Pruning bad quality causal links in sequential satisfying planning

Sergio Jiménez Celorrio
sjimenez@inf.uc3m.es
Departamento de Informática,
Universidad Carlos III de Madrid, ESPAÑA

Patrik Haslum and **Sylvie Thiebaux**
firstname.lastname@anu.edu.au
Australian National University, Canberra and
Optimisation Research Group, NICTA

Abstract

Although current sequential satisficing planners are able to find solutions for a wide range of problems, the generation of good quality plans still remains a challenge. Anytime planners, which use the cost of the last plan found to prune the next search episodes, have shown useful to improve the quality of the solutions. With this in mind this paper proposes a method that exploits the solutions found by an anytime planner to improve the quality of the subsequent ones. The method extracts a set of causal links from the first plans, the plans with worse quality, and creates a more constrained definition of the planning task that rejects the creation of these causal links. The performance of the proposed method is evaluated in domains in which optimization is particularly challenging.

Introduction

In this paper we are concerned with improving the quality of solutions in sequential satisficing planning. The mainstream approach for sequential satisficing planning is heuristic planning. Heuristic planners address the complexity of this planning task using search algorithms guided by heuristics computable in polynomial time. Recent heuristics — such as those based on relaxed plans (Hoffmann and Nebel 2001; Bonet and Geffner 2001) or on the automatic extraction of landmarks (Porteous and Sebastia 2004) — allow heuristic planners to generate sequential satisficing plans in just a few seconds on many different problems. However the generation of good quality plans is still challenging for current heuristic planners and complicates their application to many real-world problems.

In theory, sequential optimal planning is as complex as sequential satisficing planning (Bylander 1994) but in practice, many planning tasks are harder to be solved optimally (Helmert 2003). A notorious example are planning tasks with goals reachable through multiple paths, typically because of symmetries or transpositions, whose search space rapidly grow (Helmert and Röger 2008). This feature is present in many scheduling and logistics domains like the *openstacks* or the *visitall* domains from the last IPC, IPC-2011.

Anytime heuristic planners, which iteratively refine the quality of their solutions, have recently been shown useful for optimization problems in sequential satisficing planning (Richter, Thayer, and Ruml 2010). In particular, this is the approach followed by LAMA-2011, the winner of the sequential satisficing track of the IPC-2011 (Coles et al. 2012). Planners taking this approach iteratively reduce the search space of the planning task by pruning the nodes that exceed the cost of the best solution found.

This paper proposes a new method to improve the quality of solutions that exploits knowledge learned from the bad plans found by an anytime planner. The method extracts a set of causal links that appear in low quality solutions i.e., the first solutions found by the anytime strategy, and creates a new definition of the planning task that constrains the search to reject causal links from this set. Since the method does not alter the core elements that make the planner efficient, other than indirectly as a consequence of the reduction of the search space, the planner is expected to retain its efficiency throughout this process.

The paper is organized as follows. The next section gives the background necessary for presenting the proposed method. The third section explains the method in detail. The fourth section shows the empirical performance of the proposed method in domains in which optimization is particularly challenging and analyses the obtained results. The fifth section reviews the related work and finally, the sixth section poses conclusions and outlines future work.

Background

This section sets the scene for the method proposed in the paper and presents the sequential satisficing planning task and the concept of causal links in a given sequential plan.

Sequential satisficing planning

We consider the sequential satisficing planning task formulated as a tuple $\Pi = <S, A, s_0, G>$, where S is the set of propositional state variables, A is a set of ground actions, each of which is a tuple $a = <pre(a), del(a), add(a), cost(a)>$, where: $pre(a)$ are the conditions for the action's applicability, $del(a)$ are the literals removed by application of the action, $add(a)$ are the literals added by application of the action and $cost(a)$ is the cost of applying the action. Last but not least $s_0 \subseteq S$ is the initial state of the planing task and G is the set of literals that defines the goal states.

