RIFO Revisited: Detecting Relaxed Irrelevance

Jörg Hoffmann and Bernhard Nebel

Institute for Computer Science
Albert Ludwigs University
Georges-Köhler-Allee, Geb. 52
79110 Freiburg, Germany
hoffmann@informatik.uni-freiburg.de

Technical Report No. 153

April 2001

Abstract. RIFO, as has been proposed by Nebel et al. [12], is a method that can automatically detect irrelevant information in planning tasks. The idea is to remove such irrelevant information as a preprocess to planning. While RIFO has been shown to be useful in a number of domains, its main disadvantage is that it is not completeness preserving. Furthermore, the preprocess often takes more running time than nowadays state-of-the-art planners, like FF, need for solving the entire planning task.

We introduce the notion of relaxed irrelevance, concerning actions which are never needed within the relaxation that heuristic planners like FF and HSP use for computing their heuristic values. The idea is to speed up the heuristic functions by reducing the action sets considered within the relaxation. Starting from a sufficient condition for relaxed irrelevance, we introduce two preprocessing methods for filtering action sets. The first preprocessing method is proven to be completeness-preserving, and is empirically shown to terminate fast on most of our testing examples. The second method is fast on all our testing examples, and is empirically safe. Both methods have drastic pruning impacts in some domains, speeding up FF's heuristic function, and in effect the planning process.

1 Introduction

RIFO, as has been proposed by Nebel et al. [12], is a method that can automatically detect irrelevant information in planning tasks. A piece of information can be considered irrelevant if it is not necessary for generating a solution plan. The idea is to remove such irrelevant information as a pre-process in the hope to speed up the planning process. While RIFO has been shown to be useful for speeding up GRAPHPLAN in a number of domains, it does not guarantee that the removed information is really irrelevant. In effect, RIFO is not completeness preserving. Furthermore, the pre-process itself can take a lot of running time. While RIFO can be proven to terminate in polynomial time, it—or at least its implementation within IPP4.0 [11]—is on a lot of planning tasks not competitive with nowadays state-of-the-art planners. In our experiments on a large range of tasks from different domains, we found that in most examples RIFO needs more running time to finish the pre-process than FF needs for solving the entire task.

In this paper, we present a new approach towards defining and detecting irrelevance. We explore the idea of *relaxed irrelevance*, which concerns pieces of information, precisely STRIPS actions, that are not needed within the relaxation that state-of-the-art heuristic planners like FF [9] and HSP [5] use for computing their heuristic values. Those planners evaluate each search state S by estimating the solution length from S under the relaxation that all delete lists are ignored (this idea has first been proposed by Bonet et al. [6]). The main bottleneck in FF and HSP is the heuristic evaluation of states, so it is worthwhile trying to improve on the speed of such evaluations. Our idea is to speed up the heuristic functions by reducing the action sets considered within the relaxation. Actions that are relaxed irrelevant need never be considered. We define the notion of *legal generation paths*, and prove that an action is relaxed irrelevant if it does not start such a path. Deciding about legal generation paths is still NP-hard, so we introduce two approximation techniques. Both can be used as preprocessing methods for filtering the action set to be considered within the relaxation. The first preprocessing method includes all actions that start a legal generation path, and can therefore safely be applied to the relaxation. The pre-process terminates fast on most of our testing examples in the sense that it is orders of magnitude faster than FF. The second approximation method is fast on all our testing examples, and while it is not provably completeness preserving, it is empirically safe: from a large testing suite, no single example task got unsolvable because of the filtering process.

We introduce our theoretical investigations and algorithmic techniques within the STRIPS framework, and summarize how they are extended to deal with conditional effects. Both action filtering methods can in principle be used as a pre-process to either FF or HSP—or rather as a pre-process to any planner that uses the same relaxation—and both methods have drastic pruning impacts in some domains. We have implemented the methods as a pre-process to FF, and show that they significantly speed up FF's heuristic function, and in effect the plan generation process, in those cases where the pruning impact is high.

The next section gives the necessary background in terms of STRIPS notations and heuristic forward state space planning as done by FF and HSP. Section 3 defines and investigates our notions of relaxed irrelevance and legal generation paths. Section 4 explains two ways of approximating legal generation paths, yielding the above described two action filtering methods. Section 5 summarizes how our analysis is extended to ADL domains, and Section 6 describes the experiments we made for evaluating the approach. Section 7 explains two lines of work that we are currently exploring. Section 8 concludes.

2 Background

We introduce our theoretical observations and our algorithms in the STRIPS [8] framework.

Definition 1. A *state* S is a finite set of logical atoms.

We assume that all operator schemata are grounded, i.e., we only talk about *actions*.

Definition 2. A *STRIPS action* o is a triple

$$o = (\mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o))$$

where $\mathrm{pre}(o)$ are the preconditions of o, $\mathrm{add}(o)$ is the add list of o and $\mathrm{del}(o)$ is the delete list of the action, each being a set of atoms. The result of applying a single action to a state is defined as follows:

$$Result(S, \langle o \rangle) = \begin{cases} (S \cup \mathrm{add}(o)) \setminus \mathrm{del}(o) & \mathrm{pre}(o) \subseteq S \\ \text{undefined} & \text{otherwise} \end{cases}$$

The result of applying a sequence of more than one action to a state is recursively defined as

$$Result(S, \langle o_1, \ldots, o_n \rangle) = Result(Result(S, \langle o_1, \ldots, o_{n-1} \rangle), \langle o_n \rangle).$$

Definition 3. A *planning task* $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ is a triple where $\mathcal{O}$ is the set of actions, and $\mathcal{I}$ (the initial state) and $\mathcal{G}$ (the goals) are sets of atoms.

Plans are simple sequences of actions throughout the paper, i.e., we do not consider parallelism. FF and HSP do, in their current versions, not allow for any concurrency between action applications.

Definition 4. Given a planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$. A *plan* or *solution* is a sequence $P = \langle o_1, \ldots, o_n \rangle$ of actions in $\mathcal{O}$ that solves the task, i.e., for which $\mathcal{G} \subseteq Result(\mathcal{I}, P)$ holds.

A plan $P = \langle o_1, \ldots, o_n \rangle$ is called *minimal*, if no single action can be left out of the sequence without loosing the solution property, i.e., if $\langle o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n \rangle$ is not a solution for any o_i. The *length* of a plan is the number of actions in the sequence. A plan for a task $\mathcal{P}$ is *optimal* if it has minimal length among all plans for $\mathcal{P}$. Obviously, optimal plans are minimal. We will need those notations when deriving our sufficient condition for relaxed irrelevance.

FF is based on the general principle of heuristic forward state space search, as has first been implemented in HSP1.0 [3]. The idea is to search in the space of states that are reachable from the initial state, trying to minimize a heuristic value that is computed to each considered state. The heuristic evaluation in both FF and HSP is based on the following relaxation.

