University of Basel

Operator-counting Constraints for Implicit Abstractions

Master 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: Dr. Gabriele Röger
Supervisor: Clemens Büchner

Leonhard Badenberg
leonhard.badenberg@unibas.ch
2016-055-238

July 8, 2023

Acknowledgments

I want to thank my supervisor Clemens Büchner for his time and guidance. His advice and feedback was always helpful and his patience greatly appreciated. I also want to thank Dr. Gabriele Röger and Prof. Dr. Malte Helmert for giving me the opportunity to write my master thesis in the artificial intelligence research group and providing me with such an interesting topic. Furthermore, I want to thank my family and friends for giving me nothing but encouragement and helpful advice.

Abstract

Abstractions are a common way to obtain heuristic estimates that can be used for optimal planning. Abstractions typically preserve the transition behavior of the original state space to explicitly search for optimal plans in the abstract state space. Implicit abstractions on the other hand, do not preserve the transition behavior. A planning task is instead decomposed into multiple implicit abstractions such that constraints, similar to cost partitioning, are fulfilled.

Operator-counting constraints are constraints that must be fulfilled in every plan. They allow us to combine different types of constraints under a linear program formulation by using a fixed optimization function and using the result as a heuristic estimate.

In this thesis, we construct operator-counting constraints for fork abstractions. Fork abstractions are concrete instances of implicit abstractions. We derive the operator-counting constraints from the inherent cost-partition constraints of the implicit fork abstractions. Our experimental evaluation shows that the heuristic obtained from operator-counting constraints for implicit fork abstractions is computationally too expensive and does not provide a clear accuracy advantage over other operator-counting constraint based heuristics to make up for it.

Table of Contents

1 Introduction

Planning is an artificial intelligence task that focuses on finding solutions to planning tasks. These solutions are called plans and describe a sequence of state-transforming actions from an initial state to a goal. In classical planning those actions have a deterministic outcome. The goal of optimal planning is to find optimal plans with minimal cost. One way to find optimal plans is by guiding a search algorithm over all possible states with the help of a heuristic that estimates how close a state is to the goal. We generally want admissible heuristics that underestimate the true distance to the goal. When computing heuristics there is always an accuracy-computation trade-off. Higher accuracy results in better goal distance estimates and a faster search but require more time on the computation of the heuristic.

A common way to generate useful heuristics that underestimate goal distances is with the help of abstractions. Abstractions map the large state space of a planning task to a smaller state space by abstracting the planning task into one that is easier to solve. Abstractions preserve the transition behavior, such that every plan in the original state space is also a plan in the abstraction. The goal distance in an abstract state space can be used as an admissible heuristic estimate for the original state space.

Many abstraction heuristics are already well understood, such as *domain abstraction* heuristics (Hernádvölgyi and Holte, 2000), *pattern database* (PDB) heuristics (Culberson and Schaeffer, 1998, Edelkamp, 2001), *merge-and-shrink* (M&S) abstraction heuristics (Helmert et al., 2007, 2014) and *cartesian abstraction* heuristics (Seipp and Helmert, 2013, 2018). For abstraction heuristics to be tractable, we need to be able to compute the abstraction and the abstract goal distances efficiently. Abstraction heuristics, that preserve the transition behavior to *explicitly* search for optimal plans in the abstract state space, can only be computed efficiently if the size of the abstract state space is bounded. To overcome the limitation of these *explicit* abstraction heuristics, the notion of *implicit abstractions* was introduced by Katz and Domshlak (2010a). Implicit abstractions no longer preserve the transition behavior but only the costs between two states. This allows implicit abstractions to be a great fit for cost-partitioning. Cost partitioning is a way to combine multiple abstraction heuristics by partitioning the cost of the original actions over multiple abstractions such that we can add them together admissibly, without overestimating the true goal distance.

Some of the other approaches to derive and combine heuristics for optimal planning are based on linear programming. The heuristic estimate is computed by solving a linear program (LP) for every state. The LP optimizes an objective function over some constraints. Pommerening et al. (2014) introduce the *operator-counting framework*. With this framework we can combine different types of constraints, derived by different types of heuristics, by using a fixed optimization function of the LP. Operator-counting constraints are not only able to admissibly combine different heuristics but also to compare them on the basis of their constraints.

In this thesis, we want to create operator-counting constraints for implicit abstractions to investigate them further within the operator-counting framework. We start by giving the necessary preliminary definitions from optimal planning that we will be using throughout this thesis in Chapter 2. We also show how to derive operator-counting constraints from optimal cost-partitioning constraints for explicit abstractions. In Chapter 3, we introduce the general notion of implicit abstractions and specify concrete *fork abstractions* that we will focus on in this thesis. In Chapter 4, we use optimal cost-partitioning constraints for implicit fork abstractions to derive operator-counting constraints for them. Since a similar construction was done for explicit abstractions we can compare some key differences between implicit and explicit abstractions. In Chapter 5, we describe some of the design choices made for the implementation in Fast Downward (Helmert, 2006) and compare the results to four other operator-counting constraint based heuristics. The experimental evaluation shows that the forward fork abstractions we implemented are computationally too expensive and do not provide high enough heuristic accuracy to make up for it. We summarize our findings in Chapter 6 and give possible directions for future research.

2 Background

In this chapter, we introduce some concepts and terminology from the field of cost-optimal planning. This includes preliminary definitions of *classical planning* in general, *heuristic search*, *abstractions*, *operator-counting constraints*, and other notation that will be used in this thesis.

2.1 Classical Planning

In *classical planning*, an agent tries to find a sequence of actions, called *plan*, to deterministically change the agents world from an initial state to a goal state. We define the world of the agent and the actions it can take by a SAS$^+$ *planning task* (Bäckström and Nebel, 1995).

Definition 1 (SAS$^+$ Planning Task)**.** A *planning task* in SAS$^+$ is given by the tuple $\Pi = \langle V, I, O, G\rangle$ where

- V is a set of *state variables* v that each have a *finite-domain* $\mathrm{dom}(v)$,
- I is the *initial state*,
- O is a finite set of *operators* (also called *actions*) over V, and
- G is a partial assignment of V called the *goal*.

A (partial) *assignment* a of V assigns variables $v \in V$ to a specific value $a[v] \in \mathrm{dom}(v)$. A *state* s is a complete assignment of V and $S = \prod_{v \in V} \mathrm{dom}(v)$ is the *state space* of Π.[1] A state s^* is called a *goal state* of G iff $G \subseteq s^*$. An operator o, given by the tuple $o = \langle \mathrm{pre}(o), \mathrm{eff}(o), \mathit{cost}(o)\rangle$, has a *precondition* $\mathrm{pre}(o)$ and *effect* $\mathrm{eff}(o)$, which are partial assignments over V, and a *cost* $\mathit{cost}(o) \in \mathbb{R}_0^+$.

For an assignment a, $\mathcal{V}(a) \subseteq V$ denotes the set of state variables that is *instantiated* by a.

[1] We note that V is ordered. Therefore, the state space consists only of combinations of different values the variables can take and is not influenced by different variable orders.

Definition 2 (Applicability and Successor States)**.** An operator o is *applicable* in a state s iff $\text{pre}(o) \subseteq s$. Applying o in state s changes the value of each $v \in \mathcal{V}(\text{eff}(o))$ from $s[v]$ to $\text{eff}(o)[v] = s[\![o]\!][v]$ in the *successor state* $s[\![o]\!]$. If there exists no $v \in \mathcal{V}(\text{eff}(o))$ then $s[\![o]\!][v] = s[v]$

The set of values of v for which an operator o can be applied is denoted by $\text{Pre}(o)[v]$. It is either equal to $\text{pre}(o)[v]$ if $v \in \mathcal{V}(\text{pre}(o))$ or to $\text{dom}(v)$ otherwise.

Definition 3 (Transition System)**.** A *transition system* is given by the tuple $\mathcal{T} = \langle S, L, c, T, s^0, S^* \rangle$ where

- S is a finite set of *states*,
- L is a finite set of transition *labels*,
- $c : L \rightarrow \mathbb{R}_0^+$ is a *label cost* function,
- $T \subseteq S \times L \times S$ is the *transition relation*,
- $s^0 \in S$ is the *initial state*, and
- $S^* \subseteq S$ is the set of *goal states*.

A planning task $\Pi = \langle V, I, O, G \rangle$ induces the transition system $\mathcal{T}(\Pi) = \langle S, L, c, T, s^0, S^* \rangle$, where

- S is the set of all states over V,
- L is the set of operators O,
- $c(o) = \mathit{cost}(o)$ for all $o \in O$,
- $T = \{\langle s, o, s' \rangle \mid s \in S,\ o \text{ applicable in } s,\ s' = s[\![o]\!]\}$,
- $s^0 = I$, and
- $S^* = \{s^* \in S \mid G \subseteq s^*\}$.

A *path* is a sequence of operators that leads from one state to another following the transitions of $\mathcal{T}$. Any path from a state $s \in S$ to a goal state $s^* \in S^*$ is called an *s-plan.* An s-plan is just called *plan* if $s = s^0$. Any s-plan with minimal cost is called *optimal.* In classical planning we distinguish between *satisficing planning*, where the goal is to find any valid plan, and *optimal planning*, where the goal is to find optimal plans. We are interested in the latter.

Definition 4 (Causal Graph)**.** The *causal graph* $CG(\Pi)$ of a planning task $\Pi = \langle V, I, O, G \rangle$ is a digraph over nodes V. An arc $\langle v, v' \rangle$ is in $CG(\Pi)$ iff $v \neq v'$ and there exists an operator $o \in O$ such that $v \in \mathcal{V}(\text{eff}(o)) \cup \mathcal{V}(\text{pre}(o))$ and $v' \in \mathcal{V}(\text{eff}(o))$. We say that $\langle v, v' \rangle$ is *induced* by o and denote the sets of immediate successors and predecessors of v in $CG(\Pi)$ by $\text{succ}(v)$ and $\text{pred}(v)$, respectively.

Definition 5 (Domain Transition Graph)**.** The *domain transition graph* $DTG(v, \Pi)$ of a variable $v \in V$ in a planning task $\Pi = \langle V, I, O, G \rangle$ is a digraph over the nodes $\mathrm{dom}(v)$. An arc $\langle \theta, \theta' \rangle$ is in $DTG(v, \Pi)$ iff $\theta \neq \theta'$ and there exists an operator $o \in O$ such that both $\mathrm{eff}(o)[v] = \theta'$ and $\theta \in \mathrm{Pre}(o)[v]$. We say that $\langle \theta, \theta' \rangle$ is *induced* by o.

2.2 Heuristic Search and Abstractions

Heuristic search algorithms are widely used for optimal planning. They use heuristic estimates of the goal distance as guidance to find an optimal plan.

Definition 6 (Heuristic)**.** A *heuristic* of a transition system $\mathcal{T} = \langle S, L, c, T, s^0, S^* \rangle$, is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$ that maps a state $s \in S$ to an estimate of its goal distance. The *true goal distance* or *perfect heuristic* is written as h^*, where $h^*(s)$ is equal to the optimal s-plan or equal to infinity if no goal state can be reached from s.
A heuristic h is called *admissible* if for all states s: $h(s) \leq h^*(s)$.
Let h_1 and h_2 be two admissible heuristics, if $h_1(s) \leq h_2(s)$ for all $s \in S$ then we say that h_2 *dominates* h_1 as it gives a better estimate of the true goal distance h^*.

Abstractions are a way of simplifying the transition system while preserving paths to help find admissible heuristics.

Definition 7 (Abstraction)**.** An *abstraction* of a transition system $\mathcal{T} = \langle S, L, c, T, s^0, S^* \rangle$ is a function $\langle \mathcal{T}_\alpha, \alpha \rangle$ where

- $\mathcal{T}_\alpha = \langle S_\alpha, L_\alpha, c_\alpha, T_\alpha, s^0_\alpha, S^*_\alpha \rangle$ is a transition system,
- $\alpha : S \to S_\alpha$ is an abstraction function, such that
  - $\alpha(s^0) = s^0_\alpha$,
  - $\alpha(s^*) \in S^*_\alpha$ for all $s^* \in S^*$, and
  - $T_\alpha = \{\langle \alpha(s), o, \alpha(s') \rangle \mid \langle s, o, s' \rangle \in T\}$

An abstraction of a planning task Π is defined by the abstraction of its induced transition system $\mathcal{T}(\Pi)$.

Definition 8 (Abstraction Heuristic)**.** The *abstraction heuristic* h^α induced by the abstraction function $\alpha : S \to S_\alpha$ is the function that maps each state $s \in S$ to its true goal distance $h^*_{\mathcal{T}_\alpha}(\alpha(s))$ in the abstract transition system $\mathcal{T}_\alpha$.

An abstraction heuristic for a planning task Π is defined by the abstraction heuristic for its induced transition system $\mathcal{T}(\Pi)$.

2.2.1 Composition of Abstraction Heuristics

We want to find an admissible heuristic that exploits the information of different admissible heuristics by combining them. We note that higher values of admissible heuristics correspond to better estimates of the true goal distance. There are two main ways we can combine multiple admissible abstraction heuristics stemming from different abstractions of the same planning task Π. The first is to take the maximum over the different heuristic values in each state (Holte et al., 2006). This heuristic dominates its components as it always selects the strongest one. Another way is to try to add the different heuristic values such that the resulting heuristic is still admissible, as summing admissible heuristics is not admissible in general. Being able to add different heuristics allows for a better combination of heuristic that can lead to much better estimations. One way to guarantee that we can admissibly add abstraction heuristics is through *cost partitioning* (Katz and Domshlak, 2008b, Yang et al., 2008). Cost partitioning is a way to distribute the cost of a planning task Π among its abstractions such that summing over these abstraction heuristics is guaranteed to be admissible. For now we focus on a definition similar to the one by Seipp et al. (2017a).

Definition 9 (Cost Partitioning)**.** Let $\Pi = \langle V, I, O, G\rangle$ be a planning task and $\mathcal{A} = \{\langle \mathcal{T}_i, \alpha_i\rangle\}_{i=1}^m$ be a set of abstractions of $\mathcal{T}(\Pi) = \langle S, L, c, T, s^0, S^*\rangle$. An (operator) *cost partitioning* over $\mathcal{A}$ is a set of cost functions $\mathcal{C} = \{c_i\}_{i=1}^m$ whose sum is bounded by c: $\sum_{i=1}^m c_i(o) \leq c(o)$ for all $o \in O$.

Definition 10 (Cost-partitioned Heuristic)**.** The *cost-partitioned heuristic* $h_{\mathcal{C}}^{\mathcal{A}}$ is defined as $h_{\mathcal{C}}^{\mathcal{A}}(s) := \sum_{i=1}^m h_{c_i}^{\alpha_i}(s)$, where $h_{c_i}^{\alpha_i}$ is the i-th abstraction heuristic using the partitioned cost c_i.
A cost partitioning $\mathcal{C}^*$ is *optimal* for state s if $h_{\mathcal{C}^*}^{\mathcal{A}}(s) \geq h_{\mathcal{C}}^{\mathcal{A}}(s)$ for all cost partitionings $\mathcal{C}$. We write $h_{\mathcal{A}}^{\mathrm{OCP}}(s)$ for the optimal cost-partitioned value.

One simple approach to achieve such an *operator-cost partitioning* is to count the whole cost of an operator, while computing a single heuristic in the ensemble, and setting the cost of that operator to zero in all the other heuristics in the ensemble. This *zero-one cost-partitioning* was, for example, exploited for pattern database heuristics (Edelkamp, 2006, Felner et al., 2004). Another approach is to split the cost uniformly. This *uniform cost partitioning* was, for example, exploited for landmarks (Karpas and Domshlak, 2009). There exist infinitely many other approaches to partition the operator cost $c(o)$ admissibly into real numbers $c_i \in [0, c(o)]$. The restriction of c_i to take non-negative values can often be lifted, in which case we speak of *general cost-partitioning* (Pommerening et al., 2015). With infinitely many choices, the question becomes which operator-cost partitioning scheme one uses. Not all of these are necessarily better than simply taking the maximum over the admissible heuristics, which works entirely without supervision. Therefore, finding the optimal cost partitioning is desirable. As the quality of each operator-cost partition varies between search states, the optimal cost partitioning could be different for each state.

2.2.2 Optimal Cost-partitioning

Katz and Domshlak (2010b) introduced a fully unsupervised procedure to find the optimal cost partitioning for each state. The procedure takes

- a deterministic planning task Π,
- a state s, and
- a set of admissible heuristics

and finds the *optimal operator-cost partition* for s. To find the optimal operator-cost partition we have to create different linear programs that solve optimization problems. Katz and Domshlak (2008b, 2010b) show how to derive constraints to create such linear programs.

