Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

Revisiting Regression in Planning

Vidal Alcázar, Daniel Borrajo, Susana Fernández, Raquel Fuentetaja
Universidad Carlos III de Madrid
Av. Universidad, 30
28911 Leganés, Spain
valcazar@inf.uc3m.es; dborrajo@ia.uc3m.es;
sfarregu, rfuentet@inf.uc3m.es

Abstract

Heuristic search with reachability-based heuristics is arguably the most successful paradigm in Automated Planning to date. In its earlier stages of development, heuristic search was proposed as both forward and backward search. Due to the disadvantages of backward search, in the last decade researchers focused mainly on forward search, and backward search was abandoned for the most part as a valid alternative. In the last years, important advancements regarding both the theoretical understanding and the performance of heuristic search have been achieved, applied mainly to forward search planners. In this work we revisit regression in planning with reachability-based heuristics, trying to extrapolate to backward search current lines of research that were not as well understood as they are now.

1 Introduction

Automated planning (AP) is the task of finding a sequence of actions that, from a given initial state, reaches a set of goals. It is one of the oldest areas of research in AI, but only relatively recently domain-independent planners have been able to solve non-trivial tasks. Heuristic search [Bonet and Geffner, 2001] has become one of the most important paradigms in the area, as shown by the results of the most recent International Planning Competition (IPC)[1] and the much higher number of publications in AP about heuristic search compared to publications about other successful approaches like local-search [Gerevini and Serina, 2002] or SAT-based planning [Kautz and Selman, 1999].

Search can be done either advancing from the initial state towards a goal state, called forward search or progression, or from a goal state to the initial state, called backward search or regression. In heuristic search most works use combinatorial domains as benchmarks. Combinatorial domains are in most cases trivially invertible and have a single goal state. Therefore, often forward and backward search are analogous and the same techniques and conclusions are relevant to both cases. However, this is not true in AP. While the initial state is completely defined due to the closed-world assumption, goals are defined as partial states. Thus, the value of propositions not included in the goal set are unknown and multiple goal states are possible. As a consequence, backward search reasons over sets of facts and not complete states, making the search more akin to state-set search [Pang and Holte, 2011] than to traditional heuristic search.

Due to these differences backward search planners have several drawbacks: the techniques developed for forward search may not be applicable, duplicate detection is more complex due to partial states, and spurious states (partial states containing a set of facts that is unreachable from the initial state) may be generated. Early research on heuristic search in AP studied both forward and backward search [Bonet and Geffner, 2001], but these drawbacks lead to a worse performance of regression planners compared to progression planners. For this reason, research on backward search in AP was discontinued in many cases.

Over the last decade important works shed light on both the formal properties of forward search, like the relationship between different heuristics [Helmert and Geffner, 2008; Helmert and Domshlak, 2009], and the empirical impact of widely used techniques, like preferred operators and deferred heuristic evaluation [Richter and Helmert, 2009]. Nevertheless and despite the strong relationship between forward and backward search, this has not been exploited to improve over the results of previous backward search planners. In particular, some techniques that may address the shortcomings of regression in satisficing planning, like preferred operators, seem to have potential. In this work we analyze how efficient techniques commonly used in progression can be extrapolated to backward search. We define some new concepts related with regression and propose several novel techniques implemented in a new regression planner, based on Fast Downward [Helmert, 2006], called FDR (Fast Downward Regression). In particular, the concepts and techniques are the following: formalization of regression in SAS$^+$; disambiguation of states and action preconditions; action pruning using e-deletion; improvements on the computation of the applicable actions; novel definition of reasonable orders

[1] http://ipc.icaps-conference.org/

in regression; computation of reachability heuristics and preferred operators in regression; usage of P^m formulations of the planning task to infer more informed heuristics in regression; and computation of strong precedences related to the context-enhanced additive heuristic in regression.

2 Formalization

In this section we will present the current two main formalizations of the planning tasks: propositional and multi-valued. Additionally, regression in both formalizations is described.

2.1 Propositional Planning

The propositional formalization of a planning task is defined as a tuple *P=(S,A,I,G)*, where *S* is a set of atomic propositions (also known as facts), *A* is the set of grounded actions derived from the operators of the domain, $I \subseteq S$ is the initial state and $G \subseteq S$ the set of goal propositions. Each action $a \in A$ is defined as a triple *(pre(a), add(a), del(a))* (preconditions, add effects and delete effects) where *pre(a), add(a), del(a)* $\subseteq S$. Actions can have non-unitary costs. A distinction must be made between satisficing planning, that tries to find a plan preferring those of better quality with respect to a given metric, and optimal planning, where the best plan must be found. In this work we focus on satisficing planning.

