Optimal Delete-Relaxed (and Semi-Relaxed) Planning with Conditional Effects

Patrik Haslum
Australian National University & NICTA Optimisation Research Group
firstname.lastname@anu.edu.au

Abstract

Recently, several methods have been proposed for optimal delete-free planning. We present an incremental compilation approach that enables these methods to be applied to problems with conditional effects, which none of them support natively. With an h^+ solver for problems with conditional effects in hand, we also consider adapting the h^{++} anytime lower bound function to use the more space-efficient P^C_{ce} compilation. This avoids the memory limitation of the original h^{++} caused by its reliance on an exponential-space compilation. It also leads to improvements on some problems where memory is not an issue.

1 Introduction

In action description languages, a conditional effect is an action effect that takes place only if an additional condition holds when the action is applied, but which does not prevent the action from being applied if this condition is not true. Conditional effects have been part of planning languages dating back at least to ADL [Pednault, 1989], and part of PDDL since its first version [McDermott *et al.*, 1998]. They have been cited, particularly by researchers using classical planners as "black boxes", as an essential feature [Brafman *et al.*, 2012]. The now popular approach of compiling non-classical planning problems, like planning with uncertainty or temporally extended goals, into classical planning in order to leverage the recent performance gains of classical planners often makes extensive use of conditional effects [Palacios and Geffner, 2007; Bonet *et al.*, 2009; Keller and Eyerich, 2011; Baier and McIlraith, 2006]. Yet, support for conditional effects in planners, particularly cost-optimal planners, is limited: none of the twelve cost-optimal planners that participated in the last IPC had any support for conditional effects.

Conditional effects can be removed by two different problem transformations [Nebel, 2000], but there are problems with both: the compilation that preserves plans exactly is exponential in size, making it infeasible when actions have more than a few conditional effects, while the compact compilation does not preserve the problem delete relaxation, which is the basis of many planning heuristics.

Several efficient methods for optimal delete-free planning have been presented recently [e.g., Robinson *et al.* 2010; Gefen and Brafman 2012; Haslum *et al.* 2012; Pommerening and Helmert 2012], but none of them support conditional effects. Here, we introduce an incremental compilation approach through which they can be applied to problems with conditional effects. A subset (initially empty) of effects are compiled out, and the rest treated in a relaxed way; the compiled subset is grown, iteratively, only if needed to obtain a valid plan. This approach can solve problems with large numbers of conditional effects, where full compilation is infeasible, by compiling out only a small fraction.

The recent idea of "semi-relaxation" [Keyder *et al.*, 2012] provides a spectrum of intermediate points between delete-relaxed and fully non-relaxed planning, and is the basis of the h^{++} anytime lower bound function on plan cost [Haslum, 2012]. Semi-relaxed planning is delete-relaxed planning on a transformed problem. It too can be done in two ways: the P^C compilation is exponential in size, while the linear-space P^C_{ce} compilation introduces conditional effects into the problem. Having an effective h^+ solver for problems with conditional effects enables us to consider using the compact P^C_{ce} compilation in h^{++}, reducing the memory requirements drastically; surprisingly, it also improves the function on some problems where memory is not an issue.

2 Background

A planning problem is defined by a set of propositional state variables ("atoms"), a set of actions, a completely specified initial state (s_0) and a goal condition (G). Each action a is defined by a precondition ($\mathrm{pre}(a)$) and sets of positive ($\mathrm{add}(a)$) and negative ($\mathrm{del}(a)$) effects; each effect is a pair $\langle c, p\rangle$, where c is the effect condition and p the effect atom, made true ("added") by the action if $\langle c, p\rangle \in \mathrm{add}(a)$ and made false ("deleted") if $\langle c, p\rangle \in \mathrm{del}(a)$. Unconditional effects of an action are simply effects with an empty condition. A problem in which all actions have only unconditional effects is called a STRIPS problem. Note that an action may have both (conditional) effects that add and delete an atom p. If effects of both kinds trigger when the action is applied in some state, the add effect takes precedence so the result is to make the atom true. This is in accordance with PDDL semantics [Fox and Long, 2003].

Action preconditions, effect conditions, and the goal are

sets of atoms, interpreted as conjunctions. Instead of having negated atoms in conditions we assume that, when needed, each atom p has a complementary atom not-p that plays the role of $\neg p$, i.e., such that exactly one of p and not-p is true in every reachable state. Such complementary atoms can be added as needed with only a linear increase in problem size [Erol *et al.*, 1991]. As we consider cost-optimal planning, each action has a non-negative cost ($\mathrm{cost}(a)$); the cost of a plan is the sum of the costs of actions in it. The cost of an optimal plan for P is denoted $h^\star(P)$.

2.1 The Delete Relaxation

The delete relaxation of a planning problem P, denoted P^+, is exactly like P except that all negative effects are removed. The cost of an optimal plan for P^+, denoted h^+, is a lower bound on the cost of an optimal plan for P. Computing h^+ is NP-hard [Bylander, 1991].