Definition 11 (Cost-partitioning LP for Abstractions)**.** Let $\Pi = \langle V, I, O, G\rangle$ be a planning task and $\mathcal{A} = \{\langle \mathcal{T}_i, \alpha_i\rangle\}_{i=1}^{m}$ be a set of abstractions of $\mathcal{T}(\Pi) = \langle S, L, c, T, s^0, S^*\rangle$. Let $SCT_i \subseteq T_i$ be a subset containing all *state-changing* transitions $\langle s, o, s'\rangle$ with $s \neq s'$ and $s, s' \in S_i$. The estimate of the *optimal cost-partitioning heuristic* $h^{\mathrm{OCP}}_{\mathcal{A}}$ for $\mathcal{A}$ in state $s \in S$ is the objective value of the following linear program (LP) or ∞ if infeasible.

$$\text{Maximize} \sum_{i=1}^{m} h_i(\alpha_i(s)) \text{ subject to}$$

$$\begin{aligned}
d(\alpha_i(s), \alpha_i(s')) &= 0 && \text{for all } 1 \leq i \leq m \text{ and } \alpha_i(s') = \alpha_i(s)\\
d(\alpha_i(s), \alpha_i(s'')) &\leq d(\alpha_i(s), \alpha_i(s')) + c_i(o) && \text{for all } 1 \leq i \leq m \text{ and } \langle \alpha_i(s'), o, \alpha_i(s'')\rangle \in SCT_i\\
h_i(\alpha_i(s)) &\leq d(\alpha_i(s), \alpha_i(s^*)) && \text{for all } 1 \leq i \leq m \text{ and } \alpha_i(s^*) \in S_i^*\\
\sum_{i=1}^{m} c_i(o) &\leq \mathit{cost}(o) && \text{for all } o \in O,
\end{aligned}$$

where $d(\alpha_i(s), \alpha_i(s'))$ is the cost of the cheapest path from $\alpha_i(s)$ to $\alpha_i(s')$ in $\mathcal{T}_i$, and all variables ≥ 0.

We note that obtaining an optimal general cost-partitioning, by allowing negative c_i, through this LP is only possible if we lift the restriction of $\langle \alpha_i(s'), o, \alpha_i(s'')\rangle$ from being in SCT_i to being in T_i. Else, negative c_i that are part of a cycle within T_i would not appear in $h_i(\alpha_i(s))$ of this LP formulation and the LP would no longer be bounded if we lifted the restriction of c_i to be non-negative. However, the restriction to non-negative values of the d-variables and the heuristic variables h_i can always be lifted.

2.3 Operator-counting Constraints

Another way to admissibly combine heuristics can also be done with the help of the operator-counting framework introduced by Pommerening et al. (2014). This framework does not combine heuristics stemming from abstractions directly but rather combines different heuristics that are based on linear programming. When the objective function is fixed, constraints from different heuristics can be combined admissibly leading to a heuristic that is better than maximizing over the components.

Definition 12 (Operator-counting Constraints)**.** Let $\Pi = \langle V, I, O, G\rangle$ be a planning task, and let s be one of its states. Let $\mathcal{Y}$ be a set of non-negative real valued variables consisting of $\{\mathsf{Y}_o \mid o \in O\}$, where $\mathsf{Y}_o \geq 0$ is an integer variable called *operator-counting variable*, and other auxiliary variables.

A set of linear inequalities c_s over $\mathcal{Y}$ is called an *operator-counting constraint*[2] for s if for every s-plan π setting each Y_o to Y_o^π is a feasible variable assignment, where Y_o^π is the number of occurrences of o in π.
A *constraint set* $C(s)$ is a set of operator-counting constraints for s, where the only common variables between constraints are the operator-counting variables.

Based on these constraints Pommerening et al. (2014) define an *integer* or *linear program.*

Definition 13 (Operator-counting Integer/Linear Program)**.** The *operator-counting integer program* IP_C for constraint set C is:

$$\text{Minimize} \sum_{o \in O} \mathit{cost}(o) \cdot \mathsf{Y}_o \text{ subject to } C.$$

An operator-counting integer program IP_C only allows the operator-counting variables to take integer values, whereas the *operator-counting linear program* LP_C is the LP-relaxation of IP_C that allows for operator-counting variables to take continuous values.

Since an optimal plan π has a number of operator occurrences that is a feasible solution to C, the cost of an optimal plan π acts as an upper bound for the objective value of the IP, which in turn acts as an upper bound for the objective value of the LP. This induces the following admissible heuristic estimates.

Definition 14 (IP and LP Heuristic)**.** The *IP-heuristic* $h_C^{\mathrm{IP}}(s)$ is the objective value of the integer program $\mathrm{IP}_{C(s)}$. The *LP-heuristic* $h_C^{\mathrm{LP}}(s)$ is the objective value of the linear program $\mathrm{LP}_{C(s)}$. If the programs are infeasible, a heuristic value of ∞ is used instead.

We note that adding more constraints can only make the heuristic a stronger estimate of the true cost. Let C and C' be two constraint sets and $C(s) \subseteq C'(s)$ for all states s. Then the IP/LP heuristic for C' dominates the respective heuristic for C: $h_C^{\mathrm{IP}}(s) \leq h_{C'}^{\mathrm{IP}}(s)$ and $h_C^{\mathrm{LP}}(s) \leq h_{C'}^{\mathrm{LP}}(s)$.

2.3.1 Cost-partitioning Constraints for Abstractions

Recall that the optimal cost-partitioning for abstractions can be derived through linear programming. To be able to use the constraints from Definition 11 in the operator-counting framework we have to fix the objective function and derive new operator-counting constraints that result in the same objective value as the initial cost-partitioning.
We note that the cost-partitioning LP from Definition 11 is a maximization problem, whereas we defined operator-counting constraints to be a minimization problem. Pommerening et al. (2014), however, show that the *dual* of the cost-partitioning LP is a direct fit for the operator-counting framework. *Duality* in linear programs refers to the fact that every LP has an alternative view. A maximization LP can be formulated as a minimization LP and the other way around. Variables and constraints swap roles and so do the objective coefficients and bounds. The objective value of the primal view is a lower bound on the objective value

[2] Not to be confused with the non-state-dependant cost function c.

of its dual and the objective value of the dual an upper bound on its primal. If an optimal solution exists the objective values for both views are equal.

Definition 15 (Operator-counting LP for Abstractions)**.** The dual of the cost-partitioning LP is given by the following LP formulated with the operator-counting constraints $c_{i,s}^{\mathrm{OCP}}$.

$$\text{Minimize} \sum_{o \in O} \mathit{cost}(o) \cdot \mathsf{Y}_o \text{ subject to } c_{i,s}^{\mathrm{OCP}} \text{ for all } 1 \leq i \leq m$$

Where the *optimal cost-partitioning constraint* $c_{i,s}^{\mathrm{OCP}}$ consists of

1. a transition count inequality for each operator $o \in O$:

$$\mathsf{Y}_o \geq \sum_{\substack{t \in SCT_i \\ t \text{ labeled with } o}} \mathsf{Y}_t^i,$$

2. a goal inequality $\sum_{\alpha_i(s') \in S_i^*} \mathrm{G}_i(s') \geq 1$,

3. a transition flow inequality for all $\alpha_i(s') \neq \alpha_i(s)$:

$$\sum_{\substack{t \in SCT_i \\ t \text{ ends in } s'}} \mathsf{Y}_t^i - \sum_{\substack{t \in SCT_i \\ t \text{ starts in } s'}} \mathsf{Y}_t^i \geq \begin{cases} \mathrm{G}_i(s') & \text{if } \alpha_i(s') \in S_i^* \\ 0 & \text{if } \alpha_i(s) \notin S_i^* \end{cases},$$

where SCT_i is the set of state-changing transitions as defined in Definition 11 and $\mathrm{G}_i(s')$ denotes how often the goal state s' is reached.

Recall that the restriction of the operator cost variables c_i to be non-negative in the primal can only be lifted when also lifting the restriction of $\langle \alpha_i(s'), o, \alpha_i(s'') \rangle$ to be in SCT_i. In that case we can adjust the dual accordingly by also allowing $t \in SCT_i$ to be in T_i instead. The inequality 1 will turn into an equality. If the restriction to non-negative values of the d-variables and the heuristic variables h_i in the primal cost-partitioning LP are also lifted, then the inequalities 2 and 3 respectively would also turn into equalities.
The LP from Definition 15 perfectly fits the operator-counting framework and can even be interpreted as an integer program, in which case it would give a better heuristic than optimal-cost partitioning, although it cannot be computed in polynomial time.

3 Implicit Abstractions

In this chapter, we introduce the notion of *implicit abstractions* (Katz and Domshlak, 2010a). We look at the general idea behind them and then repeat the definition of *fork abstractions* according to Katz and Domshlak (2010a), for which we will create operator-counting constraints later.

3.1 General Idea

So far, an abstraction of a planning task Π was defined by the abstraction $\langle \mathcal{T}_\alpha, \alpha \rangle$ of its induced transition system $\mathcal{T}(\Pi)$. Recall from Definition 7 that each transition in the original transition system $\mathcal{T}$ is preserved explicitly in $\mathcal{T}_\alpha$. Abstractions that are created in this way can be called *explicit abstractions.* The corresponding explicit abstraction heuristic, that explicitly searches for optimal plans in the abstract state space, can only be computed efficiently if the size of the abstract space is bounded.
Motivated by the constant bound on the size of the abstract space induced by explicit abstractions, Katz and Domshlak (2010a) introduce what they call *implicit abstractions.* According to Katz (2010) the basic idea is simply not to rely on abstract problems that are easy to solve because they are small, but instead to rely on abstract problems belonging to provably tractable fragments of optimal planning. Therefore, implicit abstractions should remove the requirement on the abstraction size to be small.
Implicit abstractions want to abstract a planning task Π over states S to a different planning task Π^α directly, such that the transition system does not have to be represented explicitly and the abstraction and the induced abstraction function $\alpha : S \to S^\alpha$ can be computed in polynomial time. With that in mind Katz and Domshlak (2010a) define (additive) implicit abstractions that, instead of preserving the individual transitions, they pose a weaker condition that allows to abstract over the planning task directly and is similar to the cost partitioning.

Definition 16 (Implicit Abstraction)**.** An *implicit abstraction* of a planning task Π is given by the pair $\langle \Pi_\alpha, \alpha \rangle$, where

- $\Pi_\alpha = \langle V_\alpha, I_\alpha, O_\alpha, G_\alpha \rangle$ is an abstract planning task implicitly inducing the abstract transition system $\mathcal{T}_\alpha = \langle S_\alpha, L_\alpha, c_\alpha, T_\alpha, s^0_\alpha, S^*_\alpha \rangle$,

- $\alpha : S \to S_\alpha$ is an abstraction function, such that
  - $\alpha(s^0) = s^0_\alpha$, $\alpha(s^*) \in S^*_\alpha$ for all $s^* \in S^*$, and,
  - for all pairs of states $s, s' \in S$ it holds that

$$\mathit{cost}(\alpha(s), \alpha(s')) \leq \mathit{cost}(s, s').$$

Implicit abstractions generalize Definition 7 by no longer preserving paths in the induced transition system, allowing us to implicitly abstract the planning task without creating the transition system.

Definition 17 (Additive Set of Implicit Abstractions)**.** An *additive implicit abstraction set* of a planning task Π is a set of pairs $\mathcal{A} = \{\langle \Pi_i, \alpha_i \rangle\}_{i=1}^m$ where, for $1 \leq i \leq m$,

- $\Pi_i = \langle V_i, I_i, O_i, G_i \rangle$ is an abstract planning task inducing the abstract transition system $\mathcal{T}_i = \langle S_i, L_i, c_i, T_i, s^0_i, S^*_i \rangle$,
- $\alpha_i : S \to S_i$ is an abstraction function, such that
  - $\alpha_i(s^0) = s^0_i$, $\alpha_i(s^*) \in S^*_i$ for all $s^* \in S^*$, and,
  - for all pairs of states $s, s' \in S$ it holds that

$$\sum_{i=1}^{m} \mathit{cost}(\alpha_i(s), \alpha_i(s')) \leq \mathit{cost}(s, s'), \tag{3.1}$$

Definition 18 (Implicit Abstraction Heuristic)**.** Let Π be a planning task over the states S, and let $\mathcal{A} = \{\langle \Pi_i, \alpha_i \rangle\}_{i=1}^m$ be an additive implicit abstraction set of Π and $h^{\alpha_i}(\alpha_i(s)) = h^*_i(\alpha_i(s))$ be an *implicit abstraction heuristic*, where $h^*_i(\alpha_i(s))$ is the cost of the optimal plan for Π_i. Then $h^{\mathcal{A}}(s) = \sum_{i=1}^m h^{\alpha_i}(\alpha_i(s)) = \sum_{i=1}^m h^*_i(\alpha_i(s))$ is a cost-partitioned implicit abstraction heuristic for Π.

We can see that the implicit abstraction heuristic $h^{\mathcal{A}}(s) = \sum_{i=1}^m h^{\alpha_i}(\alpha_i(s))$ is an admissible heuristic for Π. This is because each h^*_i is an admissible heuristic for Π_i, respectively, and the condition from Equation 3.1 for the additive implicit abstraction set $\mathcal{A} = \{\langle \Pi_i, \alpha_i \rangle\}_{i=1}^m$ ensures that the sum over those admissible heuristics is admissible, as we have already seen in Definition 9.

Katz and Domshlak (2010a) show that calculating the implicit abstraction heuristic for an additive implicit abstraction set $\mathcal{A} = \{\langle \Pi_i, \alpha_i \rangle\}_{i=1}^m$ of Π is tractable, by calculating $h^*_i(\alpha_i(s))$ for each component. For all Π_i where calculating this additive fragment is not tractable, we can define a new additive implicit abstraction set $\mathcal{A}_i = \{\langle \Pi_{i,j}, \alpha_{i,j} \rangle\}_{j=1}^{m_i}$. Each $\Pi_{i,j}$ where calculating this additive fragment is still not tractable can be further abstracted in a similar fashion. We continue this until we are left with only tractable additive fragments for which we can obtain the true cost.

Because each composition $\mathcal{A}' = \bigcup_{i=1}^m \{\langle \Pi_{i,j}, \alpha_{i,j} \circ \alpha_i \rangle\}_{j=1}^{m_i}$ is an additive implicit abstraction set of the initial planning task Π, we can tractably calculate an admissible heuristic of the

components $\Pi_{i,j}$ and Π_i itself, which in turn guarantees a tractable admissible heuristic for Π. We refer to Katz and Domshlak (2010a) for a formal proof of this claim.
We note that implicit abstraction heuristics directly correspond to tractable fragments of optimal planning. We basically sum over m true goal distances h^*, where m is polynomial in the description size of Π and h^* is computable in polynomial time.

3.2 Fork Decompositions

Katz and Domshlak (2010a) specify a general framework to obtain additive implicit abstraction sets, called *acyclic causal-graph decompositions*. The idea is to decompose the given planning task Π along its causal graph $CG(\Pi)$. Because Π is abstracted along a subgraph of Π's causal graph, we obtain abstract problems with a *specific structure*.

Definition 19 (Causal Graph Abstractions)**.** Let $\Pi = \langle V, O, I, G\rangle$ be a planning task, and let $\mathbf{G} = \{\mathcal{G}_i = \langle V_{\mathcal{G}_i}, E_{\mathcal{G}_i}\rangle\}_{i=1}^m$ be a set of acyclic subgraphs of the causal graph $CG(\Pi)$ called *acyclic causal-graph decomposition*. An abstraction $\langle \Pi_{\mathcal{G}_i}, \alpha_i\rangle$ is an *acyclic causal-graph abstraction* of Π over $\mathcal{G}_i$ if $\alpha_i : S \to S_i$ is the projection mapping $\alpha_i(s) = s[V_{\mathcal{G}_i}]$ and $\Pi_{\mathcal{G}_i} = \langle V_{\mathcal{G}_i}, O_{\mathcal{G}_i}, I_{\mathcal{G}_i}, G_{\mathcal{G}_i}\rangle$, where

- $I_{\mathcal{G}_i} = I[V_{\mathcal{G}_i}], G_{\mathcal{G}_i} = G[V_{\mathcal{G}_i}]$,
- $O_{\mathcal{G}_i} = \bigcup_{o\in O} O_{\mathcal{G}_i}(o)$ where each $O_{\mathcal{G}_i}(o) = \{o^{v_1}, \dots, o^{v_k}\}$ is a set of operators with a unary-effect on a variable $v_j \in \{v_1, \dots, v_k\} = \mathcal{V}(\text{eff}(o)) \cap V_{\mathcal{G}_i}$ with

$$\text{eff}(o^{v_j})[v] = \begin{cases} \text{eff}(o)[v_j], & v = v_j \\ \text{unspecified}, & \text{otherwise} \end{cases}$$

$$\text{pre}(o^{v_j})[v] = \begin{cases} \text{pre}(o)[v_j], & v = v_j \\ \text{pre}(o)[v], & \langle v, v_j\rangle \in E_{\mathcal{G}_i} \land v \neq \mathcal{V}(\text{eff}(o)) \\ \text{eff}(o)[v], & \langle v, v_j\rangle \in E_{\mathcal{G}_i} \land v = \mathcal{V}(\text{eff}(o)) \\ \text{unspecified}, & \text{otherwise} \end{cases}$$

- For each operator $o \in O$,

$$\sum_{o' \in O_{\mathcal{G}_i}(o)} cost_{\mathcal{G}_i}(o') \leq cost(o).$$

The subgraph $\mathcal{G}_i$ induces some topological ordering of $\{v_1, \dots, v_k\} \subseteq V_{\mathcal{G}_i}$.

We note that $O_{\mathcal{G}_i}$ can be of larger size than O, and the construction of the operators in $O_{\mathcal{G}_i}$ can be understood as splitting each original operator o into unary-effect operators o' that, when executed sequentially, act in the same way as o does in $\Pi_{\mathcal{G}_i}$. The set $O_{\mathcal{G}_i}(o)$ denotes the set of unary-effect operators *induced* by o. As it will be useful to keep track of the effect of a unary-effect operator o', we also introduce the notation $O[v] = \{o \in O : v \in \mathcal{V}(\text{eff}(o))\}$ for the set of operators that have an effect on v.