2.2 Multi-Valued Formalizations of Planning

The two most common multi-valued formalizations are SAS$^+$ [Bäckström and Nebel, 1995] and the Finite-Domain Representation proposed by Helmert [2006], based on SAS$^+$. In this work we will refer only to SAS$^+$. A planning task in SAS$^+$ is defined as a tuple $\Pi=(\mathcal{V},s_0,s_\star,\mathcal{O})$. $\mathcal{V}$ is a set of state variables, and every variable $v \in \mathcal{V}$ has an associated extended domain $D_v^+ = D_v \cup \{\mathbf{u}\}$ composed of the regular domain of each variable, D_v, and the undefined value $\mathbf{u}$. The total state space is defined as $S_v^+ = D_{v_0}^+ \times \ldots \times D_{v_n}^+$ and the value of a variable $v \in \mathcal{V}$ in a given state s, also known as *fluent*, is defined as $s[v]$. Partial states are states in which at least a fluent $s[v_i] = \mathbf{u}$. s_0 is the initial state, defined over $\mathcal{V}$ such that $s_0[v_i] \neq \mathbf{u}\ \forall v_i \in \mathcal{V}$. $s_\star$ is the partial state that defines the goals. $\mathcal{O}$ is a set of operators (actions), where each operator is a tuple $o = (pre(o), post(o), prev(o))$, where $pre(o), post(o), prev(o) \in S_v^+$ represent the *pre-, post-* and *prevail-conditions* respectively. An action $o \in \mathcal{O}$ is applicable in a state s if $\forall v_i \in \mathcal{V} : (prev(o)[v_i] = s[v_i] \vee pre(o)[v_i] = s[v_i]) \wedge (pre(o)[v_i] = \mathbf{u} \vee pre(o)[v_i] = s[v_i])$. The resulting state s' from the application of o in s is equal to s except that $\forall v_i \in \mathcal{V}\ s.t.\ post(o)[v_i] \neq \mathbf{u} : s'[v_i] = post(o)[v_i]$.

Fluents in SAS$^+$ are extrapolable to propositions in a propositional representation. For instance, regarding actions, preconditions in SAS$^+$ are preconditions deleted by the action, postconditions are positive effects and prevail conditions are preconditions not deleted by the action. In this work we assume that every concept described in terms of fluents is also relevant to propositions.

2.3 Planning Formalizations in Regression

Regression search in planning starts from the goals and applies the actions backwards to find a path from the goals to the initial state. Action preconditions are partial definitions of the states in which the actions can be applied. Thus, the states generated by regression are partial states. In a propositional representation in regression, the closed world assumption cannot be enforced due to the existence of undefined values unless undefined propositions are represented explicitly.

Given a propositional planning task *P=(S,A,I,G)*, the definition of *P* for regression is a tuple *P'=(S,A,I',*S_G*)* where *I'= G* is the initial (partial) state; S_G is the set of goal states, composed by all the sets of propositions (partial states) $s \subseteq S$ for which $s \subseteq I$, $S_G = \{s \mid s \subseteq S, s \subseteq I\}$. The applicability of actions is redefined as follows: $a \in A$ is applicable in a partial state $s \subseteq S$ if it is *relevant* ($add(a) \cap s \neq \emptyset$) and *consistent* ($del(a) \cap s = \emptyset$). The resulting state $s_r \subseteq S$ obtained from regressing a in s is $s_r = (s \setminus add(a)) \cup pre(a)$. This can also be seen as a reversed action a' in which $add(a)$ is a set of disjunctive preconditions of a', $del(a)$ are negative preconditions of a' and $pre(a)$ are the *adds* of a'. Progression and regression in planning are not symmetric as each state in the regression state space may represent a set of states of the progression state space.

Similarly, a SAS$^+$ task in regression is a tuple $\Pi'=(\mathcal{V},s'_0,S'_\star,\mathcal{O})$, where $s'_0 = s_\star$ and $S'_\star$ is the set of partial states subsumed by s_0, following the next definition of subsumption.