Definition 5. Given a planning task $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$. The *relaxation* $\mathcal{P}'$ of $\mathcal{P}$ is defined as $\mathcal{P}' = (\mathcal{O}', \mathcal{I}, \mathcal{G})$, with

$$\mathcal{O}' = \{(pre(o), \mathrm{add}(o), \emptyset) \mid (\mathrm{pre}(o), \mathrm{add}(o), \mathrm{del}(o)) \in \mathcal{O}\}$$

In words, a planning task is relaxed by ignoring all delete lists. When either FF or HSP face a search state S, they estimate the length of a relaxed solution starting in S, i.e., they estimate the solution length of the task $(\mathcal{O}', S, \mathcal{G})$. In HSP, this is done by computing certain weight values for all facts, where the weight of a fact is an estimate of how difficult it is to achieve that fact from S. Computing these weight values involves a fixpoint computation that iteratively applies all actions until no more changes occur [6]. In FF, the solution length to $(\mathcal{O}', S, \mathcal{G})$ is estimated by extracting an explicit solution in a GRAPHPLAN-style manner [2,9]. The technique is based on building a relaxed version of GRAPHPLAN's planning graph, which involves, like HSP's method, repeated application of all actions.

The main bottleneck in HSP, i.e., the main source of running time consumed, is the heuristic evaluation of states [4]. The same applies to FF. While heuristic evaluation is implemented efficiently in both systems, usually no more than a few hundred state evaluations can take place in a second (for FF, Section 6 provides averaged running times per state evaluation on a large range of domains). In some huge planning tasks, we have observed that a single evaluation in FF can take up to half a second running time. This is due to the large number of actions that there are in instantiated planning tasks. With ten-thousands of actions to be considered, FF's process of building a relaxed planning graph, and HSP's process of computing a weight fixpoint, must be costly no matter how efficient the implementation is. Our idea, consequently, is to reduce the number of actions that the planners need to consider within the relaxation, i.e., to compute as a pre-process a set $\mathcal{O}|_r$ of actions that are considered relevant for the relaxation. During search, one can then estimate solution lengths to the tasks $(\mathcal{O}|'_r, S, \mathcal{G})$ as opposed to using the whole action set in the tasks $(\mathcal{O}', S, \mathcal{G})$.

Of course, the set $\mathcal{O}|_r$ can not be chosen arbitrarily small. If important actions are missed out, then the task $(\mathcal{O}|'_r, S, \mathcal{G})$ can become unsolvable for a state S though it would be solvable with the original action set. In other words, one runs the risk of loosing relaxed completeness. If the task $(\mathcal{O}|'_r, S, \mathcal{G})$ is unsolvable, which both HSP's and FF's algorithmic methods will detect, then the systems set the heuristic value of S to ∞, excluding the state from the search space. While this is normally justified—if a state can not be solved even when ignoring delete lists, then that state is unsolvable—it can lead to incompleteness if solving $(\mathcal{O}|'_r, S, \mathcal{G})$ only failed because $\mathcal{O}|_r$ does not contain some important action(s).[1] The rest of the paper is inspired by the observation that it is possible to define a notion of relevance that maintains relaxed completeness.

3 Relaxed Irrelevance

We consider an action relaxed irrelevant if it never appears in an optimal relaxed solution. Clearly, such actions can be ignored within the relaxation without loosing completeness.

Definition 6. Let $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task. An action $o \in \mathcal{O}$ is *relaxed irrelevant* if o is not part of any optimal relaxed solution from any reachable state.

One might be tempted to consider an action irrelevant already when to all reachable states there is *at least one* optimal relaxed solution without that action. While this is adequate for a single action, it would not allow us to remove more than one action: Consider the case where two actions can be used alternatively for solving a state. Each single one can be replaced by the other one, but removing both renders the state unsolvable. This can not happen with the above definition of relaxed irrelevance, where both actions must be entirely useless for being removed. Deciding about relaxed irrelevance is PSPACE-hard.

Definition 7. Let RELAXED-IRRELEVANCE denote the following problem:
Given a planning task $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ and an action $o \in \mathcal{O}$, is o relaxed irrelevant?

Theorem 8. *Deciding RELAXED-IRRELEVANCE is PSPACE-hard.*

Proof: We describe a polynomial reduction of (the complement of) PLANSAT—the decision problem of whether there exists a solution plan for a given arbitrary STRIPS planning instance [7]—to RELAXED-IRRELEVANCE. Given a task

[1] One might argue that this could be fixed by setting the heuristic value of S to a large integer instead of ∞. While this would regain completeness, it would also make the adequacy of the heuristic questionable: If a large number of states have the same high heuristic evaluation only because $\mathcal{O}|_r$ is too restrictive, then the heuristic is not very informative about the real structure of the search space.

$(\mathcal{O}, \mathcal{I}, \mathcal{G})$ and o, using the logical atoms F, first rename all atoms, yielding the set F' and the equivalent task $(\mathcal{O}', \mathcal{I}', \mathcal{G}')$. Then add the original action o to $\mathcal{O}'$, as well as the following actions:

$$\begin{array}{lcccc} \text{name} & & (\text{pre}, & \text{add}, & \text{del}) \\ o_1 & = & (\mathcal{G}', & \text{pre}(o), & F'), \\ o_2 & = & (\text{add}(o), & \mathcal{G}', & \emptyset) \end{array}$$

In the modified task, o is part of an optimal relaxed solution from and only from those states where o_1 has just been applied. As the precondition of o_1 requires the original goal to be solved, o is relaxed irrelevant in the modified task if and only if the original task is unsolvable. ■

3.1 A Sufficient Condition

As an exact decision about relaxed irrelevance is as hard as planning itself, we now derive a sufficient condition. The following definition forms the heart of the subsequent investigation.

Definition 9. Let $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task. The *generation graph* to the task is defined by the node set $\mathcal{O} \cup \{o_G\}$, with $o_G := (\mathcal{G}, \emptyset, \emptyset)$, and the edge set

$$\{(o, o') \mid \text{add}(o) \cap \text{pre}(o') \neq \emptyset\}$$

We refer to paths $P = \langle o_1, \ldots, o_n = o_G \rangle$ in this graph as *generation paths*. We call $\text{add}(o_i) \cap \text{pre}(o_{i+1})$ the *connecting facts* at position i. P is *legal* if at each position there is at least one connecting fact that is not contained in the preconditions of the previous actions, i.e., if for $1 \leq i \leq n-1$:

$$(\text{add}(o_i) \cap \text{pre}(o_{i+1})) \setminus \bigcup_{1 \leq j \leq i} \text{pre}(o_j) \neq \emptyset$$

The generation graph to a task intuitively represents all ways in which facts can be achieved. A generation path is a sequence of actions that support each other, and that end up making at least one goal true. We will see in the following that the only generation paths that are adequate in minimal relaxed solutions are those generation paths that are legal. Precisely, we will show the following.

