_{gray}\}$	2

Figure 3.1: **Different Tie-Breaking Lead to Different Cut Sets**
Full graph with three different operators (o_{blue}, o_{orange}, o_{gray}). Below the full graph is shown with operators displayed as arrows from their preconditions to their effects in the corresponding color.
Below the full graph, two iterations of the LMCut algorithm are shown. The number in brackets within each proposition is their corresponding h^{max} value. Each iteration shows the justification graph resulting from applying tie-breaking s.t. the PCF favors the proposition being the effect of the highest number of operators among highest h^{max} value (PCF^{hmax}_{e-max}, middle), or the effect of the lowest number of operators respectively (PCF^{hmax}_{e-min}, right). $PCF(o_{orange})$ is highlighted in orange. Dotted lines indicate operators in the goal zone. The table on the left lists the chosen landmarks and the current h^{LMCut} value. The (updated) operator costs are highlighted in the same color as the operator. While both strategies have the same cost in the first round, they differ in the second iteration, leading to a final heuristic value of 1 for PCF^{hmax}_{e-max} and 2 for PCF^{hmax}_{e-min}. Note that PCF^{hmax}_{e-max} stops after the first iteration.

4
Implementation

This chapter describes the details of the implementation of the different precondition choice functions (PCFs) and tie-breaking strategies in the Fast Downward planning system [7]. Here, the Fast Downward system is introduced in short, followed by a detailed description of the implementation into the Fast Downward codebase.

4.1 Fast Downward

This work uses the Fast Downward planning system [7] for classical domain-independent planning. Fast Downward is an open-source planner written in C++ and Python that is widely used in the planning community, such as the International Planning Competition (IPC). The planner is divided into two main components: the translator and the search engine.

The translator converts planning tasks in PDDL or SAS+ format into a finite-domain representation suitable for the search engine located under `src/translate`.

The search engine then finds a plan for the problem, i.e., a sequence of actions from the initial state to the goal state using heuristic search. Fast Downward uses a compact problem representation, different heuristics, and modular design, which allows search strategies and heuristics to be combined. The engine is located under `src/search`, which contains the `heuristics` directory, where our modifications were made.

4.2 Implementation Details

The primary file that was modified in this work was `lm_cut_landmarks.cc` and the corresponding header file `lm_cut_landmarks.h`, which can be found in the `src/search/heuristics` directory of the Fast Downward codebase.
To do so, the original class `LandmarkCutLandmarks` was split into three classes to improve modularity:

```
class LandmarkCutLandmarks {
  LandmarkCutCore core;
  std::unique_ptr<LandmarkCutHeuristicExploration> heuristic;
  LandmarkCutBackwardExploration backward;
  PreconditionChoiceFunction precondition_choice_function;
  // ...
};
```

Here, the class `LandmarkCutCore` contains the core logic of the LMCut heuristic. The unique pointer to the class `LandmarkCutHeuristicExploration` is used to perform the forward exploration that assigns the heuristic values to the propositions and selects the preconditions. This class implements three abstract functions `update_supporters`, `trigger_operators` and `trigger_operators_incremental` and further implements some core logic of the forward exploration phase. Additionally, the PCF mechanisms were implemented in different classes that inherit from `LandmarkCutHeuristicExploration` and use those abstract functions to apply their strategies (described later). `LandmarkCutBackwardExploration` is responsible for the backward exploration that identifies the cut sets and was left unmodified. Finally, the class `PreconditionChoiceFunction` was introduced to encapsulate the different PCFs and tie-breaking strategies. This class was primarily used as an interface between user prompts and the actual PCF implementations, that maps the input to the corresponding pointer class implementing the desired PCF or tie-breaking strategy. When using h^{max} as termination criterion, we replace the incremental exploration phase `heuristic->h_max_exploration_incremental(cut);` in the primary while loop with the full exploration `heuristic->h_max_exploration(state);`. This reduces complexity as maintaining h^{add} incrementally gets really hard.

4.2.1 Precondition Choice Functions

As mentioned above, the different PCFs were implemented in separate classes inheriting from `LandmarkCutHeuristicExploration`. As there was a broad variety of tie-breaking strategies and they all share h^{max} as base PCF, their implementation was grouped into a single class called `LandmarkCutMaxTieBreakExploration`. To avoid a bloated code base, each feature was implemented into a separate branch. The different PCFs and tie-breaking strategies, along with their corresponding class names, branches and user prompts (`pcfstrategy` keyword in the console prompt) are summarized in the implementation table in the Appendix (Table B).

4.2.2 Random Number Generation

To implement PCF^{rd}, a random number generator was needed. For this purpose, the random number generator from `utils::rng_options.h` was used that is based on the Mersenne Twister algorithm [11]. This generator is used in other parts of the Fast Downward codebase already. To ensure reproducibility of the results, the seed of the random number generator can be set using the `seed` keyword in the console prompt (with a default value of 42).

4.3 Experimental Setup

We implemented the described modifications in the Fast Downward planning system [7] on top of the existing implementation of the LMCut heuristic. For the evaluation, we included all 66 domains from the International Planning Competition (IPC) in the default optimal suite, leading to a total of 880 problems per experiment. All tests were run using the Downward Lab package [12]. The tests were run on the SciCORE scientific computing centre at the University of Basel, using the $infai\_2$ cluster with 23 nodes ($icb[01-23]$) with each 20 cores, 128GB memory, 240GB SSD (2 x 10 Core Intel Xeon Silver 4114 2.2 GHz Processor). Each run was limited with an overall time limit of 30 minutes using A^* as search algorithm.

5
Experimental Results

In this section we'll describe the experimental results we obtained in this study. We first look into the outcomes of using different PCFs and then explore the finding of applying the discussed tie-breaking strategies. This section only describes the results. They will be discussed later in the next chapter.

5.0.1 Replacing the PCF

To verify that the refactoring of the code base did not introduce any regression, we first compared the performance of the new implementation with the latest release of the official Fast Downward planner at the beginning of the project (i.e., the commit with fingerprint *766e8fe*). We found that the number of node expansions did not change in any task of any domain as visible in Table B.3 in the Appendix.

In the next step, we were interested in comparing the LMCut heuristic using an h^{max} based PCF with a PCF selecting a random precondition. The results are shown in the top row of Figure 5.1. We could see that the randomly choosing PCF yielded an increased number of necessary node expansions in all tasks except for one. $LMCut^{hmax}$ dominated almost strictly, meaning that none of the problems were able to perform better with $LMCut^{rd}$ in terms of node expansions. A large number could not even terminate within the given time limit. With $LMCut^{rd-max}$, when h^{max} was used for the termination criterion, it still performed notably worse than the original implementation but remarkably better than a fully random run. When comparing the average initial h value per domain of the random runs to their corresponding result using $LMCut^{hmax}$ in average per domain, the latter reached of 77% ($\sigma = 21\%$), while the fully random run only reached 13% ($\sigma = 12\%$). For $LMCut^{rd}$, we also compared the number of total node expansions using with different seeds, as visible in the Appendix B.1. There was no dominance of any random run for the set of seeds that were used.

Comparing $LMCut^{hmax}$ to $LMCut^{hadd}$ we observed a less clear pattern than in the random implementation (bottom row of Figure 5.1). $LMCut^{hmax}$ was still dominating, but both algorithms did not differ strongly in terms of node expansions in contrast to the clear dominance in the random runs. There were some domains where $LMCut^{hadd}$ performed better

(like *Parcprinter*, *MPrime* and *Satellite*), however on others it was noteably worse (e.g. *VisitAll*). We also looked at the initial h values of both algorithms compared domain-wise on all problems where both algorithms terminated (Table B.1). We could observe that the average ratio was 1.092 with a standard deviation σ of 0.17 in favor of $LMCut^{hmax}$.