A plan for a delete-free problem is monotonic, in the sense that any atom once made true never becomes false in a later state. (In P^+, it is possible for both atoms p and not-p in a complementary pair to be true in a state.) This means that if P is a STRIPS problem, an optimal (non-redundant) plan for P^+ does not include any action more than once. This is not true of planning problems with conditional effects, because different sets of conditional effects may need to trigger at different occurrences of the same action. Hence, we define a (sequenced) delete-relaxed plan as a sequence of pairs $\langle a_1, E_1\rangle, \ldots, \langle a_n, E_n\rangle$, where a_i is an action and $E_i \subseteq \mathrm{add}(a_i)$ is the subset of conditional add effects of a_i that must take place at step i of the plan; we call such a pair a *step*, and say the effects in E_i are *active*. A step $\langle a_i, E_i\rangle$ is applicable in state s if $\mathrm{pre}(a_i) \cup \bigcup_{\langle c,p\rangle \in E_i} c$ holds in s, and applying it adds $\{p \,|\, \langle \emptyset, p\rangle \in \mathrm{add}(a_i)\} \cup \{p \,|\, \langle c, p\rangle \in E_i\}$. In other words, a step may be viewed as an unconditional (STRIPS) action that combines the unconditional effects of a_i with the active effects in E_i. E_i does not have to contain all the effects of a_i that trigger when a_i is applied: triggering more effects can only make more atoms true, which cannot invalidate the remainder of the plan.

In an optimal (non-redundant) delete-relaxed plan, each step is a unique combination of action and effects. Although defined as a sequence, we will often consider a relaxed plan as an unordered set of steps. An unordered relaxed plan is valid iff some sequencing of the steps in it is. Because delete-relaxed execution is monotonic, deciding the validity of an unordered set of steps, and producing a sequencing if it is valid, can be done, in linear time, by greedily applying steps until all are done or no remaining step is applicable.

The relaxed plan dependency graph (RPDG) [Haslum, 2012] is a labelled directed graph that describes necessary ordering constraints in an unordered delete-relaxed plan S. It has one node n_a for each action $a \in S$, one node n_G representing the goal, and an edge $n_a \xrightarrow{\lambda} n'$ iff $\lambda \neq \emptyset$ is the subset of $\mathrm{pre}(n')$ that becomes relaxed unreachable if a is removed from the plan ($\mathrm{pre}(n_G) = G$). The RPDG of a valid relaxed plan is acyclic, and any sequencing of the plan must be a topological sort of the RPDG. We extend the RPDG to relaxed plans for problems with conditional effects by treating each step as an unconditional (STRIPS) action.

2.2 Semi-Relaxation

A semi-(delete-)relaxation of a planning problem P is a transformation of P into a new problem P' such that P' has the same optimal plan cost as P but the delete relaxation of P' keeps some information about the negative interactions caused by delete effects. Thus, $h^+(P')$ may be greater than $h^+(P)$ but remains a lower bound on $h^\star(P)$. Two semi-relaxing transformations have recently been proposed: Both take a parameter, C, which is a set of conjunctions (sets) of atoms, and add to the problem a new atom π_c for each $c \in C$. Actions in the compiled problem make π_c true only if their application in the original problem makes the condition c true. The P^C compilation [Haslum, 2012] produces a STRIPS problem that may be exponentially larger than P. The P^C_{ce} compilation [Keyder *et al.*, 2012] instead uses conditional effects to update π_c atoms, and therefore grows only linearly. Both compilations are defined only for STRIPS problems.

Both compilations are perfect in the limit, meaning that for some sufficiently large set C, $h^+(P^C) = h^+(P^C_{\mathsf{ce}}) = h^\star(P)$. The P^C_{ce} compilation is weaker, in the sense that it may require compiling in a larger set of conditions to achieve the same information gain; that is, there are cases where $h^+(P^C_{\mathsf{ce}}) < h^+(P^C)$ when both use the same set C.

2.3 Compilation of Conditional Effects

A planning problem with conditional effects can be compiled into a STRIPS problem in two ways [Nebel, 2000]: The first compilation, denoted P_{exp}, replaces each action a with one action for each possible "case", meaning a subset of effects that are triggered. As there can be up to $2^{|\,\mathrm{add}(a)\cup\mathrm{del}(a)|}$ cases, this compilation can increase the size of the problem exponentially, but it preserves plans exactly. Thus, $h^+(P_{\mathsf{exp}}) = h^+(P)$. The second compilation, denoted P_{seq}, breaks each action a into a sequence of actions, including one for each effect in $\mathrm{add}(a) \cup \mathrm{del}(a)$, and through additional control atoms forces each application of a (in the compiled problem) to be followed by the complete sequence. This compilation only increases the size of the problem polynomially, but may increase plan length polynomially. To preserve optimal plan cost, action costs must be set so that the cost of the complete sequence equals the cost of the original action. Inclusion of the complete sequence of separated effects cannot be enforced without the use of negative (delete) effects. Thus, delete relaxation of this compilation results in an additional relaxation, i.e., $h^+(P_{\mathsf{seq}}) \leq h^+(P)$.