Theorem 10. *Let $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task, S a state, and $P = \langle o_1, \ldots, o_n \rangle$ a minimal relaxed solution to S. Then for all o_i there exists a legal generation path P_i starting with o_i.*

With that, we immedeately have our sufficient condition.

Corollary 11. *Let $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task, $o \in \mathcal{O}$. If there is no legal generation path P starting with o, then o is not part of any minimal relaxed solution from any state. In particular, o is then relaxed irrelevant.*

Semantically, Definition 9 can be seen as a modification of the base technique that is used in RIFO. The relation between the techniques gives a nice picture of what is happening. Briefly, it can be explained as follows. To create an expectation of what is relevant for solving a planning task, RIFO builds a so-called fact-generation tree. This is an AND- OR-tree that is built by backchaining from the goals. The root node is an AND-node corresponding to the goals. Other AND-nodes correspond to an action's preconditions, and the OR-nodes are single atoms that can alternatively be achieved by different actions. Once this tree is generated, RIFO applies a number of simple heuristics to select the information from the tree that is likely most relevant. Now, the set of all legal generation paths can be viewed as a more restrictive version of RIFO's fact-generation tree, where an action is only allowed to achieve an OR-node if the intersection of the action's precondition with the facts on the path from the OR-node to the tree root is empty. This is adequate (only) for relaxed planning. While RIFO selects fractions of its tree as relevant, we select the whole tree. This gives us completeness in the relaxation. The proof to Theorem 10 proceeds using what we call the *needed facts*, which are the facts for whose achievement actions can be placed at a certain position in a relaxed solution.

Definition 12. Let $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task, S a state, and $P = \langle o_1, \dots, o_n \rangle$ a relaxed solution to S. The *true facts* $TF(P, i)$ *of* P *at position* i are

$$TF(P, i) := S \cup \bigcup_{1 \leq j < i} \text{add}(o_j),$$

the *open facts* $OF(P, i)$ *of* P *at position* i are

$$OF(P, i) := (\mathcal{G} \setminus \bigcup_{i < j \leq n} \text{add}(o_j)) \cup \bigcup_{i < j \leq n} (\text{pre}(o_j) \setminus \bigcup_{i < k < j} \text{add}(o_k)),$$

and the *needed facts* $NF(P, i)$ *of* P *at position* i are

$$NF(P, i) := OF(P, i) \setminus TF(P, i)$$

An action placed at position i in a relaxed plan P must add all needed facts of P at position i, and in a minimal relaxed plan there is at least one needed fact at each position. For showing this, we first need a simple characterization of relaxed plans.

Proposition 13. *Let* $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ *be a planning task,* S *a state, and* $P = \langle o_1, \dots, o_n \rangle$ *an action sequence.* P *is a relaxed solution to* S *if and only if* $pre(o_i) \subseteq S \cup \bigcup_{1 \leq j < i} add(o_j)$ *for all* i, *and* $\mathcal{G} \subseteq S \cup \bigcup_{1 \leq i \leq n} add(o_i)$.

If an action in a relaxed plan does not add a fact that is needed at the action's position, then we get a simple contradiction to Proposition 13.

Lemma 14. *Let* $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ *be a planning task,* S *a state, and* $P = \langle o_1, \dots, o_n \rangle$ *a relaxed solution to* S. *Then* $add(o_i) \supseteq NF(P, i)$ *holds for* $1 \leq i \leq n$.

Proof: Assume that $f \in NF(P,i), f \notin \text{add}(o_i)$ for some fact f and position i. In particular, $f \in OF(P,i)$. First case, f is a goal: $f \in \mathcal{G} \setminus \bigcup_{i<j\leq n} \text{add}(o_j)$. As $f \notin TF(P,i)$ and $f \notin \text{add}(o_i)$, $f \notin S \cup \bigcup_{1\leq j\leq n} \text{add}(o_j)$ follows. With $f \in \mathcal{G}$ this is a contradiction to Proposition 13. Second case, f is an open precondition of an action ahead: $f \in \text{pre}(o_j) \setminus \bigcup_{i<k<j} \text{add}(o_k)$ for some $j > i$. As $f \notin TF(P,i)$ and $f \notin \text{add}(o_i)$, we have $f \notin S \cup \bigcup_{1\leq k<j} \text{add}(o_k)$. Again, this is a contradiction to Proposition 13. ■

If no facts need to be achieved at some position in a relaxed plan, then it can easily be shown that the action at the respective position can be left out of the plan. Therefore, the needed facts are non-empty in minimal relaxed solutions.

Lemma 15. *Let $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task, S a state, and $P = \langle o_1, \ldots, o_n \rangle$ a minimal relaxed solution to S. Then $NF(P,i) \neq \emptyset$ holds for $1 \leq i \leq n$.*

Proof: Say $NF(P,i) = \emptyset$ holds for some i. We show that then, $\langle o_1, \ldots, o_{i-1}, o_{i+1}, \ldots, o_n \rangle$ is a relaxed solution from S. With Proposition 13, it suffices to show that

a) $\mathcal{G} \subseteq S \cup \bigcup_{j\neq i} \text{add}(o_j)$
b) $\text{pre}(o_j) \subseteq S \cup \bigcup_{i\neq k<j} \text{add}(o_k)$ for $j \neq i$

As for a), $NF(P,i) = \emptyset$ is $OF(P,i) \setminus TF(P,i) = \emptyset$, which in particular gives us $\mathcal{G} \setminus \bigcup_{i<j\leq n} \text{add}(o_j) \setminus \bigcup_{1\leq j<i} \text{add}(o_j) \setminus S = \emptyset$, finishing the argument. As for b), when $j < i$ there is nothing to show because P is a relaxed solution. When $j > i$, $OF(P,i) \setminus TF(P,i) = \emptyset$ gives us $\text{pre}(o_j) \setminus \bigcup_{i<k<j} \text{add}(o_k) \setminus \bigcup_{1\leq k<i} \text{add}(o_k) \setminus S = \emptyset$, and we are again finished. ■

We can now prove Theorem 10, stating that to all actions o_i in a minimal relaxed solution $P = \langle o_1, \ldots, o_n \rangle$ there is a legal generation path P_i starting with o_i.

Proof: (**to Theorem 10**) For any o_i in P, we construct the desired path P_i by starting with o_i, and successively stepping onto a successor action via the needed facts at the respective position. The process stops when we can reach a goal fact. Precisely, P_i can be constructed by the following algorithm.

1) Let $P_i := \langle o_i \rangle$, $index := i$
2) Loop
   2a) Pick some $f \in NF(P, index)$
   2b) If $f \in \mathcal{G}$, then break the loop
   2c) Pick some $j > index$ such that $f \in \text{pre}(o_j)$, and let $P_i := P_i \circ \langle o_j \rangle$, $index := j$

In step 2a), $NF(P, index) \neq \emptyset$ holds with Lemma 15. As $f \in OF(P, index)$, it must either be a goal $f \in G$, or a precondition $f \in \text{pre}(o_j)$ for some $o_j, j > i$. The index increases monotonically, so the algorithm terminates.