Remark. Any set of acyclic causal-graph abstractions $\{\langle \Pi_{\mathcal{G}_i}, \alpha_i \rangle\}_{i=1}^n$ with $\{\mathcal{G}_i\}_{i=1}^n \subseteq \mathbf{G}$ is an additive implicit abstraction set if it fulfills the *additivity constraints*

$$\sum_{i}^{n} \sum_{o' \in O_{\mathcal{G}_i}(o)} cost_{\mathcal{G}_i}(o') \leq cost(o), \tag{3.2}$$

where $c_i(o)$ denotes the cost of an operator $o \in O$ within the i-th component of the set.

We note that if the sum of the cost over all components and unary-effect operators of an operator o never exceeds the cost of the operator in the original task, then the condition from Equation 3.1 must be fulfilled and $\{\langle \Pi_{\mathcal{G}_i}, \alpha_i \rangle\}_{i=1}^n$ is an additive implicit abstraction. The concrete examples of implicit abstractions that we will be discussing in the thesis are based on specific acyclic causal-graph decompositions, called *fork decompositions.* Fork decompositions are based on two fragments of tractable cost-optimal planning (Katz and Domshlak, 2008a) for tasks with *fork* and *inverted-fork* structured causal graphs.

Definition 20 (Forward and Inverted Forks)**.** For a planning task $\Pi = \langle V, O, I, G \rangle$, and a variable $v \in V$,

1. the *forward fork* of Π with root v is a subgraph $\mathcal{G}_v^{\mathrm{f}}$ of $CG(\Pi)$ over nodes $V_{\mathcal{G}_v^{\mathrm{f}}} = \{v\} \cup \mathrm{succ}(v)$ and edges $E_{\mathcal{G}_v^{\mathrm{f}}} = \{\langle v, u \rangle \mid u \in \mathrm{succ}(v)\}$, and

2. the *inverted fork* of Π with sink v is a subgraph $\mathcal{G}_v^{\mathrm{i}}$ of $CG(\Pi)$ over nodes $V_{\mathcal{G}_v^{\mathrm{i}}} = \{v\} \cup \mathrm{pred}(v)$ and edges $E_{\mathcal{G}_v^{\mathrm{i}}} = \{\langle u, v \rangle | u \in \mathrm{pred}(v)\}$.

The sets of all forward forks and inverted forks of Π are denoted by $\mathbf{G}_{\mathcal{F}} = \{\mathcal{G}_v^{\mathrm{f}}\}_{v \in V}$ and $\mathbf{G}_{\mathcal{I}} = \{\mathcal{G}_v^{\mathrm{i}}\}_{v \in V}$, respectively.

As forward forks and inverted forks are acyclic digraphs, we can see that $\mathbf{G}_{\mathcal{F}}$ and $\mathbf{G}_{\mathcal{I}}$ are acyclic causal-graph decompositions, and so is $\mathbf{G}_{\mathcal{FI}} = \mathbf{G}_{\mathcal{F}} \cup \mathbf{G}_{\mathcal{I}}$ (Katz and Domshlak, 2010a). This allows us to define the following causal-graph abstractions.

Definition 21 (Fork Abstractions)**.** For any planning task $\Pi = \langle V, I, O, G \rangle$,

1. any acyclic causal-graph abstraction $\langle \Pi_r^{\mathrm{f}}, \alpha_r^{\mathrm{f}} \rangle$ of Π over $\mathcal{G}_r^{\mathrm{f}}$ is called a *forward fork abstraction*, and the set of all forward fork abstractions of Π over $\mathbf{G}_{\mathcal{F}}$ is given by $\mathcal{A}_{\mathcal{F}} = \{\langle \Pi_v^{\mathrm{f}}, \alpha_v^{\mathrm{f}} \rangle\}_{v \in V}$;

2. any acyclic causal-graph abstraction $\langle \Pi_r^{\mathrm{i}}, \alpha_r^{\mathrm{i}} \rangle$ of Π over $\mathcal{G}_r^{\mathrm{i}}$ is called an *inverted fork abstraction*, and the set of all inverted fork abstractions of Π over $\mathbf{G}_{\mathcal{I}}$ is given by $\mathcal{A}_{\mathcal{I}} = \{\langle \Pi_v^{\mathrm{i}}, \alpha_v^{\mathrm{i}} \rangle\}_{v \in V}$;

3. the set of all forward and inverted fork abstractions of Π over $\mathbf{G}_{\mathcal{FI}} = \mathbf{G}_{\mathcal{F}} \cup \mathbf{G}_{\mathcal{I}}$ is given by $\mathcal{A}_{\mathcal{FI}} = \{\langle \Pi_v^{\mathrm{f}}, \alpha_v^{\mathrm{f}} \rangle, \langle \Pi_v^{\mathrm{i}}, \alpha_v^{\mathrm{i}} \rangle\}_{v \in V}$.

Any set of forward fork abstractions $\mathcal{A}^{\mathrm{f}} = \{\langle \Pi_i^{\mathrm{f}}, \alpha_i^{\mathrm{f}} \rangle\}_{i=1}^m \subseteq \mathcal{A}_{\mathcal{F}}$ with $\{\mathcal{G}_i^{\mathrm{f}}\}_{i=1}^m \subseteq \mathbf{G}_{\mathcal{F}}$, inverted fork abstractions $\mathcal{A}^{\mathrm{i}} = \{\langle \Pi_i^{\mathrm{i}}, \alpha_i^{\mathrm{i}} \rangle\}_{i=1}^n \subseteq \mathcal{A}_{\mathcal{I}}$ with $\{\mathcal{G}_i^{\mathrm{i}}\}_{i=1}^n \subseteq \mathbf{G}_{\mathcal{I}}$, and forward and inverted fork abstractions $\mathcal{A}^{\mathrm{fi}} = \{\langle \Pi_i^{\mathrm{f}}, \alpha_i^{\mathrm{f}} \rangle\}_{i=1}^m \cup \{\langle \Pi_i^{\mathrm{i}}, \alpha_i^{\mathrm{i}} \rangle\}_{i=1}^n \subseteq \mathcal{A}_{\mathcal{FI}}$ with $\{\mathcal{G}_i^{\mathrm{f}}\}_{i=1}^m \cup \{\mathcal{G}_i^{\mathrm{i}}\}_{i=1}^n \subseteq \mathbf{G}_{\mathcal{FI}}$ is an additive implicit abstraction set if it fulfills the additivity constraints from Equation 3.2.

Furthermore, the cost-partitioned implicit abstraction heuristics $h^{\mathcal{A}^{\mathrm{f}}}$, $h^{\mathcal{A}^{\mathrm{i}}}$, and $h^{\mathcal{A}^{\mathrm{fi}}}$ are admissible estimates of h^* in Π.[3] Even the optimal cost-partitioning for implicit abstractions can be obtained by the linear program formulation of Katz and Domshlak (2010b). In the next chapter we will use that formulation to create operator-counting constraints for implicit fork abstractions.

But before that, we will take another look at the tractability of the fork abstractions. The causal graphs of the planning tasks in $\{\Pi^{\mathrm{f}}_v\}_{v\in V}$ and $\{\Pi^{\mathrm{i}}_v\}_{v\in V}$ form directed forks and directed inverted forks, respectively. Unfortunately optimal planning for these types of problems is inherently hard, mainly due to root variables having large domains (Helmert, 2003, 2004). By limiting the domain of the root and sink variables, Katz and Domshlak (2010a) are able to characterize tractable forks and inverted forks. This can be done by arbitrarily abstracting the domain of the root r to $\{0,1\}$ within each Π^{f}_r and by arbitrarily abstracting the domain of the sink r to $\{0,1,\ldots,k\}$ with $\mathcal{O}(1)$ within each Π^{i}_r. From now on, we always assume root variables of any forward fork abstraction $\langle \Pi^{\mathrm{f}}_r, \alpha^{\mathrm{f}}_r\rangle$ to have a binary domain.

3.2.1 Example

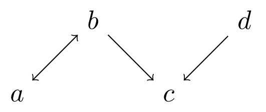
We show how to create implicit forward fork and inverted fork abstractions on a recurring example.

Let $\Pi = \langle V, I, O, G\rangle$ be the planning task of our example, with

- $V = \{a,b,c,d\}$, with $\mathrm{dom}(v) = \{0,1\}$ for all $v \in V$,
- $I = \{a=0, b=0, c=0, d=0\}$,
- $O = \{o_1, o_2\}$, with $o_1 = \langle\{b=0\},\{a=1,b=1\}\rangle$ and $o_2 = \langle\{b=1,d=0\},\{c=1\}\rangle$ with $\mathrm{cost}(o) = 1$ for all $o \in O$,
- $G = \{a=1, c=1\}$.

We can see that the only plan of this task is given by $\langle o_1, o_2\rangle$.

To create the fork abstractions we first look at the causal graph $CG(\Pi)$, given by:


$$\begin{tikzcd}
 & b \arrow[dl, leftrightarrow] \arrow[dr] & & d \arrow[dl] \\
a & & c &
\end{tikzcd}$$

We will now create the two additive implicit abstractions sets containing all forward fork and inverted fork abstractions by decomposing the planning task along its causal graph.

We will first generate the set of all forward fork abstractions $\mathcal{A}_{\mathcal{F}}$. For this, a forward fork subgraph $\mathcal{G}^{\mathrm{f}}_v$ is generated for every variable in the planning task, consisting out of the variable and its successors. The causal graph for each forward fork can be deduced by the outgoing edges in the causal graph $CG(\Pi)$:

[3] Recall that the general cost-partitioning for acyclic-causal graph abstractions, such as these fork abstractions, does not guarantee admissibility with the current construction.

$$\begin{tikzcd}
a \arrow[d] & & b \arrow[dl] \arrow[dr] & & c & d \arrow[d] \\
b & a & & c & & c
\end{tikzcd}$$

We will now show how the abstract planning tasks Π^{f}_r look for each forward fork abstraction. The variables for those tasks are given by the ordered sets $V^{\mathrm{f}}_a = \{a, b\}$, $V^{\mathrm{f}}_b = \{b, a, c\}$, $V^{\mathrm{f}}_c = \{c\}$, and $V^{\mathrm{f}}_a = \{d, c\}$, respectively. The initial states I^{f}_r and goal states G^{f}_r are given by the projection of I and G over the remaining variables in V^{f}_r. For the operator set O^{f}_r, we have to create new unary-effect operators $O^{\mathrm{f}}_r(o^v)$ from the original operators o that affect variables v in Π^{f}_r:

- $O^{\mathrm{f}}_a = \{o^a_1, o^b_1\} = \{\langle\{\,\}, \{a = 1\}\rangle,\ \langle\{a = 1, b = 0\}, \{b = 1\}\rangle\}$
- $O^{\mathrm{f}}_b = \{o^b_1, o^a_1, o^c_2\} = \{\langle\{b = 0\}, \{b = 1\}\rangle,\ \langle\{b = 1\}, \{a = 1\}\rangle,\ \langle\{b = 1\}, \{c = 1\}\rangle\}$
- $O^{\mathrm{f}}_c = \{o^c_2\} = \{\langle\{\,\}, \{c = 1\}\rangle\}$
- $O^{\mathrm{f}}_d = \{o^c_2\} = \{\langle\{d = 0\}, \{c = 1\}\rangle\}$

We will now generate the set of all inverted fork abstractions $\mathcal{A}_{\mathcal{I}}$. For this, an inverted fork subgraph $\mathcal{G}^{\mathrm{i}}_v$ is generated for every variable in the planning task, consisting out of the variable and its predecessors. The causal graph for each inverted fork can be deduced by the incoming edges in the causal graph $CG(\Pi)$:

$$\begin{tikzcd}
b \arrow[d] & & b \arrow[dr] & & d \arrow[dl] & \\
a & b & & c & & d
\end{tikzcd}$$

We will now show how the abstract planning tasks Π^{i}_r look for each inverted fork abstraction. The variables for those tasks are given by the ordered sets $V^{\mathrm{i}}_a = \{b, a\}$, $V^{\mathrm{i}}_b = \{b\}$, $V^{\mathrm{i}}_c = \{b, d, c\}$, and $V^{\mathrm{i}}_a = \{d\}$, respectively. The initial states I^{i}_r and goal states G^{i}_r are given by the projection of I and G over the remaining variables in V^{i}_r. For the operator set O^{i}_r, we have to create new unary-effect operators $O^{\mathrm{i}}_r(o^v)$ from the original operators o that effect variables v in Π^{i}_r:

- $O^{\mathrm{i}}_a = \{o^b_1, o^a_1\} = \{\langle\{b = 0\}, \{b = 1\}\rangle,\ \langle\{b = 1\}, \{a = 1\}\rangle\}$
- $O^{\mathrm{i}}_b = \{o^b_1\} = \{\langle\{b = 0\}, \{b = 1\}\rangle\}$
- $O^{\mathrm{i}}_c = \{o^b_1, o^c_2\} = \{\langle\{b = 0\}, \{b = 1\}\rangle, \langle\{b = 1, d = 0\}, \{c = 1\}\rangle\}$
- $O^{\mathrm{i}}_d = \{\,\} = \{\,\}$

With this, we have seen how to obtain the set of all forward and inverted fork abstractions $\mathcal{A}_{\mathcal{FI}}$. In the next chapter, we will see how to obtain an optimal cost-partitioning for $\mathcal{A}_{\mathcal{FI}}$.

4 Constraints for Implicit Fork Abstractions

In this chapter, we use the optimal operator-cost partitioning linear program formulation from Katz and Domshlak (2010b) for implicit abstractions to derive constraints that can be used within the operator-counting framework.

4.1 Cost-partitioning Constraints

We have seen in Section 2.2.2 and 2.3.1 how to derive operator-counting constraints from the linear programming formulation of operator-cost partitioning for explicit abstractions, by using the *dual* formulation to turn the maximization problem into a minimization problem (Pommerening et al., 2014).

We want to do something similar with the LP-formulation for implicit fork abstractions and turn it from a maximization problem, calculating the optimal cost-partitioning heuristic, to a minimization problem, fitting the operator-counting framework. First, we will see how to obtain the initial primal cost-partitioning LP for forward fork abstractions and for inverted fork abstractions as described by Katz and Domshlak (2010b). Then, we will see how we can combine them by describing the composed LP. And finally, we will derive constraints from those LPs that fit the operator-counting framework.

Definition 22 (Cost-partitioning LP for Forward Forks)**.** Let $\mathcal{A} = \{\langle \Pi^{\mathrm{f}}_i, \alpha^{\mathrm{f}}_i \rangle\}_{i=1}^{m} \subseteq \mathcal{A}_{\mathcal{F}}$ be a set of forward fork abstractions of a planning task Π. The optimal cost-partitioning heuristic $h^{\mathrm{OCP}}_{\mathcal{A}}$ for $\mathcal{A}$ in state $s \in S$ is the objective value of the following LP, where $r \in V$ denotes the root variable in the i-th component and $s_i = \alpha^{\mathrm{f}}_i(s)$.

$$\text{Maximize} \sum_{i=1}^{m} h^{\mathrm{f}}_i(s_i) \text{ subject to } C^{\mathrm{add}}(s) \text{ and } c^{\mathrm{f}}_{i,s} \text{ for all } 1 \le i \le m$$

Where $C^{\mathrm{add}}(s)$ are the *additivity constraints* $\sum_{i=1}^{m} \sum_{o' \in O^{\mathrm{f}}_r(o)} c_i(o') \le \mathit{cost}(o)$ for all $o \in O$, and the *forward fork constraint* $c^{\mathrm{f}}_{i,s}$ consists of the following sets of linear constraints:

1. Cheapest-path constraints for each goal variable $v \in V^{\mathrm{f}}_r \setminus \{r\}$ and each $\theta \in \mathrm{dom}(v)$:

$$p(v, \theta, \theta, 0) = 0, \qquad p(v, \theta, \theta, 1) = 0.$$

Likewise, for each v-changing operator $o^v \in O^{\mathrm{f}}_r[v]$ and each $\theta_r \in \mathrm{Pre}(o^v)[r]$:

$$p(v, \theta, \mathrm{eff}(o^v)[v], \theta_r) \leq c_i(o^v) + \begin{cases} p(v, \theta, \mathrm{pre}(o^v)[v], \theta_r) & \text{if } v \in \mathcal{V}(\mathrm{pre}(o^v)) \\ 0 & \text{otherwise} \end{cases}$$

2. Root-sequence-induced-distance constraints for each goal variable $v \in V^{\mathrm{f}}_r \setminus \{r\}$ and each $\theta \in \mathrm{dom}(v)$:

$$d(v, \theta, 1) \leq p(v, s_i[v], \theta, \sigma(r)[1])$$

And for each $\theta' \in \mathrm{dom}(v)$ and $2 \leq l \leq |\sigma(r)|$:

$$d(v, \theta, l) \leq d(v, \theta', l-1) + p(v, \theta', \theta, \sigma(r)[l])$$

3. Goal constraints for each goal-achieving root sequence σ^* and each pair of r-changing operators $\langle o, o' \rangle$ with $o, o' \in O^{\mathrm{f}}_r[r]$ and $\mathrm{eff}(o)[r] = 1 - s_i[r]$ and $\mathrm{eff}(o')[r] = s_i[r]$:

$$h^{\mathrm{f}}_i(s_i) \leq \left\lceil \frac{|\sigma^*| - 1}{2} \right\rceil \cdot c_i(o) + \left\lfloor \frac{|\sigma^*| - 1}{2} \right\rfloor \cdot c_i(o') + \sum_{v \in V^{\mathrm{f}}_r \setminus \{r\}} d(v, G^{\mathrm{f}}_r[v], |\sigma^*|)$$

We note *goal variables* $v \in V^{\mathrm{f}}_r \setminus \{r\}$ are leaf variables that appear in the goal G^{f}_r. We note that goal-less leaves can usually be omitted from the fork. *Root sequences* are binary sequences that, starting with $s_i[r]$, alternate between binary root values. The longest possible root sequence is $\sigma(r)$ with $|\sigma(r)| = 1 + \max_{v \in V^{\mathrm{f}}_r} |\mathrm{dom}(v)|$. A partial root sequence $\sigma \in \sigma(r)$ is a goal-achieving sequence σ^* if it ends in a root value equal to $G^{\mathrm{f}}_r[r]$.
We further note that each p-variable $p(v, \theta, \theta', \theta_r)$ stands for the cheapest path that changes the value of v from θ to θ', while the root value θ_r is fixed. Each d-variable $d(v, \theta, l)$ stands for the distance between the values $s_i[v]$ and θ of v, given the root value is changed $l - 1$ times. We can think of this as a distance measure that gets access to new fixed-root paths every time we change the root value. For now we assume that all variables ≥ 0.

