Heuristics for Planning with Penalties and Rewards using Compiled Knowledge

Blai Bonet
Departamento de Computation
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Héctor Geffner
Departamento de Tecnología
ICREA & Universitat Pompeu Fabra
08003 Barcelona, SPAIN
hector.geffner@upf.edu

Abstract
The automatic derivation of heuristic functions for guiding the search for plans in large spaces is a fundamental technique in planning. The type of heuristics that have been considered so far, however, deal only with simple planning models where costs are associated with actions but not with states. In this work we address this limitation by formulating a more expressive planning model and a corresponding heuristic where preferences in the form of penalties and rewards are associated with fluents as well. The heuristic, that is a generalization of the well-known delete-relaxation heuristic proposed in classical planning, is admissible, informative, but intractable. Exploiting however a correspondence between heuristics and preferred models, and a property of formulas compiled in d-DNNF, we show that if a suitable relaxation of the theory is compiled into d-DNNF, the heuristic can be computed for any search state in time that is linear in the size of the compiled representation. While this representation may have exponential size, as for OBDDs, this is not necessarily so. We report preliminary empirical results, discuss the application of the framework in settings where there are no goals but just preferences, and assess further variations and challenges.

Introduction
The automatic derivation of heuristic functions from problem descriptions in Strips and other action languages has been one of the key developments in recent planning research (McDermott 1996; Bonet, Loerincs, & Geffner 1997). Provided with these heuristics, the search for plans becomes more focused, and if the heuristics are admissible (do not overestimate), the optimality of plans can be ensured (Pearl 1983). The type of heuristics that have been considered so far, however, have serious limitations. Basically they are either non-admissible (Bonet & Geffner 2001; Hoffmann & Nebel 2001) or not sufficiently informative (Haslum & Geffner 2000), and in either case they are restricted to cost functions where plan costs depend on actions but not on states. As a result, the tradeoffs that can be expressed are limited; in particular, it is not possible to state a preference for achieving or avoiding an atom \( p \) in the way to the goal, or take this preference into account when searching for plans.

In this work, we address these limitations by formulating the derivation of heuristic functions in logical framework. Elsewhere we have shown that the heuristic represented by the planning graph (Blum & Furst 1995) can be understood as a precise form of deductive inference over the stratified theory that encodes the problem (Geffner 2004). Here our goal is not to reconstruct an existing heuristic but to use a logical formulation for producing a new one. The advantages of a logic framework are two: the derivation of heuristic information is an inference problem that can be made transparent with the tools of logic, and powerful algorithms have been developed that make certain types of logical inferences particularly effective. The latter includes algorithms for checking satisfiability (Moskewicz et al. 2001), computing answer sets (Simons, Niemela, & Soininen 2002), and compiling CNF formulas into tractable representations (Darwiche & Marquis 2002).

Here we consider preferences over actions \( a \) and fluents \( p \) that are expressed in terms of real costs \( c(a) \) and \( c(p) \). Action costs are assumed to be positive, while fluent or atom costs can be positive or negative. Negative costs express rewards. The cost of a plan is assumed to be given by the sum of the action costs plus the sum of atom costs for the atoms made true by the plan. We are interested in computing a plan with minimum cost. This is a well defined task, which as we will see, remains well-defined even when there are no goals but just preferences. In such a case, the best plans simply try to collect rewards while avoiding penalties, and if there are no rewards, since action costs are positive, the best plan is empty.

The preference model is not fully general but is considerably more expressive than the one underlying classical planning. As we will see, the model generalizes a recent formulation that deal with over-subscription or soft goals (Smith 2004; van den Briel et al. 2004), which in our setting can be modeled as terminal rewards, rewards that are collected when the propositions hold at the end of the plan. On the other hand, the costs and rewards are combined additively, so unlike other recent frameworks (Brafman & Chernyavsky 2005), partially-ordered preferences are not handled.

The definition of the planning model is motivated by the desire to have additional expressive power and a principled and feasible computational approach for dealing with it. For this, we want a useful heuristic with a clear semantics and a...
feasible algorithm able to capture interesting tradeoffs. We will be able to express in the model, for example, navigation problems where coins of different values are to be collected by avoiding, as much as possible, certain cells, or blocks problems where a tallest tower is to be constructed, or where the number of blocks that touch the table is to be minimized. In order to test the effectiveness of the approach we will also consider classical planning tasks where we will assess the approach empirically in relation to existing heuristics and planners.

The heuristic $h^+_c$ that we develop is simple and corresponds to the optimal cost of the relaxed problem where the delete-lists of all actions are ignored (Bonet & Geffner 2001). Since searching with this heuristic, even in the classical setting, involves an intractable computation in every state visited (Bylander 1994), planners such as HSP and FF resort to polynomial but non-admissible approximations (Bonet & Geffner 2001; Hoffmann & Nebel 2001). In this work, while considering the more general cost structure, we take a different approach: we compute the heuristic $h^+_c$ for each search state, but pay the price of an intractable computation only once, as preprocessing. This preprocessing yields what can be deemed as an evaluation network or circuit where we can plug any search state and obtain its heuristic value in linear-time. Of course, the time to construct this evaluation network and the size of the network may both be exponential, yet this is not necessarily so. The evaluation network, indeed, is nothing else but the directed acyclic graph that results from compiling a relaxation of the planning theory into d-DNNF, a form akin to OBDDs introduced in (Darwiche 2001; 2002) that renders efficient a number of otherwise intractable queries and transformations (Darwiche & Marquis 2002). The heuristic values are then obtained as the cost of the ‘best’ models, which can be computed in linear time once the relaxed theory is compiled into d-DNNF (Darwiche & Marquis 2004).

The plan for the paper is the following: we present in order the planning model, the heuristic $h^+_c$, and the correspondence between $h^+_c$ and the rank of a suitable propositional theory. We then deal with the search algorithm, which must handle negative costs, present experimental results, and summarize the contributions and discuss open problems.