With Lemma 14, $f \in \mathrm{add}(o_{index})$ holds in step 2a), so the constructed path fulfills the requirements for a generation path. It remains to show that P_i is legal, i.e., that at all positions in P_i there is at least one connecting fact not contained in the preconditions of the previous actions. The facts f picked by the above algorithm fulfill this property. We do not spell out the simple proof in detail. The argument is the following. Consider some iteration, the fact $f \in NF(P, index)$ being picked. By construction, all actions that are added to P_i before that iteration are members o_j of P with $j < index$. As P is a relaxed solution, all those action's preconditions are contained in $TF(P, index)$. But $f \notin TF(P, index)$, so f is not part of the precondition of any of those actions. ∎

3.2 Complexity

Unfortunately, deciding about the sufficient condition given by Corollary 11 is still NP-hard.

Definition 16. Let LEGAL-GENERATION-PATH denote the following problem:

Given a planning task $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ and an action $o \in \mathcal{O}$, is there a legal generation path starting with o?

Theorem 17. *Deciding LEGAL-GENERATION-PATH is NP-complete.*

Proof: Membership follows by a simple guess-and-check argument: In a legal generation path, each action o can occur at most once—the connecting facts to the second occurence are a subset of o's preconditions, so subtracting those preconditions yields the empty set. Therefore, we can non-deterministically guess an action sequence of at most length $|\mathcal{O}|$, and verify whether that sequence is a legal generation path. The verification can obviously be made in polynomial time.

We prove hardness by a polynomial reduction from 3SAT. Say we have an instance with variable set $\{v_1, \ldots, v_n\}$ and clause set $\{\{l_{11}, l_{12}, l_{13}\}, \ldots, \{l_{m1}, l_{m2}, l_{m3}\}\}$, where $l_{ij} \in \{v_1, \neg v_1, \ldots, v_n, \neg v_n\}$. The action set to our planning task is $\mathcal{O} := \{s, l_{11}, \ldots, l_{m3}, v_1, \ldots, v_n\}$, i.e., we have one starting action s, and identify actions with all literals in the clauses, and all variables. The task makes use of the logical atoms $\{S, C_1, \ldots, C_m, V_1, NOT\text{-}V_1, \ldots, V_n, NOT\text{-}V_n\}$. We set $\mathrm{pre}(s) := \emptyset, \mathrm{add}(s) := \{S\}$. For $1 \leq j \leq 3$, we initialize $\mathrm{pre}(l_{1j}) := \{S\}$ and for $i \geq 2$ $\mathrm{pre}(l_{ij}) := \{C_{i-1}\}$. Then,

$$\mathrm{pre}(l_{ij}) := \mathrm{pre}(l_{ij}) \cup \begin{cases} V_k & l_{ij} = v_k \\ NOT\text{-}V_k & l_{ij} = \neg v_k \end{cases}$$

Further, $\mathrm{add}(l_{ij}) := \{C_i\}$ for all literal actions. We define $\mathrm{pre}(v_1) := \{C_m\}, \mathrm{add}(v_1) := \{V_1, NOT\text{-}V_1\}$, and for $i \geq 2$ $\mathrm{pre}(v_i) := \{V_{i-1}, NOT\text{-}V_{i-1}\}, \mathrm{add}(v_i) := \{V_i, NOT\text{-}V_i\}$. Finally, we set $\mathcal{G} := \{V_n, NOT\text{-}V_n\}$.

There is a legal generation path starting with s if and only if the 3SAT instance is satisfiable. Figure 1 shows the generation graph to the specified planning task. There are edges between the literals from adjacent clauses—connected by the C_i facts—and edges between adjacent variables—connected by the V_i and $NOT\text{-}V_i$ facts. There are also edges that are not important for the argumentation below, namely those from variable vertices back to literal vertices—the variable actions add V_i and $NOT\text{-}V_i$ facts, which corresponding literals have as preconditions. Those edges are indicated by the dashed arrows in Figure 1.

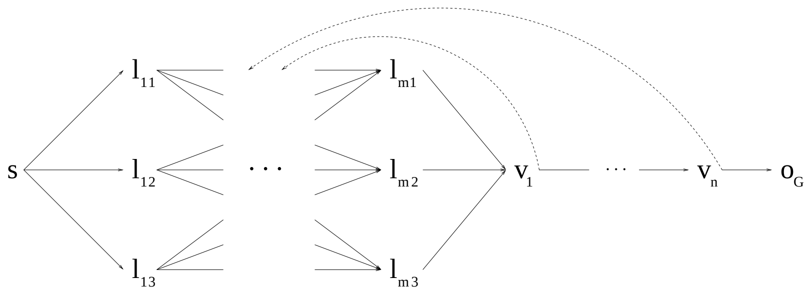

Fig. 1. Generation graph to a 3SAT encoding.

Any path from s to o_G visits each clause at least once, and then passess through all variables. The literals visited in the clauses correspond to truth value assignments. Passing a variable vertex legally checks that this variable has not been assigned contradictory values. More formally, any legal path from s to o_G defines a satisfying truth assignment and vice versa. For the first direction, given a path P define $\pi : \{v_1, \ldots, v_n\} \mapsto \{\top, \bot\}$ by

$$\pi(v_i) := \begin{cases} \top \text{ P visits } l_{ij} = v_i \\ \bot \text{ P visits } l_{ij} = \neg v_i \end{cases}$$

The function obviously satisfies at least one literal per clause. It is well defined due to the preconditions of the literal actions. If P visits a literal node $l_{ij} = v_i$ and a literal node $l_{i'j'} = \neg v_i$, then it can not pass legally through the variable vertex v_i: the connecting facts there are $\{v_i, NOT\text{-}v_i\}$, the first of which is in $\mathrm{pre}(l_{ij})$ and the other of which is in $\mathrm{pre}(l_{i'j'})$.

For the other direction, given a satisfying assignment π, start from s by stepping on to one satisfied literal in each clause. The path can legally pass through the variable vertices because π is well defined. ∎

We remark that the planning task constructed in the proof to Theorem 17 can easily be modified such that—as is the case in many of the current planning benchmark tasks—each action adds at most one precondition of another action, $|\mathrm{add}(o) \cap \mathrm{pre}(o') \leq 1|$ for $o, o' \in \mathcal{O}$.

4 Approximation Techniques

We will now introduce two polynomial-time approximations of legal generation paths, filtering action sets for relaxed planning. The first method includes all actions that start a legal path, and is therefore complete in the relaxation. As we will see in the next section, the method terminates fast in almost all of our testing examples. The second method does not give any completeness guarantees, but will be shown to be empirically safe, and to terminate extremely fast on *all* examples in our testing suite.

