Linear Programming for Heuristics in Optimal Planning

Gabriele Röger and **Florian Pommerening**
University of Basel, Switzerland
{gabriele.roeger, florian.pommerening}@unibas.ch

Abstract

Many recent planning heuristics are based on LP optimization. However, planning experts mostly use LP solvers as a black box and it is often not obvious to them which LP techniques would be most suitable for their specific applications.

To foster the communication between the planning and the optimization community, this paper gives an easily accessible overview over these recent LP-based heuristics, namely the optimal cost partitioning heuristic for abstractions, the post-hoc optimization heuristic, a landmark heuristic, the state-equation heuristic, and a delete relaxation heuristic. All these heuristics fit the framework of so-called operator-counting constraints, which we also present.

Introduction

In recent years, linear programming attracted considerable attention in the planning community to compute heuristics for cost-optimal planning. Researchers in planning usually consider the underlying LP solvers as black boxes but often lack the required knowledge to select a suitable configuration of the solver.

The aim of this paper is to foster the communication between the planning and the optimization community by describing the core of these recent heuristics in a way that is also accessible for readers who are not planning experts.

This paper comprises the following heuristics:

- the *optimal cost partitioning heuristic for abstractions* (Katz and Domshlak 2008; 2010),
- the *post-hoc optimization* heuristic (Pommerening, Röger, and Helmert 2013),
- a heuristic based on *disjunctive action landmarks* (Karpas and Domshlak 2009; Keyder, Richter, and Helmert 2010; Bonet and Helmert 2010),
- the *state equation heuristic* (van den Briel et al. 2007; Bonet 2013), and
- a *delete relaxation* heuristic (Imai and Fukunaga 2014).

We will first give just enough background in planning to enable the reader to understand the following introduction of each of the heuristics. We try to keep the explanations as high-level as possible but still sufficiently precise to give a clear idea of the core of these heuristics and of the character of the underlying linear programs.

The heuristics presented in this paper are not the first applications of linear optimization to planning. While there is no space for an extensive survey of earlier work, we still would like to give a few pointers.

The line of research on LP-based heuristics was started by a heuristic for partial-order planning, based on an LP model with a restricted plan length and explicitly represented time points (Bylander 1997). Another line of research does not derive heuristic estimates by optimization but aims to solve the planning task directly with a suitable IP model. These IP formulations (Vossen et al. 1999) were originally derived from SAT formulations for planning (Kautz and Selman 1996) but were later further refined (van den Briel and Kambhampati 2005; van den Briel, Vossen, and Kambhampati 2008).

Background

Planning Tasks The general aim of classical planning is to find a sequence of operators that transforms the world from a given initial situation (or *state*) into a situation that satisfies a given goal condition. The simplest commonly used formalism to describe such planning tasks is the STRIPS formalism.

A STRIPS planning task is given by a tuple $\Pi = \langle A, O, I, G, \mathit{cost}\rangle$, where

- A is a set of *atoms*,
- O is a set of *operators*,
- $I \subseteq A$ is the *initial state*,
- $G \subseteq A$ is the *goal*, and
- $\mathit{cost} : O \to \mathbb{R}_0^+$ is the *cost function*, which assigns every operator a non-negative cost.

A *state* is given by a subset of the atoms. If a state contains an atom we also say that the atom is *true* in this state, otherwise it is *false* in this state.

Each operator $o \in O$ has a *precondition* $pre(o) \subseteq A$, an *add effect* $add(o) \subseteq A$, and a *delete effect* $del(o) \subseteq A$. An operator o is *applicable* in a state s if all atoms in its precondition are true in s (formally $pre(o) \subseteq s$). Applying a in state s results in a *successor state* s' which is determined

Figure 1: Example logistic task.

by the operator effects: atoms in the add effect become true in s', atoms in the delete effect become false. It is also allowed that an atom occurs in the add and the delete effect of an operator. In this case the add effect wins. All other atoms are unaffected by the operator application. Formally, the successor state is $s' = (s \setminus del(o)) \cup add(o)$.

A state s *satisfies* the goal G if all atoms in G are true in s, formally $G \subseteq s$. Such a state is also called a *goal state*.

For state s, an *s-plan* is a sequence $\pi = \langle o_1, \ldots, o_n \rangle$ of operators that are subsequently applicable starting from s, and whose application leads to a goal state. The *cost of the plan* is the sum of the individual operator costs: $cost(\pi) = \sum_{1 \leq i \leq n} cost(o_i)$. A *plan* for the task is a plan for the initial state (an I-plan). The aim of *optimal planning* is to find a plan of minimum cost.

From a different perspective, a STRIPS planning task is just a concise description of the underlying *transition system* or *state space*. In general a transition system is a tuple $\langle S, s_i, G, L, T, cost \rangle$, where

- S is a finite set of *states*,
- $s_i \in S$ is the *initial state*,
- $G \subseteq S$ is the set of *goal states*,
- L is a finite set of *labels*,
- T is a set of labeled *transitions* $\langle s, l, s' \rangle$ with $s, s' \in S$ and $l \in L$, and
- $cost : L \to \mathbb{R}^+$ is the cost function.

A *plan* for a transitions system is a sequence of transitions that leads from the initial state to a goal state. Its cost is the sum of the label costs.