**Planning Model**

We consider planning problems $P = (F, I, O, G)$ where $F$ is the set of relevant atoms or fluents, $I \subseteq F$ and $G \subseteq F$ are the initial and goal situations, and $O$ is a set of (grounded) actions $a$ with preconditions $\text{Pre}(a)$ and effects $a : C \rightarrow L$ where $\text{Pre}(a)$ and $C$ are sets ( conjunctions) of atoms, and $L$ is a fluent literal (positive or negative). An effect $a : C \rightarrow L$ is conditional if $C$ is not empty, and otherwise is unconditional. When the action $a$ is clear, we write such effects as $C \rightarrow L$ or simply as $L$ when $C$ is empty.

For each action $a \in O$, there is also a cost $c(a)$, and for each fluent or atom $p \in F$, a cost $c(p)$. Action costs are assumed to be positive, while atoms costs can be positive, negative, or zero. We call positive atom costs rewards, negative atom costs penalties, and refer to the resulting planning model as PR.

A plan $\pi$ is an applicable sequence of actions $a_0, \ldots, a_n$ that maps the initial situation into the goal. If $F(\pi)$ denotes the set of atoms made true at some point during the execution of the plan $\pi$ from the initial state $I$, then the cost $c(\pi)$ of $\pi$ is given by

$$c(\pi) \overset{\text{def}}{=} \sum_{a \in \pi} c(a) + \sum_{p \in F(\pi)} c(p).$$

We are interested in the plans $\pi$ that minimize $c(\pi)$; these are the optimal or best plans. If there is a plan at all, this optimization problem is well defined, although the best plan is not necessarily unique. We denote by $c^*(P)$ the cost of a best plan for $P$ with respect to the cost function $c$:

$$c^*(P) \overset{\text{def}}{=} \min\{c(\pi) : \pi \text{ is a plan for } P\}$$

and set $c^*(P) = \infty$ when $P$ has no solutions. Clearly, when $c(a) = 1$ and $c(p) = 0$ for all actions and atoms, the cost criterion of classical planning is obtained where $c^*(P)$ measures the minimum number of actions needed to solve $P$. The resulting framework, however, is more general, as both penalties and rewards can be expressed. Indeed, it is possible to model problems with no goals but just preferences. This can be done by setting the goal to a trivial atom true that holds in $I$ and that no action deletes. In such a case, the empty plan is optimal if there are no rewards (action costs are assumed to be positive), but other plans may have a smaller cost when rewards are present. In general, the best plans must achieve the goal by trading off action and atom costs.

The cost model captured in (1) is similar to the one used in over-subscription or partial-goal planning where due to constraints or preferences, it may not be possible or convenient to achieve all the goals (Smith 2004; van den Briel et al. 2004). One important difference is that the atoms $p \in F(\pi)$ that are rewarded in (1) do not have to be true by the end of the plan but sometime during its execution. The second difference is that such atoms may express penalties or rewards. If they express penalties (positive costs), they are not atoms to be achieved but to be avoided.

In order to capture preferences on end states as opposed to preferences on executions, we add an special action End with zero cost whose preconditions are the goals $G$ of the problem. We then demand that all plans end with this action. With this convention, as we will see below, the representation of preferences on goals is also possible.

**Modeling**

The cost model is simple but flexible. Some preference patterns that can be easily expressed are the following:

- **Terminal Costs:** an atom $p$ can be rewarded or penalized if true at the end of the plan by introducing a new atom $p'$, initialized to false, and a conditional effect $p \rightarrow p'$ for the action End. A reward or penalty $c(p')$ on $p'$ then captures a reward or penalty on $p$ at the end of the plan. We call $c(p')$ a terminal cost on $p$.

- **Goals:** once costs on terminal states can be expressed, goals are not strictly required. A hard goal can be modeled as a sufficiently high terminal reward, even if this is not a good idea from a computational point of view.
• **Soft Goals:** soft goals can be modeled as terminal rewards, and the best plans will achieve them depending on the costs involved.

• **Preferences on Literals:** while the model assumes that costs are associated with positive literals $p$ but not negative ones, standard planning transformation techniques can be used to add a new atom $p'$ that is true exactly when $p$ is false (Nebel 2000). Preferences on the negation of $p$ can then be expressed as preferences on $p'$.

• **Conditional Preferences:** conditional preferences can be captured as a result of the ability to handle conditional effects. For example, if being out is good, but being out when raining is not, then a reward can be defined on $out$ and a higher penalty on $wet$, which is a conditional effect of going out when raining.

• **Rewards on Conjunctions:** it is possible to reward states in which a set of atoms $p_1, \ldots, p_n$ is true by means of an action $\text{Collect}(p_1, \ldots, p_n)$ with preconditions $p_1, \ldots, p_n$, and effect $p^*$ which is rewarded. The same trick does not work for expressing penalties on conjunctions. The reason is that optimal plans will choose to collect a free reward if possible, but will never choose to collect a free reward (as would be required if $p^*$ is a penalty and not a reward).

For example, a Blocks problem where the number of blocks that touch the table is to be kept to a minimum (even if at the price of obtaining a longer plan) can be obtained by penalizing the atoms $\text{on}(x, \text{table})$ for the blocks $x$. More interestingly, the problem of building the tallest possible tower results from assigning terminal rewards to the atoms $\text{on}(x, y)$ for all the blocks $x$ and $y$ (with non-terminal rewards the best plans would instead place every block on top of every other block). Since actions are assumed to have positive costs, the best plans will be the ones that achieve a highest tower in a minimum number of steps (i.e., choosing one of the existing tallest towers as basis). Likewise, problems where an agent is supposed to pick up some coins while avoiding a dangerous ‘wumpus’, can be modeled by rewarding the actions $\text{have(coin)}$ and penalizing the atoms $\text{at}(x, y)$ where $x, y$ is the position of the wumpus.\footnote{The ‘Wumpus’ problem in (Russell & Norvig 1994) is more interesting though as it involves uncertainty and partial observability, issues that are not addressed in the PR model.}

Among preference patterns that cannot be captured in a natural way in this setting, are costs on sets of atoms (mentioned above) and partial preferences where certain costs are not comparable (Brafman & Chernovavsky 2005; Boutilier et al. 2004). Still, the first could be accommodated by extending the planning language with ramifications or axioms (Giunchiglia, Kartha, & Lifschitz 1997; Thiébault, Hoffmann, & Nebel 2005), while the second could be dealt with, in a limited way, by considering a set of cost functions rather than a single one.