4.1 A Sufficient Approximation

Let us first introduce a notation for the set of all actions that start a legal generation path. With Corollary 11, we can restrict the actions considered by an FF or HSP style heuristic function to that set without loosing completeness.

Definition 18. Let $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task. The *legal action set* to $\mathcal{P}$ is $\mathcal{O}|_l := \{o \in \mathcal{O} \mid \exists\, P \in \mathcal{O}^* : \langle o \rangle \circ P \text{ is a legal generation path } \}$.

Our sufficient approximation collects together all actions starting generation paths that fulfill a weaker notion of legality. Reconsider Definition 9.

Definition 19. Let $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task. A generation path $P = \langle o_1, \ldots, o_n \rangle$ is *initially legal* if $(\text{add}(o_i) \cap \text{pre}(o_{i+1})) \setminus \text{pre}(o_1) \neq \emptyset$ for $1 \leq i \leq n-1$. The *initially legal action set* $\mathcal{O}|_{il}$ to $\mathcal{P}$ is defined using the following fixpoint operator $\Gamma : 2^{\mathcal{O}} \mapsto 2^{\mathcal{O}}$.

$$\Gamma(\mathcal{O}|_r) := \{o \in \mathcal{O} \mid \exists\, P \in \mathcal{O}|_r^* : \langle o \rangle \circ P \text{ is an initially legal generation path } \}$$

We set $\mathcal{O}|_{il} := \bigcup_{i=0}^{\infty} \Gamma^i(\emptyset)$.

In words, we obtain the initially legal action set by computing a fixpoint over the actions that start an initially legal generation path. A generation path is initially legal when between any two actions there is a connecting fact that is not contained in the precondition of the first action. Clearly, legal generation paths—where there are connecting facts that are not contained in the precondition of *any* previous action—fulfill this property.

Proposition 20. *Let $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ be a planning task. The initially legal action set is a superset of the legal action set, i.e., $\mathcal{O}|_{il} \supseteq \mathcal{O}|_l$ holds.*

The definition of $\mathcal{O}|_{il}$ translates directly into the fixpoint computation depicted in Figure 2.

Our implementation of the algorithm in Figure 2 is straightforward. In each iteration of the fixpoint process, check for all not yet selected actions o whether

```
𝒪|il := ∅
repeat
    Fixpoint := TRUE
    for o ∈ 𝒪 \ 𝒪|il do
        if there is an initially legal path from o to oG
                    consisting out of actions in 𝒪|il then
          𝒪|il := 𝒪|il ∪ {o}
          Fixpoint := FALSE
        endif
    endfor
until Fixpoint
```

Fig. 2. Fixpoint computation of actions starting initially legal generation paths: A sufficient approximation of legal generation paths.

there is a path to the goals, using only edges that are not excluded by o's preconditions.

We have also implemented two other sufficient approximations of $\mathcal{O}|_l$. One of those weakens $\mathcal{O}|_{il}$ by dropping the condition that the action sequences P must consist of $\mathcal{O}|_{il}$ members. The other method strengthens $\mathcal{O}|_{il}$ by incrementally building a graph of edges that start already selected paths. The required action sequences P must then traverse only edges that are in the graph already. In our experiments, both methods showed significantly worse runtime behavior than the above $\mathcal{O}|_{il}$ computation. The filtered action sets were, however, the same for all three methods in most of the cases. We therefore chose to concentrate on $\mathcal{O}|_{il}$ as a sufficient approximation.

4.2 An Insufficient but Fast Approximation

The approximation that we introduce now does not theoretically include all actions from $\mathcal{O}|_l$. The method is therefore not completeness-preserving in general. It has, however, proven to be empirically safe in our experiments. Moreover, the method terminated extremely fast in all our testing examples. Like the computation of initially legal paths, the method performs a fixpoint computation. Unlike the former computation, the method allows only edges (o, o') in the paths that are legal with respect to o. What's more, each action o is associated with at most one single edge that can be traversed from o. We call the resulting action set the set of *approximative legal* actions $\mathcal{O}|_{al}$. Have a look at the pseudo code in Figure 3.

The algorithm depicted in Figure 3 iteratively includes new actions into $\mathcal{O}|_{al}$ until a fixpoint is reached. The key feature of the algorithm is the function $e : \mathcal{O} \mapsto \mathcal{O}$, which is represented in the figure as a set of $(o, e(o))$ pairs. The function starts as the empty set of such pairs, i.e., e is initially undefined for the whole action set. If an action o is included into $\mathcal{O}|_{al}$ due to an edge (o, o'), then that edge is included into the definition of e. Initially, the only member of $\mathcal{O}|_{al}$ is o_G, so in iteration

$\mathcal{O}|_{al} := \{o_G\}$, $e := \emptyset$, $k := 0$
repeat
  Fixpoint := TRUE
  for $o \in \mathcal{O} \setminus \mathcal{O}|_{al}$ **do**
    if there is an edge (o, o'), $o' \in \mathcal{O}|_{al}$ such that
        the path $\langle o, e^0(o'), e^1(o'), \dots, e^k(o') = o_G \rangle$ is initially legal **then**
      $\mathcal{O}|_{al} := \mathcal{O}|_{al} \cup \{o\}$
      $e := e \cup \{(o, o')\}$
      Fixpoint := FALSE
    endif
  endfor
  $k := k + 1$
until Fixpoint

Fig. 3. Fixpoint computation of actions starting approximative legal generation paths: An insufficient but fast approximation of legal generation paths.

$k = 0$ the only edges that can be included are direct connections to the goals. In any later iteration k, e defines a tree of depth k where the root node is o_G, and each node—the actions for which e is defined—occurs exactly once. For the not yet selected actions o it is then checked whether they have an edge connecting them to a tree node o' such that the path $\langle o, e^0(o'), e^1(o'), \dots, e^k(o') = o_G \rangle$ is initially legal. Note here that $\langle o, e^0(o'), e^1(o'), \dots, e^k(o') \rangle$ is just the concatenation of the edge (o, o') with the path from o' to the tree root. If that path is initially legal, then o and the edge (o, o') are included into the tree.[2] While allowing only a single edge for each node may sound way to restrictive, the method turned out to be, as said, surprisingly safe in our testing examples.

5 Extension to Conditional Effects

We have extended our theoretical analysis and approximation algorithms to deal with conditional effects. Because FF compiles away all ADL constructs except the conditional effects [10,9], this enabled us to deal with planning domains specified in the ADL laguage [13], precisely in the respective sublanguage of PDDL as was used in the AIPS-2000 competition [1]. In the following, we summarize the extensions made to the definitions, proofs, and algorithms introduced in Sections 3 and 4.

A propositional action with conditional effects is a construct $o = (\mathrm{pre}(o), \Phi(o))$ where $\Phi(o)$ are the effects of o. Each single effect $\phi \in \Phi(o)$ has the form $\phi = (\mathrm{con}, \mathrm{add}, \mathrm{del})$ where con are the effect conditions. A STRIPS action in this framework has a singleton set of effects, the effect condition being empty, i.e., $o = (\mathrm{pre}(o), \{(\emptyset, \mathrm{add}(o), \mathrm{del}(o))\})$.