Definition 1 *(Subsumption of states in SAS$^+$) Given two SAS^+ states s_i and s_j, s_i subsumes s_j ($s_i \sqsubseteq s_j$) when for every $v \in \mathcal{V}$, $s_i[v] = s_j[v]$ or $s_i[v] = \mathbf{u}$.*

An action $o \in \mathcal{O}$ is applicable in a partial state s in regression if $\forall v_i \in V : s[v_i] = \mathbf{u} \vee s[v_i] = post(o)[v_i] \vee s[v_i] = prev(o)[v_i]$ and $\exists v_i \in V : s[v_i] = post(o)[v_i] \wedge s[v_i] \neq \mathbf{u}$. The resulting state s' obtained from applying o in s in regression is equal to s except that $\forall v_i \in \mathcal{V}\ s.t.\ pre(o)[v_i] \neq \mathbf{u} : s'[v_i] = pre(o)[v_i]$ and $\forall v_i \in \mathcal{V}\ s.t.\ prev(o)[v_i] \neq \mathbf{u} : s'[v_i] = prev(o)[v_i]$. Note that applicability of actions in SAS$^+$ is more restricted than in a propositional representation, as prevail preconditions must be taken into account. All the actions applicable in a propositional representation that are not applicable in SAS$^+$ lead to spurious states: the resulting state s' would contain $s[v_i]$ and $prev(o)[v_i]$, which are different values of the same variable unless one of them is undefined.

3 Common Concepts

Here we describe two important concepts used throughout the paper. These concepts are relationships of mutual exclusion and e-deletion. From this point on we will use SAS$^+$ unless otherwise stated.

3.1 Mutual Exclusivity between Facts

An important problem in backward search is the generation of spurious states (states not reachable by a forward search). Mutual exclusivity between fluents can be used to prune such states [Bonet and Geffner, 2001]. A set of fluents $M = \{p_1, \ldots, p_m\}$ is a set of mutually exclusive fluents of size m (mutex of size m) if there is no reachable state $s \subseteq S$ such that all elements in M are true in s.

The use of mutexes may not detect all the spurious states in regression, but in many domains they are able to prune a significant amount of them. Computing mutexes is exponential on the size *m*, so in most cases computing mutexes of size $m > 2$ is not practical. Invariant monotonicity [Helmert, 2006] and h^m [Bonet and Geffner, 2001] with $m = 2$ are usually used to compute mutexes of size two, although the former finds strictly fewer mutexes than the latter.

3.2 E-deletion

The negative effects of an action are not restricted to its explicit *delete* effects. In particular, whether a fluent is deleted along any path that ends with a particular action can be precomputed. This belongs to the more general definition of *e-deletion* [Vidal and Geffner, 2005].

Definition 2 *An action* a *e-deletes a fluent* f *if* f *must be false in every state resulting from the execution of an ordered set of actions whose last action is* a *from a state in which* f *is true.*

There are three cases in which an action *e-deletes* a fluent *f*: it deletes *f*; it has a set of preconditions mutex with *f* and does not add *f*; or it adds a set of fluents mutex with *f*. For instance, the action *(stack b c) e-deletes (on a b)* because it adds *(clear b)*, which is mutex with *(on a b)*. In multi-valued representations, deleting a fluent means changing the value of the variable it corresponds to, which is equivalent to adding a fluent mutex with *f*. Hence, the first case is a particular instance of the third case in multi-valued representations.

4 Regression and Backward Heuristic Search: HSPr

In satisficing planning, HSPr [Bonet and Geffner, 2001], one of the best known backward search planners, uses a propositional formalization, a weighted A* algorithm and the additive heuristic, h^{add}. h^{add} is the sum of the accumulated cost of achieving every goal proposition in a delete-free version of the problem. The main difference with HSP, its forward version, is that it caches the estimation of the cost from *I* to every proposition. It uses mutexes found with h^2 to prune spurious states. As heuristic search planners spend most of their time computing the heuristic, caching h^{add} allows HSPr to generate states at a much higher rate. However, doing regression has several important drawbacks. First, mutexes may not suffice to prune spurious states in some domains. Second, when using a propositional representation, detection of duplicate states requires modifications in the algorithm due to the presence of propositions whose value may be unknown. Furthermore, duplicate detection is unable to detect subsumption of states. Finally, h^{add} is heavily affected by partial states: if the value of a proposition is undefined, its cost is not accounted for, so h^{add} may differ greatly depending on the number of unknown propositions. For these reasons HSPr performed overall worse than HSP. These problems combined with the success of some techniques introduced by other forward heuristic planners, like Fast Forward [Hoffmann and Nebel, 2001] and Fast Downward [Helmert, 2006], made researchers abandon for the most part backward search in planning.

5 Revisiting Regression in Planning

In this section we revisit some important concepts originally defined for forward search, adapting them to their use in regression planners. Each of the subsections describes the set of techniques, proposes a novel implementation in regression and defines them formally if needed.