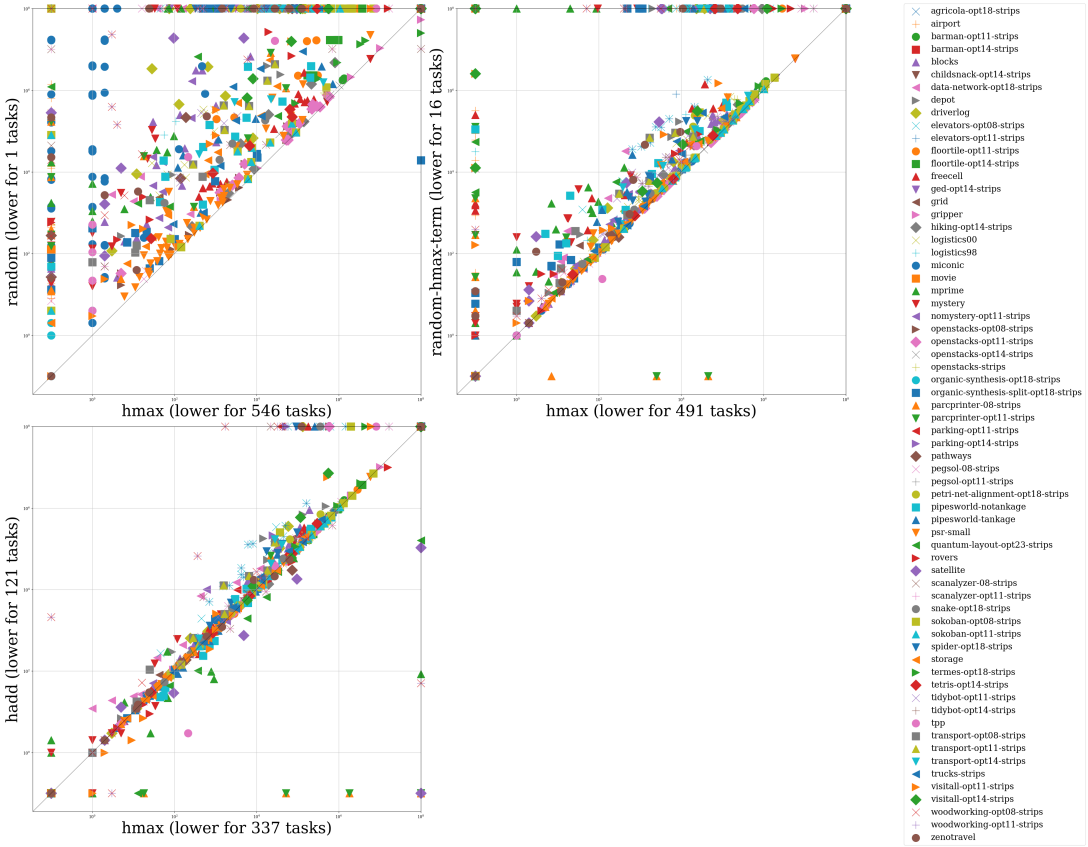


Figure 5.1: **Node Expansions of $LMCut^{hmax}$ Compared to Alternative PCF Strategies**
The three plots compare the number of node expansions required by different $LMCut$ variants using A^* search. Each point represents a single planning problem from a specific domain (legend on the right). The axis labels indicate the compared strategies and show the number of problems where each approach required less expansions. Points above the diagonal line indicate that the h^{max}-based PCF required fewer expansions, while points below favor the alternative strategy.
The h^{max}-based PCF dominated across all tasks. Against the random strategy (top left), it performs better in all but one problem, with no runs where the random approach succeeded while h^{max} failed. When the random strategy terminates on h^{max} (top right), performance improves slightly but remains inferior, requiring more expansions in 491 problems versus 16 where it performs better.
$LMCut^{hadd}$ (bottom) shows better overall performance than $LMCut$ with random PCFs but still underperforms the original implementation, outperforming $LMCut^{hmax}$ in only 121 tasks.

5.0.2 Tie-Breaking

Instead of replacing h^{max} as foundation of the PCF, we also considered tie-breaking strategies among preconditions sharing the highest h^{max} values. A variety of tie-breaking strategies were considered, from which the ones decribed in Section 3.1.3 will be evaluated here.

5.0.2.1 Considering Effects and Preconditions

First we considered the PCF_{e-max} tie-breaking strategy, favoring the precondition with the largest number of effects (top left in Figure 5.2). We can see a spread around the diagonal indicating that this strategy in fact makes a difference in terms of number of nodes expanded. However, using this strategy the spread was concentrated towards the top left corner, meaning that using $LMCut$ with default tie-breaking was more successful for most problems. However, if we change the strategy, such that instead we take the precondition having the minimum number of effects, it starts shifting towards the bottom right where tie-breaking performs better. The total number of lower problems still is in favor of non-tie-breaking though. When we instead tie-break in favor of the precondition that itself is precondition of the lowest number of operators, the total number of runs having fewer necessary node expansions switches in favor of tie-breaking. However, the overall effect is smaller, meaning the ratio between node expansions for the same problems gets closer to 1.

5.0.2.2 Reachability

Next, we took a look at the reachability tie-breaking strategy PCF_{reach} that favors propositions having lowest cost assuming all operators cost were 1. When taking the most reachable again the default tie-breaking approach numerically performs better in more tasks. Also, while beneficial effects in some problems are notable, negative effects in other problems occur to some extend.

5.0.2.3 Not Used Strategies

We received more interesting results when preferring preconditions that were not chosen as heuristic supporters in previous iterations (bottom row Figure 5.2). While technically $LMCut^{hmax}$ dominated $LMCut^{hmax}_{unused}$ in more runs, looking at the data in a closer view reveals a different picture. While most runs ended up near the diagonal, meaning that they had almost the same number of node expansions, many other problems remarkably outperformed the non-tie-breaking approach (up to orders of magnitude). This can be expressed by the ratio of node expansions $node\_exp(LMCut^{hmax})/node\_exp(LMCut^{hmax}_{unused})$. Here, the average ratio in non-tie-breaking-dominated runs was 1.06 ($\sigma = 0.14$) compared to 0.71 ($\sigma = 0.34$) in tie-breaking-dominated runs. In other words: when tie-breaking was the better choice, it was the much better choice. Additionally, the coverage was 959 for $LMCut^{hmax}$ while $LMCut^{hmax}_{unused}$ had a larger value of 980, although this was largely because the *VisitAll* domain could not be solved by the regular approach in 20 cases. When additionally counting the number of usages, preferring the less frequently used preconditions, the shift towards favoring tie-breaking becomes even clearer for many problems. However, other problems like *Airport* start getting worse heuristic estimates or even get unsolvable due to time limits.