[2] For optimization, one obviously only needs to look at actions o' that are leafs of the current tree.

We say that an effect ϕ of an action o_i in a relaxed solution $P = \langle o_1, \ldots, o_n \rangle$ *can be ignored in* P, if P without ϕ is still a relaxed solution, i.e., if

$$\langle o_1, \ldots, o_{i-1}, (\mathrm{pre}(o_i), \Phi(o_i) \setminus \phi), o_{i+1}, \ldots, o_n \rangle$$

is a relaxed solution. Recall Definition 6. We call an effect ϕ relaxed irrelevant if it can be ignored in all optimal relaxed solutions from all reachable states. The parallelity to the previous definition is that if the single effect of a STRIPS action can always be ignored, then the whole action can always be thrown out, such that the action is never part of an optimal relaxed solution. As STRIPS is a special case of conditional effects, deciding about the extended notion of relaxed irrelevance is of course also PSPACE-hard.

We extend our investigation of legal generation paths by looking at the set of all effects in a task as a set of STRIPS actions: for an action set $\mathcal{O}$ with conditional effects, look at the set $\mathrm{STRIPS}(\mathcal{O}) := \{(\mathrm{pre}(o) \cup \mathrm{con}, \mathrm{add}, \mathrm{del}) \mid \phi = (\mathrm{con}, \mathrm{add}, \mathrm{del}) \in \Phi(o) \text{ for some } o \in \mathcal{O}\}$. The key observation paralellizing Theorem 10 is the following. If an effect can not be ignored in a minimal relaxed solution from some state, then the effect starts a legal generation path in $\mathrm{STRIPS}(\mathcal{O})$. This can be proven by a natural extension of the needed facts notion. Such facts must be made TRUE by an action. In a minimal solution, there is at least one such fact at each position. If an action does no longer achieve the respective needed facts when its effect ϕ is ignored, then ϕ adds at least one needed fact. This fact is either a goal or the pre- or effect-condition of an important effect ϕ' ahead. Similar to the proof of Theorem 10, a legal generation path in $\mathrm{STRIPS}(\mathcal{O})$ can be created by stopping when a goal is reached, and moving on to ϕ' in the other case.

The parallel statement to Corollary 11 is, obviously, if an effect does not start a legal generation path in $\mathrm{STRIPS}(\mathcal{O})$, then the effect can be ignored in all minimal relaxed solutions, which in particular means that the effect is relaxed irrelevant. Extending the filtering methods from Section 4 comes down to implementing them on the set $\mathrm{STRIPS}(\mathcal{O})$. If an effect does not start an initially or approximative legal generation path in $\mathrm{STRIPS}(\mathcal{O})$, then the effect is removed from the respective action in the sense that the effect is not considered within the relaxation. If all effects of an action are removed, then the whole action is ignored. Remember that STRIPS actions have singleton effect sets, so in that special case the more general techniques simplify exactly to what we described in the previous sections.

6 Empirical Evaluation

We evaluated our approach by running a number of large scale experiments. We used 20 benchmark planning domains, including all examples from the AIPS-1998 and AIPS-2000 competitions. The domains were *Assembly*, two *Blocksworlds* (three- and four-operator representation), *Briefcaseworld*, *Bulldozer*, *Freecell*, *Fridge*, *Grid*, *Gripper*, *Hanoi*, *Logistics*, *Miconic*-ADL, *Miconic*-SIMPLE, *Miconic*-STRIPS, *Movie*, *Mprime*, *Mystery*, *Schedule*, *Tireworld*, and *Tsp*. In each of these domains, we generated instances by using randomized generation software.[3] We ran experiments for evaluating

1. RIFO's runtime behavior when compared to FF,
2. the runtime behavior and pruning impact of $\mathcal{O}|_{il}$ and $\mathcal{O}|_{al}$,
3. and the empirical safety of $\mathcal{O}|_{al}$.

For each single experiment, we set up a large testing suite containing up to 200 instances from each domain. The testing suites differed in terms of the size of the instances that we generated.

In the first experiment, we ran the RIFO implementation within IPP4.0 versus FF on a suite of 681 instances that were small enough for the IPP4.0 instantiation routine to cope with.[4] Test runs were given 300 seconds time and 400 MBytes memory on a Sun machine running at 163 MHz. We show the number of instances handled successfully, and the average running time per domain. For FF, we count as successfully handled those instances were a plan was found. For RIFO, success on an instance means termination of the pre-process within the given time and memory bounds. We count only those such instances for which we know they are solvable—those were FF found a plan. Times are averaged over those instances that both implementations handled successfully. Running time for RIFO does *not* include IPP's instantiation time. See the data in Table 4.

In 3 of the 20 domains shown (*Assembly*, *Bulldozer* and *Hanoi*) does RIFO terminate faster than FF solves the tasks. In 10 domains, RIFO's average running time is orders of magnitude higher than that of FF. In some domains, RIFO exhausts resources on a number of instances that FF manages to solve. We conclude that RIFO is, as a pre-process, not competitive with FF, at least in its implementation within IPP4.0. In any case, we demonstrate in the following that (the implementations of) our new action filtering methods behave much better in terms of running time consumed.

In our second experiment, we evaluated the $\mathcal{O}|_{il}$ and $\mathcal{O}|_{al}$ methods in terms of runtime behavior and pruning impact. Test runs were given 300 seconds and 200 MBytes memory on a Sun machine running at 300 MHz. We used a total of 2334 large instances generated to be of a size challenging for FF, but still within its range of solvability within the given resources. On each task, we ran three implementations: FF-v2.2 [9], and two versions of the same code were $\mathcal{O}|_{il}$ respectively $\mathcal{O}|_{al}$ were computed as a pre-process. In the latter two versions, FF's heuristic function was changed to consider only those effects contained in the

[3] Descriptions of the randomization strategies and the source code of all generators are publicly available at http://www.informatik.uni-freiburg.de/~ hoffmann/ff-domains.html.