A STRIPS planning task $\Pi = \langle A, O, I, G, cost \rangle$ *induces* a transition system $\langle S, s_i, G', L, T, cost' \rangle$ in the intuitive way: The set of states S consists of all possible states of the task, which correspond to all possible subsets of A: $S = \mathcal{P}(A)$. The initial state is the same in both perspectives. The set of goal states of the transition system is the set of goal states of the planning task: $G' = \{s \in S \mid G \subseteq s\}$. The labels correspond to the operators ($L = O$) and both perspectives use the same cost function ($cost' = cost$). There is a labeled transition $\langle s, o, s' \rangle \in T$ iff o is applicable in s and the successor state is s'. With this definition, a plan for the task corresponds to a plan for the induced transition system and vice versa.

Running Example Throughout the paper we use the planning task in Figure 1 as a running example where a truck needs to transport a package from location B to location A and then move back to location B again. Initially the truck is at location A. There are six different operators: *load-A* and *load-B* load the package into the truck at location A and B, respectively, *unload-A* and *unload-B* are the inverse operators, and *drive-A-B* and *drive-B-A* move the truck from A to B and vice versa.

We can formulate this task with the set of atoms $\{$*package-at-A*, *package-at-B*, *package-in-truck*, *truck-at-A*, *truck-at-B*$\}$. The initial state is $\{$*truck-at-A*, *package-at-B*$\}$ and the goal is $\{$*truck-at-B*, *package-at-A*$\}$.

Consider *load-B* as an example for an operator. To load the package at location B the package and the truck must be there: $pre($*load-B*$) = \{$*truck-at-B*, *package-at-B*$\}$. Loading the package has the effect that the package is no longer at location B but in the truck: $del($*load-B*$) = \{$*package-at-B*$\}$ and $add($*load-B*$) = \{$*package-in-truck*$\}$. This operator can obviously be applied in state $\{$*truck-at-B*, *package-at-B*$\}$ resulting in successor state $\{$*truck-at-B*, *package-in-truck*$\}$. Assuming a suitable formulation of the other operators, a plan for this task would be $\langle$*drive-A-B*, *load-B*, *drive-B-A*, *unload-A*, *drive-A-B*$\rangle$. If each drive operator costs 5 and all other operators cost 1, then the cost of this plan is 17.

We will discuss the state space induced by this example in the next section.

Heuristics The most common approach for optimal planning is heuristic search with the A* algorithm (Hart, Nilsson, and Raphael 1968). The search algorithm requires a *heuristic* that estimates for a given state s the cost of an s-plan. A* guarantees optimality of the solution if the heuristic is *admissible*, i. e. the estimate is a lower bound for the cost of every s-plan. Informally, an admissible heuristic never overestimates the goal distance.

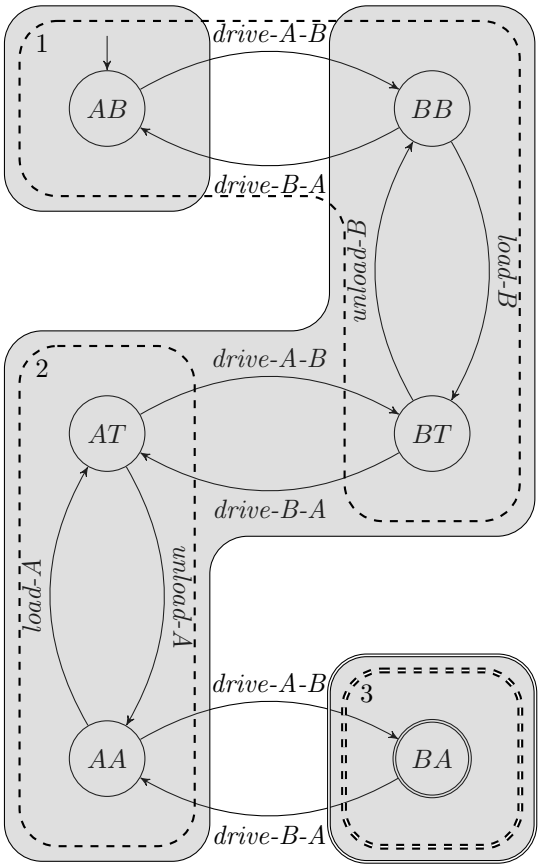
We say that an admissible heuristic h *dominates* a heuristic h' if for all states s, it holds that $h(s) \geq h'(s)$. Higher heuristic estimates allow A* to prune more nodes from its search space.

In the following, we will introduce several admissible heuristics. Depending on what is more intuitive, we will switch between the STRIPS task and the transition system perspective.

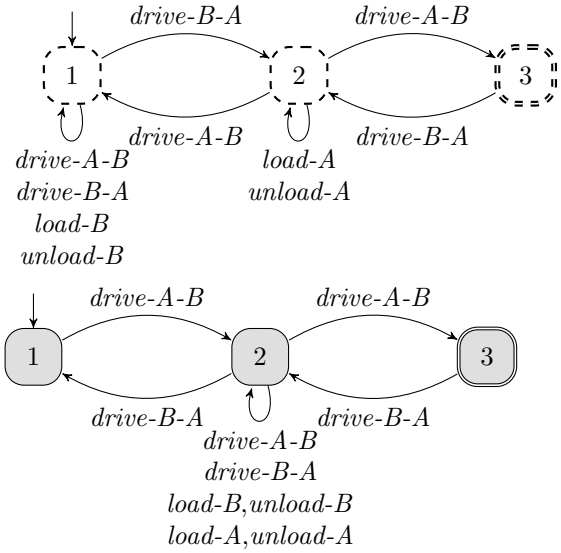
Optimal Cost Partitioning for Abstractions

An *abstraction heuristic* is computed from a simplified version of the original transition system. It is based on an *abstraction function* α that maps the original states to states of the abstract transition system. Typically, we require that every path from s to s' in the original space induces an equally labeled path in the abstract space from $\alpha(s)$ to $\alpha(s')$ (i. e., the abstraction function is a homomorphism) and that the induced path has less or equal cost. If in addition all goal states are mapped to abstract goal states this ensures that the optimal abstract plan cost for $\alpha(s)$ is an admissible heuristic estimate for the original state s.