Applying an action $a \in A$ in a state s produces a state s' such that $s' = (s \setminus del(a)) \cup add(a)$. A solution π to a sequential satisficing planning task Π is defined as a sequence of actions $\pi = [a_1, \ldots, a_n]$ corresponding to a sequence of state transitions $[s_0, \ldots, s_n]$ such that, a_i is applicable in state s_{i-1}; applying action a_i in s_{i-1} produces state s_i; and s_n is a state that satisfies all the goal conditions defined in G. The cost of the solution π is the sum of its action costs: formally, $cost(\pi) = \sum_{a_i \in \pi} cost(a_i)$. An optimal solution is one that minimizes this sum. Since we consider satisficing planning, we do not seek plans that are guaranteed to be optimal, but we still try to minimize cost.

As an example, Figure 1 shows a solution plan for problem *p01* from the *Openstacks* domain of the sequential satisficing track of IPC-2008.

	#stacks	cost
`(open-new-stack n0 n1)`	1	1
`(start-order o1 n1 n0)`	0	
`(open-new-stack n0 n1)`	1	1
`(start-order o2 n1 n0)`	0	
`(make-product p2)`		
`(ship-order o1 n0 n1)`	1	
`(make-product p1)`		
`(ship-order o2 n1 n2)`	2	
`(start-order o3 n2 n1)`	1	
`(start-order o4 n1 n0)`	0	
`(make-product p3)`		
`(ship-order o3 n0 n1)`	1	
`(make-product p4)`		
`(ship-order o4 n1 n2)`	2	
`(start-order o5 n2 n1)`	1	
`(make-product p5)`		
`(ship-order o5 n1 n2)`	2	

Figure 1: Example plan for problem *p01* from the *Openstacks* domain of the sequential satisficing track of the IPC-2008. Column "#stacks" shows the number of stacks available (free) after each action, where it changes. The last column shows (non-zero) action costs.

The *Openstacks* domain will be used as a running example throughout the paper. In this domain, a manufacturer has a number of orders to ship and each order requires the combination of different products. The manufacturer can only make one type of product at a time but the total quantity required for one type of product is made at the same time. From the time that the first product in an order is made to the time that all products in the order are made, the order is said to be *open* and during this time it requires a *stack*, a temporary storage space. When the order is complete, it can be shipped and at that time the stack it occupied becomes free for use again. This is illustrated by the plan in Figure 1. The objective is to sequence the production of the orders so as to minimize the maximum number of stacks simultaneously in use. The planning formulation of this problem models this with an extra action, `open-new-stack`, which increases the number of stacks available by one. This action has a cost of 1, while all other actions have zero cost. Finding any solution to this planning task is easy, because is is always possible to open enough stacks for all the orders, but finding plans of good quality is difficult since it depends on the sequencing of the actions.

Extracting causal links from a sequential plan

Let $\pi = [a_1, \ldots a_n]$ be a solution to a planning task Π. The triple $cl = <a_i, a_j, p>$ that comprises two actions from the plan, $a_i, a_j \in \pi$, and one proposition p forms a causal link in π if: (1) p is added by a_i, (2) p is also a precondition of a_j and (3), p is not deleted or added by any action that occurs between a_i and a_j. Formally, $i<j$, $p \in add(a_i)$, $p \in pre(a_j)$ and $\nexists a_k$ s.t., $i < k < j$, $p \in (del(a_k) \cup add(a_k))$. In such a causal link, a_i is called the *producer* and a_j is called the *consumer*. Note that this definition restricts the producer in a causal link to be the last achiever of a proposition, and also that links from propositions provided by the initial state are not considered.

Figure 2 shows the algorithm for extracting the set of causal links CL_π in a given plan π for a planning task Π. This algorithm traverses the plan forward, registering the propositions added by its actions in a list of *started* causal links. Each entry in this list is a pair of the producer action and the proposition produced. When an action of the plan deletes a proposition, or when the last achiever changes, the algorithm removes from the list of started causal links any entry with this proposition. For each proposition required by the precondition of an action in the plan, a complete causal link is created and added to the set of causal links of the plan if there is a started causal link for that proposition.

For each proposition p, we denote by $A_{threat}(p) \subseteq A$ the subset of actions that delete p. These are the actions that can *threaten* a given causal link for p.