The PR model can be extended to deal with repeated penalties or rewards, as when a cost is paid each time an atom is made true. We do not consider such an extension in this work, however, for two reasons: semantically, with repeated rewards, some problems do not have a well-defined cost (cyclic plans for example could accumulate infinite reward);\footnote{This same problem arises in Markov Decision Processes where the usual work around is to discount future costs (Bertsekas 1995).} and computationally, the proposed heuristics do not capture the specific features of that model.

### Heuristic $h^+$

Heuristics are fundamental for searching in large spaces. In the classical setting, several effective heuristics have been proposed, most of which are defined in terms of the delete-relaxation: a simplification of the problem where the deletelists of the operators are dropped. Delete-free planning is simpler than planning, in the sense that plans can be computed in polynomial time; still optimal delete-free planning is intractable too (Bylander 1994). Thus, on top of this relaxation, the heuristics used in many classical planners rely on other simplifications; the formulation in (Hoffmann & Nebel 2001) drops the optimality requirement in the relaxed problem, while the one in (McDermott 1996; Bonet, Loerincs, & Geffner 1997), assumes that subgoals are independent. In both cases, the resulting heuristics are not admissible.

The heuristic that we formulate for the $\mathsf{PR}$ model builds on and extends the optimal delete-relaxation heuristic proposed in classical planning. If $P^+$ is the delete-relaxation of problem $P$ and $c$ is the cost function, the heuristic $h^+_c(P)$, that estimates an upper bound of the cost of solving $P$ given $c$, is defined as:

$$h^+_c(P) \triangleq c^+(P^+).$$

For the 0/1 cost function that characterizes classical planning, where the cost of all atoms is 0 and the cost of all actions is 1, this definition yields the (optimal) delete-relaxation heuristic which provides an estimate of the number of steps to the goal. The heuristic is admissible and tends to be quite informative too (see the empirical analysis in (Hoffmann 2003)). Expression (3) generalizes this heuristic to the larger class of cost functions where actions may have non-uniform costs and atoms can be rewarded or penalized.

In the general PR setting, however, the heuristic $h^+_c$ is not always admissible. For example, consider a planning problem $P$ with initial situation $I = \{p\}$, goal $G = \{r\}$, and actions $a_1, a_2,$ and $a_3$ with effects

$$a_1 : p \rightarrow q,$$
$$a_2 : q \rightarrow r,$$
$$a_2 : p, q \rightarrow s,$$
$$a_3 : p \rightarrow \neg p$$

and let the cost function be such that $c(a_1) = 1$ for $i = 1, 2, 3$ and $c(s) = 10$ for atom $s$. The best plan for this problem is the action sequence $\pi^* = \langle a_1, a_3, a_2 \rangle$ with cost $c(\pi^*) = 3$. The plan $\pi' = \langle a_1, a_2 \rangle$ is shorter but makes the atom $s \perp$ true for a total cost $c(\pi') = 12$. The action $a_3$, skipped in $\pi'$, is used in $\pi^*$ for deleting $p$ before $a_2$ is done, thus preventing the conditional effect $a_2 : p, q \rightarrow s$ from triggering and adding the penalty $c(s) = 10$. In the delete-relaxation, this
The delete-relaxation does not yield admissible heuristic in the PR model when conditional effects combine with penalties in a certain way. Indeed, when there are no conditional effects (as in Strips) or there are no positive costs associated with fluents, the heuristic $h^+_p$ is admissible.

Let us say that the head $p$ of a conditional effect $a : C \rightarrow p$ is a conditional atom if the body $C$ contains an atom $q$ that is in turn the head of another conditional effect $b : C' \rightarrow q$ with $C' \neq 0$. Then we say that a cost $c(p)$ is a conditional penalty when $p$ is a conditional atom and $c(p) > 0$. In the absence of conditional penalties, the heuristic $h^+_p$ is admissible, and if not, an alternative, weaker but admissible relaxation can be defined:

**Proposition 1 (Admissibility)** In the absence of conditional penalties, the heuristic $h^+_p$ is admissible and hence $h^+_p (P) \leq c^*(P)$. The heuristic $h^+_p (P)$, for the cost function $c'$ that is like $c$ except that $c'(p) = 0$ for all conditional penalties $c(p)$, is always admissible.

Thus, if admissibility is required in the presence of conditional penalties, either the weaker heuristic $h^+_P$ needs to be used, or the culprit atoms must be rendered unconditional by mapping some operators into Strips (Nebel 2000).

If we let $P[I = s]$ and $P[G = g]$ refer to the planning problems that are like $P$ but with initial and goal situations $I = s$ and $G = g$ respectively, then (optimal) forward heuristic-search planners aimed at solving $P$ need to compute $h^+_p (P[I = s])$ for all states $s$ encountered, while regression planners need to compute $h^+_p (P[G = g])$ for all encountered subgoals $g$. Since each such computation is intractable, even for the 0/1 cost function, classical planners like HSP and FF settle on polynomial but non-admissible approximations. In this work we take a different path: we use the $h^+_p$ heuristic in the more general cost setting, but rather than performing an intractable computation for every search state encountered, we perform an intractable computation *only once*. This is done by compiling a propositional theory whose preferred models, for any state $s$ and goal $g$, can be computed in polynomial time and have rank equal to the heuristic values $h^+_p (P[I = s])$ and $h^+_p (P[G = g])$ respectively.

**Heuristics and Preferred Models**

Following (Kautz & Selman 1992: 1996), a propositional encoding for a sequential planning problem $P$ with horizon $n$ can be obtained by introducing fluent and action variables $p_i$ and $a_i$ for each fluent $p$, action $a$, and time step $i$ in a theory $T^n(P)$ comprised of the following formulas:

1. **Init:** $p_0$ for $p \in I$, $\neg q_0$ for $q \in F - I$
2. **Goal:** $p_n$ for $p \in G$
3. **Actions:** For $i = 0, 1, \ldots, n - 1$ and all $a$

$$a_i \supset p_i \text{ for } p \in Pre(a)$$

$$C_i \land a_i \supset L_{i+1}$$

for each effect $a : C \rightarrow L$

For a sufficiently large horizon $n$, the models of $T^n(P)$ are in correspondence with the plans for $P$: each model encodes a plan, and each plan determines a model.