[4] In some domains, like *Freecell*, the routine can handle only comparatively small instances which is, we think, due to the implementation: this is intended to deal with full scale ADL constructs [10], and fails to efficiently handle the simple STRIPS special case.

domain	success RIFO	success FF	running time RIFO	running time FF
Assembly	33	33	1.08	9.16
Blocksworld-3ops	21	21	4.45	2.90
Blocksworld-4ops	21	21	0.91	0.07
Briefcaseworld	20	20	1.86	1.12
Bulldozer	17	17	1.97	4.54
Freecell	33	50	21.90	0.06
Fridge	22	22	0.23	0.22
Grid	22	35	43.77	7.72
Gripper	25	25	0.45	0.31
Hanoi	8	8	0.34	4.79
Logistics	35	35	46.80	1.18
Miconic-ADL	22	40	14.03	3.77
Miconic-SIMPLE	25	25	0.64	0.54
Miconic-STRIPS	25	25	0.64	0.37
Movie	30	30	0.00	0.00
Mprime	48	61	16.47	1.19
Mystery	23	36	27.16	12.51
Schedule	15	28	28.34	14.06
Tireworld	20	20	6.03	0.48
Tsp	25	25	4.90	0.12

Fig. 4. Instances handled successfully, and average running times for RIFO and FF per domain. The successfully handled instances for FF are those for which a plan was found. The successfully handled instances for RIFO are those solvable ones where RIFO terminated within the given time and memory bounds.

filtered action set. We measured the overhead produced by the filtering methods, the total running times, the time taken for state evaluations, and the number of effects in the complete respectively filtered action sets. See the data in Table 5.

All measured values were averaged over those instances were all three methods succeeded in finding a plan (we tried inserting default values in the other cases, but found that this generally obfuscated the results more than it helped understanding them). In 12 domains, the solved instances were exactly the same across all methods anyway. In another 3 domains, differences occured only in very few instances (1 - 2 out of 90 - 181). In *Grid* and *Mprime*, computing $\mathcal{O}|_{il}$ sometimes exhausted resources (in *Grid*, 41 of 179 cases, in *Mprime*, 51 of 196 cases). In *Assembly* and *Logistics*, the speed-up produced by the filtering methods helped FF to solve some more instances (165 instead of 159 in *Assembly*, 87 instead of 75 in *Logistics*). In *Schedule*, original FF solved 85 instances instead of 74 solved with $\mathcal{O}|_{il}$ or $\mathcal{O}|_{al}$ on. We will come back to this later.

Let us first focus on the overheads produced. Compare the first two columns with the third column, showing average solving time for FF. The overhead for $\mathcal{O}|_{il}$ is neglectible (i.e., below 0.2 seconds on average) in 11 of our domains, and orders of magnitude smaller than FF's average time in another 4 domains. In the 3-operator *Blocksworld*, the overhead is a third of FF's time, and below a second

domain	overhead $\mathcal{O}\vert_{il}$	overhead $\mathcal{O}\vert_{al}$	total time FF	total time $+\mathcal{O}\vert_{il}$	total time $+\mathcal{O}\vert_{al}$	single evaluation FF	single evaluation $+\mathcal{O}\vert_{il}$	single evaluation $+\mathcal{O}\vert_{al}$	number of effects $\mathcal{O}$	number of effects $\mathcal{O}\vert_{il}$	number of effects $\mathcal{O}\vert_{al}$
Assembly	0.01	0.00	12.83	12.01	11.53	1.75	1.64	1.57	426.72	358.64	358.64
Blocksworld-3ops	0.59	0.08	1.62	2.24	1.61	3.41	3.47	3.21	1854.62	1854.62	1819.09
Blocksworld-4ops	0.04	0.00	1.04	1.08	0.99	0.76	0.76	0.72	290.06	290.06	286.94
Briefcaseworld	0.04	0.01	5.51	1.10	1.01	4.26	0.82	0.77	4106.50	670.00	670.00
Bulldozer	0.02	0.01	6.89	7.00	6.67	1.27	1.29	1.23	599.22	599.22	599.17
Freecell	11.14	0.53	17.47	28.77	17.33	8.19	8.26	7.87	4725.37	4668.17	4668.17
Fridge	0.00	0.00	1.71	1.72	1.70	0.96	0.97	0.95	302.22	302.22	302.22
Grid	76.12	0.54	11.57	87.90	11.93	7.95	8.09	7.89	6424.35	6424.35	6417.28
Gripper	0.03	0.01	0.33	0.36	0.27	1.38	1.39	1.11	478.00	478.00	359.00
Hanoi	0.00	0.00	4.73	4.80	4.58	0.83	0.84	0.80	244.50	244.50	244.50
Logistics	2.24	0.22	83.57	45.51	43.52	37.45	19.39	19.40	19904.53	15347.80	15347.80
Miconic-ADL	1.09	0.29	13.91	13.48	12.23	12.72	11.33	10.92	2988.20	2700.52	2700.52
Miconic-SIMPLE	0.17	0.02	0.52	0.69	0.51	2.14	2.15	2.04	1504.00	1504.00	1504.00
Miconic-STRIPS	0.16	0.02	0.39	0.55	0.38	1.92	1.94	1.82	1504.00	1504.00	1504.00
Movie	0.00	0.00	0.00	0.00	0.00	0.33	0.23	0.17	7.00	7.00	7.00
Mprime	60.20	0.79	5.40	65.66	6.13	16.38	16.54	16.18	12138.00	12136.97	12136.32
Mystery	12.87	1.05	20.11	33.26	21.14	15.59	15.80	15.57	14644.20	14644.20	14641.38
Schedule	0.48	0.01	52.31	55.56	54.52	10.86	7.07	6.99	3049.84	917.43	916.82
Tireworld	1.34	0.08	23.23	13.27	7.07	19.31	9.92	5.81	7105.50	4479.00	3646.00
Tsp	0.01	0.09	0.13	0.14	0.22	2.09	2.11	2.13	4390.00	4390.00	4390.00

Fig. 5. Average overhead for pre-processing, average total running time, average running time per state evaluation, and average number of effects, shown per planning domain and filtering method used. Times are in seconds except for state evaluations, where milliseconds are specified.

anyway. In the remaining four domains, the pre-process can hurt: In *Freecell* and *Mystery*, it takes almost as much time as FF, and in *Grid* and *Mprime* it can take much longer time (we will later describe an approach to automatic recognition of the cases were the pre-process takes a lot of time). The overhead for $\mathcal{O}|_{al}$ is neglectible in 14 of the domains, and still a lot smaller than FF's running time in the other cases.

Concerning the impact that the filtering methods have on the number of effects in the action set, the speed of the heuristic function, and the total running time, it is easiest to start by looking at the rightmost three columns in Table 5. The methods do not prune any effects in 6 of our domains, and prune very few effects in another 7 domains. Moderately many effects are pruned in the *Assembly*, *Gripper* and *Miconic*-ADL domains. In the *Briefcaseworld*, *Logistics*, *Schedule* and *Tireworld* domains, the pruning is drastic.[5] As a consequence, the average time taken for a single state evaluation (total evaluation time divided by number of evaluated states) is, when using the filtering methods, significantly lower in the four domains with drastic pruning, and slightly lower in the three domains with moderate pruning. Look at the respective columns, specifying the average state

[5] In the *Briefcaseworld*, for example, amongst other things all actions are thrown out that take objects out of the briefcase—taking objects out of the briefcase is not necessary within the relaxation, where keeping them inside never hurts.

evaluation time in milliseconds. In *Briefcaseworld*, *Logistics* and *Tireworld*, the faster heuristic functions translate directly into improved total running time. In *Schedule*, there seems to be some interaction between the filtering methods and FF's internal algorithmic techniques: though the heuristic function is faster, total running time gets worse. This is because FF evaluates, with the filtered action sets, more states before finding the goal. An explanation for this might be FF's helpful actions heuristic, which biases the actions selected to those that could also be selected by the heuristic function [9]. For $\mathcal{O}|_{al}$, it might also be that some states become unsolvable—though we did not find such a case in the experiment described below.