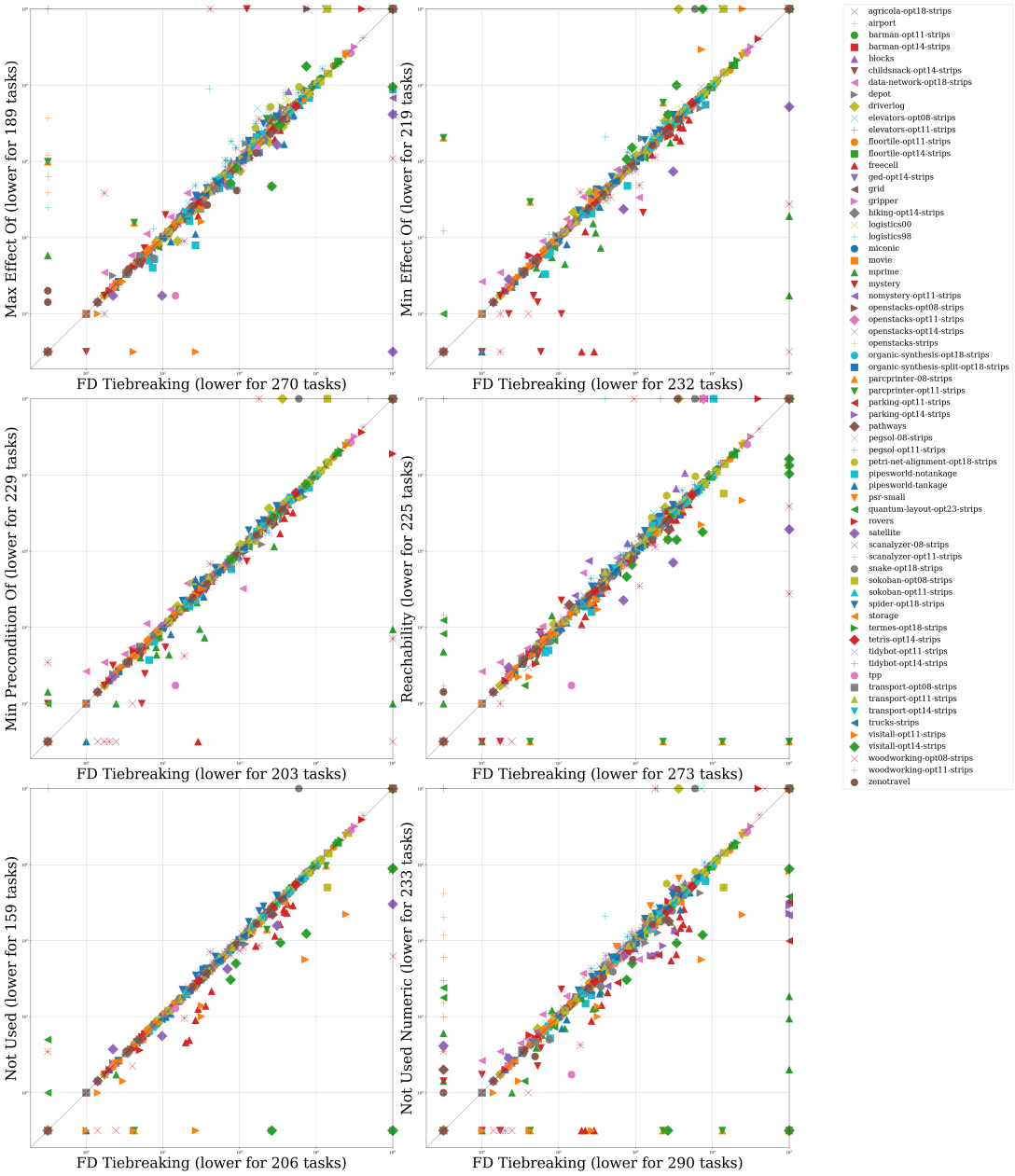


Figure 5.2: **Node Expansions of LMCut with Different Tie-Breaking Strategies** The six plots compare node expansions for different tie-breaking strategies in *LMCut* using A^* search against the official implementation with FD tie-breaking preferring first pick. Each point represents a single problem from a specific domain (legend on the right). The axis labels indicate the tie-breaking strategy and show the number of problems where it required less expansions. Points below the diagonal line indicate that the tie-breaking strategy required fewer expansions than the official variant, while points above favored the official variant.
The results demonstrate varying effectiveness across strategies. PCF_{e-max} (top left) performs worse than the FD tie-breaking in most cases. Conversely, PCF_{e-min} (top right) and PCF_{p-min} (middle left) show improvements in several problems. The PCF_{reach} strategy (middle right) yielded mixed results. Most notably, PCF_{unused} (bottom left) and $PCF_{unused-n}$ (bottom right) achieve substantial improvements in many problems, particularly in the *VisitAll* domain.

6 Discussion

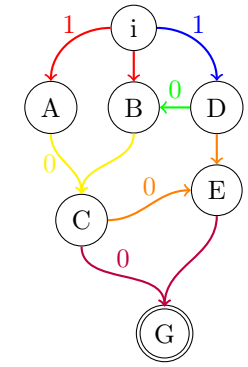
The primary focus of this work was to investigate whether replacing the PCF with an alternate function could yield improved results. However, while working on this thesis, we collected evidence that h^{max} seems to be a reasonable choice, but tie-breaking can make a significant difference. We will first discuss why we believe h^{max} appears to be a good choice and then explore how we can further improve $LMCut$ through reasonable tie-breaking strategies. In the following sections, we will reflect on our results and provide an outlook on potential future directions.

6.1 Impact of replacing the PCF

From the results comparing the PCF^{hmax} with PCF^{rd}, we can see that $LMCut$ with PCF^{hmax} almost consistently outperforms the randomly mapping PCF approach in terms of node expansions. This is visible in the top two plots in Figure 5.1. There were only a few instances where $LMCut^{rd}$ could finish tasks that could not be completed with $LMCut^{hmax}$. We did not necessarily expect this with such clarity, as the random selection could have potentially revealed more effective paths just by chance. If h^{max} would have been a poor choice, we would have expected at least a notable portion having fewer node expansions in the random PCF. This was not the case except for a few tasks where $LMCut^{hmax}$ did not terminate but $LMCut^{rd}$ did. Also, making a preselection among precondition candidates with $LMCut^{rd-max}$ improved the results, but did not change the overall picture. This suggests that $LMCut^{hmax}$ evidently is a good heuristic for selecting preconditions. However, it also demonstrates by contradiction that it isn't the best choice in general. This opens room for further investigation as there may be better choices.

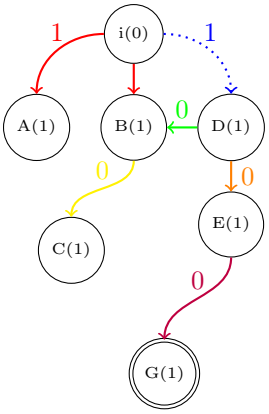
In the next step, we investigated the impact of replacing h^{max} with h^{add} in the LMCut heuristic. From the results in the bottom line of Figure 5.1, we can see that the h^{add} based PCF performs worse in terms of node expansions in most tasks. We could further see that $LMCut^{hadd}$ did not terminate in a notable amount of tasks. This could be due to two reasons. The obvious but less interesting one is that calculating h^{add} values is computationally more expensive. This can trigger time limits leading to uncompleted calculations. However, this will not explain differences in node expansions. The other reason is less straightforward

and the intuition goes as follows: h^{max} is an admissible heuristic that assumes the *best case*. Here, we select the most expensive precondition, assuming that we can reach it by taking advantage of other operators' effects. h^{add}, on the other hand, is conservative and assumes the *worst-case* scenario where we have to satisfy all preconditions separately. So, by choosing h^{add} for the PCF in $LMCut$, we tend to favor preconditions that are more deeply woven into the graph, as having more preconditions will increase the resulting h^{add} value. This could create larger cut sets and therefore lower heuristic values.
In Figure 6.1, we could show that using $LMCut^{hadd}$ can lead to lower estimates than $LMCut^{hmax}$ when using a shared tie-breaking strategy. While $LMCut^{hmax}$ maintains free choice among all candidates for the PCF, $LMCut^{hadd}$ can force the algorithm to choose preconditions in such a way that the resulting cut sets will include more operators. Thus, the resulting heuristic value for the latter will be an underestimation. So far, we were only able to construct examples where the differing estimates depend on tie-breaking. Also, switching to another tie-breaking strategy could neglect this effect. As $LMCut$ does not define how to tie-break, the difference between applying both PCF strategies becomes a statistical problem. In the current example, if tie-breaking randomly, $LMCut^{hadd}$ will estimate correctly in 50% of the runs, while $LMCut^{hmax}$ is correct in 62.5% of all cases. This can be verified by creating all possible justification graphs and calculating their heuristic value (Figure B.4). Therefore, it would be interesting to further explore whether it is possible to construct an example where switching PCFs causes a change in propositional h^{add}/h^{max} values, such that $LMCut$ returns different estimates without requiring tie-breaking decisions at all.
Our empirical results support the hypothesis that different estimates between $LMCut^{hmax}$ and $LMCut^{hadd}$ are likely caused by tie-breaking decisions. Several domains observed to be sensitive to tie-breaking (Figure 5.2) show different estimates, including *VisitAll*, *MPrime*, *Satellite*, and *Parcprinter* (Figure 5.1). For the *VisitAll* domain, we can explain the underlying mechanism causing this sensitivity, as discussed in the next section. This observation confirms that tie-breaking effects manifest in larger, more complex problems as well. However, this does not definitively answer whether replacing the PCF with h^{add} only influences tie-breaking or has additional effects. Future research could implement deterministic tie-breaking for both algorithms to isolate and compare their intrinsic differences, which would put more light on this question.