For any cost function $c()$, if we define the rank of a model $M$ as $r(M) = c(\pi(M))$ where $\pi(M)$ stands for the sequence of actions made true in $M$, and the rank of a theory $T$ as

$$r^*(T) \overset{\text{def}}{=} \min_{M \vdash T} r(M)$$

with $r^*(T) = \infty$ when $T$ has no models, it follows that the cost of $P$ and the rank of its propositional encoding $T^n(P)$ can be related as follows:

**Proposition 2 (Costs and Ranks)** For a sufficiently large time horizon $n$ (exponential in the worst case), $c^*(P) = r^*(T^n(P))$, where the model rank $r(M)$ is given by the cost $c(\pi(M))$ of the plan defined by $M$.

This correspondence, which follows directly from the definitions, does not give us much unless we have a way to derive theory ranks effectively. A result in this direction comes from (Darwiche & Marquis 2004) that shows how to compute theory ranks $r^*(T)$ efficiently when $r$ is a literal-ranking function and the theory $T$ is in d-DNNF (Darwiche 2002). A literal ranking function ranks models in terms of the rank of the literals $l$ that are true:

$$r(M) = \sum_{l: M \models l} r(l)$$

For literal-ranking functions $r$ and propositional theories $T$ compiled into d-DNNF, Darwiche and Marquis show that

**Proposition 3 (Darwiche and Marquis)** If a propositional theory $T$ is in d-DNNF and $r$ is a literal-ranking function, then the rank $r^*(T)$ can be computed in time linear in the size of $T$.

This result suggests that we could compute the optimal cost $c^*(P)$ of $P$ by compiling first the theory $T^n(P)$ into d-DNNF and then computing its rank $r^*(T^n(P))$ in time linear in the size of the compilation. There are two obstacles for this however. The first is that the model ranking function $r(M) = c(\pi(M))$ in Prop. 2 is defined in terms of the cost of the atoms made true during the execution of the plan, not in terms of the literals true in the model, and hence it is not exactly a literal-ranking function. The second, and more critical, is that the horizon $n$ needed for ensuring Prop. 2 is normally too large for $T^n(P)$ to compile. We show below though that these problems can be handled better when the computation of the heuristic $h^+_p (P)$, that approximates the real cost $c^*(P)$, is considered instead.

$^3$Darwiche and Marquis use the name ‘normal weighted bases’ rather than literal-ranking functions.
Stratified Encodings

Since the heuristic $h^+_c(P)$ is defined in terms of the optimal cost of the relaxed, delete-free problem $P^+$, it is natural to consider the computation of the heuristic in terms of the theory $T^n(P^+)$ of the relaxed problem. We will do this but first simplify the theory $T^n(P^+)$ by dropping the seriality constraints that are no longer needed in the delete-free setting where any parallel plan can be easily serialized retaining its cost. In addition, we will drop from $T^n(P^+)$ the init and goal clauses as we want to be able to compute the heuristic values $h^+_c(P[I = s])$ and $h^+_c(P[G = g])$ for any possible initial state $s$ and subgoals $g$ that might arise in a progression or regression search respectively. We call the set of clauses that are left in $T^n(P^+)$, the stratified (relaxed) encoding and denote it by $T^n_1(P)$. Later on we will consider another encoding that does not involve time at all.

The first crucial difference between the problem $P$ and its delete-free relaxation $P^+$ is the horizon $n$ needed for having a correspondence between models and plans. For $P$, the optimal plans may have exponential length due to the number of different states that a plan may visit. On the other hand, the optimal plans for $P^+$ have at most linear length, as without deletes, actions can only add atoms, and thus the number of different states that can be visited is bounded by the number of fluents. Longer plans are possible but they will not be optimal as they will contain useless actions.

The second difference is that the optimal cost of the delete-free problem can be put in correspondence with the rank of its propositional encoding using a simple literal-ranking function compatible with Prop. 3, as any atom achieved in a delete-free plan remains true until the end of the plan, and no action needs to be repeated.

If we let $s_0$ and $g_0$ stand for the init and goal clauses in $T^n(P)$ encoding the initial and goal situations of problem $P[I = s, G = g]$, the following correspondence between heuristic values and theory ranks can be established:

**Proposition 4 (Heuristics and Ranks)** For a sufficiently large horizon $n$ (linear in the worst case) and any initial and goal situations $s$ and $g,$

$$h^+_c(P[I = s, G = g]) = r^n_1(T^n_1(P) \land s_0 \land g_0),$$

where $r$ is the literal ranking function such that $r(p_a) = c(p)$ for every fluent $p$, $r(a_i) = c(a)$ for every action $a$ and $i \in [0, n - 1]$, otherwise $r(1) = 0$.

Exploiting then Proposition 3 and the ability of d-DNNF formulas to be conjoined with *literals* in linear-time (Darwiche 2001), we get:

**Theorem 5 (Compilation and Heuristics)** Let $\Pi_1(P, n)$ refer to the compilation of theory $T^n_1(P)$ into d-DNNF where $n$ is a sufficiently large horizon (linear in the worst case). Then the heuristic values $h^+_c(P[I = s, G = g])$ for any initial and goal situations $s$ and $g$, and any cost function $c$, can be computed from $\Pi_1(P, n)$ in linear time.

This theorem tells us that a single compilation suffices for computing a huge set of heuristic values in time that is linear in the size of the compilation. The heuristic values $h^+_c(P[I = s, G = g])$ provide estimates of the cost of achieving any goal $g$ from any initial state $s$. During a forward search, however, only the values $h^+_c(P[I = s])$ are needed, while in a regression search, only the values $h^+_c(P[G = g])$ are needed. The formulation, however, yields a larger number of heuristic values that can be used, for example, in a bidirectional search. The computation of such values is linear in the size of the compilation $\Pi_1(P, n)$ which may be exponential in the size of the original encoding $T_1(P, n)$. This however, as for OBDDs, is not necessarily so.