Figure 2a shows the part of the state space of our example task that is reachable from the initial state. The figure shows two abstraction functions (shaded and dashed regions) that map all states that are in the same region to the same abstract state. The corresponding abstract state spaces are shown in Figure 2b. It is easy to check that all labeled paths in the original state space have a corresponding path in the abstract



(a) Reachable state space. The state labeled XY is the state where the truck is at position X and the package at Y. The shaded and dashed areas show two abstraction functions.



(b) Abstract state spaces induced by the two abstraction functions.

Figure 2: Abstractions for our example task.

spaces. The cheapest paths in the abstract state spaces are $1 \xrightarrow{\textit{drive-B-A}} 2 \xrightarrow{\textit{drive-A-B}} 3$ and $1 \xrightarrow{\textit{drive-A-B}} 2 \xrightarrow{\textit{drive-A-B}} 3$, respectively. Since drive operators have a cost of 5, the heuristic estimate from each abstraction is 10.

Different abstraction heuristics can be combined into a dominating estimate by taking their maximum. In our example, this is not very informative, as both heuristic estimates are 10. Taking the sum is not admissible in general. This can be seen in the example, where the sum of both estimates is 20 while the optimal cost is 17. In this case, the cost of operator *drive-A-B* contributes too much to the total cost. However, if we suitably adapt the cost functions of the abstract systems we can add up the estimates in a way that dominates the maximum and preserves admissibility. A sufficient condition for this is that the sum of the cost we assign to an operator in each abstract system does not exceed its original cost. In our example, we can set the cost operator *drive-A-B* to 0 in the first and to 5 in the second abstraction. If we also set the cost of *drive-B-A* to 5 and 0 respectively, the heuristic values of the abstractions are 5 and 10. We can now add up these estimates for a total of 15. If heuristic estimates can be added like this without losing admissibility, we call the heuristics *additive*.

There are infinitely many ways to distribute the costs between different abstractions. The *optimal cost partitioning heuristic* (Katz and Domshlak 2008; 2010) optimizes the cost distribution to get the best admissible estimate for a state. Let $\mathcal{A}$ be the set of abstractions over which we want to distribute the costs. Each abstraction $\alpha \in \mathcal{A}$ has its own transition system $\mathcal{T}^\alpha$ with the abstract states S^α, the abstract goal states G^α and the abstract transitions T^α. We are mostly interested in the subset of transitions that are not self-loops which we call *state-changing* transitions $\mathit{SCT}^\alpha \subseteq T^\alpha$. The estimate of the optimal cost partitioning heuristic for the set of abstractions $\mathcal{A}$ in state s is the objective value of the following LP or ∞ if the LP is not bounded feasible:

Linear Program 1 (Optimal cost partitioning)
Optimization variables:

- C_o^α for $\alpha \in \mathcal{A}, o \in O$,
  describing the cost of operator o in $\mathcal{T}^\alpha$,
- $\mathsf{D}_{s'}^\alpha$ for $\alpha \in \mathcal{A}, s' \in S^\alpha$,
  measuring the cheapest cost to reach s' from $\alpha(s)$ in $\mathcal{T}^\alpha$ under the cost partitioning given by the variables C_o^α, and
- H^α for $\alpha \in \mathcal{A}$
  for the heuristic estimate for abstraction α.

Maximize: $\sum_{\alpha \in \mathcal{A}} \mathsf{H}^\alpha$
Subject to:

$$\mathsf{D}_{s'}^\alpha = 0 \qquad \text{for all } \alpha \in \mathcal{A} \text{ and } s' = \alpha(s)$$

$$\mathsf{D}_{s''}^\alpha \leq \mathsf{D}_{s'}^\alpha + \mathsf{C}_o^\alpha \qquad \text{for all } \alpha \in \mathcal{A} \text{ and } \langle s', o, s'' \rangle \in \mathit{SCT}^\alpha$$

$$\mathsf{H}^\alpha \leq \mathsf{D}_{s'}^\alpha \qquad \text{for all } \alpha \in \mathcal{A} \text{ and } s' \in G^\alpha$$

$$\sum_{\alpha \in \mathcal{A}} \mathsf{C}_o^\alpha \leq \mathit{cost}(o) \qquad \text{for all } o \in O$$

$$0 \leq \mathsf{C}_o^\alpha, \mathsf{D}_{s'}^\alpha, \mathsf{H}^\alpha \text{ for all } o \in O, \alpha \in \mathcal{A} \text{ and } s' \in S^\alpha$$

Pommerening et al. (2015) have recently shown that it is not necessary to restrict cost partitioning to *non-negative* op-

erator costs and that it is beneficial to allow arbitrary values for C_o^α.