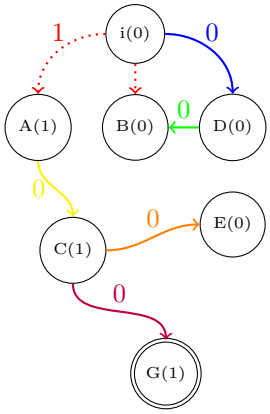


$o_{red} = \langle\{i\}, \{A, B\}, \{\}, 1\rangle$
$o_{blue} = \langle\{i\}, \{D\}, \{\}, 1\rangle$
$o_{yellow} = \langle\{A, B\}, \{C\}, \{\}, 0\rangle$
$o_{green} = \langle\{D\}, \{B\}, \{\}, 0\rangle$
$o_{orange} = \langle\{C, D\}, \{E\}, \{\}, 0\rangle$
$o_{purple} = \langle\{C, E\}, \{G\}, \{\}, 0\rangle$

Iteration 1: $h^{\text{LMCut}}_{\text{hmax}}(I) = 1$



Iteration 2: $h^{\text{LMCut}}_{\text{hmax}}(I) = 2$



Iteration 1: $h^{\text{LMCut}}_{\text{hadd}}(I) = 1$

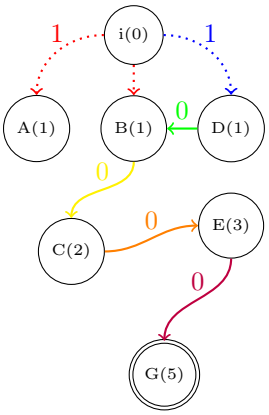


Figure 6.1: **Differing Heuristic Value for $LMCut^{hmax}$ and $LMCut^{hadd}$ Under Common Tie-Breaking Rules**
Each circle represents a proposition in the state space graph, each arrow represents an operator with its cost highlighted aside in the same color, according to the legend. Operators in the cut set are display with dotted lines. The initial state is labeled on top with a lower case i. The goal state G is marked with doubled border lines.
The upper illustration shows the complete graph. On the left side the justification graphs for the first and second iteration with $LMCut^{hmax}$ is shown. The h^{max} value is highlighted in brackets within the proposition node. The right side shows the justification graphs for the first and only iteration with $LMCut^{hadd}$. The corresponding h^{max} resp. h^{add} value of a proposition is highlighted in brackets within the node.
Note that tie-breaking is essential in this example. Here, we tie-break in favor of the alphabetically subsequent proposition, i.e. B for yellow , E for purple and D for orange. Thus, the final heuristic value for $LMCut^{hmax}$ is 2. $LMCut^{hadd}$ is forced to chose proposition E for purple and C for orange, leading to a final heuristic value of 1. Further note that it matters which precondition is selected by the yellow operator, as only B is connected to D over the green operator s.t. blue ends up in the cut set. This is enforced by the commom tie-breaking rule.

6.1.1 Termination Criterion in $LMCut^{rd}$

Adjusting the termination criterion for the random choice PCF is consecutive, as we could demonstrate by examining search time (top right in Figure B.2). Even though $LMCut^{rd}$ performs worse in terms of node expansions, search time can be lower than for $LMCut^{hmax}$. As Figure B.3 displays, it can terminate quickly (while still returning optimal solutions guaranteed by combining $LMCut$ and A^*). Since randomly choosing preconditions is inherently a poor strategy, reducing search time is the only real benefit we can potentially obtain. When terminating based on h^{max} instead, we end up in a lose-lose situation where we incur high computational costs due to maintaining the h^{max} queue without using this additional information to select optimal candidates. Especially if would not avoid selecting preconditions with h^{max} values of zero, we would repeatedly select poor preconditions that are already in the goal zone. Thus, the larger the problem gets, the more likely the algorithm will end up stuck (not in a theoretical sense though). In our results we could confirm this, as our algorithm did not terminate in many cases due to time limits.

6.2 Effects of Tie-Breaking in h^{max}

As we could see in the previous section, h^{max} seems to be a good choice for the PCF. However, we could also see that there is potential for improvement. Thus, we further investigated the impact of tie-breaking strategies among preconditions sharing the highest h^{max} value.

Overall, we could observe in our results that tie-breaking can have a positive effect on the final result. Although there is no strategy that improves performance in general, there are some clearly improving the overall result. In general, tie-breaking can create better estimates by reducing the number of operators in cut sets, as demonstrated in Figure 3.1. Selecting the PCF_{p-min} strategy is a conservative approach that appears to be slightly better in general. The PCF_{unused} and $PCF_{unused-n}$ strategies, however, can reduce node expansions remarkably. We can also observe that tie-breaking strategies seem to have a trade-off — meaning that the more they improve some problems, the weaker they make others.

So, what do these improvements have in common? In general, we know that $h^{max} \leq h^{LMCut} \leq h^+ \leq h^*$ [2], [8]. As the $LMCut$ heuristic performs well on delete-free tasks, one hypothesis was that it performs especially well on tasks where h^+ is close to h^*. Betz and Helmert [1] showed empirically for problems like *Miconic-STRIPS*, *Satellite*, and *Logistics* that h^+ is a close estimation to the perfect heuristic h^*. However, in our experimental data, we could not observe a pattern among these domains. More interestingly, we can examine a domain that has frequently drawn our attention and where Hoffmann [9] proved that $h^+ = h^*$ (for non-Hamiltonian paths), which is *VisitAll*.

6.2.1 VisitAll

The *VisitAll* domain is a special case of planning problems, as it has a very specific graph structure. In this problem, a robot is required to visit a given number of locations on a field of squares. Each square can be reached from its neighboring squares by moving up, down, left, or right by an operator describing this exact movement (i.e. move_from_to). All problems consist of a large number of operators with only a single precondition and a single effect (i.e. the field the robot is currently on and the one it goes to). The only exception to that is the artificial operator created by the $LMCut$ heuristic to reach the artificial goal state. What the PCF_{unused} tie-breaking strategy enforces is preferring preconditions that have not been used yet in the current $LMCut$ iteration. In this scenario, this is a good choice, as we can observe in Figure 6.2. Here, we can see the expansion of the $LMCut$ heuristic in the *VisitAll* domain with and without tie-breaking. Both strategies behave the same in the first four iterations. However, in the fifth iteration, the PCF_{unused} strategy selects a precondition it has not chosen so far (orange cell), while the PCF without tie-breaking chooses a precondition that has already been selected. Thus, it has zero-cost operators to neighboring cells (yellow), ultimately leading to larger cut sets. This is, of course, an underestimation, since in a non-delete-free world we cannot reach two fields by applying just a single action. The PCF_{unused} strategy, on the other hand, selects a precondition that has not been used so far, which prevents the goal zone from growing in this case (although not in general). This results in higher heuristic values, better-informed search, and thus fewer node expansions, as we could observe in the experimental results (bottom rows in Figure 5.2).

Hence, we saw that tie-breaking choices can improve the estimates of $LMCut$. Lauer and Fickert [10] also observed this behavior for the *Freecell* and *VisitAll* domains, which we could now empirically confirm and provide an explanation for why this is the case (for the latter). However, the general mechanism — if there is one at all — remains unknown. Thus, further research is needed to identify patterns that enable us to handle them effectively. Finding these could help to develop more general tie-breaking strategies that can be applied to a larger set of problems.