We note that obtaining an optimal general cost-partitioning through this LP is not possible, as this LP formulation would no longer be bounded if we lifted the restriction of c_i to be non-negative. However, the restriction to non-negative values of the d-variables, the p-variables and the heuristic variables h^{f}_i can be lifted.
To see how Definition 22 works in practice, we will create the cost-partitioning LP for the complete set of forward fork abstractions $\mathcal{A}_{\mathcal{F}} = \{\langle \Pi^{\mathrm{f}}_a, \alpha^{\mathrm{f}}_a \rangle, \langle \Pi^{\mathrm{f}}_b, \alpha^{\mathrm{f}}_b \rangle, \langle \Pi^{\mathrm{f}}_c, \alpha^{\mathrm{f}}_c \rangle, \langle \Pi^{\mathrm{f}}_d, \alpha^{\mathrm{f}}_d \rangle\}$ for the initial state $s^0 = I = \{a = 0, b = 0, c = 0, d = 0\}$ from our recurring example. The objective function is

$$\text{Maximize } h^{\mathrm{f}}_a(s^0_a) + h^{\mathrm{f}}_b(s^0_b) + h^{\mathrm{f}}_c(s^0_c) + h^{\mathrm{f}}_d(s^0_d) \text{ subject to } C^{\mathrm{add}}(s^0) \text{ and } c^{\mathrm{f}}_{v,s^0} \text{ for all } v \in V.$$

The only constraints that use variables from different forward forks are the additivity constraints $C^{\mathrm{add}}(s^o)$ given by:

$$\begin{aligned} c_a(o^a_1) + c_a(o^b_1) + c_b(o^b_1) + c_b(o^a_1) &\leq \mathit{cost}(o_1) = 1 \\ c_b(o^c_2) + c_c(o^c_2) + c_d(o^c_2) &\leq \mathit{cost}(o_2) = 1 \end{aligned}$$

The forward fork constraints c^{f}_{a,s^0} and c^{f}_{c,s^0} are simply given by:

$$h^{\mathrm{f}}_a(s^0_a) \leq c_a(o^a_1) \qquad h^{\mathrm{f}}_c(s^0_c) \leq c_c(o^c_2)$$

The forward fork constraint c^{f}_{b,s^0} is given by:

$$p(a,0,0,0)=0 \quad p(a,0,0,1)=0 \quad p(a,1,1,0)=0 \quad p(a,1,1,1)=0 \quad p(a,0,1,1) \leq c_b(o^a_1)$$

$$p(c,0,0,0)=0 \quad p(c,0,0,1)=0 \quad p(c,1,1,0)=0 \quad p(c,1,1,1)=0 \quad p(c,0,1,1) \leq c_b(o^c_2)$$

$$\begin{aligned}
d(a,0,1) &\leq p(a,0,0,0) & d(a,1,1) &\leq p(a,0,1,0)\\
d(a,0,2) &\leq d(a,0,1)+p(a,0,0,1) & d(a,1,2) &\leq d(a,0,1)+p(a,0,1,1)\\
d(a,0,2) &\leq d(a,1,1)+p(a,1,0,1) & d(a,1,2) &\leq d(a,1,1)+p(a,1,1,1)\\
d(a,0,3) &\leq d(a,0,2)+p(a,0,0,0) & d(a,1,3) &\leq d(a,0,2)+p(a,0,1,0)\\
d(a,0,3) &\leq d(a,1,2)+p(a,1,0,0) & d(a,1,3) &\leq d(a,1,2)+p(a,1,1,0)
\end{aligned}$$

$$\begin{aligned}
d(c,0,1) &\leq p(c,0,0,0) & d(c,1,1) &\leq p(c,0,1,0)\\
d(c,0,2) &\leq d(c,0,1)+p(c,0,0,1) & d(c,1,2) &\leq d(c,0,1)+p(c,0,1,1)\\
d(c,0,2) &\leq d(c,1,1)+p(c,1,0,1) & d(c,1,2) &\leq d(c,1,1)+p(c,1,1,1)\\
d(c,0,3) &\leq d(c,0,2)+p(c,0,0,0) & d(c,1,3) &\leq d(c,0,2)+p(c,0,1,0)\\
d(c,0,3) &\leq d(c,1,2)+p(c,1,0,0) & d(c,1,3) &\leq d(c,1,2)+p(c,1,1,0)
\end{aligned}$$

$$\begin{aligned}
h^{\mathrm{f}}_b(s^0_b) &\leq d(a,1,1)+d(c,1,1)\\
h^{\mathrm{f}}_b(s^0_b) &\leq c_b(o^b_1)+d(a,1,2)+d(c,1,2)\\
h^{\mathrm{f}}_b(s^0_b) &\leq c_b(o^b_1)+d(a,1,3)+d(c,1,3)
\end{aligned}$$

The last forward fork constraint c^{f}_{d,s^0} is given by:

$$p(c,0,0,0)=0 \quad p(c,0,0,1)=0 \quad p(c,1,1,0)=0 \quad p(c,1,1,1)=0 \quad p(c,0,1,0) \leq c_b(o^c_2)$$

$$\begin{aligned}
d(c,0,1) &\leq p(c,0,0,0) & d(c,1,1) &\leq p(c,0,1,0)\\
d(c,0,2) &\leq d(c,0,1)+p(c,0,0,1) & d(c,1,2) &\leq d(c,0,1)+p(c,0,1,1)\\
d(c,0,2) &\leq d(c,1,1)+p(c,1,0,1) & d(c,1,2) &\leq d(c,1,1)+p(c,1,1,1)\\
d(c,0,3) &\leq d(c,0,2)+p(c,0,0,0) & d(c,1,3) &\leq d(c,0,2)+p(c,0,1,0)\\
d(c,0,3) &\leq d(c,1,2)+p(c,1,0,0) & d(c,1,3) &\leq d(c,1,2)+p(c,1,1,0)
\end{aligned}$$

$$\begin{aligned}
h^{\mathrm{f}}_d(s^0_d) &\leq d(c,1,1)\\
h^{\mathrm{f}}_d(s^0_d) &\leq d(c,1,2)\\
h^{\mathrm{f}}_d(s^0_d) &\leq d(c,1,3)
\end{aligned}$$

We note that the p-variables and d-variables are fork dependant and stand for different variables in Π^{f}_b and Π^{f}_d.

Definition 23 (Cost-partitioning LP for Inverted Forks)**.** Let $\mathcal{A} = \{\langle \Pi^{\mathrm{i}}_i, \alpha^{\mathrm{i}}_i \rangle\}_{i=1}^n \subseteq \mathcal{A}_{\mathcal{I}}$ be a set of inverted fork abstractions of a planning task Π. The optimal cost-partitioning heuristic $h^{\mathrm{OCP}}_{\mathcal{A}}$ for $\mathcal{A}$ in state $s \in S$ is the objective value of the following LP, where $r \in V$ denotes the sink variable in the i-th component and $s_i = \alpha^{\mathrm{i}}_i(s)$.

$$\text{Maximize} \sum_{i=1}^{n} h^{\mathrm{i}}_i(s_i) \text{ subject to } C^{\mathrm{add}}(s) \text{ and } c^{\mathrm{i}}_{i,s} \text{ for all } 1 \leq i \leq n$$

Where $C^{\mathrm{add}}(s)$ are the *additivity constraints* $\sum_{i=1}^n \sum_{o' \in O^{\mathrm{i}}_r(o)} c_i(o') \leq \mathit{cost}(o)$ for all $o \in O$, and the *inverted fork constraint* $c^{\mathrm{i}}_{i,s}$ consists of the following sets of linear constraints:

1. Cheapest-path constraints for each $v \in V^{\mathrm{i}}_r \setminus \{r\}$ and each $\theta \in \mathrm{dom}(v)$:

$$d(v, \theta, \theta) = 0$$

Likewise, for each v-changing operator $o^v \in O^{\mathrm{i}}_r[v]$:

$$d(v, \theta, \mathrm{eff}(o^v)[v]) \leq c_i(o^v) + \begin{cases} d(v, \theta, \mathrm{pre}(o^v)[v]) & \text{if } v \in \mathcal{V}(\mathrm{pre}(o^v)) \\ 0 & \text{otherwise} \end{cases}$$

2. Goal constraints for each cycle-free path $\pi^* = \langle o^r_1, \dots, o^r_m \rangle$ from $s_i[r]$ to $G^{\mathrm{i}}_r[r]$ in $DTG(r, \Pi^{\mathrm{i}}_r)$, where $o^r_j \in O^{\mathrm{i}}_r[r]$:

$$h^{\mathrm{i}}_i(s_i) \leq \sum_{j=1}^{m} c_i(o^r_j) + \sum_{j=0}^{m} \sum_{v \in V^{\mathrm{i}}_r \setminus \{r\}} d(v, p_j[v], p_{j+1}[v])$$

Where $p_0[v] = s_i[v]$, $p_{m+1}[v] = G^{\mathrm{i}}_r[v]$ if $G^{\mathrm{i}}_r[v]$ is specified and $p_m[v]$ otherwise, and $p_j[v] = \mathrm{pre}(o_j)[v]$ if $\mathrm{pre}(o_j)[v]$ is specified and p_{j-1} otherwise. The p-function basically describes the value each parent v has to take along each step of the path π, such that the operators of π can be taken and v ends up at its goal value (if defined). Each d-variable $d(v, \theta, \theta')$ stands for the cheapest path that changes the value of v from θ to θ' also known as the distance between the two values. For now we assume that all variables ≥ 0.

We note that obtaining an optimal general cost-partitioning through this LP is not possible, as this LP formulation would no longer be bounded if we lifted the restriction of c_i to be non-negative. However, the restriction to non-negative values of the d-variables and the heuristic variables h^{i}_i can be lifted.
To see how Definition 23 works in practice, we will create the cost-partitioning LP for the complete set of inverted fork abstractions $\mathcal{A}_{\mathcal{I}} = \{\langle \Pi^{\mathrm{i}}_a, \alpha^{\mathrm{i}}_a \rangle, \langle \Pi^{\mathrm{i}}_b, \alpha^{\mathrm{i}}_b \rangle, \langle \Pi^{\mathrm{i}}_c, \alpha^{\mathrm{i}}_c \rangle, \langle \Pi^{\mathrm{i}}_d, \alpha^{\mathrm{i}}_d \rangle\}$ for the initial state $s^0 = I = \{a = 0, b = 0, c = 0, d = 0\}$ from our recurring example. The objective function is

$$\text{Maximize } h^{\mathrm{i}}_a(s^0_a) + h^{\mathrm{i}}_b(s^0_b) + h^{\mathrm{i}}_c(s^0_c) + h^{\mathrm{i}}_d(s^0_d) \text{ subject to } C^{\mathrm{add}}(s^0) \text{ and } c^{\mathrm{i}}_{v,s^0} \text{ for all } v \in V.$$

The only constraints that use variables from different inverted forks are the additivity constraints $C^{\mathrm{add}}(s^o)$ given by:

$$\begin{aligned} c_a(o^b_1) + c_a(o^a_1) + c_b(o^b_1) + c_c(o^b_1) &\leq \mathit{cost}(o_1) = 1 \\ c_c(o^c_2) &\leq \mathit{cost}(o_2) = 1 \end{aligned}$$

The inverted fork constraints c^{i}_{b,s^0} and c^{i}_{d,s^0} are simply given by:

$$h^{\mathrm{i}}_b(s^0_b) \leq 0 \qquad h^{\mathrm{i}}_b(s^0_b) \leq c_b(o^b_1) \qquad\qquad h^{\mathrm{i}}_d(s^0_d) \leq 0$$

The inverted fork constraint c^{i}_{a,s^0} is given by:

$$d(b,0,0) = 0 \qquad d(b,0,1) \leq c_a(o^b_1) + d(b,0,0) \qquad d(b,1,1) = 0$$

$$h^{\mathrm{i}}_a(s^0_a) \leq c_a(o^a_1) + d(b,0,1) + d(b,1,1)$$

The last inverted fork constraint c^{i}_{c,s^0} is given by:

$$\begin{array}{lll} d(b,0,0) = 0 & d(b,0,1) \leq c_c(o^b_1) + d(b,0,0) & d(b,1,1) = 0 \\ d(d,0,0) = 0 & & d(d,1,1) = 0 \end{array}$$

$$h^{\mathrm{i}}_c(s^0_c) \leq c_c(o^c_2) + d(b,0,1) + d(b,1,1) + d(d,0,0) + d(d,0,0)$$

We note that the d-variables are fork dependant and stand for different variables in Π^{i}_a and Π^{i}_c.

Definition 24 (Composed Cost-partitioning LP for Forks)**.** The *composition* of an LP for a set of forward fork abstractions $\{\langle \Pi^{\mathrm{f}}_i, \alpha^{\mathrm{f}}_i \rangle\}^m_{i=1} \subseteq \mathcal{A}_{\mathcal{F}}$ and an LP for a set of inverted fork abstractions $\{\langle \Pi^{\mathrm{i}}_i, \alpha^{\mathrm{i}}_i \rangle\}^n_{i=1} \subseteq \mathcal{A}_{\mathcal{I}}$ is given by the following LP, where in the i-th forward fork component $s_i = \alpha^{\mathrm{f}}_i(s)$ and $r \in V$ denotes the root, and in the i-th inverted fork component $s_i = \alpha^{\mathrm{i}}_i(s)$ and $r \in V$ denotes the sink.

$$\text{Maximize } \sum_{i=1}^{m} h^{\mathrm{f}}_i(s_i) + \sum_{i=1}^{n} h^{\mathrm{i}}_i(s_i) \text{ subject to } C^{\mathrm{add}}(s),$$

$$c^{\mathrm{f}}_{i,s} \text{ for all } 1 \leq i \leq m, \qquad\qquad \text{and } c^{\mathrm{i}}_{i,s} \text{ for all } 1 \leq i \leq n$$

Where $C^{\mathrm{add}}(s)$ are the *additivity constraints* over all fork abstractions

$$\sum_{i=1}^{m} \sum_{o' \in O^{\mathrm{f}}_r(o)} c_i(o') + \sum_{i=1}^{n} \sum_{o'' \in O^{\mathrm{i}}_r(o)} c_i(o'') \leq \mathit{cost}(o) \text{ for all } o \in O,$$

and the forward fork constraint $c^{\mathrm{f}}_{i,s}$ and inverted fork constraint $c^{\mathrm{i}}_{i,s}$ are exactly as defined in Definition 22 and Definition 23.
Let $\mathcal{A} = \{\langle \Pi^{\mathrm{f}}_i, \alpha^{\mathrm{f}}_i \rangle\}^m_{i=1} \cup \{\langle \Pi^{\mathrm{i}}_i, \alpha^{\mathrm{i}}_i \rangle\}^n_{i=1} \subseteq \mathcal{A}_{\mathcal{FI}}$ be a set of forward and inverted fork abstractions of a planning task Π. The *composed* optimal cost-partitioning heuristic $h^{\mathrm{OCP}}_{\mathcal{A}}$ in state $s \in S$ is the objective value of the composition LP of its component.

4.2 Operator-counting Constraints

To obtain the dual of the cost-partitioning LP we basically swap the roles of the variables and constraints, and the objective coefficients and constraint bounds. The sign constraints of the variables are dependant on the sign before the constraint bounds. It is beneficial to look at the dual problem as its own problem and take a closer look at the new constraints and auxiliary variables, used for the operator-counting formulation, to compare them to other known operator-counting constraints.

Definition 25 (Operator-counting LP for Forward Forks)**.** The operator-counting LP for a set of forward fork abstractions $\mathcal{A} = \{\langle \Pi^{\mathrm{f}}_i, \alpha^{\mathrm{f}}_i \rangle\}_{i=1}^m \subseteq \mathcal{A}_{\mathcal{F}}$ of a planning task Π is given by the following LP formulated with operator-counting constraints $c^{\mathrm{f}}_{i,s}$, for each $\langle \Pi^{\mathrm{f}}_i, \alpha^{\mathrm{f}}_i \rangle$, where $r \in V$ denotes the root variable in the i-th component and $s_i = \alpha^{\mathrm{f}}_i(s)$.

$$\text{Minimize} \sum_{o \in O} \mathit{cost}(o) \cdot \mathsf{Y}_o \text{ subject to } c^{\mathrm{f}}_{i,s} \text{ for all } 1 \leq i \leq m$$

Where the *forward fork constraint* $c^{\mathrm{f}}_{i,s}$ makes use of the operator-counting variables, which will be denoted as Y_o, and three different types of auxiliary variables that only exist if the corresponding constraint exists within the primal:

1. Cheapest fixed-root path variables $\mathsf{Y}^i_{\theta_r}(v, \theta, \theta', o)$ that denote how often the cheapest path is taken that changes the variable v from θ to θ' using operator o while the root is at the fixed value θ_r, where $\theta, \theta' \in \mathrm{dom}(v)$, $\theta_r \in \mathrm{dom}(r)$, and $o \in O^{\mathrm{f}}_r[v] \cup \{\square\}$.