The main disadvantage of optimal cost partitioning is that for each heuristic computation, the LP has to compute the cheapest plan in all abstract transition systems while in parallel adapting the cost functions. This is expensive because the cost function is optimized for every state. The usual alternative is to fix the cost function for the abstractions in advance for the duration of the search. With a fixed cost function, the goal distance of every abstract state can be precomputed with one backwards traversal of the abstract space and determined with a cheap hash table look-up during the search. While optimal cost partitioning leads to better heuristic estimates than provided by precomputed heuristics, the reduction of the size of the search space does not usually outweigh the overhead in the heuristic computation.

Post-hoc Optimization of Heuristic Values

Post-hoc optimization is inspired by a domain-specific observation by Felner, Korf, and Hanan (2004) for the sliding tile puzzle. They considered a large set of heuristics, each reporting a lower bound on the movements required for a small subset of all tiles. They noticed that they can derive additional information from the interactions of these bounds, using a domain-specific NP-hard optimization method. Post-hoc optimization exploits a generalization of these interactions with polynomial-time linear optimization.

Consider using a fixed cost function for the abstraction heuristics introduced in Figure 2b. The heuristic estimate corresponds to a cheapest plan in the abstract system, and there always is a cheapest plan without self-loops because all operator costs are non-negative. Therefore, an operator that only occurs as a label on self-loops (e.g. *load-B*) cannot contribute to the heuristic estimate. Thus, only the operators incurring true abstract state transitions are relevant for this heuristic.

This notion of relevance can be extended to arbitrary heuristics: an operator is *irrelevant* for a heuristic estimate if the same estimate is also admissible for the task that only differs from the original one in that this operator is free of cost. Otherwise the operator is *relevant*.

If we know for several admissible heuristics that already a subset of the operators is sufficient to justify the heuristic estimate, we can perform further reasoning about the true optimal plan costs.

As an example consider a task with three operators o_A, o_B and o_C. Assume we have three admissible heuristics h^{AB}, h^{AC} and h^{BC} for which only the operators indicated in the superscripts are relevant. For state s, all heuristics report a heuristic estimate of 2. From $h^{AB}(s) = 2$, we can conclude that the cost incurred by operators o_A and o_B in any s-plan is at least 2. Let Y_X denote the number of occurrences of o_X ($X \in \{A, B, C\}$) in an arbitrary s-plan. Then this statement can be expressed by the inequality $\mathit{cost}(o_A)\mathsf{Y}_A + \mathit{cost}(o_B)\mathsf{Y}_B \geq 2$. We can derive analogous constraints from the heuristics h^{AC} and h^{BC}. These constraints are satisfied by *any* s-plan. To get a lower bound on the *optimal* plan cost, we need to minimize the overall plan cost, considering all operators. Since the number of occurrences of each operator in an s-plan must be integral, this gives rise to the following integer program:

Minimize: $\mathit{cost}(o_A)\mathsf{Y}_A + \mathit{cost}(o_B)\mathsf{Y}_B + \mathit{cost}(o_C)\mathsf{Y}_C$
Subject to:

$$\begin{aligned} h^{AB} = 2 &\leq \mathit{cost}(o_A)\mathsf{Y}_A + \mathit{cost}(o_B)\mathsf{Y}_B \\ h^{AC} = 2 &\leq \mathit{cost}(o_A)\mathsf{Y}_A + \mathit{cost}(o_C)\mathsf{Y}_C \\ h^{BC} = 2 &\leq \mathit{cost}(o_B)\mathsf{Y}_B + \mathit{cost}(o_C)\mathsf{Y}_C \end{aligned}$$

By summing up the three constraints we can easily see that the objective value is at least 3, which dominates the best individual heuristic estimate of 2.

This general approach is called *post-hoc optimization* (Pommerening, Röger, and Helmert 2013). It only cares about *which* operator costs are relevant for the heuristic estimate, not *how* they influence the heuristic estimate.

For a task with operator set O and a set H of admissible heuristics, the estimate of the *post-hoc optimization heuristic* for a state s is the objective value of the linear program

Linear Program 2 (Post-hoc optimization)
Optimization variables:

- Y_o for $o \in O$ for the number of occurrences of o in a plan

Minimize: $\sum_{o \in O} \mathit{cost}(o)\mathsf{Y}_o$
Subject to:

$$\begin{aligned} h(s) &\leq \sum_{\substack{o \in O \\ o \text{ is relevant for } h \text{ in } s}} \mathit{cost}(o)\mathsf{Y}_o && \text{for all } h \in H \\ 0 &\leq \mathsf{Y}_o && \text{for all } o \in O \end{aligned}$$

Note that also the objective value of the corresponding IP would be an admissible estimate but we usually use the weaker LP because it allows one to compute the estimate in polynomial time. So far, post-hoc optimization has only been used with relevance information that was independent of the evaluated state. This has the advantage that the matrix of the LP is stable over the individual heuristic evaluations.

In comparison to the optimal cost partitioning in the previous section it can be shown that the post-hoc optimization also computes a state-specific cost partitioning, albeit under additional restrictions: it cannot change individual operator costs but can only scale all operator costs within each heuristic by a factor that depends on the heuristic but not on the operator (Pommerening, Röger, and Helmert 2013).

Disjunctive Action Landmarks

A *disjunctive action landmark* (Zhu and Givan 2003; Helmert and Domshlak 2009) for a state s is a set of operators of which at least one must be part of any s-plan.