6.3 Conclusion

As an overall summary for this thesis, we could confirm that h^{max} in most cases is a good choice for the PCF, even though it does not always guarantee better performance. We could demonstrate that h^{add} can cause suboptimal tie-breaking choices and we could bring some light into why h^{max} performs better in general. Additionally, we could show why choosing suitable tie-breaking mechanisms is advisable, especially when the domain characteristics are well understood.

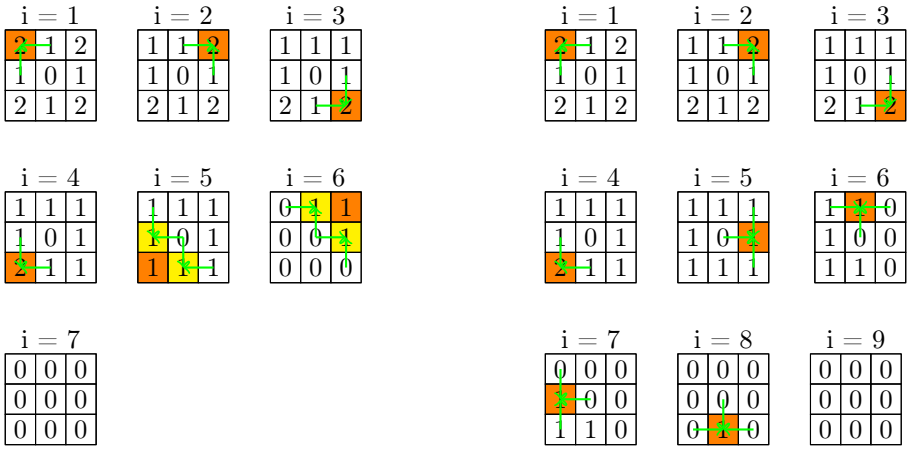


Figure 6.2: **Visualization of LMCut Expansion in the Visit-all Domain With and Without Tie Breaking.**
Each cell represents a location in the Visit-all domain, with the number indicating the h^{max} value for that proposition. The orange background color highlights the precondition that was selected by the PCF for the artificial goal state. The yellow background indicates the goal zone (i.e. the propositions that can be reached from the artificial goal with zero cost actions in the current justification graph). Green arrows indicate the operators that are in the current cut set (and thus will be reduced to cost 0 in the next iteration). The expansion goes from top to bottom and left to right (the iterations are numbered above each state). The left side shows the expansion without tie breaking, resulting in a total of seven iterations and an initial heuristic value of 6. We can see a growing goal zone in iteration 5 and 6. The right side illustrates the expansion with tie breaking, which results in eight iterations with a different state exploration path and a higher initial heuristic value of 8. Here the goal zone only ever consists of a single proposition.

A

Literature

Bibliography

[1] Christoph Betz and Malte Helmert. Planning with h^+ in theory and practice. In Bärbel Mertsching, Marcus Hund, and Zaheer Aziz, editors, *Proceedings of the 32nd Annual German Conference on Artificial Intelligence (KI 2009)*, volume 5803 of *Lecture Notes in Artificial Intelligence*, pages 9–16. Springer-Verlag, 2009.

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

[3] Clemens Büchner, Remo Christen, Salomé Eriksson, and Thomas Keller. DALAI 2023. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*, 2023.

[4] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[5] Santiago Franco, Álvaro Torralba, Levi H. S. Lelis, and Mike Barley. On creating complementary pattern databases. In Carles Sierra, editor, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pages 4302–4309. IJCAI, 2017.

[6] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[7] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[8] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 162–169. AAAI Press, 2009.

[9] Jörg Hoffmann. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.

[10] Pascal Lauer and Maximilian Fickert. Beating LM-cut with LM-cut: Quick cutting and practical tie breaking for the precondition choice function. In *ICAPS 2020 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, pages 9–15, 2020.

[11] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[12] Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. Downward Lab. https://doi.org/10.5281/zenodo.790461, 2017.

[13] Jendrik Seipp, Thomas Keller, and Malte Helmert. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67:129–167, 2020.

A.1 Repositories

The following repositories were used and created for this thesis:

- **Downward Fork**: https://github.com/derschiw/downward
- **Downward Lab**: https://github.com/derschiw/downward-lab
- **Downward Test**: https://github.com/derschiw/downward-test

A.2 AI Declaration

AI-based tool	**Type of use**	**Affected parts**	**Remarks**
Chat GPT (GPT-5.1, GPT-4.1), Proton LUMO, Claude Sonnet 4 + 4.5	Fix typos and make stylistic corrections while preserving the meaning. Highlight areas with inconsistencies. Prompt: "Fix spelling and make wording smoother without changing the content. Give me feedback on anything you think is worth mentioning."	Entire paper	
Github Copilot using GPT-4.1, GPT-4o, GPT-5 mini, Claude Sonnet 4	Code Improvement: - Code explanation - Minor performance improvements - API suggestions, C++ language understanding - Code debugging and line autocompletion	Entire paper	
Claude Sonnet 4	Help for creating the Figure 6.2 that shows colored h^{max} values in the Visit-all domain and applied operators as green arrows	Figure 6.2	
Chat GPT Chat (GPT-5.1)	Generated pseudo code for the h_max algorithm (revised and edited by author later) Prompt: "can you create me a pseudo code for the h_max algorithm?"	Section 2.3.2	
Chat GPT Chat (GPT-5.1)	A^* formal definition help (revised and edited by author later) Prompt: "Define A-Star formally"	Section 2.2.1	