3 Relaxed Planning with Conditional Effects

We approach optimal delete-relaxed planning for a problem P with conditional effects by incremental compilation: We first apply a transformation, similar to P_{seq}, to produce a delete-free STRIPS problem that is a relaxation of P^+; we call this the *floating effects* relaxation of P^+. The relaxed problem is solved, optimally, using any method for optimal delete-free planning available for STRIPS problems [e.g., Robinson *et al.* 2010; Gefen and Brafman 2012; Haslum *et al.* 2012; Pommerening and Helmert 2012]. If the resulting plan can be *scheduled* into a plan for P^+, this plan is optimal

and we are done; if not, we apply the exponential compilation to a subset of conditional effects, and the relaxation to the remaining conditional effects, and repeat. We call this procedure h^+_{ce}.

In the worst case, this may end up compiling away all conditional effects, thus faring no better than applying the exponential compilation up front and solving the STRIPS problem only once. But in practice, as will be shown, we often need to compile only a small fraction of them, thus making this approach able to solve problems whose exponential compilation is prohibitively large.

3.1 The Floating Effects Relaxation

The idea of making conditional effects into separate actions has been used in delete relaxation-based heuristics since FF [Hoffmann and Nebel, 2001]. Typically, the condition of each effect is augmented with the precondition of its action to ensure the action is (relaxed) applicable when the effect is used.

The floating effects relaxation is based on the same idea, but differs on two points: First, it ensures that whenever a conditional effect appears in the relaxed plan, so does at least one instance of the action that the effect belongs to. This allows setting costs so that the relaxation yields a lower bound, and simplifies scheduling the relaxed plan. Second, a subset of conditional effects (a parameter of the transformation) are compiled out rather than relaxed. This gives a sliding scale, from full relaxation to full exponential compilation of all conditional effects. To keep track of the compiled out effects, we associate with each action a in the relaxation a set $\mathrm{ce}(a)$ of conditional effects.

Definition 1 *Let P^+ be a delete-free problem with conditional effects and E a subset of its effects (with non-empty effect conditions). The* (partially compiled) floating effects relaxation*, denoted P^+_{fe}/E, is defined as follows. Atoms in P^+_{fe}/E are all atoms of P^+, plus one atom* done-a *for each action a in P^+. For each action a in P^+, P^+_{fe}/E has two types of actions:*

- *For each $A \subseteq (\mathrm{add}(a) \cap E)$, an action a_A with $\mathrm{pre}(a_A) = \mathrm{pre}(a) \cup \bigcup_{\langle c,p\rangle \in A} c$, $\mathrm{add}(a_A) = \{p \,|\, \langle \emptyset, p\rangle \in \mathrm{add}(a)\} \cup \{p \,|\, \langle c, p\rangle \in A\} \cup \{$*done-$a\}$*, and $\mathrm{cost}(a_A) = \mathrm{cost}(a)$. This is called an* anchor action *(for a). We set $\mathrm{ce}(a_A) = A$.*
- *For each conditional effect $\langle c, p\rangle \in (\mathrm{add}(a) - E)$ (where $c \neq \emptyset$), an action $a_{\langle c,p\rangle}$ with $\mathrm{pre}(a_{\langle c,p\rangle}) = c \cup \{$*done-$a\}$*, $\mathrm{add}(a_{\langle c,p\rangle}) = p$ and $\mathrm{cost}(a_{\langle c,p\rangle}) = 0$. This is called a* floating effect *(of a). We set $\mathrm{ce}(a_{\langle c,p\rangle}) = \emptyset$.*

The initial state of P^+_{fe}/E is that of P^+, with all done-a *atoms false, and the goal of P^+_{fe}/E is that of P^+.*

The size of P^+_{fe}/E is exponential, but only in the maximum number of effects per action that are compiled out. Note that the size of $P^+_{\mathsf{fe}}/\emptyset$ is linear in P.

It is easy to see that P^+_{fe}/E is a relaxation of P^+: Given a step $\langle a_i, E_i\rangle$ in an optimal plan for P^+, let $A_i = E_i \cap E$ be the subset of E_i that has been compiled out. P^+_{fe}/E, by construction, includes an anchor action a_{iA_i} with $\mathrm{ce}(a_{iA_i}) = A_i$. The precondition of this action is $\mathrm{pre}(a_i) \cup \bigcup_{\langle c,p\rangle \in A_i} c$, which must hold in the state where the step takes place, since a_i is applicable and all effects in E_i trigger, and it adds all of $\{p \,|\, \langle c, p\rangle \in A_i\}$. Thus, replacing each step $\langle a_i, E_i\rangle$ with a_{iA_i} followed by the floating effect action $a_{\langle c,p\rangle}$ for each effect $\langle c, p\rangle \in (E_i - A_i)$ yields a plan for P^+_{fe}/E, with equal cost. The converse, however, is not necessarily true (unless E contains all conditional effects in P^+). This is shown by the following example.

Example 1 Consider (a simplified version of) the Miconic domain, which involves scheduling the stops of an elevator to serve a queue of waiting passengers. The action (stop ?f), for each floor ?f, has two types of conditional add effects: one effect $\langle\{$(waiting ?p)$\},$ (boarded ?p)$\rangle$ for each passenger ?p waiting at ?f and one effect $\langle\{$(boarded ?p)$\},$ (served ?p)$\rangle$ for each passenger ?p whose destination is ?f.