   This auxiliary variable only exists if the path is trivial and $\theta = \theta'$ or $\mathrm{eff}(o)[v] = \theta'$ and $\theta_r \in \mathrm{Pre}(o)[v]$,

2. Distance variables induced by a partial root-sequence $\mathsf{Y}^i_l(v, \theta, \theta')$ that denote how often the cheapest path is taken that changes v from θ to θ' after having changed the binary root value $l - 1$ times, where $\theta, \theta' \in \mathrm{dom}(v)$ and $l = |\sigma|$ is the length of a partial root sequence σ.

   This auxiliary variable only exists when the partial root sequence σ changes the root and $2 \leq l \leq |\sigma(r)|$ or if there is no root change ($l = 1$) and the path starts in the current state $s_i[v] = \theta$,

3. Goal-achieving root sequence variables $\mathsf{Y}^i_{\sigma^*_l}(o, o')$ that denote how often a particular goal-achieving root sequence $\sigma^*_l(o, o')$ of length l, using $o, o' \in O^{\mathrm{f}}_r[r]$, is taken.

   This auxiliary variable exists for each goal-achieving root sequence $\sigma^* \in \sigma(r)$ and each pair of operators $\langle o, o' \rangle$ with $o, o' \in O^{\mathrm{f}}_r[r]$ and $\mathrm{eff}(o)[r] = 1 - s_i[r]$ and $\mathrm{eff}(o')[r] = s_i[r]$. We assume these operators to always exist, as we can simply create the non-root-changing operators $o^0 = \langle \{r = 0\}, \{r = 0\}, 0 \rangle$ and $o^1 = \langle \{r = 1\}, \{r = 1\}, 0 \rangle$.

We note that the first two auxiliary variables only exist for goal variables $v \in V^{\mathrm{f}}_r \setminus \{r\}$. *Goal variables* and *root sequences* are defined just like they were in Definition 22. All of the variables with the exception of the trivial path $\mathsf{Y}^i_{\theta_r}(v, \theta, \theta, o)$ are restricted to be non-negative.
The *forward fork constraint* $c^{\mathrm{f}}_{i,s}$ consists of the following sets of linear constraints over these variables:

1. Operator count inequalities for the unary-effect operator $o^r \in O^{\mathrm{f}}_r[r](o)$ and $o^v \in O^{\mathrm{f}}_r[v](o)$ for all $v \in V^{\mathrm{f}}_r$, where $\mathrm{Pre}(o^v)[r]$ is the set of root values for which o^v can

be applied. For each operator $o \in O$:

$$\mathsf{Y}_o \geq \begin{cases} \sum\limits_{\sigma_l^* \in \sigma(r)} \sum\limits_{\substack{o' \in O_r^{\mathrm{f}}[r] \\ \mathrm{eff}(o')[r] = 1 - \mathrm{eff}(o^r)[r]}} \left\lceil \frac{l-1}{2} \right\rceil \cdot \mathsf{Y}^i_{\sigma_l^*}(o^r, o') & \text{if } \mathrm{eff}(o^r)[r] \neq s_i[r] \\ \sum\limits_{\sigma_l^* \in \sigma(r)} \sum\limits_{\substack{o' \in O_r^{\mathrm{f}}[r] \\ \mathrm{eff}(o')[r] = 1 - \mathrm{eff}(o^r)[r]}} \left\lfloor \frac{l-1}{2} \right\rfloor \cdot \mathsf{Y}^i_{\sigma_l^*}(o', o^r) & \text{if } \mathrm{eff}(o^r)[r] = s_i[r] \end{cases},$$

$$\mathsf{Y}_o \geq \sum_{\substack{\theta \in \mathrm{dom}(v) \\ \theta \neq \mathrm{eff}(o^v)[r]}} \sum_{\theta_r \in \mathrm{Pre}(o^v)[r]} \mathsf{Y}^i_{\theta_r}(v, \theta, \mathrm{eff}(o^v)[r], o^v),$$

$$\vdots$$

Semantics: Each original operator o has to be used at least as often as its unary-root-effect operator o^r is used in goal achieving root sequences and its unary-leaf-effect operator o^v is used on v-changing paths for all $o^v \in O_r^{\mathrm{f}}(o) \setminus \{o^r\}$.

2. Cheapest fixed-root path inequalities for all goal variables $v \in V_r^{\mathrm{f}} \setminus \{r\}$, each $\theta, \theta' \in \mathrm{dom}(v)$, and $\theta_r \in \{0, 1\}$. Let $l \geq 1$ if $s_i[v] = \theta$, and $l \geq 2$ otherwise:

   For $\theta = \theta'$, we have:

$$\mathsf{Y}^i_{\theta_r}(v, \theta, \theta, \square) \geq \sum_{\substack{l \leq |\sigma(r)| \\ \sigma(r)[l] = \theta_r}} \mathsf{Y}^i_l(v, \theta, \theta) + \sum_{\substack{o' \in O_r^{\mathrm{f}}[v] \\ \mathrm{pre}(o')[v] = \theta \\ \theta \neq \mathrm{eff}(o')[v] \\ \theta_r \in \mathrm{Pre}(o')[r]}} \mathsf{Y}^i_{\theta_r}(v, \theta, \mathrm{eff}(o')[v], o')$$

   For $\theta \neq \theta'$, we have:

$$\sum_{\substack{o \in O_r^{\mathrm{f}}[v] \\ \mathrm{eff}(o)[v] = \theta' \\ \theta_r \in \mathrm{Pre}(o)[r]}} \mathsf{Y}^i_{\theta_r}(v, \theta, \theta', o) \geq \sum_{\substack{l \leq |\sigma(r)| \\ \sigma(r)[l] = \theta_r}} \mathsf{Y}^i_l(v, \theta, \theta') + \sum_{\substack{o' \in O_r^{\mathrm{f}}[v] \\ \mathrm{pre}(o')[v] = \theta' \\ \theta \neq \mathrm{eff}(o')[v] \\ \theta_r \in \mathrm{Pre}(o')[r]}} \mathsf{Y}^i_{\theta_r}(v, \theta, \mathrm{eff}(o')[v], o')$$

   Semantics: Each cheapest fixed-root path is used as often as it appears in all partial root-sequence paths, where it is usable, and in all other cheapest fixed-root paths it is guaranteed to be a part of, as they continue from its final state.

3. Root-sequence-induced-distance flow inequalities for all goal variables $v \in V_r^{\mathrm{f}} \setminus \{r\}$, each $\theta' \in \mathrm{dom}(v)$, and $1 \leq l \leq |\sigma(r)|$:

   For $l = 1$, we have:

$$\mathsf{Y}^i_1(v, s_i[v], \theta') - \sum_{\theta \in \mathrm{dom}(v)} \mathsf{Y}^i_2(v, \theta', \theta) \geq \begin{cases} \sum_{\sigma_1^*(o,o')} \mathsf{Y}^i_{\sigma_1^*}(o, o') & \text{if } \theta' = G_r^{\mathrm{f}}[v] \\ 0 & \text{otherwise} \end{cases}$$

   For $l \geq 2$, we have:

$$\sum_{\theta \in \mathrm{dom}(v)} \mathsf{Y}^i_l(v, \theta, \theta') - \sum_{\theta'' \in \mathrm{dom}(v)} \mathsf{Y}^i_{l+1}(v, \theta', \theta'') \geq \begin{cases} \sum_{\sigma_l^*(o,o')} \mathsf{Y}^i_{\sigma_l^*}(o, o') & \text{if } \theta' = G_r^{\mathrm{f}}[v] \\ 0 & \text{otherwise} \end{cases}$$

Where $\sum_{\sigma_l^*(o,o')}$ is defined as:

$$\sum_{\substack{\sigma^* \in \sigma(r) \\ |\sigma^*|=l}} \quad \sum_{\substack{o \in O_r^{\mathrm{f}}[r] \\ \mathrm{eff}(o)[r]=1-s_i[r]}} \quad \sum_{\substack{o' \in O_r^{\mathrm{f}}[r] \\ \mathrm{eff}(o')[r]=s_i[r]}}$$

Semantics: For every time variable v is changed into a value θ' it has to be changed out of the value θ' again. Except if θ' is the goal value of the variable, in which case it has to be changed into θ' once more than out of θ'. For $l = 1$ we only consider changes from $s_i[v]$ to θ' as we always start in the initial before the first root value change.

4. A goal inequality:

$$\sum_{\sigma_l^* \in \sigma(r)} \quad \sum_{\substack{o \in O_r^{\mathrm{f}}[r] \\ \mathrm{eff}(o)[r]=1-s_i[r]}} \quad \sum_{\substack{o' \in O_r^{\mathrm{f}}[r] \\ \mathrm{eff}(o')[r]=s_i[r]}} \mathsf{Y}^i_{\sigma_l^*}(o, o') \geq 1$$

Semantics: At least one goal-achieving root sequence is taken.

Let $C^{\mathrm{f}}(s) = \{c^{\mathrm{f}}_{i,s}\}_{i=1}^m$ be the set of forward fork constraints of $\mathcal{A}$. The operator-counting LP-heuristic $h^{\mathrm{LP}}_{C^{\mathrm{f}}}(s)$ is the objective value of this LP.

Recall that the restriction of the operator cost variables c_i to be non-negative is necessary for the primal to be bounded. Therefore the dual would not be feasible if this restriction was lifted. However, the restriction to non-negative values for the p-variables, the d-variables, and the heuristic variables h^{f}_i in the primal cost-partitioning LP can be lifted, in which case the inequalities 2, 3, and 4 respectively would turn into equalities. As we will see in Chapter 5, there is little practical incentive to do so.
We note that the root-sequence-induced flow inequalities provide constraints very similar to the transition flow inequalities from Definition 15. The main difference is that we do not measure the flow of explicit transitions but some flow of root-sequence paths, where the root-sequence length increases when taking a step.

Proposition 1. *Forward fork constraints as defined in Definition 25 are operator-counting constraints.*

Even though the constraints are very intuitive and the proposition seems to be clear, the proof for this proposition appears to be much more difficult than anticipated. Below is an attempt at the proof that showcases some of the general ideas necessary.

Proof. Let $\pi = \langle o_1, \ldots, o_m \rangle$ be an s-plan of length m for state $s \in S$ in the original planning task Π, and let Y_o^π denote the number of occurrences of the operator $o \in O$ in π. We know that $s_i[r]$ is equal to 0 or 1 within any forward fork abstraction. We assume, without loss of generality, that $s_i[r] = 0$. Let $O_r^{\mathrm{f}}[r = 1]$ denote the set of operators o with $\mathrm{eff}(o)[r] = 1$ and $O_r^{\mathrm{f}}[r = 0]$ denote the set of operators o with $\mathrm{eff}(o)[r] = 0$. For any pair of operators $o^1, o^0 \in \pi$ with $o^1 \in O_r^{\mathrm{f}}[r = 1]$ and $o^0 \in O_r^{\mathrm{f}}[r = 0]$ we set $\mathsf{Y}_{\sigma_{l+1}^*}(o^1, o^0) = \frac{l}{n}$, where n is the number of times the root value is changed in π and l is the number of times the pair $\langle o^1, o^0 \rangle$ was responsible for said change. We can get l by counting how often o^0 appears for the first time after o^1 in π and how often o^1 appears for the first time or for the first time after o^0. From this construction we can see that

$$\sum_{\sigma_l^* \in \sigma(r)} \sum_{\substack{o \in O_r^{\mathrm{f}}[r] \\ \mathrm{eff}(o)[r]=1-s_i[r]}} \sum_{\substack{o' \in O_r^{\mathrm{f}}[r] \\ \mathrm{eff}(o')[r]=s_i[r]}} \mathsf{Y}^i_{\sigma_l^*}(o, o') = 1$$

and inequality 4 is satisfied.
For any goal variable $v \in V_r^{\mathrm{f}} \setminus \{r\}$ let o be the operator in π that last changed v and let l be equal to the number of times the pair $\langle o^1, o^0 \rangle$ changed the root value before o last changed v. We set $\mathsf{Y}^i_{l+1}(v, \theta, \mathrm{eff}(o)[v]) = \mathsf{Y}^i_{\sigma^*_{l+1}}(o^1, o^0)$, where θ is equal to the value v had before o^1 was applied for the first time. For different pairs $\langle o^1, o^0 \rangle$, we will get different values for l or θ. We go through all pairs like this and set all the other $\mathsf{Y}^i_l(v, \theta', \theta)$ equal to $\mathsf{Y}^i_{l+1}(v, \theta, \mathrm{eff}(o)[v])$, as they need to be reached first. From this we should get

$$\sum_{\theta' \in \mathrm{dom}(v)} \mathsf{Y}^i_l(v, \theta', \theta) - \sum_{\theta \in \mathrm{dom}(v)} \mathsf{Y}^i_{l+1}(v, \theta, \mathrm{eff}(o)[v]) = \sum_{\sigma_l^*(o,o')} \mathsf{Y}^i_{\sigma^*_{l+1}}(o^1, o^0)$$

and satisfy the inequalities 3.
For any goal variable $v \in V_r^{\mathrm{f}} \setminus \{r\}$ let $\langle o_{j+1} \ldots, o_{j+k} \rangle$ describe a partial sequence of π between two root-changing operators o_j and o_{j+k+1} that does not contain any root-changing operators. For every v-changing operator o within the first partial sequence, before the first root change, we set $\mathsf{Y}^i_0(v, \theta, \theta', o)$ to $\mathsf{Y}^i_1(v, \theta, \mathrm{eff}(o)[v])$ if we use o to change v from initial value $\theta = s_i[v]$ to θ' and it was the last change to v of the partial sequence, and to $\mathsf{Y}^i_0(v, \theta, \mathrm{eff}(o')[v], o')$ otherwise, where o' is the next v-changing operator in the partial sequence. For later partial sequences we set $\mathsf{Y}^i_{\mathrm{eff}(o_j)[r]}(v, \theta, \theta', o)$ to $\mathsf{Y}^i_{l+1}(v, \theta, \mathrm{eff}(o)[v])$ if we use o to change v from initial value $\theta = s_i[v]$ to θ' and it was the last change to v of the partial sequence, and to $\mathsf{Y}^i_{\mathrm{eff}(o_j)[r]}(v, \theta, \mathrm{eff}(o')[v], o')$ otherwise, where o' is the next v-changing operator in the partial sequence. We set all $\mathsf{Y}^i_{\theta_r}(v, \theta, \theta, \square)$ to 0. From this construction it should follow that

$$\sum_{\substack{o \in O_r^{\mathrm{f}}[v] \\ \mathrm{eff}(o)[v]=\theta' \\ \theta_r \in \mathrm{Pre}(o)[r]}} \mathsf{Y}^i_{\theta_r}(v, \theta, \theta', o) \geq \sum_{\substack{l \leq |\sigma(r)| \\ \sigma(r)[l]=\theta_r}} \mathsf{Y}^i_l(v, \theta, \theta') + \sum_{\substack{o' \in O_r^{\mathrm{f}}[v] \\ \mathrm{pre}(o')[v]=\theta' \\ \theta \neq \mathrm{eff}(o')[v] \\ \theta_r \in \mathrm{Pre}(o')[r]}} \mathsf{Y}^i_{\theta_r}(v, \theta, \mathrm{eff}(o')[v], o')$$

and the inequalities 2 are satisfied.
Through our construction (recall inequality 4) the right side of the unary-root effect inequality is given by $\lceil \frac{l-1}{2} \rceil$, as we can assume $\mathrm{eff}(o)[r] = 1$ without loss of generality, which is equal to the amount of times o is used to change the root value. The right side of each unary-effect inequality is simply given by the amount of times o was used in all partial sequences to change v (recall construction for inequality 2) and therefore by how often it is used in π to change v. To satisfy the inequalities 1 we set Y_o to Y_o^π. As the inequalities hold for Y_o^π the inequalities are guaranteed to hold for Y_o. $\square$

The LP formulation in Definition 25 is the dual of the primal cost-partitioning LP from Definition 22 for forward forks and indeed fits the operator-counting framework perfectly.