Method

This section describes our method for finding plans of better quality. The method is structured in three phases: First, a learning phase that takes a set of plans, of different quality, and extracts a set of causal links that appear in plans of worse quality, but not in the best plan. This set is further filtered to focus on causal links that appear early in the plan. The intuition is that these are most important for the search to avoid. Second, a compilation phase uses these causal links to generate a more constrained definition of the planning task, that disallows the creation of these "bad" causal links. The third phase is to apply a planner to the modified definition of the planning task. Since the restrictions placed on the modified planning task are not strong enough to make the first plan found match the quality of the best plan, we use an anytime planner to continue searching for better plans. The remainder of this section describes each of these three phases in detail.

Phase I: Learning causal links to reject

Planning tasks, and particularly planning tasks in which optimization is the main difficulty, usually have many valid solution plans with different quality. In practice solution plans

$extractCausalLinks(\pi, \Pi)$:
Input: π, a plan, and Π, a planning task
Output: CL_π, the set of causal links of the plan

$CL_\pi = \varnothing$
$CL_{started} = \varnothing$

for $a \in \pi$ **do**
  for $p \in pre(a)$ **do**
    if $\exists <a', p> \in CL_{started}$ **then**
      $add(<a', a, p>, CL_\pi)$
    end if
  end for

  for $p \in add(a)$ **do**
    if $\exists <a', p> \in CL_{started}$ **then**
      $replace(<a', p>, <a, p>, CL_{started})$
    else
      $add(<a, p>, CL_{started})$
    end if
  end for
  for $p \in del(a)$ **do**
    $remove(<\star, p>, CL_{started})$
  end for
end for

return CL_π

Figure 2: Algorithm for extracting the set of causal links from a sequential plan

with different quality can be generated by adding some variation to the planner, for example varying choices normally made arbitrarily, the weight of the heuristics or the cost bound, and then running the planner several times. Since current sequential satisficing planners tend to be fast, it is often feasible to repeat this process, collect a wealth of information about the planning process and thus obtain a basis for learning which choices impact on the quality of the generated plans.

The learning phase uses a state-of-the-art planner to generate a collection of solutions for a given planning task and examines the decisions that led to different plan qualities to discover knowledge that helps focusing next planning episodes on solutions with better quality. Specifically, this knowledge takes the form of causal links that appear only in worse plans. The input to the learning phase is a planning task Π and a planner P that is able to generate different solution plans π_i, for example by implementing an anytime strategy that iteratively bounds the cost of the solutions. The output of this phase is the set of causal links to reject in the following planning episodes, CL_{rej}.

Figure 3 presents the algorithm that implements the learning phase. First the planner P is run, up to some time limit, to generate diverse solution plans for the planning task Π. After that the set of causal links CL_{π_i}, is computed for each solution plan π_i, using the algorithm of Figure 2. The algorithm identifies the plan with the best quality i.e., with minimum cost, among all the generated plans. For each of the other plans, $\pi_i \neq \pi_{best}$, the algorithm filters its set of causal links CL_{π_i} and creates a subset $CL'_{\pi_i} \subseteq CL_{\pi_i}$ that only includes the causal links contained in the prefix of π_i that does not exceed the cost of the best plan. In other words, if the summed cost of the actions in the prefix $a_1, \ldots, a_j$ of a plan π_i is higher than the cost of the best plan, π_{best}, then the causal link $cl = <a_i, a_j, p>$ in CL_{π_i} is not included in CL'_{π_i}. The purpose of this is to focus learning on bad choices that appear early in the search process. The compilation, described in the next subsection, prunes from the problem plans that contain the rejected causal links, but this pruning may not occur until the causal link has been completed, i.e., until the consumer action a_j is added to a path in the search space. If the cost of the prefix at this point already exceeds the cost of the best plan, it will be pruned by the much simpler mechanism of imposing a cost bound on the planner. Thus, we reserve the compilation for those bad plan prefixes that should be pruned earlier.