Consider a problem with two passengers, A and B, where A wants to go from F1 to F2 and B from F2 to F1. An optimal plan for the floating effects relaxation (with $E = \emptyset$) of this problem is (stop F1), $\langle\{$(waiting A)$\}$,(boarded A)$\rangle$, (stop F2), $\langle\{$(boarded A)$\}$,(served A)$\rangle$, $\langle\{$(waiting B)$\}$,(boarded B)$\rangle$, $\langle\{$(boarded B)$\}$,(served B)$\rangle$, with a cost of 2 (since floating effect actions have cost 0). But this plan cannot be rewritten into a plan for the delete relaxation of the original problem: The last floating effect, $\langle\{$(boarded B)$\}$,(served B)$\rangle$, belongs to action (stop F1), so must take place at the same time as that action (whereas the relaxation only requires it to happen after the anchor action), but depends on the fact (boarded B) which is only added by an effect of the second action, (stop F2). Swapping the order of the two stops only leads to the same problem with passenger A. Due to the cyclic dependency, any plan for the delete relaxation of the original problem must stop at one floor twice, for a minimum cost of 3.

3.2 Scheduling a Relaxed Plan

A scheduling of a plan S for a partially compiled floating effects relaxation P^+_{fe}/E is a plan S' for P^+ that has one step for each anchor action $a \in S$, whose active effect set includes $\mathrm{ce}(a)$, and where each floating effect in S is included in the effect set of a (not necessarily adjacent) step with the action that the effect belongs to. The cost of S' is the same as the cost of S; thus, if S is an optimal plan for P^+_{fe}/E then S' is an optimal plan for P^+. Example 1 shows it may be impossible to schedule a plan for P^+_{fe}/E, but if S contains no floating effect action, then clearly a valid scheduling of S exists; thus for large enough E, scheduling succeeds.

The complexity of deciding if a plan for P^+_{fe}/E can be scheduled is an open question. Keyder et al. [2012] show NP-hardness, but for a slightly different scheduling problem; their reduction does not apply when both the set of anchor actions and floating effects is fixed. On the other hand, we also do not have a polynomial-time scheduling algorithm.

To schedule a relaxed plan we use plain depth-first search, with a constraint propagation mechanism for efficiency. With slight abuse of terminology, let S' be a partially scheduled plan, i.e., a set of steps where some actions are floating effects. Let $\langle a', E'\rangle$ and $\langle a_{\langle c,p\rangle}, \emptyset\rangle$ be two steps in S', where $a_{\langle c,p\rangle}$ is a floating effect and a' is an instance of the action that this effect belongs to. *Placing $a_{\langle c,p\rangle}$ with $\langle a', E'\rangle$* trans-

			FSC (1 state)						FSC (2 states)						T0 Grid				T0 IPC5					
	Assembly (30)	Miconic (150)	Blocks (14)	Grid A1 (16)	Grid A2 (2)	Grid R (16)	Hall (2)	Visual marker (7)	Blocks (14)	Grid A1 (16)	Grid A2 (2)	Grid R (16)	Hall (2)	Visual marker (7)	Dispose (15)	Push-To (12)	To-Trash (8)	Look-&-Grab (17)	Adder (2)	Coins (30)	Comm (25)	Sortnet (9)	UTS (29)	Alt. Sortnet (10)
h^+_{ce}	30	150	11	16	2	15	2	7	4	15	1	11	2	7	12	6	3	11	2	25	25	9	8	9
P_{exp}	30	150	0	2	1	0	1	0	0	2	1	0	1	0	3	1	0	0	0	20	25	3	18	1
h^+	30	150	0	2	0	0	1	0	0	1	1	0	1	0	1	0	0	0	0	20	25	2	6	1

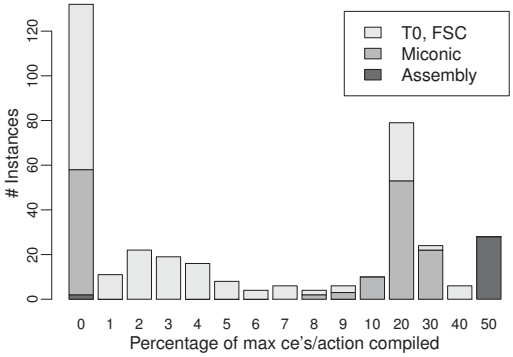

Table 1: Left: Number of delete-relaxed problems solved with h^+_{ce}; where P_{exp} could be built; and $h^+(P_{\mathsf{exp}})$ computed. Both methods are limited to 1h CPU time and 3Gb memory for each problem. Right: Maximum number of conditional effects per action compiled out, as a percentage of the instance maximum.

forms S' into a new plan S'' by (*i*) replacing $\langle a', E'\rangle$ with $\langle a', E' \cup \{\langle c, p\rangle\}\rangle$ and (*ii*) removing $\langle a_{\langle c,p\rangle}, \emptyset\rangle$.