With this we can create the operator-counting LP for the complete set of forward fork abstractions $\mathcal{A}_{\mathcal{F}} = \{\langle \Pi^{\mathrm{f}}_a, \alpha^{\mathrm{f}}_a \rangle, \langle \Pi^{\mathrm{f}}_b, \alpha^{\mathrm{f}}_b \rangle, \langle \Pi^{\mathrm{f}}_c, \alpha^{\mathrm{f}}_c \rangle, \langle \Pi^{\mathrm{f}}_d, \alpha^{\mathrm{f}}_d \rangle\}$ for the initial state $s^0 = I = \{a = 0, b = 0, c = 0, d = 0\}$ from our recurring example. The objective function is

$$\text{Minimize} \sum_{o \in O} \mathit{cost}(o) \cdot \mathsf{Y}_o \text{ subject to } c^{\mathrm{f}}_{i,s} \text{ for all } 1 \leq i \leq m$$

The forward fork constraint c^{f}_{a,s^0} is given by:

$$\mathsf{Y}_{o_1} \geq \mathsf{Y}^a_{\sigma^*_2}(o_1, o^0) \qquad \mathsf{Y}_{o_1} \geq 0$$

$$\mathsf{Y}^a_{\sigma^*_2}(o_1, o^0) \geq 1$$

The forward fork constraint c^{f}_{b,s^0} is given by:

$$\mathsf{Y}_{o_1} \geq \mathsf{Y}^b_{\sigma^*_2}(o_1, o^0) \qquad \mathsf{Y}_{o_1} \geq \mathsf{Y}^b_{\theta_r=1}(a, 0, 1, o^a_1) \qquad \mathsf{Y}_{o_2} \geq \mathsf{Y}^b_{\theta_r=1}(c, 0, 1, o^c_2)$$

$$\begin{aligned}
\mathsf{Y}^b_{\theta_r=0}(a,0,0,\square) &\geq \mathsf{Y}^b_{l=1}(a,0,0) + \mathsf{Y}^b_{l=3}(a,0,0) & \mathsf{Y}^b_{\theta_r=1}(a,0,0,\square) &\geq \mathsf{Y}^b_{l=2}(a,0,0)\\
0 &\geq \mathsf{Y}^b_{l=1}(a,0,1) + \mathsf{Y}^b_{l=3}(a,0,1) & \mathsf{Y}^b_{\theta_r=1}(a,0,1,o^a_1) &\geq \mathsf{Y}^b_{l=2}(a,0,1)\\
0 &\geq \mathsf{Y}^b_{l=3}(a,1,0) & 0 &\geq \mathsf{Y}^b_{l=2}(a,1,0)\\
\mathsf{Y}^b_{\theta_r=0}(a,1,1,\square) &\geq \mathsf{Y}^b_{l=3}(a,1,1) & \mathsf{Y}^b_{\theta_r=1}(a,1,1,\square) &\geq \mathsf{Y}^b_{l=2}(a,1,1)
\end{aligned}$$

$$\begin{aligned}
\mathsf{Y}^b_{\theta_r=0}(c,0,0,\square) &\geq \mathsf{Y}^b_{l=1}(c,0,0) + \mathsf{Y}^b_{l=3}(c,0,0) & \mathsf{Y}^b_{\theta_r=1}(c,0,0,\square) &\geq \mathsf{Y}^b_{l=2}(c,0,0)\\
0 &\geq \mathsf{Y}^b_{l=1}(c,0,1) + \mathsf{Y}^b_{l=3}(c,0,1) & \mathsf{Y}^b_{\theta_r=1}(c,0,1,o^c_2) &\geq \mathsf{Y}^b_{l=2}(c,0,1)\\
0 &\geq \mathsf{Y}^b_{l=3}(c,1,0) & 0 &\geq \mathsf{Y}^b_{l=2}(c,1,0)\\
\mathsf{Y}^b_{\theta_r=0}(c,1,1,\square) &\geq \mathsf{Y}^b_{l=3}(c,1,1) & \mathsf{Y}^b_{\theta_r=1}(c,1,1,\square) &\geq \mathsf{Y}^b_{l=2}(c,1,1)
\end{aligned}$$

$$\begin{aligned}
\mathsf{Y}^b_{l=1}(a,0,0) - \mathsf{Y}^b_{l=2}(a,0,0) - \mathsf{Y}^b_{l=2}(a,0,1) &\geq 0\\
\mathsf{Y}^b_{l=1}(a,0,1) - \mathsf{Y}^b_{l=2}(a,1,0) - \mathsf{Y}^b_{l=2}(a,1,1) &\geq \mathsf{Y}^b_{\sigma^*_1}(o_1, o^0)\\
\mathsf{Y}^b_{l=2}(a,0,0) + \mathsf{Y}^b_{l=2}(a,1,0) - \mathsf{Y}^b_{l=3}(a,0,0) - \mathsf{Y}^b_{l=3}(a,0,1) &\geq 0\\
\mathsf{Y}^b_{l=2}(a,0,1) + \mathsf{Y}^b_{l=2}(a,1,1) - \mathsf{Y}^b_{l=3}(a,1,0) - \mathsf{Y}^b_{l=3}(a,1,1) &\geq \mathsf{Y}^b_{\sigma^*_2}(o_1, o^0)\\
\mathsf{Y}^b_{l=3}(a,0,0) + \mathsf{Y}^b_{l=3}(a,1,0) &\geq 0\\
\mathsf{Y}^b_{l=3}(a,0,1) + \mathsf{Y}^b_{l=3}(a,1,1) &\geq \mathsf{Y}^b_{\sigma^*_3}(o_1, o^0)
\end{aligned}$$

$$\begin{aligned}
\mathsf{Y}^b_{l=1}(c,0,0) - \mathsf{Y}^b_{l=2}(c,0,0) - \mathsf{Y}^b_{l=2}(c,0,1) &\geq 0\\
\mathsf{Y}^b_{l=1}(c,0,1) - \mathsf{Y}^b_{l=2}(c,1,0) - \mathsf{Y}^b_{l=2}(c,1,1) &\geq \mathsf{Y}^b_{\sigma^*_1}(o_1, o^0)\\
\mathsf{Y}^b_{l=2}(c,0,0) + \mathsf{Y}^b_{l=2}(c,1,0) - \mathsf{Y}^b_{l=3}(c,0,0) - \mathsf{Y}^b_{l=3}(c,0,1) &\geq 0\\
\mathsf{Y}^b_{l=2}(c,0,1) + \mathsf{Y}^b_{l=2}(c,1,1) - \mathsf{Y}^b_{l=3}(c,1,0) - \mathsf{Y}^b_{l=3}(c,1,1) &\geq \mathsf{Y}^b_{\sigma^*_2}(o_1, o^0)\\
\mathsf{Y}^b_{l=3}(c,0,0) + \mathsf{Y}^b_{l=3}(c,1,0) &\geq 0\\
\mathsf{Y}^b_{l=3}(c,0,1) + \mathsf{Y}^b_{l=3}(c,1,1) &\geq \mathsf{Y}^b_{\sigma^*_3}(o_1, o^0)
\end{aligned}$$

$$\mathsf{Y}^b_{\sigma_1^*}(o_1, o^0) + \mathsf{Y}^b_{\sigma_2^*}(o_1, o^0) + \mathsf{Y}^b_{\sigma_3^*}(o_1, o^0) \geq 1$$

The forward fork constraint c^{f}_{c,s^0} is given by:

$$\mathsf{Y}_{o_2} \geq \mathsf{Y}^c_{\sigma_2^*}(o_2, o^0)$$
$$\mathsf{Y}^c_{\sigma_2^*}(o_2, o^0) \geq 1$$

The last forward fork constraint c^{f}_{d,s^0} is given by:

$$\mathsf{Y}_{o_2} \geq \mathsf{Y}^d_{\theta_r=0}(c, 0, 1, o^c_2)$$

$$\begin{aligned}
\mathsf{Y}^d_{\theta_r=0}(c,0,0,\square) &\geq \mathsf{Y}^d_{l=1}(c,0,0) + \mathsf{Y}^d_{l=3}(c,0,0) & \mathsf{Y}^d_{\theta_r=1}(c,0,0,\square) &\geq \mathsf{Y}^d_{l=2}(c,0,0)\\
\mathsf{Y}^d_{\theta_r=0}(c,0,1,o^c_2) &\geq \mathsf{Y}^d_{l=1}(c,0,1) + \mathsf{Y}^d_{l=3}(c,0,1) & 0 &\geq \mathsf{Y}^d_{l=2}(c,0,1)\\
0 &\geq \mathsf{Y}^d_{l=3}(c,1,0) & 0 &\geq \mathsf{Y}^d_{l=2}(c,1,0)\\
\mathsf{Y}^d_{\theta_r=0}(c,1,1,\square) &\geq \mathsf{Y}^d_{l=3}(c,1,1) & \mathsf{Y}^d_{\theta_r=1}(c,1,1,\square) &\geq \mathsf{Y}^d_{l=2}(c,1,1)
\end{aligned}$$

$$\begin{aligned}
\mathsf{Y}^d_{l=1}(c,0,0) - \mathsf{Y}^d_{l=2}(c,0,0) - \mathsf{Y}^d_{l=2}(c,0,1) &\geq 0\\
\mathsf{Y}^d_{l=1}(c,0,1) - \mathsf{Y}^d_{l=2}(c,1,0) - \mathsf{Y}^d_{l=2}(c,1,1) &\geq \mathsf{Y}^d_{\sigma_1^*}(o^1, o^0)\\
\mathsf{Y}^d_{l=2}(c,0,0) + \mathsf{Y}^d_{l=2}(c,1,0) - \mathsf{Y}^d_{l=3}(c,0,0) - \mathsf{Y}^d_{l=3}(c,0,1) &\geq 0\\
\mathsf{Y}^d_{l=2}(c,0,1) + \mathsf{Y}^d_{l=2}(c,1,1) - \mathsf{Y}^d_{l=3}(c,1,0) - \mathsf{Y}^d_{l=3}(c,1,1) &\geq \mathsf{Y}^d_{\sigma_2^*}(o^1, o^0)\\
\mathsf{Y}^d_{l=3}(c,0,0) + \mathsf{Y}^d_{l=3}(c,1,0) &\geq 0\\
\mathsf{Y}^d_{l=3}(c,0,1) + \mathsf{Y}^d_{l=3}(c,1,1) &\geq \mathsf{Y}^d_{\sigma_3^*}(o^1, o^0)
\end{aligned}$$

$$\mathsf{Y}^d_{\sigma_1^*}(o^1, o^0) + \mathsf{Y}^d_{\sigma_2^*}(o^1, o^0) + \mathsf{Y}^d_{\sigma_3^*}(o^1, o^0) \geq 1$$

We note that $o^0 = \langle \{r = 0\}, \{r = 0\}, 0\rangle$ and $o^1 = \langle \{r = 1\}, \{r = 1\}, 0\rangle$.

Definition 26 (Operator-counting LP for Inverted Forks)**.** The operator-counting LP for a set of inverted fork abstractions $\mathcal{A} = \{\langle \Pi^{\mathrm{i}}_i, \alpha^{\mathrm{i}}_i\rangle\}^n_{i=1} \subseteq \mathcal{A}_{\mathcal{I}}$ of a planning task Π is given by the following LP formulated with operator-counting constraints $c^{\mathrm{i}}_{i,s}$, for each $\langle \Pi^{\mathrm{i}}_i, \alpha^{\mathrm{i}}_i\rangle$, where $r \in V$ denotes the sink variable in the i-th component and $s_i = \alpha^{\mathrm{i}}_i(s)$.

$$\text{Minimize} \sum_{o \in O} \mathit{cost}(o) \cdot \mathsf{Y}_o \text{ subject to } c^{\mathrm{i}}_{i,s} \text{ for all } 1 \leq i \leq n$$

Where the *inverted fork constraint* $c^{\mathrm{i}}_{i,s}$ makes use of the operator-counting variables, which will be denoted as Y_o, and two different types of auxiliary variables that only exist if the corresponding constraint exists within the primal:

1. Cheapest path variables $\mathsf{Y}^i(v, \theta, \theta', o)$ that denote how often the cheapest path is taken that changes variable v from θ to θ' using operator o, where $\theta, \theta' \in \mathrm{dom}(v)$ and $o \in O^{\mathrm{i}}_r[v] \cup \{\square\}$.

   This auxiliary variable only exists for parent variables $v \in V^{\mathrm{i}}_r \setminus \{r\}$, and if the path is trivial and $\theta = \theta'$ or $\mathrm{eff}(o)[v] = \theta'$.

2. Goal-achieving cycle-free path variables $\mathsf{Y}^i_{\pi^*_m}$ that denote how often a particular cycle-free path $\pi^* \in \mathcal{P}(r)$ of length m is taken.

   This auxiliary variable exists for each cycle-free path $\pi^*_m = \langle o^r_1, \dots, o^r_m \rangle \in \mathcal{P}(r)$, where $\mathcal{P}(r)$ is the set of all cycle-free paths from $s_i[r]$ to $G^{\mathrm{i}}_r[r]$ in $DTG(r, \Pi^{\mathrm{i}}_r)$ and $o^r_j \in O^{\mathrm{i}}_r[r]$.

All of the variables with the exception of the trivial path $\mathsf{Y}^i(v, \theta, \theta, \Box)$ are restricted to be non-negative.
The *inverted fork constraint* $c^{\mathrm{i}}_{i,s}$ consists of the following sets of linear constraints over these variables:

1. Operator count inequalities for the unary-effect operator $o^v \in O^{\mathrm{i}}_r[v](o)$ for all $v \in V^{\mathrm{i}}_r$ and $o^r \in O^{\mathrm{i}}_r[r](o)$. For each operator $o \in O$:

$$\mathsf{Y}_o \geq \sum_{\substack{\theta \in \mathrm{dom}(v) \\ \theta \neq \mathrm{eff}(o^v)[v]}} \mathsf{Y}^i(v, \theta, \mathrm{eff}(o^v)[v], o^v),$$

$$\vdots$$

$$\mathsf{Y}_o \geq \sum_{\pi^*_m \in \mathcal{P}(r)} Y^{\pi^*_m}_{o^r} \cdot \mathsf{Y}^i_{\pi^*_m},$$

   where $Y^{\pi^*_m}_{o^r}$ denotes the number of occurrences of o^r in π^*_m.

   Semantics: Each original operator o has to be used at least as often than its unary-parent-effect operator o^v is used on v-changing paths for all $o^v \in O^{\mathrm{i}}_r(o) \setminus \{o^r\}$ and at least as often as its unary-sink-effect operator o^r is used in cycle-free paths from $s_i[r]$ to $G^{\mathrm{i}}_r[r]$.

2. Cheapest path inequalities for all parent variables $v \in V^{\mathrm{i}}_r \setminus \{r\}$ and each $\theta, \theta' \in \mathrm{dom}(v)$:

   For $\theta = \theta'$, we have:

$$\mathsf{Y}^i(v, \theta, \theta, \Box) \geq \sum_{\pi^*_m \in \mathcal{P}(r)} \sum_{\substack{0 \leq j \leq m \\ p_j[v] = \theta \\ p_{j+1}[v] = \theta'}} \mathsf{Y}^i_{\pi^*_m} + \sum_{\substack{o' \in O^{\mathrm{i}}_r[v] \\ \mathrm{pre}(o')[v] = \theta \\ \theta \neq \mathrm{eff}(o')[v]}} \mathsf{Y}^i(v, \theta, \mathrm{eff}(o')[v], o')$$

   For $\theta \neq \theta'$, we have:

$$\sum_{\substack{o \in O^{\mathrm{i}}_r[v] \\ \mathrm{eff}(o)[v] = \theta'}} \mathsf{Y}^i(v, \theta, \theta', o) \geq \sum_{\pi^*_m \in \mathcal{P}(r)} \sum_{\substack{0 \leq j \leq m \\ p_j[v] = \theta \\ p_{j+1}[v] = \theta'}} \mathsf{Y}^i_{\pi^*_m} + \sum_{\substack{o' \in O^{\mathrm{i}}_r[v] \\ \mathrm{pre}(o')[v] = \theta' \\ \theta \neq \mathrm{eff}(o')[v]}} \mathsf{Y}^i(v, \theta, \mathrm{eff}(o')[v], o')$$

   Semantics: Each cheapest parent-changing path is used as often as it appears in all cycle-free paths from $s_i[r]$ to $G^{\mathrm{i}}_r[r]$ to change the value of the parent, such that the path is applicable and the goal of the parent is reached afterwards, and in all other cheapest paths it is guaranteed to be a part of, as they continue from its final state.

3. A goal inequality:

$$\sum_{\pi^*_m \in \mathcal{P}(r)} \mathsf{Y}^i_{\pi^*_m} \geq 1$$

   Semantics: At least one cycle-free path from $s_i[r]$ to $G^{\mathrm{i}}_r[r]$ is taken.

We note that we used the same p-function notation as in Definition 23: $p_0[v] = s_i[v]$, $p_{m+1}[v] = G^{\mathrm{i}}_r[v]$ if $G^{\mathrm{i}}_r[v]$ is specified and $p_m[v]$ otherwise, and $p_j[v] = \mathrm{pre}(o_j)[v]$ if $\mathrm{pre}(o_j)[v]$ is specified and p_{j-1} otherwise. It describes the value each parent v has to take along each step of the path π, such that the operators of π can be taken and v ends up at its goal value (if defined).
Let $C^{\mathrm{i}}(s) = \{c^{\mathrm{i}}_{i,s}\}_{i=1}^{n}$ be the set of inverted fork constraints of $\mathcal{A}$. The operator-counting LP-heuristic $h^{\mathrm{LP}}_{C^{\mathrm{i}}}(s)$ is the objective value of this LP.

Recall that the restriction of the operator cost variables c_i to be non-negative is necessary for the primal to be bounded. Therefore the dual would not be feasible if this restriction was lifted. However, the restriction to non-negative values for the d-variables and the heuristic variables h^{i}_i in the primal cost-partitioning LP can be lifted, in which case the inequalities 2 and 3 respectively would turn into equalities.