For an example of how we can exploit such landmarks, consider a task with three operators o_1, o_2, o_3 with $\mathit{cost}(o_1) = \mathit{cost}(o_3) = 3$ and $\mathit{cost}(o_2) = 5$. Let $\{o_1, o_2\}$ and $\{o_2, o_3\}$ be disjunctive action landmarks for state s. From each individual landmark we can determine that every s-plan must cost at least 3 (the cost of the cheapest operator in the landmark). However, we can also use variables Y_i for the number of occurrences of operator o_i in an arbitrary s-plan,

requiring that $\mathsf{Y}_i \geq 0$. From the given landmarks we can then derive the constraints $\mathsf{Y}_1 + \mathsf{Y}_2 \geq 1$ and $\mathsf{Y}_2 + \mathsf{Y}_3 \geq 1$, respectively. Since all these constraints are satisfied by every s-plan, we can determine an admissible estimate by minimizing the total plan cost $3\mathsf{Y}_1 + 5\mathsf{Y}_2 + 3\mathsf{Y}_3$ subject to these constraints and establish a better heuristic estimate of 5.

For the general case, consider an arbitrary task with operator set O. Let $\mathcal{L}$ be a set of disjunctive action landmarks for state s. Then the estimate of the optimal landmark heuristic for state s is the objective value of the LP

Linear Program 3 (Disjunctive action landmarks)
Optimization variables:

- Y_o for $o \in O$ for the number of occurrences of o in a plan

Minimize: $\sum_{o \in O} \mathit{cost}(o)\mathsf{Y}_o$
Subject to:

$$\sum_{o \in L} \mathsf{Y}_o \geq 1 \qquad \text{for all } L \in \mathcal{L}$$
$$\mathsf{Y}_o \geq 0 \qquad \text{for all } o \in O$$

Using linear programming to derive heuristic estimates from landmarks was introduced by Karpas and Domshlak (2009) as cost partitioning for landmarks. Their LP formulation was improved by Keyder, Richter, and Helmert (2010). The formulation presented here is due to Bonet and Helmert (2010) and corresponds to the dual of the representation by Keyder et al.

State Equation Heuristic

For the next heuristic, consider our running example task from Figure 1 again. It is clear that we only can unload the package from the truck if it is in the truck, so it must previously have been loaded. Similarly, we cannot unload the package more often than we load it. Again, using variables Y_o to denote the number of occurrences of o in an arbitrary plan, we can express the latter of these statements with a constraint $\mathsf{Y}_{\textit{load-A}} + \mathsf{Y}_{\textit{load-B}} - \mathsf{Y}_{\textit{unload-A}} - \mathsf{Y}_{\textit{unload-B}} \geq 0$.

Such a constraint can be defined for each atom of the task. Consider atom *package-in-truck*. We say that operators *load-A* and *load-B produce* this atom because previous to the operator application it is false and afterwards it is true. Analogously, operators *unload-A* and *unload-B consume* this atom. We call the difference of the number of producers and consumers in a plan the *net change* of the atom. The previous expression $\mathsf{Y}_{\textit{load-A}} + \mathsf{Y}_{\textit{load-B}} - \mathsf{Y}_{\textit{unload-A}} - \mathsf{Y}_{\textit{unload-B}}$ is the net change of atom *package-in-truck*. Since in each plan, we cannot consume the atom more often than we produce it, the net change cannot be negative.

Now consider atom *package-at-A*. We know that in the goal this atom must be true and in the initial state it is not. Therefore, the corresponding net change $\mathsf{Y}_{\textit{unload-A}} - \mathsf{Y}_{\textit{load-A}}$ must be at least 1.[1] For atom *package-at-B* we have the opposite situation: It is initially true but in the goal it may be false. Therefore, a plan may consume it once more than it produces the atom: $\mathsf{Y}_{\textit{unload-B}} - \mathsf{Y}_{\textit{load-B}} \geq -1$.

[1] In this case, we could also fix the net change to exactly 1 but as we will see later this is not possible in general and does not add new information to the heuristic.

If we collect these *net change constraints* for the entire task, we can again "reason" about all possible plans of the task and determine a lower bound on the optimal plan cost by minimizing the corresponding sum (where O denotes the set of all six operators of the task):

Minimize: $\sum_{o \in O} \mathit{cost}(o)\mathsf{Y}_o$
Subject to:

$$\begin{aligned}
\textit{truck-at-A}: &\quad \mathsf{Y}_{\textit{drive-B-A}} - \mathsf{Y}_{\textit{drive-A-B}} \geq -1 \\
\textit{truck-at-B}: &\quad \mathsf{Y}_{\textit{drive-A-B}} - \mathsf{Y}_{\textit{drive-B-A}} \geq 1 \\
\textit{package-at-A}: &\quad \mathsf{Y}_{\textit{unload-A}} - \mathsf{Y}_{\textit{load-A}} \geq 1 \\
\textit{package-at-B}: &\quad \mathsf{Y}_{\textit{unload-B}} - \mathsf{Y}_{\textit{load-B}} \geq -1 \\
\textit{package-in-truck}: &\quad \mathsf{Y}_{\textit{load-A}} + \mathsf{Y}_{\textit{load-B}} \\
&\quad - \mathsf{Y}_{\textit{unload-A}} - \mathsf{Y}_{\textit{unload-B}} \geq 0 \\
\text{for all } o \in O: &\quad \mathsf{Y}_o \geq 0
\end{aligned}$$

The cheapest solution is to set the variables $\mathsf{Y}_{\textit{load-B}}$, $\mathsf{Y}_{\textit{unload-A}}$ and $\mathsf{Y}_{\textit{drive-A-B}}$ to 1 and all other variables to 0. The resulting heuristic estimate is 7 which in this task significantly underestimates the optimal plan cost of 17. The reason is that the LP is not aware of the fact that the truck has to move to drop off the package. The atom *truck-at-A* is a precondition of the operator *unload-A* but is not consumed by it, so $\mathsf{Y}_{\textit{unload-A}}$ does not occur in the constraint for *truck-at-A*.