Finally the set of causal links to reject is computed as the causal links present in the subsets, after filtering, extracted from plans of worse quality that are not present in the best plan. Formally,

$$CL_{rej} = (\bigcup_{i \neq best} CL'_{\pi_i}) \setminus CL_{\Pi_{best}}$$

.

$learningCausalLinks(\Pi, P)$:
Input: Π, a planning task and P, a planner able to find different solutions.
Output: CL_{rej}, set of causal links to reject.

$plans = Plan(P, \Pi)$

for $\pi_i \in plans$ **do**
  $CL_{\pi_i} = extractCausalLinks(\pi_i, \Pi)$
end for

$\pi_{best} = \underset{\pi_i \in plans}{\operatorname{argmin}}\ cost(\pi_i)$
for $\pi_i \in plans$ and $\pi_i \neq \pi_{best}$ **do**
  $CL'_{\pi_i} = filter(CL_{\pi_i}, \pi_{best})$
end for

$CL_{rej} = (\bigcup_{i \neq best} CL'_{\pi_i}) \setminus CL_{\Pi_{best}}$
return CL_{rej}

Figure 3: Algorithm for extracting the set of causal links to reject from a set of solution plans.

Phase II: Compilation of the causal links to reject

Machine learning has been used to improve planning processes since the early days of automated planning, and a wide range of different mechanisms have been developed to exploit the learned knowledge. Because we aspire for our learning method to be as generally applicable as possible, it

exploits the learned knowledge without introducing modifications to the planner. Instead, we create a new planning task by compiling the learned knowledge into the original planning task. The new planning task resulting from the compilation is more constrained because it prevents the creation of the "bad" causal links learned in the previous phase. In particular, the new planning task introduces extra preconditions that only allow the application of actions when they are not creating any of the learned causal links. The input to this phase is the planning task to solve Π and the set of causal links to reject CL_{rej}. The output of this phase is the more constrained planning task Π'.

In the new planning task Π', the rejected causal links are represented explicitly by objects of a new type `causalLink`. The initial state and action definitions of Π' are modified to monitor the state of each of these causal links, whether they are started or not, and prevent their appearance in solution plans.

The new initial state The initial state s'_0 of the new planning task Π' is created by extending the initial state of the original planning task with static facts that describe the causal links to reject and their threats. Accordingly, for each causal link $cl_{id} = <a_i, a_j, p>$ such that $cl_{id} \in CL_{rej}$ the initial state is extended with:

- A new static fact that describes the producer, `(isproducer-`$<n(a_i)>$`-of-`$<n(p)>$ $cl<id>$ $<arg(a_i)>$`)`. The name of the predicate is the concatenation of the name of the producer action, $n(a_i)$, and the name of the proposition of the causal link, $n(p)$. The arguments of the predicate are the object $cl<id>$ that represents the causal link, and the arguments of the producer.
- A new static fact that describes the consumer, `(isconsumer-`$<n(a_j)>$`-of-`$<n(p)>$ $cl<id>$ $<arg(a_j)>$`)`. As above, the name of the predicate contains the name of the consumer, $n(a_j)$, and the name of the proposition of the causal link. The arguments are the object $cl<id>$ that represents the causal link and the arguments of the consumer action.
- New static facts describing each threat to the causal link. That is, for each action $a_k \in A_{threat}(p)$ that threatens the causal link, `(isthreat-`$<n(a_k)>$`-for-`$<n(p)>$ $cl<id>$ $<arg(p)>$`)`. The name of the predicate is the concatenation of the name of the threatening action, $n(a_k)$, and the name of the proposition of the causal link. The arguments are the causal link object and the arguments of the proposition.