B

Appendix

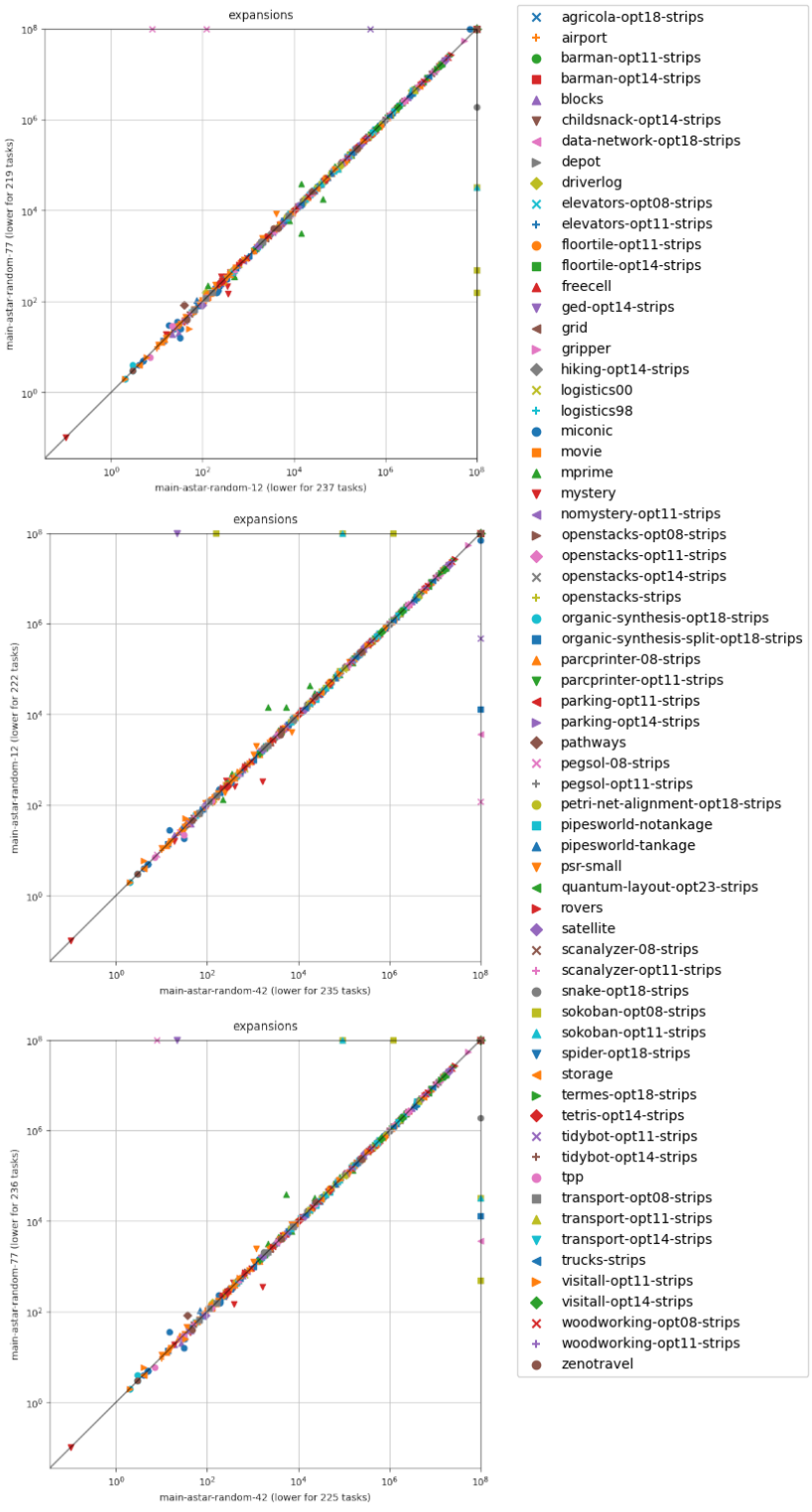


Figure B.1: **Node Expansions of $LMCut^{random}$ Using Different Seeds** The figure shows three runs of the $LMCut^{random}$ algorithm on A^* with the seeds 77, 12 and 42 passed as arguments. Each dot represents the number of node expansions (including last jump) of a run for a problem of a specific domain (shown in the legend on the left) for both seeds. Most runs had almost the same number of node expansions with a few outliers on both sides. There was no clear dominance for any seed.

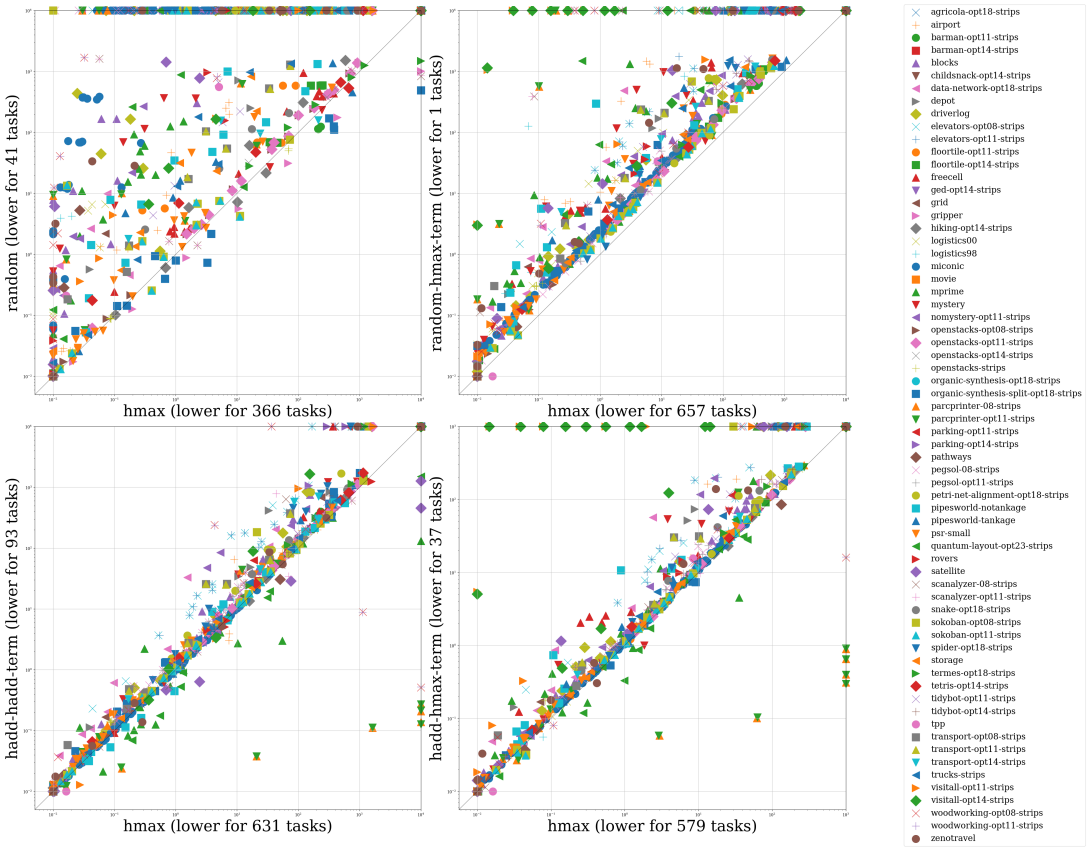


Figure B.2: **Search Time of LMCut with h_{max} based PCF Compared to Alternate PCF Strategies**
The three plots show the number of node search time the alogrithm needed to run each LMCut variant with the given PCF strategy using A^*. Each point represents one problem for a given domain. The domain names are listed in the legend on the right side. The axis label on the left indicate the comparison strategies and highlights the number of runs where they finished with less search time. Similarily, points above the diagonal line indicate runs where the h_{max} based PCF approach needed less search time and vice verca.

Problem	h_hadd	h_hmax	h_hmax/h_hadd
agricola-opt18-strips (20)	3066	3414	1.114
airport (50)	9276	9312	1.004
barman-opt11-strips (20)	505	676	1.339
barman-opt14-strips (14)	250	194	0.776
blocks (35)	587	607	1.034
childsnack-opt14-strips (20)	308	308	1
data-network-opt18-strips (20)	1549	1786	1.153
depot (22)	397	464	1.169
driverlog (20)	437	473	1.082
elevators-opt08-strips (27)	552	947	1.715
elevators-opt11-strips (19)	400	641	1.603
floortile-opt11-strips (20)	1190	1199	1.008
floortile-opt14-strips (20)	1145	1143	0.998
freecell (80)	776	971	1.251
ged-opt14-strips (18)	17	17	1
grid (5)	66	86	1.303
gripper (20)	940	940	1
hiking-opt14-strips (20)	255	247	0.969
logistics00 (28)	1040	1041	1.001
logistics98 (35)	2951	2952	1
miconic (150)	7570	7570	1
movie (30)	210	210	1
mprime (35)	167	166	0.994
mystery (30)	148	152	1.027
nomystery-opt11-strips (20)	380	394	1.037
openstacks-opt08-strips (30)	30	30	1
openstacks-opt11-strips (20)	20	20	1
openstacks-opt14-strips (20)	20	20	1
openstacks-strips (30)	1539	2383	1.548
organic-synthesis-opt18-strips (7)	12	12	1
organic-synthesis-split-opt18-strips (20)	2385	2698	1.131
parcprinter-08-strips (30)	36681678	37139851	1.012
parcprinter-opt11-strips (20)	23031480	23113284	1.004
parking-opt11-strips (20)	282	341	1.209
parking-opt14-strips (20)	276	323	1.170
pathways (30)	1997	2114	1.059
pegsol-08-strips (30)	56	76	1.357
pegsol-opt11-strips (20)	45	59	1.311
petri-net-alignment-opt18-strips (20)	859	988	1.150
pipesworld-notankage (50)	418	468	1.119
pipesworld-tankage (50)	398	474	1.191
psr-small (50)	157	157	1
quantum-layout-opt23-strips (20)	637	633	0.994
rovers (40)	1987	2043	1.028
satellite (36)	3988	3979	0.998
scanalyzer-08-strips (30)	982	994	1.012
scanalyzer-opt11-strips (20)	654	661	1.011
snake-opt18-strips (20)	120	129	1.075
sokoban-opt08-strips (30)	412	408	0.990
sokoban-opt11-strips (20)	285	282	0.989
spider-opt18-strips (20)	146	175	1.199
storage (30)	519	531	1.023
termes-opt18-strips (20)	268	272	1.015
tetris-opt14-strips (17)	166	193	1.163
tidybot-opt11-strips (20)	340	377	1.109
tidybot-opt14-strips (20)	355	396	1.115
tpp (30)	1413	1350	0.955
transport-opt08-strips (30)	7021	8778	1.250
transport-opt11-strips (20)	4280	5429	1.268
transport-opt14-strips (20)	4774	6098	1.277
trucks-strips (30)	1065	1061	0.996
visitall-opt11-strips (20)	571	542	0.949
visitall-opt14-strips (20)	1765	1378	0.781
woodworking-opt08-strips (30)	8330	8355	1.003
woodworking-opt11-strips (20)	5500	5505	1.001
zenotravel (20)	459	461	1.004
Finite sum (1828)	59801871	60349238	1.092 ($\sigma = 0.17$)