The general heuristic that is based on this kind of reasoning is known by different names, inspired by different perspectives: van den Briel et al. (2007) call it an *order-relaxation* heuristic because it only considers *how often* operators are applied in a plan ignoring the order of the applications. Bonet (2013) calls it the *state equation heuristic* because he motivates it from the state equation associated to the Petri-net representation of the planning task. The Petri-net perspective also gave rise to the name *flow heuristic*.

The main obstacle in generalizing the approach is that we must determine whether an operator actually produces (or consumes) an atom. Planning operators can "add" an atom to the state in which they are applied even though this atom is already true in the state. In this case, the operator would not produce the atom and should not be counted as a producer. Likewise, operators that delete atoms that are already false should not be counted as consumers in this state. Since the constraints should hold for all possible plans we do not know the states in which the operators are applied. We therefore use a special classification for such operators: for each atom, we classify the operators adding the atom into those that *always produce* the atom (i.e. guarantee that it was false before) and those that *sometimes produce* it (i.e. cannot guarantee that is was previously false). Likewise, we classify operators deleting an atom into those that *always consume* and those that *sometimes consume* it. With this classification, the expression $\sum_{o \text{ always produces } a} \mathsf{Y}_o + \sum_{o \text{ sometimes produces } a} \mathsf{Y}_o - \sum_{o \text{ always consumes } a} \mathsf{Y}_o$ is an upper bound on the actual net change of atom a in a specific plan because it overestimates the producers and underestimates the consumers.

To specify a lower bound on each net change, we need to consider the current state s and the goal. For atom a, define

$S(a) = 1$ if a is true in s and 0 otherwise. Similarly, define $G(a) = 1$ if it is required in the goal and 0 otherwise. Then it is easy to see that in every s-plan the difference $G(a) - S(a)$ is a lower bound for the net change of a.

Since these bounds on the actual net change are valid for each possible s-plan, we can again determine an admissible heuristic estimate by optimizing the overall plan cost. The state equation heuristic for a state s is the objective value of the LP:

Linear Program 4 (State equation heuristic)
Optimization variables:

- Y_o for $o \in O$ for the number of occurrences of o in a plan

Minimize: $\sum_{o \in O} \mathit{cost}(o)\mathsf{Y}_o$
Subject to:

$$G(a) - S(a) \leq \sum_{o \text{ always produces } a} \mathsf{Y}_o + \sum_{o \text{ sometimes produces } a} \mathsf{Y}_o - \sum_{o \text{ always consumes } a} \mathsf{Y}_o \qquad \text{for all atoms } a$$

$$0 \leq \mathsf{Y}_o \qquad \text{for all } o \in O$$

Typically, we can determine only for very few STRIPS operators that they always produce or consume an atom based on their syntactic structure. For this reason, we usually strengthen this analysis with information on mutual exclusivity of atoms.[2]

We only have presented net change constraints based on the operators that sometimes produce an atom where the numeric bound of the constraint underestimates the actual net change. Pommerening et al. (2014) call such constraints *lower-bound net change constraints*. They also consider the opposite notion of *upper-bound net change constraints* that use a numeric *upper* bound and are based on the operators that sometimes consume an atom. However, these upper-bound constraints do not carry additional information because every feasible solution for the set of all lower-bound net change constraint is also feasible for all upper-bound net change constraints.

Bonet and van den Briel (2014) have further extended the state equation heuristic and also allow constraints for *sets* of atoms. The details on this extension would be beyond the scope of this paper.

Delete Relaxations

The last heuristic we want to introduce is based on the concept of *delete relaxation* (Bonet and Geffner 2001; Hoffmann and Nebel 2001). Similar to abstractions, the delete relaxation of a planning task is a simpler version of the original task with the property that every plan for the original task is also a plan for the relaxation. Delete relaxation ignores the negative side effects of all operators, i.e. operators of the delete relaxation only add atoms and never remove them. Delete relaxed tasks are easier in several ways: first, applying an operator never makes a previously applicable operator inapplicable. Second, once an operator was applied, applying it again cannot make additional atoms true because all atoms in the add effect are already true and cannot be falsified by other operators. As a direct consequence, the maximal length of an optimal plan can be limited to the number of operators.

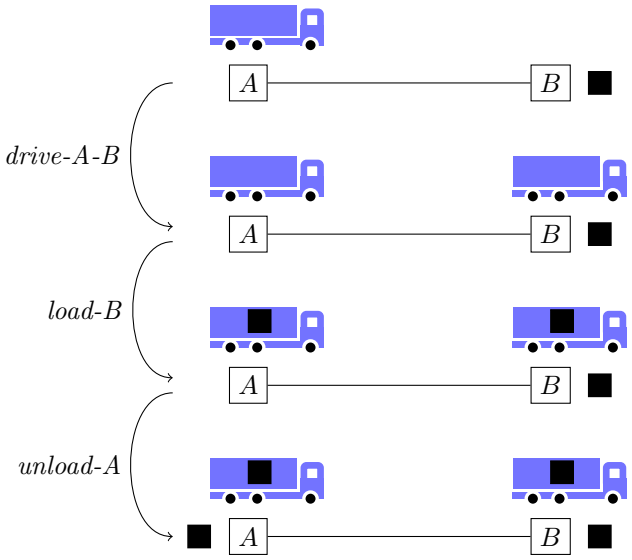


Figure 3: Solution for the example task's delete relaxation.