An example of an extended initial state s'_0 is shown in Figure 4. The example illustrates the compilation of two causal links on the planning task *p01* from the *Openstacks* domain of the sequential satisficing track of the IPC-2008. The causal links compiled in the example are cl_1 and cl_2, where $cl_1 =$ `< (start-order o2 n1 n0), (open-new-stack n0 n1), (stacks-avail n0) >` and $cl_2 =$ `< (open-new-stack n0 n1), (start-order o10 n1 n0), (stacks-avail n1) >`.

```
(:init
  ;;; Begin - Initial state from the original task
  (next-count n0 n1) (next-count n1 n2) (next-count n2 n3)
  (next-count n3 n4) ...
  ;;; End - Initial state from the original task

  ;;; Begin - Extension to the original initial state
  ;; for the causal link cl1
  (isproducer-start-order-of-stacks-avail cl1 o2 n1 n0)
  (isconsumer-open-new-stack-of-stacks-avail cl1 n0 n1)
  (isthreat-ship-order-for-stacks-avail cl1 n0)
  ;; for the causal link cl2
  (isproducer-open-new-stack-of-stacks-avail cl2 n0 n1)
  (isconsumer-start-order-of-stacks-avail cl2 o10 n1 n0)
  (isthreat-ship-order-for-stacks-avail cl2 n1)
  ;;; End - Extension to the original initial state
 )
```

Figure 4: Example of the extensions introduced to the initial state of a planning task to constrain the creation of learned causal links.

The new action model The action model of the planning task Π' is also extended. In particular, for each causal link to reject:

- A new literal `(clstarted` $cl<id>$`)` is added to the positive effects of the action a_i, the producer of the causal link. Adding this literal makes the application of the action a_i modify the state of the causal link to `started`. This is implemented by introducing a new quantified conditional effect in the model of the producer action. Figure 5 shows the new PDDL model of action `ship-order` after compiling the causal link $cl_3 =$ `< (ship-order o9 n0 n1), (start-order o7 n1 n0), (stacks-avail n1) >`. Although this implementation increases the size of the planing task, and may make instantiation more expensive, it results in a more compact and understandable model of the modified planning task.

```
(:action ship-order
 :parameters (?o - order ?avail - count ?new-avail - count)
 :precondition (and (started ?o) (stacks-avail ?avail)
                    (next-count ?avail ?new-avail))
                    (forall (?p - product)
                      (or (not (includes ?o ?p)) (made ?p)))
 :effect (and (not (started ?o)) (shipped ?o)
              (not (stacks-avail ?avail))
              (stacks-avail ?new-avail)
              ;;; Begin - New quantified conditional effect
              (forall (?clid - causalLink)
                (when (isproducer-ship-order-of-stacks-avail
                        ?clid ?o ?avail ?new-avail)
                  (clstarted ?clid)))
              ;;; End - New quantified conditional effect
              ))
```

Figure 5: Example of the extension to the PDDL model of an action that is producer in a causal link to reject.

- A new precondition

```
(or (not (clstarted cl<id>))
    (not (isconsumer-<n(a_j)>-of-<n(p)>
          cl<id> <arg(a_j)>)))
```

is added to the action a_j, the consumer of the causal link. This new precondition makes the action applicable only when it is not creating the causal link. Similar to the previous, this extension is implemented with a quantified precondition in the model of the consumer action. Figure 6 shows the new PDDL model of the action `start-order` after compiling causal link cl_3.

```
(:action start-order
 :parameters (?o - order ?avail - count ?new-avail - count)
 :precondition
  (and (waiting ?o) (stacks-avail ?avail)
       (next-count ?new-avail ?avail)
       ;;; Begin - New quantified precondition
       (forall (?clid - causalLink)
         (or (not (clstarted ?clid))
             (not (isconsumer-start-order-of-stacks-avail
                    ?clid ?o ?avail ?new-avail)))))
       ;;; End - New quantified precondition
 :effect (and (not (waiting ?o)) (started ?o)
              (not (stacks-avail ?avail))
              (stacks-avail ?new-avail)))
```

Figure 6: Example of the extension to the PDDL model of an action that is consumer in a causal link to reject.