5.1 SAS$^+$, Invariant Groups and Disambiguation

Partial states require the definition of the undefined value **u**. SAS$^+$ allows it by adding it to the domain of all the variables in a preprocessing step. Hence, any SAS$^+$ planner can do regression with no changes in their search algorithm apart from changing the applicability and effects of actions. Identical partial states are detected as duplicates, but subsumption of states is not. For example, given two states s_0 and s_1 such that $\forall v_i \in V : s_0[v_i] = s_1[v_i]$, they are detected as duplicate. However, if this is true except for some $v_j \in V$ such that $s_0[v_j] = u$ and $s_1[v_j] \neq u$, $s_0 \sqsubseteq s_1$ is not detected.

Planners that use SAS$^+$ transform a planning task expressed in the Planning Domain Definition Language (PDDL) into SAS$^+$ by computing invariant groups.

Definition 3 *An invariant group is a set of fluents θ such that every fluent $f_i \in \theta$ is mutex with every other fluent $f_j \in \theta$ and $f_j \neq f_i$. Exactly one fluent $f_i \in \theta$ must be true in every non-spurious complete state.*

In partial states the value of some variables may be unknown.[2] Nevertheless, the value of other known variables may reduce the domain of the unknown variables. We call *disambiguation* to the process of reducing the valid domains of the unknown variables of a partial state. There are three relevant cases for any given invariant group θ: (1) only one fluent $f \in \theta$ can be true in the partial state; (2) no fluent $f \in \theta$ can be true; and (3) more than one fluent $f \in \theta$ can be true. The first case means that f must be true in the state and thus the variable f corresponds to is no longer unknown. The second case means the partial state is spurious, as it is not possible for exactly one fluent $f \in \theta$ to be true. And in the third case, no additional information can be inferred.

For example, given $V = \{v_0, v_1, v_2\}$ and $D_{v_0} = \{a, b, c\}$, if in a given partial state $s : s[v_0] = u$, $s[v_1]$ is mutex with $v_0 = a$ and $s[v_2]$ is mutex with $v_0 = b$, then we can infer that $s[v_0] = c$; it is the only possible value for v_0. If additionally $s[v_1]$ or $s[v_2]$ are mutex with $s[v_0] = c$, then v_0 has no possible value and the state is spurious.

Disambiguation of partial states fulfills two purposes: adding information to partial states by reducing the number of unknown variables upon generation, and pruning spurious states undetectable by only using binary static mutexes. Having fewer unknown variables impacts the performance in two ways. First, heuristics tend to be more accurate, as the cost of fluents that otherwise would be ignored is accounted for. Second, it reduces the number of cases in which there is subsumption of expanded states thanks to cases in which $s_i, s_j \subseteq S$, $s_i \sqsubseteq s_j$ before disambiguation and $s_i = s_j$ after disambiguation. Figure 1 shows an example of pruning by disambiguation in the *floortile* domain.

[2] Note that every variable in SAS$^+$ corresponds to an invariant

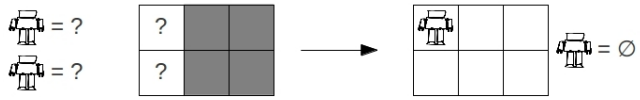

Figure 1: Pruning a spurious state in *floortile* by disambiguation. The second robot in the state at the right has no valid location because all cells are clear or occupied.

Problems in *floortile* consist of two or more robots that have to paint the cells of a grid. The initial state contains the locations where the robots are. The goal contains the painted cells. It is possible to find a plan doing regression in which a single robot traverses and paints the whole grid. This means that there may be a partial state in which all the cells are either painted, clear or occupied by the first robot. When disambiguating the state, we can see that there are no legal values for the variables that represent the position of the other robots, as a robot cannot be at a painted cell, a clear cell or a cell occupied by another robot. Such a state can be safely pruned. Binary mutexes with no disambiguation are ineffective to detect this case in regression.

Disambiguation can also be done over the preconditions of actions after grounding, as the set of preconditions define a partial state. More preconditions lead to fewer unknown variables in partial states. Also, operators whose set of preconditions is spurious can be discarded.

5.2 Enhanced Applicability in Regression and Decision Trees for Successor Generation

HSPr evaluates states at a faster rate than HSP. However, many of those states are spurious and are pruned using mutexes anyway, so often no performance gain is achieved. Consider for example the *blocksworld* domain: if there is a tower of blocks and *(arm-empty)* is true in the partial state in regression, half of the actions of the domain are aplicable because they add *(arm-empty)*, whereas only one (stacking the upper block) leads to a non-spurious state. The other applicable actions produce states that contain mutexes. For example, if *(on a b)* and *(on b c)* are true and *(stack b c)* is applied in regression, the generated state would contain the mutex {*(on a b), (holding b)*}, as *(holding b)* is a precondition of *(stack b c)*.