Let S'' be the result of placing a floating effect $a_{\langle c,p\rangle}$ with some step $\langle a', E'\rangle$ in S': for any valid sequencing of S'' there is a corresponding valid sequencing of S', with $a_{\langle c,p\rangle}$ applied immediately following the step $\langle a', E'\rangle$. This implies that the RPDG of S', excluding the $a_{\langle c,p\rangle}$ node, is an edge-label-subgraph of the RPDG of S'', i.e., if $n \xrightarrow{\lambda} n'$ is an edge in the former, then $n \xrightarrow{\lambda'} n'$, with $\lambda \subseteq \lambda'$, is an edge in the latter. We exploit this to build a sound approximation (i.e., an edge-label-subgraph) of the RPDG of the final scheduled plan incrementally: Starting from the RPDG of the plan for the relaxation, whenever a floating effect $a_{\langle c,p\rangle}$ is placed with a step $\langle a', E'\rangle$, the $a_{\langle c,p\rangle}$ node is removed and all its incident edges redirccted to the $\langle a', E'\rangle$ node. Because the RPDG of a valid plan is acyclic, a cycle in the approximate graph implies that the current partially scheduled plan cannot be valid.

Scheduling starts from $S' = \{\langle a, \mathrm{ce}(a_A)\rangle \,|\, a_A \in S\} \cup \{\langle a_{\langle c,p\rangle}, \emptyset\rangle \,|\, a_{\langle c,p\rangle} \in S\}$. If there is a floating effect $a_{\langle c,p\rangle}$ in S', then for each candidate step $\langle a', E'\rangle$ in turn, try placing $a_{\langle c,p\rangle}$ with $\langle a', E'\rangle$. If the placement causes a cycle in the RPDG, or the new plan S'' cannot be sequenced, proceed to the next candidate; otherwise try to schedule S''. If any placement succeeds, return the plan, else add the effect $\langle c, p\rangle$ to a set *Failed* and backtrack.

If the scheduling procedure returns failure, the set *Failed* contains at least one conditional effect, and no effect in *Failed* is in the set E of effects that have been compiled out. In this case, we set $E' = \mathit{Failed} \cup E$, construct P^+_{fe}/E', call the STRIPS h^+ solver to obtain a new relaxed plan, and try again. This repeats until scheduling succeeds, which will happen eventually since the set of compiled out effects grows at each iteration. We then have an optimal plan for P^+.

3.3 Evaluation: h^+_{ce} in Practice

We compare the effectiveness of h^+_{ce} with building the full exponential compilation of the delete-relaxed problem, P^+_{exp}, and computing h^+ on that. The exponential compilation is optimised so that no redundant actions are generated: when an action has two conditional effects $\langle c, p\rangle$ and $\langle c', p'\rangle$ such that c implies c', any compiled action that includes $\langle c, p\rangle$ also includes $\langle c', p'\rangle$. We make one modification to h^+_{ce}, adding at most one conditional effect per action from *Failed* to the set of compiled effects, E, per iteration.[1] Both methods use the iterative landmarks h^+ solver [Haslum *et al.*, 2012].

Of the usual planning benchmark domains only Assembly (from IPC-1) and the "full" version of Miconic (from IPC-2), have conditional effects (after grounding). Both of them are easily dealt with by compilation because the maximum number of effects per action is small. Therefore, we also use problems generated by the conformat-to-classical planning compilation (T0) of Palacios and Geffner [2009] and the finite-state controller synthesis compilation (FSC) by Bonet et al. [2009]. These have significantly larger maximum numbers of conditional effects per action.

Table 1 (left) summarises the number of problems solved (i.e., h^+ computed) by the two methods, within 1h CPU and 3Gb memory per problem. Assembly and Miconic are easily solved by full compilation, but in the more difficult domains, incremental compilation has a clear advantage. In these, the number of conditional effects per action compiled (the exponent in the size of the compilation) is often a small fraction of the maximum; in about a third of the problems, no effects need to be compiled. The number of h^+_{ce} iterations varies from 1 to 101 (median 3, mean 7.9). There is, of course, a time overhead for computing h^+ several times: on those problems that can be solved with the full exponential compilation, the median slow-down of h^+_{ce} is a factor of 2.4 (mean 309).

4 Incremental Lower Bounds Beyond h^+

The h^{++} incremental lower bound function [Haslum, 2012] uses the P^C compilation to iteratively strengthen h^+. It computes an optimal delete-relaxed plan, checks if the relaxed

[1] We also tried a different compilation of relaxed planning with conditional effects, based on time-indexing atoms and actions in a way similar to the relaxed planning graph (Joerg Hoffmann, pers. comm.). Although it is polynomial in size, it was more difficult for the h^+ solver, and led to fewer problems solved.

plan works for the real, unrelaxed problem, and if not, computes a set of binary conjunctions C, called flaws, and repeats the process with P^C, thus creating a series of less and less relaxed problems. Each flaw set is chosen so that the current relaxed plan does not solve the delete relaxation of the next problem. This ensures the process eventually converges to a point where $h^+((\cdots(P^{C_1})^{C_2}\cdots)^{C_k}) = h^\star(P)$. However, because the P^C compilation is exponential it may run out of space, or out of time computing h^+. The derelaxation is built iteratively, instead of rebuilding P^C each time with a larger set C, because flaw extraction finds only binary conjunctions; after the the second iteration, the conjuncts can be π_c atoms, which lets it find (implicit) conjunctions of arbitrary size.