Proposition 2. *Inverted fork constraints as defined in Definition 26 are operator-counting constraints.*

Proof. Let $\pi = \langle o_1, \ldots, o_l \rangle$ be an s-plan of length l for state $s \in S$ in the original planning task Π, and let Y^{π}_o denote the number of occurrences of the operator $o \in O$ in π. We set $\mathsf{Y}^{i}_{\pi^*_m}$ to 1 if π^*_m is equal to the sink-changing operator sequence of π with omitted cycles and 0 otherwise. This satisfies inequality 3.
For all $v \in V^{\mathrm{i}}_r$ we set $\mathsf{Y}^{i}(v, \theta, \theta', o)$ to 1 if $s_i[v] = \theta$ and o is the last v-changing operator in π with $\mathrm{eff}(o)[v] = \theta'$ and to 0 otherwise. We further set $\mathsf{Y}^{i}(v, \theta, \theta, \Box)$ to 1 if $s_i[v] = \theta$. This satisfies the inequalities 2.
We now set $\mathsf{Y}_o = Y^{\pi}_o$. Because π^*_m is equal to the sink-changing operator sequence of π with omitted cycles we have $Y^{\pi}_o \geq Y^{\pi^*_m}_{o^r}$ and $\sum_{\pi^*_m \in \mathcal{P}(r)} \mathsf{Y}^{i}_{\pi^*_m} = 1$ we have

$$\mathsf{Y}_o \geq \sum_{\pi^*_m \in \mathcal{P}(r)} Y^{\pi^*_m}_{o^r} \cdot \mathsf{Y}^{i}_{\pi^*_m}.$$

For all $v \in V^{\mathrm{i}}_r$ we only have a non-zero variable $\mathsf{Y}^{i}(v, \theta, \theta', o)$ with $\theta \neq \theta'$ and $\mathrm{eff}(o)[v] = \theta'$ if o was the last v-changing operator and therefore part of the plan. This gives us

$$\mathsf{Y}_o \geq \sum_{\substack{\theta \in \mathrm{dom}(v) \\ \theta \neq \mathrm{eff}(o^v)[v]}} \mathsf{Y}^{i}(v, \theta, \mathrm{eff}(o^v)[v], o^v)$$

and therefore all inequalities hold. $\Box$

The LP formulation in Definition 26 is the dual of the primal cost-partitioning LP from Definition 23 for inverted forks and indeed fits the operator-counting framework perfectly.
With this we can create the operator-counting LP for the complete set of inverted fork abstractions $\mathcal{A}_{\mathcal{I}} = \{\langle \Pi^{\mathrm{i}}_a, \alpha^{\mathrm{i}}_a \rangle, \langle \Pi^{\mathrm{i}}_b, \alpha^{\mathrm{i}}_b \rangle, \langle \Pi^{\mathrm{i}}_c, \alpha^{\mathrm{i}}_c \rangle, \langle \Pi^{\mathrm{i}}_d, \alpha^{\mathrm{i}}_d \rangle\}$ for the initial state $s^0 = I = \{a = 0, b = 0, c = 0, d = 0\}$ from our recurring example. The objective function is

$$\text{Minimize} \sum_{o \in O} \mathit{cost}(o) \cdot \mathsf{Y}_o \text{ subject to } c^{\mathrm{i}}_{i,s} \text{ for all } 1 \leq i \leq n$$

The inverted fork constraint c^{i}_{a,s^0} is given by:

$$\mathsf{Y}_{o_1} \geq \mathsf{Y}^{a}(b, 0, 1, o^b_1) \qquad \mathsf{Y}_{o_1} \geq \mathsf{Y}^{a}_{\pi^*_1}$$

$$\mathsf{Y}^a(b,0,0,\square) \geq \mathsf{Y}^a(b,0,1) \qquad \mathsf{Y}^a(b,0,1,o_1^b) \geq \mathsf{Y}^a_{\pi_1^*}$$
$$0 \geq 0 \qquad \mathsf{Y}^a(b,1,1,\square) \geq \mathsf{Y}^a_{\pi_1^*}$$
$$\mathsf{Y}^a_{\pi_1^*} \geq 1$$

The inverted fork constraint c^{i}_{b,s^0} is given by:

$$\mathsf{Y}_{o_1} \geq \mathsf{Y}^b_{\pi_1^*}$$
$$\mathsf{Y}^b_{\pi_0^*} + \mathsf{Y}^b_{\pi_1^*} \geq 1$$

The inverted fork constraint c^{i}_{c,s^0} is given by:

$$\mathsf{Y}_{o_1} \geq \mathsf{Y}^c(b,0,1,o_1^b) \qquad \mathsf{Y}_{o_2} \geq \mathsf{Y}^c_{\pi_1^*}$$

$$\mathsf{Y}^c(b,0,0,\square) \geq \mathsf{Y}^c(b,0,1) \qquad \mathsf{Y}^c(b,0,1,o_1^b) \geq \mathsf{Y}^c_{\pi_1^*}$$
$$0 \geq 0 \qquad \mathsf{Y}^c(b,1,1,\square) \geq \mathsf{Y}^c_{\pi_1^*}$$

$$\mathsf{Y}^c(d,0,0,\square) \geq 2 \cdot \mathsf{Y}^c_{\pi_1^*} \qquad 0 \geq 0$$
$$0 \geq 0 \qquad \mathsf{Y}^c(d,1,1,\square) \geq 0$$

$$\mathsf{Y}^c_{\pi_1^*} \geq 1$$

The last inverted fork constraint c^{i}_{d,s^0} is given by:

$$\mathsf{Y}^d_{\pi_0^*} \geq 1$$

We note that $\mathsf{Y}^i_{\pi_m^*}$ is unique for each length m in our example. Otherwise we would have to add a notation to differentiate between different paths of same length within an inverted fork.

Definition 27 (Composed Operator-counting LP for Forks)**.** The *composition* of an LP for a set of forward fork abstractions $\{\langle \Pi^{\mathrm{f}}_i, \alpha^{\mathrm{f}}_i \rangle\}_{i=1}^m \subseteq \mathcal{A}_{\mathcal{F}}$ and an LP for a set of inverted fork abstractions $\{\langle \Pi^{\mathrm{i}}_i, \alpha^{\mathrm{i}}_i \rangle\}_{i=1}^n \subseteq \mathcal{A}_{\mathcal{I}}$ is given by the following LP, where in the i-th forward fork component $s_i = \alpha^{\mathrm{f}}_i(s)$ and $r \in V$ denotes the root, and in the i-th inverted fork component $s_i = \alpha^{\mathrm{i}}_i(s)$ and $r \in V$ denotes the sink.

$$\text{Minimize} \sum_{o \in O} \mathit{cost}(o) \cdot \mathsf{Y}_o \text{ subject to } c^{\mathrm{f}}_{i,s} \text{ for all } 1 \leq i \leq m \text{ and } c^{\mathrm{i}}_{i,s} \text{ for all } 1 \leq i \leq n$$

Where the forward fork constraint $c^{\mathrm{f}}_{i,s}$ and inverted fork constraint $c^{\mathrm{i}}_{i,s}$ are exactly as defined in Definition 25 and Definition 26.
Let $\mathcal{A} = \{\langle \Pi^{\mathrm{f}}_i, \alpha^{\mathrm{f}}_i \rangle\}_{i=1}^m \cup \{\langle \Pi^{\mathrm{i}}_i, \alpha^{\mathrm{i}}_i \rangle\}_{i=1}^n \subseteq \mathcal{A}_{\mathcal{FI}}$ be a set of forward and inverted fork abstractions of a planning task Π. Let $C^{\mathrm{fi}}(s) = C^{\mathrm{f}}(s) \cup C^{\mathrm{i}}(s)$ be the set of forward fork and inverted fork constraints of $\mathcal{A}$. The *composed* operator-counting LP-heuristic $h^{\mathrm{LP}}_{C^{\mathrm{fi}}}(s)$ is the objective value of the composed LP and dominates $h^{\mathrm{LP}}_{C^{\mathrm{f}}}$ and $h^{\mathrm{LP}}_{C^{\mathrm{i}}}$.

Remark. Because forward fork constraints and inverted fork constraints are both operator-counting constraints, the combined constraint set $C^{\mathrm{fi}}(s) = C^{\mathrm{f}}(s) \cup C^{\mathrm{i}}(s)$ is, by definition, a set of operator-counting constraints.

5 Results

In this chapter, we discuss some of the design choices that were made for the implementation of operator-counting constraints for implicit fork abstractions in Fast Downward (Helmert, 2006). We then show some experimental results that highlight how expensive their computation and how accurate the resulting heuristic is.

5.1 Implementation

We implemented the implicit forward fork constraints derived in Chapter 4 in Fast Downward. We focused on the forward fork constraints as they do not need paths over the domain transition graph of the sink $DTG(r, \Pi)$, like the inverted forks do. We implemented both, the cost-partitioning and the operator-counting constraints, to ensure that the constraints we derived were indeed correct and yielded the same objective value when solving the LP. We implemented the forward fork operator-counting constraints C^{f} directly within the already existing operator-counting framework, such that it could be easily combined with, and tested against, other operator-counting constraints.

Design Choices We mainly followed the description in Chapter 3 to create forward fork abstractions for a planning task Π. We could create forward fork abstractions for any set of variables in V but, as we have not looked in detail at what the characteristics of a good set of fork abstractions is, decided to create them for all variables in V and only omit forward fork abstractions from our set if they are clearly of no use. Therefore, we only excluded fork abstractions that contained no goal variables from our set. We abstracted the root domain to binary values by randomly mapping values ≥ 2 to values in $\{0, 1\}$. The other considerations for the mapping were to alternate between 0 and 1 and to map the first half of the domain to 0 and the second half to 1. To avoid implicit biases, we decided to go with the random mapping. It is likely that the other mappings would perform much better for certain planning tasks. When looking at the operator-counting constraints from Definition 25, it is worth to note that we can only achieve root sequences of length $l \geq 2$ if at least one unary root-effect operator $o \in O^{\mathrm{f}}_r[r]$ exists, with $\mathrm{eff}(o) = 1 - s_i[r]$, and we can only achieve root sequences of length $l \geq 3$ if at least one pair $o, o' \in O^{\mathrm{f}}_r[r]$ exists, with $\mathrm{eff}(o) = 1 - \mathrm{eff}(o')$.

Therefore, we decided to only generate constraints for root sequences of longer length if those operators actually exist. We also implemented a more generalized version of the operator-counting constraints that allow for most of the variables in the primal cost-partitioning view to no longer be restricted to non-negative values, to see the effect that this can have. In the experimental evaluation that follows, we will see that this only increases the computational cost with no benefit to the heuristic and are focusing on the original constraints as described in Definition 25. It is important to note, however, that the constraints of the generalized version are **not** constraints for a general cost-partitioning, as we are still not allowing the costs c_i to take negative values. Their verdict is still open.

5.2 Experimental Evaluation

We tested our implementation of the operator-counting LP heuristic obtained for implicit forward forks in version 22.12 of the Fast Downward planner (Helmert, 2006) with CPLEX 22.1.1 as LP solver. We consider a benchmark set consisting of all 1827 planning tasks without conditional effects from the optimal sequential tracks of the International Planning Competitions 1998–2018. We use Lab (Seipp et al., 2017b) for running our experiments. All experiments are conducted on Intel Xeon Silver 4114 processors running on 2.2 GHz with a time limit of 30 minutes and a memory limit of 3.5 GB. The calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing center at University of Basel.
We compare the operator-counting constraints obtained for implicit forward forks to four other types of operator-counting constraints, already implemented in Fast Downward:

- Delete Relaxation Constraints (Imai and Fukunaga, 2015),
- LM-Cut Landmark Constraints (Bonet, 2013, Pommerening et al., 2014),
- Post-Hoc Optimization Constraints (Pommerening et al., 2013),
- State Equation Constraints (Bonet, 2013, Pommerening et al., 2014, Van Den Briel et al., 2007).

We denote the LP heuristic h_C^{LP} corresponding to the operator-counting LP obtained from those constraints with **Delete Relaxation**, **LM-Cut**, **Post-Hoc**, and **State Equation**, respectively. The operator-counting LP heuristic obtained for implicit forward forks is denoted by **Implicit**. We note that 56 619 out of 173 555 fork abstractions did not contain any goal variables and were therefore not created. We compare these different heuristics by executing an A* search with them on our benchmark set of 1827 planning tasks.

	Success	Out-of-Memory	Out-of-Time
Implicit	281	637	894
Implicit-General	274	650	**888**
Delete Relaxation	577	207	1027
LM-Cut	**909**	**0**	901
Post-Hoc	748	2	1058
State Equation	770	**0**	1041

Table 5.1: Coverage comparison of all 1827 planning tasks. The winner of each category is highlighted in bold. We note that the reason for **Implicit-General** having the lowest out-of-time error is due to it running out of memory for those tasks before running out of time.

We can see the coverage of the different operator-counting heuristics in Table 5.1. **Implicit-General** is the operator-counting heuristic for implicit forward forks, where we lift the restriction of all non-cost variables in the primal view to be positive. We found the heuristic value of these more generalized fork constraints to be the equal to the restricted one. As they seem to be computationally more expensive and run out of memory more often, lifting those restrictions does not seem worth it. The question if a general-cost partitioning, where also the restriction on the cost variables of the primal view is lifted, is possible and useful, remains. We further note that **LM-Cut** performs the best, coverage wise, and **Implicit** the worst.

We are generating constraints exponential to the variables and their domains, which is what makes the computation so expensive. That is why the domain abstraction of the root was crucial in making the forks tractable at all. Another limiting factor for the coverage of implicit fork constraints is the fact that all operator-counting constraints generated are state dependant and have to be generated at every step of the search. For the

5.2.1 Heuristic Accuracy

We will now compare the accuracy of two heuristics, by comparing them directly and with its combined heuristic, which is the operator-counting heuristic that uses the combined constraint sets of the two. To compare the accuracy of heuristics, we look at the *initial h-value*, which is the value obtained for the initial state of a planning task, and the *expansions until last jump*, which is the number of state expansions before the last f-layer of A*, from which on the true goal distance h^* is used. Since the heuristics compared are all admissible, higher h-values are preferred, whereas less expansions until last jump are preferred, as that means that the heuristic estimate manages to match h^* sooner.

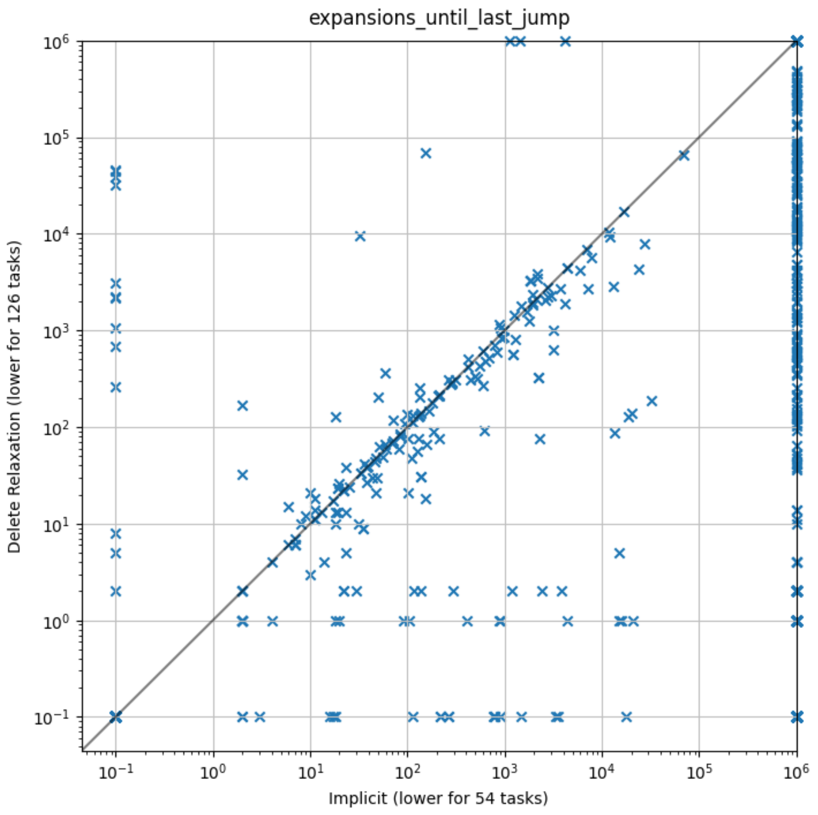


Figure 5.1: Number of expansions before the last f-layer. The tasks in which **Delete Relaxation** needs to expand less states are below the diagonal, the tasks in which **Implicit** expands less are above. Tasks on the right edge were not solved by **Implicit** before running out of memory/time. Tasks lying on the x- and y-axis show tasks where **Delete Relaxation** and **Implicit**, respectively, had the perfect estimate for the initial state.

	Implicit	**Delete Relaxation**	**Combined**
Implicit	–	270	0
Delete Relaxation	**493**	–	0
Combined	**172**	**411**	–

Table 5.2: Comparison of the initial h-value. We compare the row heuristic to the column heuristic and denote in each cell for how many tasks it yields a higher value in the initial state. The winner of each pairwise comparison is highlighted in bold.

Implicit vs Delete Relaxation We can see from Table 5.2 that the combined heuristic dominates its component as it should. Even though the **Delete Relaxation** beats the **Implicit** heuristic in more tasks, the combined heuristic beats **Delete Relaxation** more often than **Implicit**. This is interesting, as this is not the case for any of the other operator-counting constraints and means that **Delete Relaxation** seems to benefit more for its initial estimate from the combination with **Implicit** than the other way around.