Table B.1: **Domainwise Comparison of Initial H-Values of $LMCut^{hmax}$ vs $LMCut^{add}$**
The table shows the sum of the initial h values per domain and their respective ratio. Only problems where both algorithms succeeded were considered. The average ratio was 1.092 with a standard deviation σ of 0.17. Domains where $LMCut^{hadd}$ was more successful are highlighted in red.

PCF / Tie Breaking	**Class Name**	**Branch**	**pcfstrategy**
h^{max}	LandmarkCutMaxExploration	main	hmax
h^{add} (3.1.2)	LandmarkCutHAddExploration	main	hadd
Random (3.1.1.1)	LandmarkCutTotallyRandomExploration	main	random
Random Excluding Goal Zone (3.1.1.2)	LandmarkCutAlmostRandomExploration	main	random
h^{add} Terminating on h^{max}(3.1.2)	LandmarkCutHAddExploration	hadd-hmax-terminated	hadd
Random Terminating on h^{max}(3.1.2)	LandmarkCutTotallyRandomExploration	random-excluding-zero-hmax	hadd
Max Effect_of (3.1.3.1)	LandmarkCutMaxTieBreakExploration	tiebreaking-max-effect-of	hmaxtie
Min Effect_of (3.1.3.2)	LandmarkCutMaxTieBreakExploration	tiebreaking-min-effect-of	hmaxtie
Max Precondition_of (**??**)	LandmarkCutMaxTieBreakExploration	tiebreaking-max-precondition-of	hmaxtie
Min Precondition_of (3.1.3.3)	LandmarkCutMaxTieBreakExploration	tiebreaking-min-precondition-of	hmaxtie
Total Number of Operators (**??**)	LandmarkCutMaxTieBreakExploration	tiebreaking-num-ops	hmaxtie
Used (**??**)	LandmarkCutMaxTieBreakExploration	tiebreaking-used	hmaxtie
Not Used (3.1.3.5)	LandmarkCutMaxTieBreakExploration	tiebreaking-used-neg	hmaxtie
Not Used Numeric(3.1.3.6)	LandmarkCutMaxTieBreakExploration	tiebreaking-used-numeric-neg	hmaxtie
Reachability (3.1.3.4)	LandmarkCutMaxTieBreakExploration	tiebreaking-reachability	hmaxtie
Reachability Negative (**??**)	LandmarkCutMaxTieBreakExploration	tiebreaking-reachability-neg	hmaxtie

Table B.2: **Implementation Table** This table shows where to find the implementation of the specific PCF function or tiebreaking strategy. The *ClassName* describes the name of the C++ class in the `lm_cut_landmarks.h` and corresponding `lm_cut_landmarks.cc` file. The branch name is highlighted in the *Branch* column. The *pcfstrategy* describes the keyword that must be provided to `lmcut()` in the `--search` parameter when executing `fast-downward.py` (e.g. `--search "astar(lmcut(pcfstrategy=hadd))"`)

Domain (Tasks)	Refactored	Official	Difference
agricola-opt18-strips (0)	–	–	–
airport (28)	16,879	16,879	0.0
barman-opt11-strips (4)	5,035,782	5,035,782	0.0
barman-opt14-strips (0)	–	–	–
blocks (28)	816,775	816,775	0.0
childsnack-opt14-strips (0)	–	–	–
data-network-opt18-strips (12)	34,242	34,242	0.0
depot (6)	202,677	202,677	0.0
driverlog (13)	265,983	265,983	0.0
elevators-opt08-strips (18)	459,990	459,990	0.0
elevators-opt11-strips (15)	429,875	429,875	0.0
floortile-opt11-strips (6)	861,876	861,876	0.0
floortile-opt14-strips (5)	2,924,453	2,924,453	0.0
freecell (13)	796,086	796,086	0.0
ged-opt14-strips (15)	4,892,449	4,892,449	0.0
grid (2)	76,854	76,854	0.0
gripper (6)	2,426,433	2,426,433	0.0
hiking-opt14-strips (8)	455,864	455,864	0.0
logistics00 (20)	859,645	859,645	0.0
logistics98 (6)	107,126	107,126	0.0
miconic (140)	39,121	39,121	0.0
movie (30)	0	0	0.0
mprime (21)	3,456	3,456	0.0
mystery (17)	6,421,021	6,421,021	0.0
nomystery-opt11-strips (14)	67,418	67,418	0.0
openstacks-opt08-strips (18)	1,229,210	1,229,210	0.0
openstacks-opt11-strips (13)	1,223,197	1,223,197	0.0
openstacks-opt14-strips (2)	650,885	650,885	0.0
openstacks-strips (7)	914,091	914,091	0.0
organic-synthesis-opt18-strips (7)	0	0	0.0
organic-synthesis-split-opt18-strips (10)	215	215	0.0
parcprinter-08-strips (18)	299,182	299,182	0.0
parcprinter-opt11-strips (13)	299,177	299,177	0.0
parking-opt11-strips (1)	3,380	3,380	0.0
parking-opt14-strips (1)	24,364	24,364	0.0
pathways (5)	73,013	73,013	0.0
pegsol-08-strips (27)	2,045,292	2,045,292	0.0
pegsol-opt11-strips (17)	2,187,310	2,187,310	0.0
petri-net-alignment-opt18-strips (8)	216,708	216,708	0.0
pipesworld-notankage (15)	543,431	543,431	0.0
pipesworld-tankage (8)	43,951	43,951	0.0
psr-small (48)	445,141	445,141	0.0
quantum-layout-opt23-strips (11)	57,020	57,020	0.0
rovers (7)	110,002	110,002	0.0
satellite (7)	100,571	100,571	0.0
scanalyzer-08-strips (13)	19,715	19,715	0.0
scanalyzer-opt11-strips (10)	19,711	19,711	0.0
snake-opt18-strips (4)	19,866	19,866	0.0
sokoban-opt08-strips (27)	5,664,459	5,664,459	0.0
sokoban-opt11-strips (20)	3,168,370	3,168,370	0.0
spider-opt18-strips (9)	154,434	154,434	0.0
storage (15)	221,144	221,144	0.0
termes-opt18-strips (3)	1,322,580	1,322,580	0.0
tetris-opt14-strips (3)	7,568	7,568	0.0
tidybot-opt11-strips (12)	48,859	48,859	0.0
tidybot-opt14-strips (4)	18,624	18,624	0.0
tpp (6)	27,810	27,810	0.0
transport-opt08-strips (11)	56,653	56,653	0.0
transport-opt11-strips (6)	56,126	56,126	0.0
transport-opt14-strips (6)	511,729	511,729	0.0
trucks-strips (9)	226,272	226,272	0.0
visitall-opt11-strips (10)	591,596	591,596	0.0
visitall-opt14-strips (5)	751,853	751,853	0.0
woodworking-opt08-strips (15)	200,417	200,417	0.0
woodworking-opt11-strips (10)	200,393	200,393	0.0
zenotravel (12)	100,702	100,702	0.0
Total (880)	**51,049,026**	**51,049,026**	**0.0**
Sum of differences			**0.00**
Arithmetic mean of differences			**0.00**