With a method of computing optimal delete-relaxed plans for problems with conditional effects in hand, we can use the linear-space P^C_{ce} compilation in place of P^C in h^{++}. This may avoid the exponential blow-up, as well as allow its application to problems with conditional effects. There are two obstacles: First, P^C_{ce}, as defined by Keyder *et al.* [2012], is defined only for STRIPS problems. Generalising it to work with conditional effects is straightforward, but the size of the generalised compilation is no longer polynomial in the worst case. Second, the existing flaw extraction method only works for STRIPS problems, and requires substantial revision.

4.1 The P^C_{ce} Compilation with Conditional Effects

The P^C_{ce} compilation [Keyder *et al.*, 2012] augments the planning problem P with a new atom π_c for each conjunction $c \in C$, and modifies actions so that π_c is made true only when the conjunction c holds. Here, we define the generalised P^C_{ce} compilation only for binary conjunctions.

Let C be a set of (binary) conjunctions, and X any set of atoms: X^C denotes the set $\{\pi_c \,|\, c \subseteq X, c \in C\}$. Let a be an action, and $\{p, q\}$ a binary conjunction. The conditional effects of a on $\{p, q\}$ are characterised as follows:

- For an atom p, $\mathrm{add}(a, p) = \min_\subset\{c \,|\, \langle c, p\rangle \in \mathrm{add}(a)\}$; that is, $\mathrm{add}(a, p)$ is the set of minimal (w.r.t. subset) conditions which cause action a to add p. Likewise, $\mathrm{del}(a, p) = \min_\subset\{c \,|\, \langle c, p\rangle \in \mathrm{del}(a)\}$.
- Let $\mathrm{del}(a, p) = \{c_1, \ldots, c_k\}$: $\cancel{\mathrm{del}}(a, p)$ is the set of minimal conditions in which a does not delete p, and is given by $\{\{\mathsf{not}\text{-}p_1, \ldots, \mathsf{not}\text{-}p_k\} \,|\, p_1 \in c_1, \ldots, p_k \in c_k\}$, i.e., all ways of taking the complement of one atom in each condition that causes a to delete p. Likewise, $\cancel{\mathrm{add}}(a, p)$ is the set of conditions in which a does not add p and is constructed analogously.
- Finally, $\mathrm{add}(a, \{p, q\})$ denotes the conditions in which a makes $\{p, q\}$ true. It is the union of three sets:

$$\begin{aligned}&\{c \cup c' \,|\, c \in \mathrm{add}(a, p), c' \in \mathrm{add}(a, q)\}\\ \cup &\{c \cup \{q\} \cup c' \,|\, c \in \mathrm{add}(a, p), c' \in \cancel{\mathrm{del}}(a, q)\}\\ \cup &\{c \cup \{p\} \cup c' \,|\, c \in \mathrm{add}(a, q), c' \in \cancel{\mathrm{del}}(a, p)\}\end{aligned}$$

- Likewise, $\mathrm{del}(a, \{p, q\})$ denotes the conditions in which a makes $\{p, q\}$ false. Because add effects take precedence over deletes, this happens only if a deletes, and does not add, one of the conjuncts: $\mathrm{del}(a, \{p, q\}) = \{c \cup c' \,|\, c \in \mathrm{del}(a, p), c' \in \cancel{\mathrm{add}}(a, p)\} \cup \{c \cup c' \,|\, c \in \mathrm{del}(a, q), c' \in \cancel{\mathrm{add}}(a, q)\}$.

Definition 2 *Let P be a planning problem (with conditional effects) and C a set of binary conjunctions. Atoms in P^C_{ce} are all atoms of P, plus one atom π_c for each $c \in C$. For each action a in P, P^C_{ce} has an action a' with*

- $\mathrm{pre}(a') = \mathrm{pre}(a) \cup \mathrm{pre}(a)^C$,
- $\mathrm{add}(a') = \{\langle c \cup (c \cup \mathrm{pre}(a))^C, p\rangle \,|\, \langle c, p\rangle \in \mathrm{add}(a)\}$ $\cup\{\langle c \cup (c \cup \mathrm{pre}(a))^C, \pi_{\{p,q\}}\rangle \,|\, c \in \mathrm{add}(a, \{p, q\})\}$,
- $\mathrm{del}(a') = \mathrm{del}(a) \cup \{\langle c, \pi_{\{p,q\}}\rangle \,|\, c \in \mathrm{del}(a, \{p, q\})\}$,

and $\mathrm{cost}(a') = \mathrm{cost}(a)$. Each atom π_c is initially true iff c is; the initial state of all other atoms in P^C_{ce} is as in P. The goal of P^C_{ce} is G^C, where G is the goal of P.

The size of P^C_{ce} is potentially exponential, but only if some actions conditionally add and delete atoms in some $c \in C$, and it is exponential only in the number of minimal conditions which cause the actions to add or delete those atoms.

The key property of P^C_{ce} is that in any reachable state, π_c is true if and only if c holds. (The proof of this is a straightforward adaptation of the proof for P^C [Haslum, 2012].) Since pre-, effect and goal conditions in P^C_{ce} include π_c only when the condition contains c, it follows that $a_1, \ldots, a_n$ is a plan for P if and only if $a'_1, \ldots, a'_n$ is a plan for P^C_{ce}. Thus, since corresponding actions have the same cost, $h^\star(P^C_{\mathsf{ce}}) = h^\star(P)$.