Consider the delete relaxation of our running example. Dropping the delete effects means that after driving the truck, it is at both locations at the same time and after loading or unloading the package, it is both in its new and its old position. After the first driving operator has been applied, the package can thus be loaded into the truck at B and then unloaded at A without moving the truck again. Figure 3 shows a solution for the delete relaxation.

Solving a delete relaxed problem optimally is still a hard problem (NP-equivalent) so most heuristics based on the delete relaxation approximate its solution in some way.

Imai and Fukunaga (2014) present an integer program that exactly defines the cost of an optimal delete relaxed plan for a state s and consider several of its relaxations. Their formulation consists of inequalities that are necessary and sufficient properties of a relaxed plan. It exploits that with relaxed planning the length of an optimal plan can be bounded by the number of operators in O. Therefore, it is sufficient to distinguish up to $|O| + 1$ different time points 0 to $|O|$.

The cost of the optimal plan for the delete relaxation is the objective value of the following integer program (described below). In addition to the optimization variables, it uses the constant function $S(a)$ that maps atoms in s to 1 and all other atoms to 0 and a sufficiently large constant M.

[2] Two atoms a and b are mutually exclusive if in every reachable state at most one of them can be true. If for such atoms an operator precondition requires a to be true and the effect adds b we can conclude that the operator always produces b. Mutually exclusive atoms can be precomputed (e. g., Helmert 2009).

Linear Program 5 (Delete Relaxation)
Optimization variables:

- $\mathsf{U}_o \in \{0,1\}$ for $o \in O$
  indicating whether o is part of the plan
- $\mathsf{R}_a \in \{0,1\}$ for $a \in A$
  indicating whether a is reached in the plan
- $\mathsf{F}_{o,a} \in \{0,1\}$ for $o \in O, a \in A$
  indicating whether o is the first operator in the plan that adds a
- $\mathsf{T}_o \in \{0,\ldots,|O|\}$ for $o \in O$
  indicating the time operator o is applied in the plan. The first operator can be applied at time point 0 and value $|O|$ indicates that the operator was not used.
- $\mathsf{T}_a \in \{0,\ldots,|O|\}$ for $a \in A$
  indicating the time atom a was true for the first time. Value 0 means that a was already true in the state s.

Minimize: $\sum_{o \in O} \mathit{cost}(o)\mathsf{U}_o$
Subject to:

$$\mathsf{R}_a, \mathsf{U}_o, \mathsf{F}_{o,a} \in \{0,1\}$$
$$\mathsf{T}_a, \mathsf{T}_o \in \{0,\ldots,|O|\} \quad \text{f.a. } o \in O, a \in A$$
$$\mathsf{R}_a = 1 \quad \text{f.a. } a \in G \quad (1)$$
$$S(a) + \sum_{o \text{ adds } a} \mathsf{F}_{o,a} \geq \mathsf{R}_a \quad \text{f.a. } a \in A \quad (2)$$
$$\mathsf{U}_o \geq \mathsf{F}_{o,a} \quad \text{f.a. } o \in O, a \in \mathit{add}(o) \quad (3)$$
$$\mathsf{R}_a \geq \mathsf{U}_o \quad \text{f.a. } o \in O, a \in \mathit{pre}(o) \quad (4)$$
$$\mathsf{T}_a \leq \mathsf{T}_o \quad \text{f.a. } o \in O, a \in \mathit{pre}(o) \quad (5)$$
$$\mathsf{T}_o + 1 \leq \mathsf{T}_a + M(1 - \mathsf{F}_{o,a}) \quad \text{f.a. } o \in O, a \in \mathit{add}(o) \quad (6)$$

A plan for the delete relaxation must reach every goal atom (1). An atom is reached if it is true in s or there is an operator that adds the atom for the first time (2). An operator can only be a first achiever for a atom if it is used (3). An operator can only be used if all its preconditions are reached (4) before it is applied (5) Finally, if an operator is the first achiever of an atom then this atom must be added for the first time directly after the operator is used (6). In the last equation M is a constant that is large enough to satisfy the inequality if $\mathsf{F}_{o,a} = 0$. Since the time steps are limited to the number of operators, $M = |O| + 1$ is sufficient.

Every feasible solution of the integer program can be transformed into a plan for the delete relaxation and the objective function calculates the cost of this plan. The plan shown for our example in Figure 3 corresponds to a solution that sets $\mathsf{U}_o = 1$ for the operators *drive-A-B*, *load-B* and *unload-A* with time steps T_o of 0, 1, and 2 respectively. The operator *drive-A-B* adds the atom *truck-at-B* for the first time in time step 1, so $\mathsf{F}_{\textit{drive-A-B,truck-at-B}} = 1$. Likewise, *load-B* and *unload-A* are first achievers for the package to be in the truck and at A. All other atoms are true in s so $\mathsf{R}_a = 1$ for all atoms a. For unused operators such as *drive-B-A* the solution sets $\mathsf{U}_o = 0$ and $\mathsf{T}_o = |O|$.

Imai and Fukunaga experimented with solving this IP and its LP relaxation and report that the LP solution often gives a very close or perfect approximation of the IP solution. They also considered several variable elimination techniques based on operators or atoms. Variables for operators can be eliminated if the respective operator either is necessary in every plan (an *action landmarks*) or can be shown to be unnecessary in the presence of an other operator. Similarly, variables for atoms can be eliminated if they are reached by every plan (*fact landmarks*) or can be detected *irrelevant* for reaching the goal.