This is due to the current definition of applicability of actions in regression. In fact, the current definition makes planners knowingly generate spurious states that will be pruned using mutexes. This is a waste of effort that can be avoided using *e-deletion* instead of just regular *deletes*. The applicability of an action in regression can be modified to reflect this.

Definition 4 *If* $e\text{-}del(a)$ *is the set of fluents e-deleted by action* a*,* a *is consistent with partial state* s *if* $e\text{-}del(a) \cap s = \emptyset$*.*

Using *e-deletion* causes the actions to have a bigger number of negative preconditions (i.e., a regular *move* operator *e-deletes* the moving agent at every location other than the destination). Also, regular *adds* are disjuntive preconditions in regression, which imposes an additional complexity when checking applicability. This may cause an important overhead during search. A solution is to use precomputed decision trees as successor generators [Helmert, 2006].

Forward search uses decision trees (similar to RETE networks used for matching rule-based systems [Forgy, 1982]) to avoid checking the applicability of all the actions of the problem. Every inner node represents a variable $v \in \mathcal{V}$ and has an edge for each $d \in D_v$ plus the *don't care* value. Leaf nodes contain the sets of actions that are applicable on that node. Each edge leads to another inner node representing the next variable or to a leaf node. When building the leaf (action) nodes, an action a is propagated at every inner node down the edge that corresponds to $v = d$ if $v = d \in pre(a) \vee v = d \in prev(a)$. If $v = d \notin pre(a) \wedge v = d \notin prev(a)$ then a is propagated down the *don't care* edge. So, the path from the root to the leaf node represents the preconditions of the action. Every action occurs in at most one leaf node, so the number of leaf nodes is bounded by the number of actions of the problem. When computing the applicable actions of a state s, at every inner node the algorithm follows the *don't care* edge and the edge that corresponds to $s[v]$. The union of the sets of actions of the visited leaf nodes is the set of applicable actions.

The main difference with the decision trees used by Fast Downward is that FDR must deal with disjunctive preconditions, negative preconditions and unknown variables. To take this into account, the inner nodes represent fluents (values of variables) instead of variables. They have three edges for the cases $\top$, $\bot$ and *don't care* through which actions that add, *e-delete* and do not change the variable of the fluent are respectively propagated. If the fluent is unknown in a partial state, all the three children must be explored, as opposed to either $\top$ or $\bot$ and *don't care*. This combines the efficiency of decision trees in progression and the greater expressiveness required in regression.

6 Reachability Heuristics in Regression

In this section, we describe how to compute recent state-of-the-art heuristics in regression, exploiting advantages of regression like caching schemes. We use a propositional formalization of the task as it is the *de facto* implementation even in planners that use a multi-valued formalization, like FD.

6.1 Best Supporter Caching

Most forward planners use heuristics based on a delete-relaxation of the problem. There are several heuristics of this kind [Keyder and Geffner, 2009], but HSPr is limited to h^{add}. Delete-relaxation and critical path heuristics belong to the same family of heuristics. They differ mainly in the functions used to (1) select the best supporter of propositions, (2) aggregate the supporters of preconditions (subgoals) and goals, and (3) compute a cost estimation from a set of aggregated supporters [Fuentetaja *et al.*, 2009]. Supporters and costs are derived from a forward reachability analysis that depends exclusively on the source state. If the source state does not change, all the necessary information can be cached. This al-

group.

lows planners to compute these heuristics in regression without performing additional reachability analysis, which greatly speeds up their computation. In practice, we cache the best supporters determined by Equation 1, which defines how to select the best supporter a_p of a proposition p among the set of actions $A(p)$ that add p.[3]

$$a_p \in \underset{a \in A(p)}{\operatorname{argmin}} \left(cost(a) + h^{max}(pre(a), I)\right) \quad (1)$$

Caching the best supporters allows computing the FF heuristic [Hoffmann and Nebel, 2001]: the cost of a plan that reaches a partial state from the initial state in a delete-free version of the problem. Figure 2 shows the difference between caching in HSPr and best supporter caching. In this problem an agent must go from I to K to pick up a key and then carry it to G. h^{add} adds the distance from I to K to the distance from I to G and to the cost of picking up the key. So, if the cost of all actions is one, $h^{add} = 9$. FF computes a relaxed plan by tracing back from the fluents of the partial state to the initial state using best supporters. Moving from I to K and G with no deletes requires 5 actions, so $h^{FF} = 6$. In either case if the partial state changes no new reachability analysis is needed; both heuristics can be computed with the cached information.