4.2 Flaw Extraction

The flaw extraction procedure for STRIPS problems has two steps: First, if a relaxed plan fails when executed in the real problem, it is because some atom, p, required by an action precondition or the goal, was deleted by a previous action. The nodes in the RPDG corresponding to these are identified as the *failed node*, n_f, and the *deleter*, n_d, respectively. Second, the set of flaws is extracted from the edge labels of the RPDG: A dependency closure from n to n' is a recursively defined subgraph of the RPDG that contains a path from n to n', and a path from n to any node that adds an atom that labels an edge in the closure. If n_f is a descendant of n_d, flaws are pairs $\{p, q\}$ where q labels an edge in a dependency closure from n_d to n_f. If not, dependency closures from n_d and n_f to one of their nearest common descendants are chosen, and flaws are pairs of one atom from each closure, with p added to the second. For a full description of the procedure we must refer to the earlier paper [Haslum, 2012]. Adapting it to relaxed plans with conditional effects requires several changes, which we can only sketch briefly here.

Relaxed Plan Normalisation The relaxed plan is normalised by (1) including in the active effect set of each step $\langle a, E\rangle$ all $\langle c, p\rangle \in \mathrm{add}(a)$ that are *implied* by the combined precondition of the step, i.e., such that $c \subset \mathrm{pre}(a_i) \cup \bigcup_{\langle c', p\rangle \in E_i} c'$; and (2) iteratively removing from each step any redundant non-implied effect, and any redundant step from the plan.

Splitting the RPDG Next, we need to separate active effects from actions, and track their dependencies explicitly. Computing the RPDG on the floating effects relaxation is not sufficient, because this does not reveal all necessary dependencies. Instead, we compute a *split RPDG* in two steps. Recall that each step $\langle a_i, E_i\rangle$ in a relaxed plan can be viewed as a STRIPS action; we compute a standard RPDG, G^1, on this plan, and transform it into the split RPDG, G^{split}, as follows: 1. Each node $n_{\langle a,E\rangle}$ in G^1 is split into one node, n_a, for the action and one node $n_{\langle c,p\rangle}$ for each active effect $\langle c, p\rangle \in E$.

2. For each incoming edge $n \xrightarrow{\lambda} n_{\langle a,E\rangle}$ in G^1, each label atom $q \in \lambda$ is considered separately: If $q \in \mathrm{pre}(a)$, redraw the edge $n \xrightarrow{q}$ to each node that $n_{\langle a,E\rangle}$ was split into (or add q to the label where an edge already exists). If $q \notin \mathrm{pre}(a)$, q must belong to the condition of some active effects; redraw the edge only to each node $n_{\langle c,p\rangle}$ where $q \in c$.

3. For each outgoing edge $n_{\langle a,E\rangle} \xrightarrow{\lambda} n$ in G^1 and each label atom $q \in \lambda$: If q is an unconditional effect of a, draw the edge $\xrightarrow{q} n$ from n_a (or add q to the label if the edge already exists). Otherwise, q is added by one of the active effects $\langle c, q\rangle \in E$; in this case, draw the edge $n_{\langle c,q\rangle} \xrightarrow{q} n$.

Assigning Failures If a step in the relaxed plan fails only because some of its effects' conditions are unsatisfied, each node $n_{\langle c,p\rangle}$ corresponding to such an effect is a failed node. If a step fails because the precondition of the action is unsatisfied, the action node, n_a, and all of its associated conditional effect nodes are failed. Flaws are generated separately for each failed node. In the split RPDG, nodes other than the goal node may have no outgoing edges; these cannot be used to derive flaws. However, due to normalisation, whenever there is a failed node with no outgoing edge, there is also another failed node that has at least one outgoing edge.

Selecting Deleters For each failed node, flaw extraction from the split RPDG proceeds in the same way as for STRIPS plans. However, since the node of the action that deleted the missing atom may have been split, we may have a choice of deleter node. This choice is made to minimise the size of the flaw set, with the restriction that if any candidate deleter node has a path to the failed node, one that does must be chosen.

Convergence The flaw extraction procedure for STRIPS plans returns a set of conjunctions such that compiling those in is guaranteed to rule out the failed relaxed plan. This is not true of flaw extraction, as described above, for conditional effect plans. The difficulty arises in the recursive part of a dependency closure, which must include a path from the root node of the closure to every node that adds an atom that labels an edge in the closure. In the standard RPDG, such a path always exists [Haslum, 2012, Theorem 2, item 5]. In the split RPDG, that path may need to be traced "backwards" from a conditional effect node to the action node of the corresponding step. How to define a closure in the split RPDG to ensure convergence is an open question.

Flaw Extraction for Non-Sequenced Plans The procedure used to compute a small flaw set that covers each sequencing of a partially ordered relaxed plan in the STRIPS version [Haslum, 2012] does not easily generalise to the conditional effects case. However, with the P^C_{ce} compilation minimising the size of the flaw set is not as critical. Thus, we adopt the simple enumerate-to-first-failure approach for non-sequenced relaxed plans.