We can see in Figure 5.1 that **Delete Relaxation** expands less states than **Implicit** before the last f-layer. While **Delete Relaxation** might give a better estimate than **Implicit** on average, for a few tasks **Implicit** estimates h^* perfectly in the initial state and **Delete**

Relaxation still needs to expand a lot of states. We note that there are even tasks that were solved by **Implicit** but not by **Delete Relaxation**. Because the number of expanded states for different tasks seem to differ the most between the two, as we can see from comparing the spread of the different figures, combining them will be very beneficial.

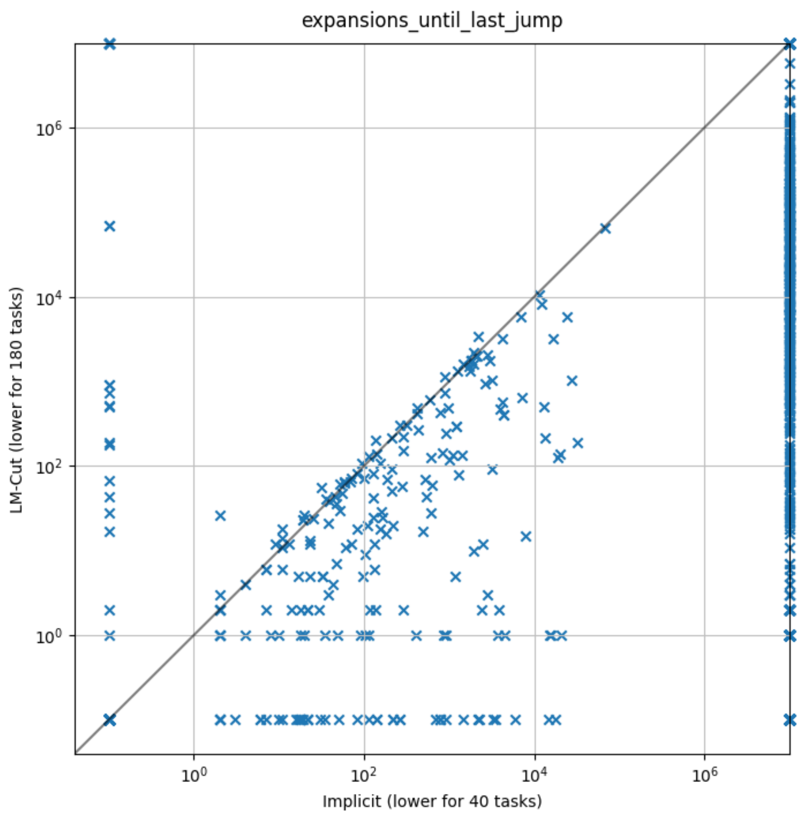


Figure 5.2: Number of expansions before the last f-layer for **LM-Cut** and **Implicit**.

	Implicit	**LM-Cut**	**Combined**
Implicit	–	135	0
LM-Cut	**903**	–	0
Combined	**958**	**208**	–

Table 5.3: Pairwise comparison of the initial h-value.

Implicit vs LM-Cut We can see from Table 5.3 and Figure 5.2 that **LM-Cut** completely outperforms **Implicit** on all fronts. Not only does it derive higher initial h-values but it also expands less states before the last f-layer for more planning tasks. **Implicit** would benefit way more from a combination with **LM-Cut** than the other way around. It is worth mentioning, however, that there are tasks for which **Implicit** estimates h^* perfectly in the initial state and **LM-Cut** does not find a solution.

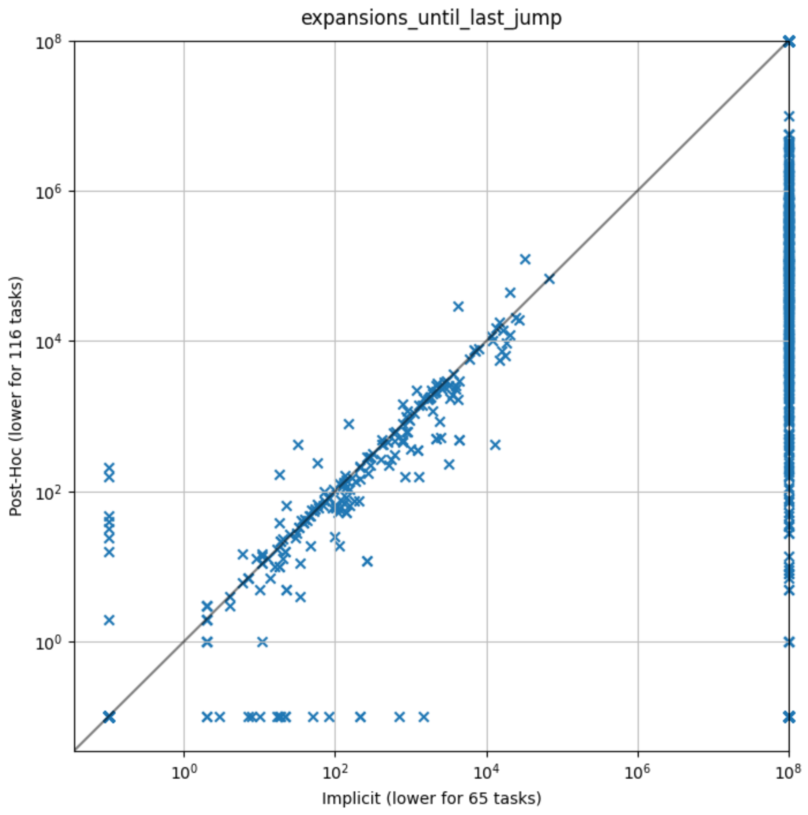


Figure 5.3: Number of expansions before the last f-layer for **Post-Hoc** and **Implicit**.

	Implicit	**Post-Hoc**	**Combined**
Implicit	–	313	0
Post-Hoc	**486**	–	0
Combined	**525**	**484**	–

Table 5.4: Pairwise comparison of the initial h-value.

Implicit vs Post-Hoc Among all the operator-counting constraints we compared, the **Post-Hoc** heuristic accuracy is the closest match to the accuracy of **Implicit**. We can see in Figure 5.3 that they expand a similar amount of states before the last f-layer for most tasks and have fewer outliers between them than between **Implicit** and any of the other operator-counting constraints.

We can see in Table 5.4 that, when comparing the initial h-value, **Impicit** is much more competitive with **Post-Hoc** than with the other operator-counting constraints, as it loses to it the least. Combining the heuristics seems to benefit the initial estimate of **Post-Hoc** almost as often as it benefits **Implicit**.

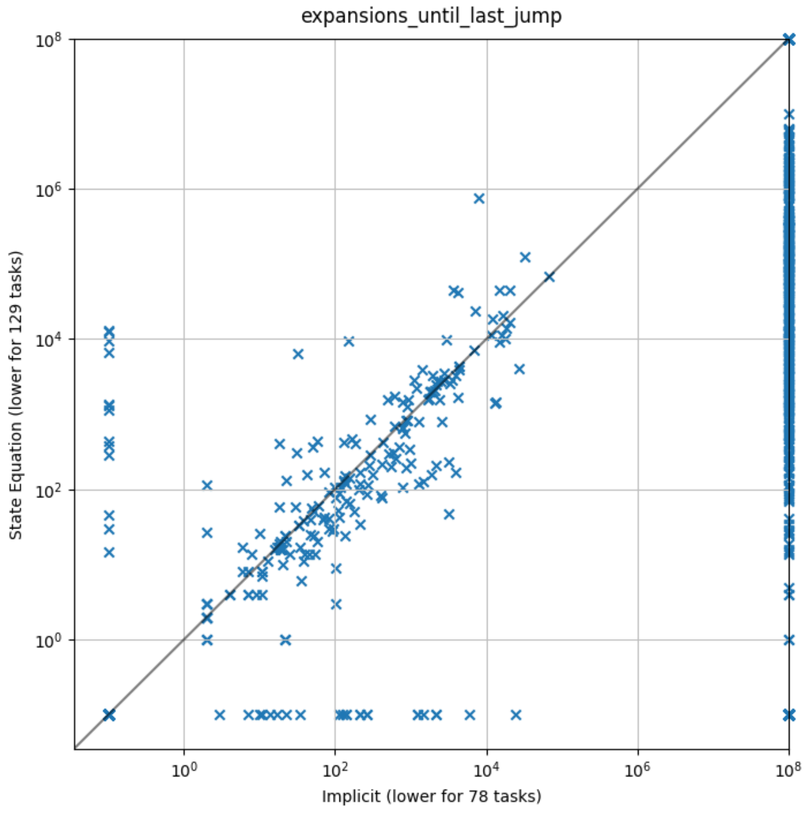


Figure 5.4: Number of expansions before the last f-layer for **State Equation** and **Implicit**.

	Implicit	**State Equation**	**Combined**
Implicit	–	396	0
State Equation	**570**	–	0
Combined	**714**	**619**	–

Table 5.5: Pairwise comparison of the initial h-value.

Implicit vs State Equation **Implicit** beats **State Equation** in more tasks than any other operator-counting heuristic when comparing initial h-values as well as expansions before until last jump.

We can see in Figure 5.4 that for most tasks **State Equation** still expands less states before the last f-layer than **Implicit**. We note, however, that the tasks where **Implicit** expands less states lie not as close to the diagonal as it was the case in other comparisons. That means that on those tasks **Implicit** beats **State Equation** by more than just a small margin. The spread also suggests that combining the heuristics could be very beneficial.

As we can see from Table 5.5, when it comes to the initial h-value, **State Equation** is indeed the heuristic that benefits the most from a combination with **Implicit**, and the initial estimate of **Implicit** benefits even more.

5.2.2 Summary

We have seen that the **Implicit** heuristic is computationally very expensive, especially for problems with many variables and large domains.
The **Implicit** heuristic is beaten by all other operator-counting heuristics when it comes to the h-value in the initial state, with the closest match being the **Post-Hoc** heuristic.
When it comes to the number of expanded states before the last f-layer, **Implicit** seems to be much more competitive with the other operator-counting constraints, only being clearly beaten by **LM-Cut**.
We can see, that the **Implicit** heuristic is not the best at estimating the goal distance for the initial state, even though it made perfect estimates for some tasks where others did not, but improves quite a lot when it gets closer to a goal state.
None of the operator-counting heuristics completely dominates the **Implicit** heuristic, in the initial h-value or the expansions until last jump, although **LM-Cut** outperforms it in almost all ways. This means that combining the operator-counting heuristics with the **Implicit** heuristic is, at least theoretically, always useful. We can see that the combination with the **Implicit** heuristic is especially helpful in reducing the number of expanded states before the last f-layer. The operator-counting heuristic that would benefit the most from a combination with **Implicit** are **Delete Relaxation** (for expansions until last jump) and **State Equation** (for initial h-value). In practice, however, combining operator-counting constraints C with operator-counting constraints for implicit fork abstractions C^{f} increases the computation by too much for the trade-off, of getting a better heuristic estimate, to be worth it.

6 Conclusion

In this thesis, we have created operator-counting constraints for implicit abstractions and investigated them within the operator-counting framework by comparing them to other operator-counting constraints.

We have done this by introducing fork abstractions (Katz and Domshlak, 2008a, 2010a) as concrete instances of implicit abstractions and looking at the linear program formulation of the optimal cost-partitioning for them (Katz and Domshlak, 2008b, 2010b). We then used the cost-partitioning constraints, from the linear program formulation, to derive operator-counting constraints for forward fork and inverted fork abstractions, respectively. To do this, we have made use of the duality of linear programs (LP) to create the duals of the cost-partitioning LPs, and we have shown that those dual LPs are a fit for the operator-counting framework. We have also seen how to compose the constraints for forward fork and inverted fork abstractions. We have seen for the derived operator-counting constraints for implicit abstractions that there are similarities between the path constraints of inverted fork abstractions and forward fork abstractions, as well as similarities between the flow constraints of forward fork abstractions and the flow constraints derived from cost-partitioning constraints of explicit abstractions.

We implemented the operator-counting constraint heuristic for forward fork abstractions in Fast Downward (Helmert, 2006) and compared them to four other types of operator-counting constraints. The experimental evaluation shows that the accuracy of the heuristic is subpar in its initial estimate but seems to improve the closer a search gets to a goal state. Although LM-Cut constraints (Bonet, 2013, Pommerening et al., 2014) seem to beat implicit forward fork constraints on all fronts, it is never fully dominated. Because of this, combining other operator-counting constraints with implicit forward fork constraints can, in theory, considerably improve the heuristic estimate. In practice, however, implicit forward forks have shown to be computationally too expensive to make the accuracy-computation trade-off beneficial.

6.1 Future Work

One of the main questions for future work is how to make the computation more efficient. For the cost-partitioning constraints of implicit fork abstractions it is possible to precompute some of the constraints that are shared between states and store them in some form of database to speed up the computation (Katz and Domshlak, 2010a). Because the operator-counting constraints for fork abstractions are all state dependant and have to be computed individually for every state, we have to look for different ways to make them more applicable in practice.

The other two main factors causing the computation to be so expensive, is the fact that it creates constraints exponential to the number of variables and domain size. Therefore limiting the amount of fork abstractions we use for the computation and their domains can play a substantial role in making them easier to compute. As we do not want to sacrifice heuristic accuracy, it is important to come up with a notion to describe useful forks, that should always be created, and irrelevant forks, that can be omitted without much decrease in accuracy. We have already discussed very basic concepts of irrelevant forks but there are still a lot more insights to be gained on how to best select a set of fork abstractions for a planning task. It would also be interesting to see if there are ways to combine forward and inverted fork abstractions such that a minimal amount of computation is repeated and a maximal increase in accuracy is achieved. Another area where we could decrease the computation cost is by abstracting more than just the root domain or coming up with methods that abstract the root domain of each planning task in an optimal manner.

Future work might also include the search for other types of usable implicit abstractions that are not directly based on fork decompositions.

The last interesting question from this thesis remaining open for future research is how a general cost-partitioning for implicit abstractions would look. We saw that the constrains as they are described in this thesis do not allow for a general cost-partitioning. In theory, however, there is nothing preventing the modification of those constraints in such a way that they allow for general cost-partitioning. One of the requirements for such constraints will likely be that path-costs for non-goal leaf variables for forward forks have to be included. Constraints for general cost-partitioning will therefore likely be even more expensive to compute than the current constraints. If they achieve a favourable accuracy-computation trade-off remains to be seen.

Bibliography

Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

Blai Bonet. An admissible heuristic for sas+ planning obtained from the state equation. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 2268–2274, 2013.

Joseph C Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

Stefan Edelkamp. Planning with pattern databases. In *Proceedings of the Sixth European Conference on Planning (ECP-01)*, pages 13–24, 2001.

Stefan Edelkamp. Automated creation of pattern database search heuristics. pages 35–50, 2006.

Ariel Felner, Richard E Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 22:279–318, 2004.

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

Malte Helmert. A planning heuristic based on causal graph analysis. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pages 161–170, 2004.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, et al. Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 176–183, 2007.

Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, 61(3):1–63, 2014.

István T Hernádvölgyi and Robert C Holte. Experiments with automatically created memory-based heuristics. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 281–290. Springer, 2000.

Robert C Holte, Ariel Felner, Jack Newton, Ram Meshulam, and David Furcy. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence*, 170 (16-17):1123–1136, 2006.

Tatsuya Imai and Alex Fukunaga. On a practical, integer-linear programming model for delete-free tasks and its use as a heuristic for cost-optimal planning. *Journal of Artificial Intelligence Research*, 54:631–677, 2015.

Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *IJCAI*, pages 1728–1733. Pasadena, CA, 2009.

Michael Katz. *Implicit abstraction heuristics for cost-optimal planning.* Research thesis, Senate of the Technion - Israel Institute of Technology, Av, 5770 Haifa, Israel, 2010. URL https://fai.cs.uni-saarland.de/katz/PHD/MichaelKatzPhD.pdf.

Michael Katz and Carmel Domshlak. Structural patterns heuristics via fork decomposition. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, pages 182–189, 2008a.

Michael Katz and Carmel Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, pages 174–181, 2008b.

Michael Katz and Carmel Domshlak. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research*, 39:51–126, 2010a.

Michael Katz and Carmel Domshlak. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12-13):767–798, 2010b.

Florian Pommerening, Gabriele Röger, and Malte Helmert. Getting the most out of pattern databases for classical planning. *International Joint Conference on Artificial Intelligence*, pages 2357–2364, 2013.

Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. Lp-based heuristics for cost-optimal planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, pages 226–234, 2014.

Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 3335–3341, 2015.

Jendrik Seipp and Malte Helmert. Counterexample-guided cartesian abstraction refinement. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, pages 347–351, 2013.

Jendrik Seipp and Malte Helmert. Counterexample-guided cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research*, 62:535–577, 2018.

Jendrik Seipp, Thomas Keller, and Malte Helmert. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3651–3657, 2017a.

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

Menkes Van Den Briel, J Benton, Subbarao Kambhampati, and Thomas Vossen. An lp-based heuristic for optimal planning. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, pages 651–665. Springer, 2007.

Fan Yang, Joseph Culberson, Robert Holte, Uzi Zahavi, and Ariel Felner. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, 32:631–662, 2008.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

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

Author — Autor
Leonhard Badenberg

Matriculation number — Matrikelnummer
2016-055-238

Title of work — Titel der Arbeit
Operator-counting Constraints for Implicit Abstractions

Type of work — Typ der Arbeit
Master 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, July 8, 2023

Badenberg

Signature — Unterschrift