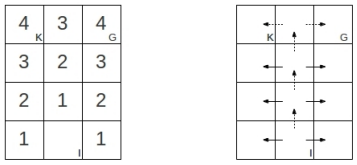

Figure 2: Caching costs (left) and best supporters (right); dashed arrows are part of the relaxed plan.

Additional information can be extracted by caching supporters. To compute preferred operators [Helmert, 2006], a technique that improves the performance in progression significantly [Richter and Helmert, 2009], an aggregated set of actions is required. We redefine preferred operators in regression as the applicable actions *in regression* that appear in the relaxed plan. In the former example moving to G from the left is the only preferred operator.

6.2 Computing Heuristics in P^m

Reachability heuristics are usually computed by ignoring the *delete* effects of actions. These effects can be partially included by using the alternative version of the problem P^m [Haslum, 2009]. P^m is a redefinition of the planning task in which every fluent in P^m is a set of fluents of size m. h^{max} in P^m is equivalent to h^m in P^1; this is not exclusive to h^{max}, any other reachability heuristic can be computed in P^m. Computing such heuristics is exponential on m, so in practice it is not viable to use them in progression, as it would require an expensive reachability analysis per evaluated state. New developments regarding semi-relaxed plans in progression [Keyder *et al.*, 2012] have been proposed, although they require complex implementations and are not as informative as the heuristics computed in P^m. Nevertheless, in regression only one reachability analysis is needed, which makes their computation tractable thanks to caching.

We propose a general definition of best supporters in P^1 that takes into account sets of propositions of size m based on Equation 1. Best supporters are now determined by Equation 2, that defines the best supporter a_P of a set of propositions P with size $|P| \leq m$.

$$a_P \in \underset{a \in A(P)}{\operatorname{argmin}} \left(cost(a) + h^m(Reg(P, a), I)\right) \quad (2)$$

$A(P)$ is the set of actions that generate at least a proposition in P without deleting any other:

$$A(P) = \{a \in \mathcal{A} \mid add(a) \cap P \neq \emptyset \wedge del(a) \cap P = \emptyset\} \quad (3)$$

$Reg(P, a)$ is the result of regressing P through a: $Reg(P, a) = P \setminus add(a) \cup pre(a)$. The cost of reaching the partial state s from the initial state I is defined as:

$$h(s, I) = \sum_{a \in \pi(s,I)} cost(a) \quad (4)$$

where

$$\pi(P, I) = \begin{cases} \emptyset & \text{if } P \subseteq I \\ \pi(a_P, P, I) & \text{if } P \nsubseteq I, |P| \leq m \\ \bigcup_{P_m} \pi(P_m, I) & \text{if } P \nsubseteq I, |P| > m \end{cases} \quad (5)$$

and

$$\pi(a, P, I) = \{a\} \cup \pi(Reg(P, a), I) \quad (6)$$

In our implementation we cache best supporters for atom pairs when computing h^2 to find mutexes. This allows computing a relaxed plan in P^2 (essentially a FF^2 version of the original FF heuristic). For partial state s we compute the atom pairs $P_i \in s$. The best supporter a_{P_i} of each atom pair P_i is added to the relaxed plan and the atom pairs obtained from $Reg(P_i, a_{P_i})$ that are not true in I are added as open preconditions. This is repeated until no open preconditions remain.

6.3 Reasonable Orders, Contexts and Paths through Invariants

Reasonable orders between goal propositions were first proposed to create a goal agenda [Koehler and Hoffmann, 2000] and later on extended to landmarks and arbitrary pairs of facts [Hoffmann *et al.*, 2004]. *e-deletion* allows us a simple and general way to extend the definition to sets of facts.

Definition 5 *A fact f is reasonably ordered before a set of fluents F_r ($f <_r F_r$) if all the supporters of f are e-deleters of some fact $f' \in F_r$.*

Reasonable orders were deemed "unsound" in progression, because it is not true that, if $f <_r f'$, then f must be achieved first in every solution plan. Nevertheless, some sound information can be derived: if $f <_r f'$ and both fluents must be true until the end of the plan, then f' must be achieved *last*. In progression this can only be proved for top level goals. But, in regression, this holds for every $f <_r f'$ if f and f' are true in a partial state, as partial states in regression can be treated as top level goals. In this case, the supporters of f can be pruned because f' must be supported first.

If consistency is defined as in Definition 4 no additional pruning is obtained. However, reasonable orders in regression also allow us inferring stronger precedence constraints.

[3] Ties in Equation 1 are broken arbitrarily.