**LP Encodings**

The encoding $T^n(P)$ for computing the optimal cost of $P$ requires an horizon $n$ that is exponential in the worst case, while the encoding $T^n_1(P)$ for computing the heuristic $h^+_c$ requires an horizon that is linear. However, a much more compact encoding for computing $h^+_c$, which requires no time or horizon at all, can be obtained. We call it the LP (for Logic Program) encoding as it is obtained from a set of positive Horn clauses (Lloyd 1987).

The LP encoding of a planning problem $P$ for computing the heuristic $h^+_c$ is obtained from the propositional LP rules of the form

$$p \leftarrow C, Pre(a), a$$  \hspace{1cm} (6)

for each (positive) effect $a : C \rightarrow p$ associated with an action $a$ with preconditions $Pre(a)$ in $P$, where $Pre(a)$, $C(a, p)$, or both may be empty. For convenience, as we explain below, for each atom $p$ in $P$, we introduce also a ‘dummy’ action $set(p)$ which has no precondition and unique effect $p$ encoded as:

$$p \leftarrow set(p).$$  \hspace{1cm} (7)

These actions will be formal devices for ‘setting’ the initial situation to $s$ when computing the heuristic values $h^+_c(P[I = s])$ for any state $s$. No such encoding trick is needed for the goals $g$.

The LP encoding, that will enable us to compute the $h^+_c$ heuristic in a more effective way, has two features that distinguish it from the previous stratified encodings. The first is that there is no time. Time, however, is not necessary as we will focus on a class of minimal models that have an implicit stratification that is in correspondence with the temporal stratification. Such minimal models will be grounded on the actions as all fluents will have a well-founded support based on them. The second distinctive feature is that actions do not imply their preconditions. This will not be a problem either as actions have all positive costs and, in this encoding, all require their preconditions in order to have some effect. So while models that make actions true without their preconditions are possible, such models will not be preferred.

For a planning problem $P$, let $T_2(P)$ refer to the collection of rules (6) and (7) encoding the effects of the actions in $P$, including the $set(p)$ actions, and let $wffc(T_2(P))$ stand for the well-founded fluent completion of $T_2(P)$: a completion formula defined below that forces each fluent $p$ to have a well-founded support. Then if we let $set(s)$ refer to the collection of unit clauses $set(p)$ that represent a situation $s$, namely $set(p) \in set(s)$ if $p \in s$, and $\neg set(p) \in set(s)$ if
for all actions except for \( \text{move}(A, B) \) with cost \( c(\text{move}(A, B)) = 10 \).

The best plan for this state-goal pair in the delete-relaxation is \( \pi = \{ \text{move}(A, B), \text{move}(B, C) \} \) which is also the best plan without the relaxation, so
\[
h_+^*(P[I = s, G = g]) = c^*(P[I = s, G = g]) = 11.
\]

Theorem 7 says that this heuristic value must correspond to the rank of the well-founded fluent completion of \( T_2(P) \), \( \text{wffc}(T_2(P)) \), extended with the set of literals given by
\[
\text{set}(s) = \{ \text{set}(at(A)), \neg\text{set}(at(B)), \neg\text{set}(at(C)) \},
\]
and \( g = \{ at(C) \} \). While we will not spell out the theory \( \text{wffc}(T_2(P)) \) in detail, let us illustrate why it must be stronger than Clark’s completion. For this problem, Clark’s completion for the fluent atoms gives us the theory:
\[
\begin{align*}
\text{at}(C) & \equiv (\text{at}(B) \land \text{move}(B, C)) \lor \text{set}(at(C)), \\
\text{at}(B) & \equiv (\text{at}(A) \land \text{move}(A, B)) \lor \text{set}(at(B)) \lor (\text{at}(C) \land \text{move}(C, B)), \\
\text{at}(A) & \equiv (\text{at}(B) \land \text{move}(B, A)) \lor \text{set}(at(A)).
\end{align*}
\]
For the literal ranking function \( r \) that corresponds to \( c^* \), the best ranked model of Clark’s completion extended with the literals in \( \text{set}(s) \) and \( g \), has rank 2 which is different from \( h_+^*(P[I = s, G = g]) = 11 \). In such a model, the costly \( \text{move}(A, B) \) action is avoided, and \( at(C) \) is made true through a circular justification that involves the cheaper actions \( \text{move}(B, C) \) and \( \text{move}(C, B) \). This arises because the program \( T_2(P) \) contains a cycle involving the atoms \( \text{at}(B), \text{at}(C), \text{move}(B, C), \) and \( \text{move}(B, C) \). In the well-founded completion defined in (Lin & Zhao 2003), Clark’s completion is applied to a program which is different than \( T_2(P) \) and where the circularities are broken with the addition of an extra predicate that encodes the possible precedences in the models that are well-founded.

It is worth pointing out that while the heuristic \( h_+^* \) and the optimal cost \( c^* \) coincide for this state, goal, and cost function, they do not coincide in general for other combinations. For example, if the goal \( g \) is set to the initial state \( s = \{ at(A) \} \) and the atom \( at(C) \) is given cost \(-20\), then \( c^*(P[I = s, G = g]) = -7 \) while \( h_+^*(P[I = s, G = g]) = -11 \).
−9. The reason is that in the delete-relaxation the two actions \(move(C, B)\) and \(move(B, A)\) that get the agent back to \(at(A)\) are not needed as the atom \(at(A)\) is never deleted.

This last variation illustrates that in the PR model, heuristics and costs can both be negative, and even if the initial situation is also the goal situation, the optimal plan is not necessarily empty. This all means that we cannot just plug the heuristic into an algorithm like A* and expect to get back optimal solutions. Indeed, in the above variation the root node of the search is a goal node, and yet the empty plan is not optimal. In order to use the heuristic \(h_+^*\) to guide the search for plans in the PR model, these and other issues need to be considered in the search.