Note that for the IP and its LP relaxation we can alternatively use a formulation with the previously seen objective function $\sum_{o \in O} \mathit{cost}(o)\mathsf{Y}_o$, adding an additional inequality $\mathsf{U}_o \leq \mathsf{Y}_o$ for each $o \in O$.

Operator-counting Constraints

It is no coincidence that all but the optimal cost partitioning heuristic for abstractions use the same objective function. We intentionally formulated the heuristics this way so that they fit the general framework of so-called *operator-counting constraints* (Pommerening et al. 2014). This framework allows one to combine constraints from different sources of information. The idea was not new: Bonet (2013) already suggested to use landmark constraints for strengthening the estimate of the state-equation heuristic.

The generalized framework is defined on the notion of *operator-counting constraints*. Such constraints are based on *operator-counting variables* Y_o for each operator o of the task. An operator-counting constraint for state s is a set of inequalities such that for every s-plan there exists a feasible solution that sets each Y_o to the number of occurrences of o in this plan. The constraints are not restricted to operator-counting variables. However, we can only combine them into a *constraint set* if the only common variables between constraints are the operator-counting variables.

Pommerening et al. (2014) have shown that for a constraint set C for state s the objective value of the following linear program is an admissible heuristic estimate for s. This is equally true if we restrict the operator-counting variables to integers.

Linear Program 6 (General operator-counting constraints)
Optimization variables:

- Y_o for $o \in O$ for the number of occurrences of o in a plan
- all other optimization variables occurring in C

Minimize: $\sum_{o \in \mathcal{O}} \mathit{cost}(o)\mathsf{Y}_o$
Subject to: C

The advantage of the framework is that it allows to arbitrarily combine the constraints of the previously seen heuristics with the guarantee of an admissible heuristic estimate. Adding additional operator-counting constraints results in a dominating heuristic because it can only reduce the set of feasible solutions. The resulting heuristic estimate can also be strictly larger than all individual heuristic estimates.

For the optimal cost partitioning heuristic for abstractions we presented a formulation that does not fit this framework. The reason is that this representation is more easily accessible. The heuristic is nevertheless covered by the general framework because the dual of the shown formalization fits the requirements of operator-counting constraints (Pommerening et al. 2014).

Conclusion

We have presented several recent LP-based heuristics for cost-optimal planning. Apart from the optimal cost partitioning for abstractions, the underlying LPs are usually not overly large.

It is hard to characterize the size for a typical planning instance because even in the standard benchmark suites for optimal planning from the International Planning Competitions this varies a lot. Lots of the instances have only a few hundred operators but there are also many with tens of thousands of operators and in the worst case there can be almost a million. Also the number of atoms is in most cases far below one hundred but it can go up to more than twenty thousands on exceptionally hard tasks (which are beyond the current capabilities of optimal planning).

While the individual LPs are usually solved very quickly, the challenge is that during one A^* search we easily need to compute the heuristic for several million states. Indeed, for a long time it was considered impractical to solve a linear program for every single state. The recent heuristics show that this is indeed possible and that it can be very beneficial.

However, very likely these heuristics do not realize their full potential. On the one hand, LP solvers implement many different solving strategies and for a non-expert it is not clear which one is best suited for the problem at hand. On the other hand, there is probably also some potential in reformulations of the underlying LPs. For example, the original definition of the post-hoc optimization aggregates variables that always occur together in a constraint. This aggregation gives a significant speed-up and we assume that there are many other such tricks that are obvious to experts in linear programming and optimization.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Abstraction Heuristics for Planning and Combinatorial Search" (AHPACS).

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 129(1):5–33.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proc. ECAI 2010*, 329–334.

Bonet, B., and van den Briel, M. 2014. Flow-based heuristics for optimal planning: Landmarks and merges. In *Proc. ICAPS 2014*, 47–55.

Bonet, B. 2013. An admissible heuristic for SAS^+ planning obtained from the state equation. In *Proc. IJCAI 2013*, 2268–2274.

Bylander, T. 1997. A linear programming heuristic for optimal planning. In *Proc. AAAI 1997*, 694–699.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive pattern database heuristics. *JAIR* 22:279–318.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Imai, T., and Fukunaga, A. 2014. A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. In *Proc. ECAI 2014*, 459–464.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proc. IJCAI 2009*, 1728–1733.

Katz, M., and Domshlak, C. 2008. Optimal additive composition of abstraction-based admissible heuristics. In *Proc. ICAPS 2008*, 174–181.

Katz, M., and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *AIJ* 174(12–13):767–798.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI 1996*, 1194–1201.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *Proc. ECAI 2010*, 335–340.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *Proc. ICAPS 2014*, 226–234.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proc. AAAI 2015*. To appear.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In *Proc. IJCAI 2013*, 2357–2364.

van den Briel, M., and Kambhampati, S. 2005. Optiplan: A planner based on integer programming. *JAIR* 24:919–931.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proc. CP 2007*, 651–665.

van den Briel, M.; Vossen, T.; and Kambhampati, S. 2008. Loosely coupled formulation for automated planning: An integer programming perspective. *JAIR* 31:217–257.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *Proc. IJCAI 1999*, 304–309.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *ICAPS 2003 Doctoral Consortium*, 156–160.