The context-enhanced additive heuristic, h^{cea} [Helmert and Geffner, 2008], is a variation of h^{add}. Several contexts or pivots are selected during the heuristic computation and the cost of some goals is computed from a state that results from achieving the pivots. In Figure 2 the most informative context is *agent-at=K*, in which case h^{cea} would add the distance from I to K to the distance from K to G and to the cost of picking up the key, yielding $h^{add} = 7$, the cost of the optimal solution. Choosing the right context is still an open question. Reasonable orders were proposed as a way of computing contexts [Cai *et al.*, 2009], although in progression they may be misleading. In regression, though, if $f <_r f'$ and f and f' are true at the same time, f' must be supported after having achieved f. This hints from which context f' is achieved in most solution plans. A likely context is the conflicting fluents that cause the *e-deletion* of f' by the best supporter of f [Nguyen and Kambhampati, 2001]. In Figure 2, the best supporter of *have-key*=$\top$ is *pick-key(K)*, whose precondition *agent-at=K* is the cause of the *e-deletion* of *agent-at=G*. This suggests that *agent-at=K* is indeed a useful context.

Best supporter caching is also possible in h^{cea}. If the conflicting fluents belong to the same invariant group θ, the best supporters can be cached when a fluent f is used as context. For this, a Dijsktra algorithm in the Domain Transition Graph (DTG) defined by θ is done with f as the source. This allows computing relaxed plans and preferred operators in DTGs. In our implementation relaxed plans are computed instead of adding costs.

7 Experimentation

The proposed techniques were implemented on top of Fast Downward (FD). We used the benchmark suite from IPC2011 and compared FDR against HSPr, FD with greedy best-first search (GBFS), h^{FF} and delayed evaluation and Mp [Rintanen, 2010]. Mp was included because it includes several modifications that make the SAT solver prefer actions that support open goals and preconditions like in regression. Four configurations were tested: FDr^{FF}, FDr^{add}, FDr^{cea}, FDr^2, which use h^{FF}, h^{add}, h^{cea} and h^{FF} in P^2 respectively. FDr's search algorithm is GBFS with regular evaluation. Since the focus of the experimentation is coverage, action costs were ignored. Time score was omitted for lack of space.

Preferred operators were not used. Although promising *a priori*, they were not helpful. This may seem counterintuitive, as the reduced branching factor and the additional heuristic guidance should help, but in partial states they often commit strongly to unpromising areas of the search space. When enabled, FDr^{FF} only solved one more problem in *woodworking* while losing coverage in *parcprinter*, *transport* and *visitall*.

As seen in Table 1, FDr^{FF} solves consistently more problems than HSPr and is close to Mp in coverage. FDr^2 performed worse except in *parcprinter*, in which it solved the whole set of problems. The overhead of h^{cea} does not pay off, surpassing other heuristics only in two domains. No regression planner was able to solve any instance in *barman* due to spurious states that could not be detected (*i.e.* due to some imprecisions in the domain formulation h^2 cannot detect mutexes like a shaker being clean and containing an ingredient at the same time). The regression planners fare worse than FD in total coverage, although this varies among domains. FDR is superior to FD in *floortile*, as this domain contains a high number of dead ends in progression that are difficult to detect. FDr^{add} also solves many more problems in *visitall*, a domain in which heuristics like FF are affected by big plateaus. *openstacks* and *sokoban* are the opposite case: problems trivial for FD are hard for all the regression planners. As hypothesized, Mp seems to behave similarly to regression planners, with the exceptions of *nomystery* and *pegsol*. This confirms the impact of directionality in planning [Massey, 1999].

Domain	FD	r^{FF}	r^{add}	r^{cea}	r^2	HSPr	Mp
barman	18	0	0	0	0	0	6
elevators	18	16	16	10	0	**20**	17
floortile	3	**20**	**20**	**20**	**20**	18	**20**
nomystery	9	6	6	7	5	4	17
openstacks	**20**	0	0	0	1	0	0
parcprinter	11	12	12	12	**20**	11	0
parking	19	5	0	4	0	3	0
pegsol	**20**	14	11	15	11	10	**20**
scanalyzer	17	**20**	**20**	17	19	**20**	17
sokoban	19	3	2	3	2	3	2
tidybot	14	1	0	0	0	0	17
transport	0	5	0	2	0	0	0
visitall	4	5	**17**	3	7	5	0
woodworking	19	19	19	11	16	17	**20**
Total	**191**	126	128	104	101	111	136

Table 1: Coverage in IPC11. *r* stands for FDR.