**From Heuristics to Search**

We will focus first on the use of the heuristic in a progression search from the initial state, and then briefly mention what needs to be changed for a regression search. First of all, in the PR model, a search node needs to keep track not only of the state of the system \(s\) but also of the set of fluents \(t\) with non-zero costs that have been achieved in the way to \(s\). This is because in the model penalties and rewards associated with such atoms are paid only once. Thus, search nodes \(n\) must be pairs \((s, t)\), and the heuristic for those nodes \(h_+^*(n)\) must be set to \(h_+^*(s)\) where \(c'(x) = c(x)\) for all actions and fluents \(x\), except that \(c'(x) = 0\) if \(x \in t\). As usual, the evaluation function \(f(n)\) for a node \(n\) is given by the sum \(g(n) + h(n)\) where \(h(n) = h_+^*(n)\) and \(g(n)\) is the accumulated cost along the path \(n_0, n_1, \ldots, n_i, n_{i+1}\) from the root \(n_0\) to \(n = n_{i+1}\). This accumulated cost is the sum \(c(n_0) + c(a_0, n_0) + c(a_1, n_1) + \cdots + c(a_i, n_i)\) where \(c(a_i, n_i)\) is \(c(a_i)\) plus the cost \(c(p)\) of the atoms \(p \notin t_i\) that the action \(a_i\) makes true in \(s_{i+1}\), while \(c(n_0)\) is the sum of all costs \(c(p)\) for \(p \in s_0\). For the root node \(n_0 = \langle s_0, t_0\rangle\), \(s_0\) is the initial state and \(t_0\) is the set of atoms \(p \in s_0\) with non-zero costs.

In this search space, the search algorithm must handle both negative heuristics and costs, while ensuring optimality. This rules out algorithms such as Dijkstra or A* that do not handle negative costs (Pearl 1983), yet a simple Best-First Search (BFS) algorithm can be defined that reduces to A*: the algorithm maintains the (accumulated) cost \(g(n)\) of the best solution node if \(h(n)\) in OPEN is a solution, but it terminates then with \(n'\) as the solution node if \(h(n')\) is non-negative.

It is simple to show that the algorithm is correct when the heuristic is monotone (like \(h_+^*\)). In such a case, even if \(h\) is negative, the evaluation function \(f(n)\) cannot decrease along any path from the root, and hence if a solution has been found with cost \(g(n)\) which is no greater than \(f(n')\) for every node in OPEN, then \(g(n)\) will be no greater than the solutions that go through those nodes.

Unlike A*, the algorithm may terminate by reporting a node \(n\) in the CLOSED list as a solution. This happens for example when there are no goals but the heuristic \(h(n_0)\) deems a certain reward worth the cost of obtaining it, when it is not. For example, if the reward is −10, and the estimated and real cost for achieving it are 9 and 11 respectively, then the best plan is to do nothing with cost 0. However, initially \(g(n_0) = 0\) and \(h(n_0) = -1 < 0\), so the algorithm expands \(n_0\) and keeps going until the best node \(n'\) in OPEN satisfies \(f(n') \geq g(n_0) = 0\), returning \(n_0\) as the solution node. In (van den Briel et al. 2004), the termination condition of A* is also modified for dealing with (terminal) rewards (soft goals) but the proposed termination condition does not ensure optimality, as in particular, it will never report a solution node from the CLOSED list.

Most of this discussion carries directly to regression search where classical regression needs to be modified slightly: while in the classical setting, an action \(a\) can be used to regress a subgoal \(g\) when \(a\) ‘adds’ an atom \(p\) in \(g\), in the penalties and reward setting, \(a\) can also be used when it adds an atom \(p\), that while not in \(g\), has a negative cost \(c(p) < 0\).

**Empirical Results**

We report some empirical results that illustrate the range of problems that can be handled using the proposed techniques. We derive the heuristic using the LP encodings and Theorem 7. The compilation into d-DNNF is done using Darwiche’s publicly available c2d compiler.\(^7\) We actually consider a ‘forward’ theory used for guiding a progression search, and a ‘backward’ theory used for guiding a regression search. The first is obtained from the compilation of the formula \(wffc(T_2(P)) \land G\) where \(G\) is the goal, while the second is obtained from the compilation of the formula \(wffc(T_2(P)) \land \text{set}(I)\) where \(I\) is the initial situation. The heuristic \(h_+^*\) for the regression search is complemented with structural ‘mutex’ information, meaning that the heuristic values associated with subgoals \(g\) are set to \(\infty\) when \(g\) contains a pair of structurally mutex fluents. This is because, the regression search tends to generate such impossible states (Bonet & Geffner 2001).

All the experiments are carried on a Linux machine running at 2.80GHz with 2Gb of RAM, and terminated after taking more than 2 hours or more than 1Gb of memory. Four domains are considered: two classical domains, Logistics and Blocks, where the \(h_+^*\) heuristic can be compared with classical heuristics, and two navigation domains, Wumpus and Elevator.

**Logistics.** Table 2 shows the time taken by the compilation of some ‘forward’ and ‘backward’ logistic theories, along with the size of the resulting d-DNNF formula. These are all serialized instances from the 2nd Int. Planning Competition (Bacchus 2001), with several packages, cities, trucks, and airplanes, some having plans with more than 40 actions.

\(^7\)At [http://reasoning.cs.ucla.edu/c2d](http://reasoning.cs.ucla.edu/c2d).
Almost all of these instances compile, although backward theories, where the initial state is fixed, take much less time and yield much smaller representations. Table 1 provides information about the quality and effectiveness of the heuristic \( h_e^+ \) for the classical 0/1 cost function, in relation with the classical, admissible heuristic \( h^2 \) (Haslum & Geffner 2000), a generalization of the heuristic used in Graphplan (Blum & Furst 1995). The table shows the heuristic and real-cost values associated with the root nodes of the search, along with the time taken by the search and the number of nodes expanded. It can be seen that the \( h_e^+ \) heuristic is more informed than \( h^2 \) in this case, and when used for guiding a regression search, scales up to problems that \( h^2 \) cannot solve.