4.3 Evaluation: h^{++}_{ce} vs. h^{++}

We compare use of the P^C_{ce} compilation in h^{++} (called h^{++}_{ce}) with the original procedure, using the P^C compilation, on two sets of domains: Airport, 4-ops Blocksworld and Openstacks are domains where the original h^{++} most often runs out of memory. Here, we want to see if using P^C_{ce} instead leads to better lower bounds. APPN, ParcPrinter and Woodworking are among the domains where h^{++} is at its best, compared to lower bounds based on A* search. Here, we want to see if using the weaker P^C_{ce} compilation also leads to weaker bounds.

		h^{++}_{ce}		h^{++}		
		Sol.	$>$	Sol.	$>$	Mem.
Airport	(50)	21	0	24	9	15
4-ops Blocksworld	(35)	12	15	8	8	5
Openstacks	(30)	0	10	0	0	19
APPN	(35)	35	1	33	0	0
ParcPrinter	(30)	26	0	27	4	0
Woodworking	(30)	11	7	9	9	0

Table 2: Comparison of h^{++}, using the P^C_{ce} (h^{++}_{ce}) and P^C compilations. Columns show the number of problems solved ("Sol."); where each method proves a strictly higher lower bound ($>$); and for the P^C compilation, where h^{++} runs out of memory ("Mem.").

Both solvers are given 1h CPU time and 3Gb memory per problem. Table 2 shows the number of problems optimally solved, and how often each method proves a lower bound (on $h^\star$) strictly higher than the other. h^{++}_{ce}, of course, never runs out of memory, and does improve over h^{++} where it does, but the improvement does not seem directly related to how often that happens. Moreover, h^{++}_{ce} even fares (slightly) better in some of the domains where h^{++} already does well. Clearly, more factors than just the size of the compilation determine the effectiveness of this iterative lower bounding approach.

5 Conclusions

Adapting optimal planning techniques to problems with conditional effects is difficult, but important for optimal planners to be applicable in many cases, particularly to problems generated by compilations. The exponential conditional effects compilation is convenient, since it allows such problems to be reduced to the STRIPS case, but infeasible whenever actions have more than a few conditional effects. We have shown that *incremental* compilation may be a promising approach, though it remains to investigate how this idea can be applied in other contexts than for computing h^+.

Having an h^+ solver for problems with conditional effects allows us to contemplate using the more space-efficient P^C_{ce} construction within the h^{++} anytime lower bound function. This requires modifications to both the compilation and the original h^{++} procedure, and establishing the convergence guarantee that holds in the STRIPS case remains an open problem. However, preliminary results suggest that there are more benefits to making this switch than just reducing memory requirements.

Acknowledgements

NICTA is funded by the Australian Government represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[Baier and McIlraith, 2006] J. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *Proc. 21st National Conference on AI (AAAI'06)*, 2006.

[Bonet *et al.*, 2009] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 34–41, 2009.

[Brafman *et al.*, 2012] R. Brafman, G. Shani, and R. Taig. Leveraging classical planners through translations. In *Proc. ICAPS'12 workshop on the International Planning Competition*, pages 6–9, 2012.

[Bylander, 1991] T. Bylander. Complexity results for planning. In *Proc. 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 274–279, 1991.

[Erol *et al.*, 1991] K. Erol, D.S. Nau, and V.S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning: A detailed analysis. Technical Report CS-TR-2797, Computer Science Department, University of Maryland, 1991.

[Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of AI Research*, 20:61–124, 2003.

[Gefen and Brafman, 2012] A. Gefen and R. Brafman. Pruning methods for optimal delete-free planning. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 2012.

[Haslum *et al.*, 2012] P. Haslum, J. Slaney, and S. Thiébaux. Minimal landmarks for optimal delete-free planning. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pages 353–357, 2012.

[Haslum, 2012] P. Haslum. Incremental lower bounds for additive cost planning problems. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pages 74–82, 2012.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of AI Research*, 14:253–302, 2001.

[Keller and Eyerich, 2011] T. Keller and P. Eyerich. A polynomial all outcome determinization for probabilistic planning. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, pages 331–334, 2011.

[Keyder *et al.*, 2012] E. Keyder, J. Hoffmann, and P. Haslum. Semi-relaxed plan heuristics. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, pages 128–136, 2012.

[McDermott *et al.*, 1998] D. McDermott, M. Ghallab, A. Howe, C. A. Knoblock, A. Ram, Veloso. M., D. Weld, and D. Wikins. PDDL – the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[Nebel, 2000] B. Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of AI Research*, 12:271–315, 2000.

[Palacios and Geffner, 2007] H. Palacios and H. Geffner. From conformant into classical planning: Efficient translations that may be complete too. In *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 264–271, 2007.

[Palacios and Geffner, 2009] H. Palacios and H. Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of AI Research*, 35:623–675, 2009.

[Pednault, 1989] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332, 1989.

[Pommerening and Helmert, 2012] F. Pommerening and M. Helmert. Optimal planning for delete-free tasks with incremental LM-cut. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 2012.

[Robinson *et al.*, 2010] N. Robinson, C. Gretton, D. Pham, and A. Sattar. Partial weighted MaxSAT for optimal planning. In *Proc. 11th Pacific Rim International Conference on AI (PRICAI'10)*, 2010.