An ablation study was also done disabling disambiguation and decision trees as successor generators. 8 fewer problems were solved using FDr^{add} after disabling disambiguation. The number of expansions increased by an order of magnitude in *parcprinter* and *woodworking*, staying the same or very similar in other domains. When disabling decision trees, 5 fewer problems were solved. The impact was proportional to the number of grounded actions and the number of effects of those actions, increasing search time by an order of magnitude in domains like *floortile* and *woodworking*.

8 Conclusions

In this work we analyzed several state-of-the-art techniques in forward search and explored their potential in regression. Some novel definitions were also proposed. We implemented a new regression planner, FDR, which performed consistently better than its predecessor HSPr. Overall performance seems to depend greatly on the topography of the search space, which means that one should not rule out regression in domain-independent planning. This also implies that there may be a high synergy between FD and FDr, potentially leading to a bidirectional planner better than the combination of the individual planners in a portfolio. The implementation of such a planner remains as future work along with the use of other state-of-the-art techniques which are simple to employ in regression, like landmarks [Richter and Westphal, 2010] and multiple queues [Röger and Helmert, 2010].

Acknowledgments

This work has been partially supported by the Spanish government through MICINN projects TIN2008-06701-C03-03 (as a FPI grant) and TIN2011-27652-C03-02.

References

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–656, 1995.

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

[Cai *et al.*, 2009] Dunbo Cai, Jörg Hoffmann, and Malte Helmert. Enhancing the context-enhanced additive heuristic with precedence constraints. In *International Conference on Automated Planning and Scheduling*, pages 50–57, 2009.

[Forgy, 1982] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[Fuentetaja *et al.*, 2009] Raquel Fuentetaja, Daniel Borrajo, and Carlos Linares López. A unified view of cost-based heuristics. In *Workshop on Heuristics for Domain-Independent Planning, ICAPS'09*, Thessaloniki (Greece), September 2009.

[Gerevini and Serina, 2002] Alfonso Gerevini and Ivan Serina. LPG: A planner based on local search for planning graphs with action costs. In *Conference on Artificial Intelligence Planning Systems*, pages 13–22. AAAI, 2002.

[Haslum, 2009] Patrik Haslum. $h^m(P) = h^1(P^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m. In *International Conference on Automated Planning and Scheduling*, pages 354–357, 2009.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *International Conference on Automated Planning and Scheduling*, pages 162–169, 2009.

[Helmert and Geffner, 2008] Malte Helmert and Hector Geffner. Unifying the causal graph and additive heuristics. In *International Conference on Automated Planning and Scheduling*, pages 140–147, 2008.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)*, 26:191–246, 2006.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)*, 14:253–302, 2001.

[Hoffmann *et al.*, 2004] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)*, 22:215–278, 2004.

[Kautz and Selman, 1999] Henry A. Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *International Joint Conference on Artificial Intelligence*, pages 318–325, 1999.

[Keyder and Geffner, 2009] Emil Keyder and Hector Geffner. Trees of shortest paths vs. steiner trees: Understanding and improving delete relaxation heuristics. In *International Joint Conference on Artificial Intelligence*, pages 1734–1739, 2009.

[Keyder *et al.*, 2012] Emil Keyder, Jörg Hoffmann, and Patrik Haslum. Semi-relaxed plan heuristics. In *International Conference on Automated Planning and Scheduling*, pages 128–136, 2012.

[Koehler and Hoffmann, 2000] Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *J. Artif. Intell. Res. (JAIR)*, 12:338–386, 2000.

[Massey, 1999] Bart Massey. *Directions In Planning: Understanding The Flow Of Time In Planning*. PhD thesis, Computational Intelligence Research Laboratory, University of Oregon, 1999.

[Nguyen and Kambhampati, 2001] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *International Joint Conference on Automated Planning*, pages 459–466, 2001.

[Pang and Holte, 2011] Bo Pang and Robert C. Holte. State-set search. In *Symposium on Combinatorial Search*, pages 125–133, 2011.

[Richter and Helmert, 2009] Silvia Richter and Malte Helmert. Preferred operators and deferred evaluation in satisficing planning. In *International Conference on Automated Planning and Scheduling*, pages 273–280, 2009.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)*, 39:127–177, 2010.

[Rintanen, 2010] Jussi Rintanen. Heuristics for planning with SAT. In *Principles and Practice of Constraint Programming (CP)*, pages 414–428, 2010.

[Röger and Helmert, 2010] Gabriele Röger and Malte Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In *International Conference on Automated Planning and Scheduling*, pages 246–249, 2010.

[Vidal and Geffner, 2005] Vincent Vidal and Héctor Geffner. Solving simple planning problems with more inference and no search. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *LNCS*, pages 682–696, Sitges, Spain, October 2005. Springer.