It is important to emphasize that once the theories are compiled they can be used for any cost function. So these logistics theories can be used for settings where, for example, packages have different priorities, loading them in various tracks involves different costs, etc. This applies to all domains.

**Blocks World.** Blocks instances do not compile as well as logistic instances. We do not report actual figures as we managed to compile only the first 8 instances from the 2nd IPC. These are rather small instances having at most 6 blocks, where use of the heuristic \( h_e^+ \) does not pay off.

**Wumpus.** In this world, simplified from (Russell & Norvig 1994), an agent moves in a grid \( n \times n \) collecting coins, each with cost \(-8\), while avoiding ‘wumpuses’ at cost 20 each. For this problem, we managed to compile instances with a few coins and wumpus, and sizes no greater than \( n = 4 \). We then considered a further relaxation with all wumpuses removed from the problem, resulting in a less informed heuristic but which enabled the compilation of larger instances.

### Table 2: Compilation data for the first 18 logistic problems from 2nd IPC (serialized), some having plans with more than 40 actions. Time refers to compilation time in seconds, while Nodes to the number of nodes in the DAG representing the d-DNNF formula.

<table>
<thead>
<tr>
<th>Problem</th>
<th>backward theory</th>
<th>forward theory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Nodes</td>
</tr>
<tr>
<td>4-0</td>
<td>0.34</td>
<td>1,163</td>
</tr>
<tr>
<td>4-1</td>
<td>0.20</td>
<td>1,163</td>
</tr>
<tr>
<td>4-2</td>
<td>0.21</td>
<td>1,155</td>
</tr>
<tr>
<td>5-0</td>
<td>0.20</td>
<td>1,155</td>
</tr>
<tr>
<td>5-1</td>
<td>0.20</td>
<td>1,155</td>
</tr>
<tr>
<td>5-2</td>
<td>0.33</td>
<td>1,163</td>
</tr>
<tr>
<td>6-0</td>
<td>0.20</td>
<td>1,155</td>
</tr>
<tr>
<td>6-1</td>
<td>0.20</td>
<td>1,155</td>
</tr>
<tr>
<td>6-2</td>
<td>0.21</td>
<td>1,163</td>
</tr>
<tr>
<td>6-3</td>
<td>0.32</td>
<td>1,163</td>
</tr>
<tr>
<td>7-0</td>
<td>1.26</td>
<td>3,833</td>
</tr>
<tr>
<td>7-1</td>
<td>1.38</td>
<td>3,837</td>
</tr>
<tr>
<td>8-0</td>
<td>1.30</td>
<td>3,833</td>
</tr>
<tr>
<td>8-1</td>
<td>1.37</td>
<td>3,837</td>
</tr>
<tr>
<td>9-0</td>
<td>1.98</td>
<td>3,854</td>
</tr>
<tr>
<td>9-1</td>
<td>1.27</td>
<td>3,833</td>
</tr>
<tr>
<td>10-0</td>
<td>6.86</td>
<td>13,153</td>
</tr>
<tr>
<td>10-1</td>
<td>6.87</td>
<td>13,090</td>
</tr>
</tbody>
</table>

For example, problems involving grids \( 10 \times 10 \) with 8 coins are compiled in a few seconds. Table 3 shows the results of the forward search over a family of \( 10 \times 10 \) problems with 8 coins and a variable number of wumpuses. These are problems with soft goals as not all the coins are collected, and in all cases, an optimal path among the collected coins that avoids the wumpuses, if that is convenient, must be found. The result for the forward search guided by the ‘relaxed’ \( h_e^+ \) heuristic is compared with a ‘blind’ search guided by the \( h_0 \) heuristic. This is not the \( h = 0 \) heuristic which is not admissible in this setting (optimal costs are negative), but the heuristic that results from adding all potential (uncollected) rewards. Since this heuristic is very fast, even if the search guided by \( h_e^+ \) expands an order-of-magnitude less nodes, the search with \( h_0 \) is usually faster.

**Elevator:** The last domain consists of a building with \( n \) floors, \( m \) positions in each floor ordered linearly, and \( k \) elevators. There are no hard goals but various rewards and penalties associated with certain positions as in Wumpus, and all actions have cost 1. Figure 1 shows the instance 10-5-1 with 10 floors, 5 positions per floor, and 1 elevator aligned at position 1 on the left. We consider also an instance 10-5-2 where there is an additional elevator on the right, performs steps but obtains a better cost of \(-5\). The LP encoding for computing the \( h_e^+(P) \) heuristic doesn’t compile for this domain except for very small instances. However, a good and admissible approximation \( h_e^+(P') \) can be obtained by relaxing the problem \( P \) slightly by simply dropping the fluent \((\text{inside} e)\) from all the operators (this a so-called pattern-database relaxation (Culberson & Schaeffer 1998), where certain atoms are dropped from the problem (Edelkamp 2001; Haslum,
In this work we have combined ideas from a number of areas, such as search, planning, knowledge compilation, and answer set programming to develop

1. a simple planning model PR that accommodates fluent penalties and rewards,
2. an admissible heuristic $h^+_c$ for informing the search in this model,
3. an account of $h^+_c$ in terms of the rank of the preferred models of a suitable theory,
4. a correspondence between this theory and the strong completion of a non-temporal logic program,
5. an approach that exploits these correspondences and the properties of d-DNNF for computing all heuristic values $h^+_c$ in time linear in the size of the compilation, and
6. a best-first algorithm able to use this heuristic and handle negative costs, while ensuring optimality.

All these ideas are combined in an actual PR planner that we tested on a number of problems.

The computational bottleneck in this approach is the compilation. We have seen that a number of non-trivial domains such as Logistics compile well, while others such as Blocks

Bonet, & Geffner 2005)). Using this technique, we were able to compile theories with up to 10 floors and 10 positions in less than a second. The problem in Fig. 1 is then solved optimally in 0.39 seconds, expanding 238 nodes. As a reference, a ‘blind’ search based on the non-informative admissible heuristic $h_0$ above takes 161 seconds and expands 445,956 nodes. More results appear in Table 4 where it is shown that the heuristic is cost-effective in this case, and enables the optimal solution of problems that are not trivial.