Table B.3: Comparison of Node Expansions: Refactored Implementation vs Official Fast Downward

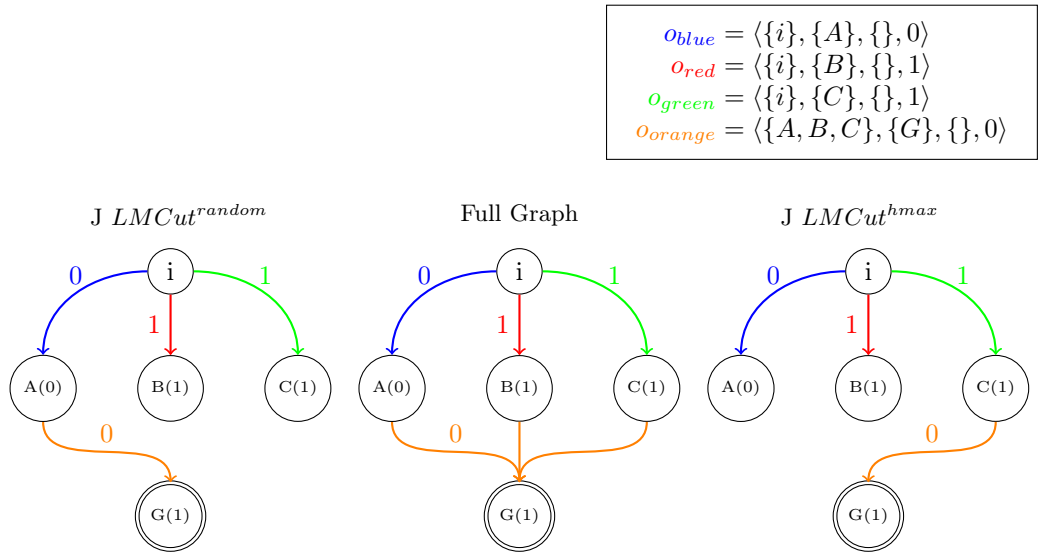


Figure B.3: **Early Termination of** $LMCut^{random}$
Each circle represents a proposition in the state space graph, each arrow represents an operator from the legend above. The h^{max} value is highlighted in brackets in the proposition. The operator costs are displayed above the arrows. The initial state is i. The goal state G is marked with doubled border lines.
The center shows the full graph. The left side shows a possible justification graph (J) for the first iteration of $LMCut^{random}$ when arbitrarily chosing node A, while the right side shows a justification graph for $LMCut^{hmax}$. Here, only the random strategy will terminate after the first iteration, since it has a zero cost path from the initial state to the goal in the justification graph. This is not the case if $LMCut^{random}$ terminates when the goal state has to have a h^{max} value of zero. Also, $LMCut^{hmax}$ will have to make another iteration to catch the red operator.

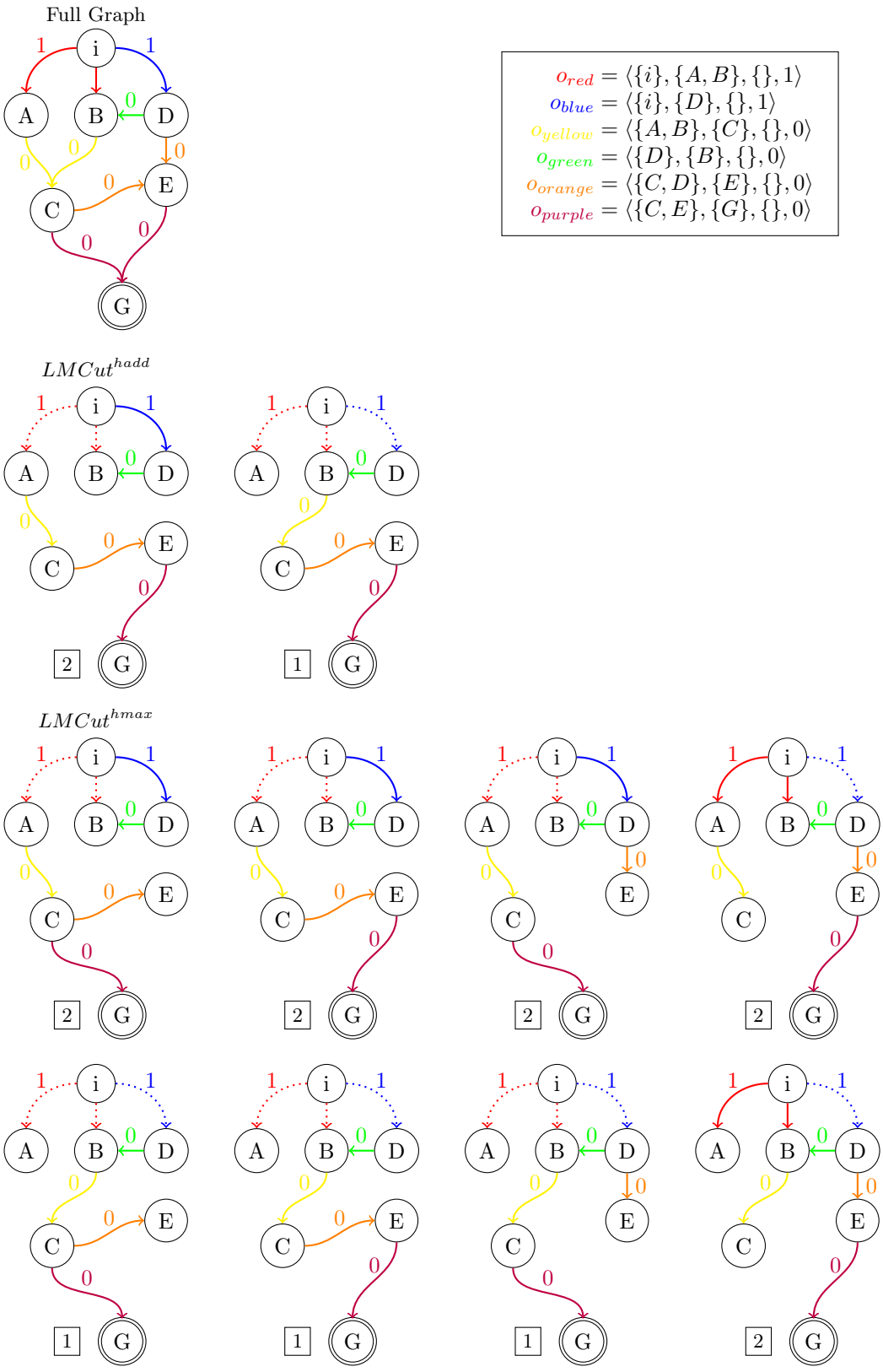


Figure B.4: **Possible $LMCut^{hmax}$ and $LMCut^{hadd}$ Justification Graphs Without Tie-Breaking Rules**
Each circle represents a proposition, each arrow represents an operator with its cost highlighted aside in the same color, according to the legend. Operators in the cut set are display with dotted lines. The initial state is labeled on top with a lower case i. The goal state G is marked with doubled border lines. The box left of the goal node shows the final heuristic value. On top is the full graph. The others are all justification graphs for the first iteration of $LMCut$ for the variant labeled above. $LMCut^{hadd}$ will estimate 2 in 50% of the cases and $LMCut^{hmax}$ in 62.5%.