We finally consider the safety of the $\mathcal{O}|_{al}$ filtering method with respect to completeness in the relaxation. The method is empirically safe in the sense that, from the 2334 examples used in the above described experiment, only 11 *Schedule* instances could not be solved with the method on though they could be solved with original FF. The failures were only due to the runtime restrictions we applied in the experiment: given slightly more time, FF with $\mathcal{O}|_{al}$ filtering *could* solve those 11 instances. In addition to this result, we ran the following experiment. We generated a total of 2099 instances from our 20 domains, small enough to build an explicit representation of the state space. To each instance, we looked at all reachable states, and verified whether the goal was reachable when ignoring delete lists, using the whole action set $\mathcal{O}$, or the filtered action set $\mathcal{O}|_{al}$. In 19 of our 20 domains, all states solvable with $\mathcal{O}$ were still solvable with $\mathcal{O}|_{al}$. Only in *Grid* did we find states that became unsolvable. This occured in 19 of 100 instances. In all those instances, the states becoming unsolvable were less than 1% of the state space.

7 Current Work

Our current results reveal two drawbacks of the presented approach:

1. $\mathcal{O}|_{il}$ filtering sometimes hurts in the sense that it can take a lot of running time.
2. While $\mathcal{O}|_{il}$ is provably and $\mathcal{O}|_{al}$ empirically safe, both methods have strong pruning impacts only in a few domains.

We address these difficulties in two lines of work that we are currently pursuing. One idea to avoid the first problem is to calculate an estimation value for the runtime that would be necessary for computing $\mathcal{O}|_{il}$. One can then simply skip the pre-process if it appears to be too costly. Reconsider the algorithm depicted in Figure 2. The running time is consumed by the repeated search for legal generation paths. Finding a path in a graph is of course more costly the more edges there are in the graph. Now, an obvious upper limit to the number of edges in a generation graph is the following:

$$\sum_{f \in F} |\{o \in \mathcal{O} \mid f \in \text{add}(o)\}| * |\{o \in \mathcal{O} \mid f \in \text{pre}(o)\}|$$

Here, F denotes the set of all logical atoms that appear in the actions $\mathcal{O}$. The maximal number of action pairs supporting each other is given by summing up, for all facts f, the product of the number of actions supporting f with the nuber of actions that need f. We have computed, for the 2334 large instances from the second experiment described in the previous section, the above upper limit, as well as the real number of edges in the generation graph. In 8 domains, the values are the same across all instances. In the remaining domains, the values are close. Moreover, there seems to be a close correspondence to the running time consumed by the $\mathcal{O}|_{il}$ computation: the averaged upper limit values are between 3 and 11 millions in those four domains were $\mathcal{O}|_{il}$ takes a lot of computation time, and below one million in all other domains. It remains to establish an exact criterion that uses this correspondence for deciding about whether to compute $\mathcal{O}|_{il}$ or not.

Addressing the second problem, lack of strong pruning impacts in many domains, appears to us to be a much harder task. If one wants to obtain stronger pruning impacts, there does not seem to be a way around sacrificing empirical, let alone theoretical safety. So there appears to be a natural trade-off between speeding up the heuristic function, and maintaining completeness in the relaxation. We are currently experimenting with combining our techniques and RIFO's information selection heuristics. Reconsider the algorithm depicted in Figure 3, computation of $\mathcal{O}|_{al}$. The algorithm is based on computing the e function, which defines a tree upon termination. Now, as was said before, RIFO is also based on computing a tree, namely the so-called fact-generation tree. RIFO applies a number of heuristics to select the probably most relevant information from this tree. The idea is to apply the same heuristics to the tree defined by the e function. We have implemented some first strategies. As expected, the pruning impact became more drastic in some examples. However—as we also expected—a lot of states became unsolvable for the heuristic. In some cases, all paths to the goal were interrupted by such a state, rendering the whole planning task unsolvable for FF.

8 Conclusion and Outlook

We have presented a new approach towards defining irrelevance in planning tasks, concerning actions that are not necessary within the relaxation used in the heuristic functions of state-of-the-art heuristic planners like HSP and FF. We have derived a sufficient condition for relaxed irrelevance, and we have presented two approximation methods that can be used for filtering action sets. One of those methods, $\mathcal{O}|_{il}$ computation, has been proven to be complete within the relaxation, the other method, $\mathcal{O}|_{al}$ computation, has been shown to be empirically safe. The methods have drastic pruning impacts in some domains, speeding up FF's heuristic function, and in effect the planning process (except in *Schedule*, where there appears to be some interaction with FF's internal techniques). Computing $\mathcal{O}|_{al}$ never hurts in the sense that the required overhead is neglectible in most of the cases, and always small compared to FF's running time. Computing $\mathcal{O}|_{il}$ does not hurt in 16 of our 20 domains. We have outlined an approach how the other cases might be recognizable automatically. The challenge remains to find filtering methods that are still empirically safe in most of the cases, but have stronger pruning impacts.

References

1. Fahiem Bacchus. *Subset of PDDL for the AIPS2000 Planning Competition*. The AIPS-00 Planning Competition Comitee, 2000.
2. Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):279–298, 1997.
3. Blai Bonet and Héctor Geffner. HSP: Heuristic search planner. In *AIPS-98 Planning Competition*, Pittsburgh, PA, 1998.
4. Blai Bonet and Héctor Geffner. Planning as heuristic search: New results. In S. Biundo and M. Fox, editors, *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, Durham, UK, September 1999. Springer-Verlag.
5. Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 2001. Forthcoming.
6. Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI-97)*, pages 714–719. MIT Press, July 1997.
7. Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.
8. Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
9. Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 2001. Accepted for Publication.
10. Jana Koehler and Jörg Hoffmann. On the instantiation of ADL operators involving arbitrary first-order formulas. In *Proceedings ECAI-00 Workshop on New Results in Planning, Scheduling and Design*, 2000.
11. Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In Steel and Alami [14], pages 273–285.
12. Bernhard Nebel, Yannis Dimopoulos, and Jana Koehler. Ignoring irrelevant facts and operators in plan generation. In Steel and Alami [14], pages 338–350.
13. Edwin P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR-89)*, pages 324–331, Toronto, ON, May 1989. Morgan Kaufmann.
14. S. Steel and R. Alami, editors. *Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97)*, volume 1348 of *Lecture Notes in Artificial Intelligence*, Toulouse, France, September 1997. Springer-Verlag.