### Summary and Discussion

Table 1: Search results for serialized logistics problems $P$ using the heuristics $h^2$ and $h^+_c$, the second used to search in both directions. Both heuristics complemented with structural mutexes. Time and Nodes stand for search time and number of expanded nodes. A dash means time or memory exceeded. The cost function is the classical cost function and $c^*$ stands for the optimal cost.

<table>
<thead>
<tr>
<th>Problem</th>
<th>$c^*(P)$</th>
<th>$h^2(P)$</th>
<th>Time</th>
<th>Nodes</th>
<th>$h^+_c$ with mutex backward</th>
<th>Time</th>
<th>Nodes</th>
<th>$h^+_c$ forward</th>
<th>Time</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-0</td>
<td>20</td>
<td>12</td>
<td>0.23</td>
<td>4,259</td>
<td>19</td>
<td>0.02</td>
<td>40</td>
<td>19</td>
<td>8.24</td>
<td>76</td>
</tr>
<tr>
<td>4-1</td>
<td>19</td>
<td>10</td>
<td>0.40</td>
<td>7,079</td>
<td>17</td>
<td>0.14</td>
<td>109</td>
<td>17</td>
<td>26.95</td>
<td>259</td>
</tr>
<tr>
<td>4-2</td>
<td>15</td>
<td>10</td>
<td>0.02</td>
<td>537</td>
<td>13</td>
<td>0.01</td>
<td>25</td>
<td>13</td>
<td>6.94</td>
<td>72</td>
</tr>
<tr>
<td>5-0</td>
<td>27</td>
<td>12</td>
<td>8.00</td>
<td>118,389</td>
<td>25</td>
<td>0.75</td>
<td>490</td>
<td>25</td>
<td>113.26</td>
<td>1,075</td>
</tr>
<tr>
<td>5-1</td>
<td>17</td>
<td>9</td>
<td>0.52</td>
<td>7,904</td>
<td>15</td>
<td>0.08</td>
<td>103</td>
<td>15</td>
<td>21.50</td>
<td>212</td>
</tr>
<tr>
<td>5-2</td>
<td>8</td>
<td>4</td>
<td>0.00</td>
<td>143</td>
<td>8</td>
<td>0.00</td>
<td>8</td>
<td>8</td>
<td>0.79</td>
<td>8</td>
</tr>
<tr>
<td>6-0</td>
<td>25</td>
<td>10</td>
<td>28.32</td>
<td>316,175</td>
<td>23</td>
<td>1.00</td>
<td>668</td>
<td>23</td>
<td>94.98</td>
<td>932</td>
</tr>
<tr>
<td>6-1</td>
<td>14</td>
<td>9</td>
<td>0.10</td>
<td>1,489</td>
<td>13</td>
<td>0.01</td>
<td>19</td>
<td>13</td>
<td>3.39</td>
<td>33</td>
</tr>
<tr>
<td>6-2</td>
<td>25</td>
<td>10</td>
<td>25.49</td>
<td>301,054</td>
<td>23</td>
<td>0.89</td>
<td>517</td>
<td>23</td>
<td>53.52</td>
<td>516</td>
</tr>
<tr>
<td>6-3</td>
<td>24</td>
<td>12</td>
<td>7.87</td>
<td>99,827</td>
<td>21</td>
<td>0.84</td>
<td>727</td>
<td>21</td>
<td>52.53</td>
<td>537</td>
</tr>
<tr>
<td>7-0</td>
<td>36</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>33</td>
<td>9.74</td>
<td>4,973</td>
<td>33</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>7-1</td>
<td>44</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>39</td>
<td>4,157.70</td>
<td>175,886</td>
<td>39</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>8-0</td>
<td>31</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>29</td>
<td>11.64</td>
<td>591</td>
<td>29</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>8-1</td>
<td>44</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>41</td>
<td>283.32</td>
<td>12,913</td>
<td>41</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>9-0</td>
<td>36</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>33</td>
<td>6.58</td>
<td>3,083</td>
<td>33</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>9-1</td>
<td>30</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>29</td>
<td>1.54</td>
<td>81</td>
<td>29</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10-0</td>
<td>?</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>41</td>
<td>—</td>
<td>—</td>
<td>41</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10-1</td>
<td>42</td>
<td>12</td>
<td>—</td>
<td>—</td>
<td>39</td>
<td>5,699.2</td>
<td>20,220</td>
<td>39</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
Table 4: Results for the regression search over elevator instances \(n-m-k\) with \(n\) floors, \(m\) positions, and \(k\) elevators. A dash means time or memory exceeded. The ‘relaxed’ heuristic \(h_r^+\) is defined over problem with the atom \(\text{inside } e\) relaxed. The ‘blind’ heuristic \(h_0\) just adds up the uncollected rewards. Both heuristics complemented with structural mutexes.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Len/c*</th>
<th>(h_r^+) with mutex backward</th>
<th>(h_r^+ (P))</th>
<th>Time</th>
<th>Nodes</th>
<th>(h_0) with mutex backward</th>
<th>(h_0 (P))</th>
<th>Time</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-4-2</td>
<td>12/—9</td>
<td>—18</td>
<td>0.35</td>
<td>1,382</td>
<td>—28</td>
<td>4.19</td>
<td>29,247</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-6-2</td>
<td>23/—14</td>
<td>—26</td>
<td>21.44</td>
<td>24,386</td>
<td>—49</td>
<td>2,965.90</td>
<td>6,229,815</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-6-3</td>
<td>23/—14</td>
<td>—29</td>
<td>133.48</td>
<td>76,128</td>
<td>—49</td>
<td>—</td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-5-1</td>
<td>11/—3</td>
<td>—7</td>
<td>0.39</td>
<td>238</td>
<td>—42</td>
<td>161.85</td>
<td>445,956</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-5-2</td>
<td>32/—5</td>
<td>—23</td>
<td>330.72</td>
<td>189,131</td>
<td>—42</td>
<td>—</td>
<td>—</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

References