- A new negative effect `(not (clstarted cl<id>))` is added to each action $a_k \in A_{threat}(p)$ that threatens the causal link. The new delete effect makes the application of action a_k modify the state of the causal link. Again, in order to make the compilation more compact, it is implemented by introducing a new quantified conditional effect in the model of the action is a threat. Figure 7 shows the new PDDL model of the action `open-new-stack` after compiling causal link cl_3.

```
(:action open-new-stack
 :parameters (?open - count ?new-open - count)
 :precondition (and (stacks-avail ?open)
                    (next-count ?open ?new-open))
 :effect (and (not (stacks-avail ?open))
              (stacks-avail ?new-open)
              ;;; Begin - New quantified conditional effect
              (forall (?clid - causalLink)
                (when (isthreat-open-new-stack-for-stacks-avail
                        ?clid ?open)
                      (not (clstarted ?clid))))
              ;;; End - New quantified conditional effect
              (increase (total-cost) 1)))
```

Figure 7: Example of extension to the PDDL model of an action that threatens a causal link to reject.

Phase III: Planning rejecting bad causal links

The aim the modified planning task is to focus planners on solutions with better quality, even planners that are not necessarily good optimizers, like planners that ignore action cost. Moreover, planners are expected to retain their efficiency since that the core elements that make a planner efficient, particularly the heuristic, are not altered other than indirectly as a consequence of the modification of the search space. The inputs to this third phase are the new planning task Π' and a state-of-the-art sequential satisficing planner P. The output is, if found, a solution plan that does not include causal links from the set CL_{rej}.

Planning with the new planning task Π' is correct in the sense that any solution to Π' is also a solution to the original planning task Π. Briefly, this is because the new planning task is a more constrained version of the original one. In detail, both the initial state and goals from the original task Π are also present in the new task Π'. Furthermore the new predicates only serve to monitor the state of the causal links, whether they are started or not. The new effects added to the actions of Π' that either work as producers or threats in causal links from CL_{rej} only modify the state of these causal links. Examples of these new effects are shown in Figures 5 and 7. Finally, the new preconditions added to actions that are consumers of causal links from CL_{rej} only constrain the application of these actions, as shown in the example in Figure 6.

Planning with the new planning task Π' is not complete, in the sense that optimal solutions for the original planning task Π may be pruned by the new preconditions of the planning task Π'. The explanation for this is that the set of plans used in the learning phase may be only a subset of plans. The plan identified as the plan with the best quality in the learning phase, π_{best}, may not be optimal. In fact, an optimal plan may include causal links that are rejected. Therefore, there is no guarantee that optimal solutions are not pruned from the new planning task.

Planning with the new planning task Π' can be computationally more expensive than planning with the original task Π. In particular, the size of the state space increases with the introduction of the new predicate `(clstarted cl<id>)` that monitors the state of the selected causal links. The complexity of actions is also increased by including quantified conditional effects in the actions that act as producers or threats of the causal links in CL_{rej}. However, the new preconditions added to consumer actions are expected to reduce the size of the search space by constraining the states in which these actions can be applied. Thus, the final impact of the proposed compilation on planner performance depends on this trade-off between the size of the new planning task and the benefits achieved by pruning causal links from bad quality solutions. An experimental analysis of the performance of the proposed method examining this trade-off is provided along the next section.

Results

This section shows the experimental evaluation of the proposed method. First it details the design of the experiments

and next it analyses the performance of the proposed method in its three phases: learning, compilation and planning.

Experimental setup

The presented evaluation is a direct comparison of the performance of the winner of the last IPC, LAMA-2011 and the proposed method. Like in the competition LAMA-2011 is run with a time bound of 1800 seconds per problem. On the other hand the proposed method is run with the same time bound which is distributed as follows: 300 seconds for the learning phase, 100 seconds for the compilation phase, despite in practice it is always completed in milliseconds time, and 1400 secs for the planning phase. In more detail the three phases of the proposed method are configured as follows:

I. Learning causal links to reject.

(a) Run LAMA-2011 on the original planning task for 300 seconds.

(b) Compute the causal links to reject following the learning algorithm of Figure 3 and using the solution plans generated by the anytime strategy of LAMA-2011 during the 300 seconds.

II. Compilation of the causal links to reject.

(a) A maximum of 100 causal links is compiled to not overload the size of the new planning task

(b) When the number of learned causal links for a given problem is greater than 100, a subset is selected that only comprises the most expensive causal links. The cost of a given causal link is computed adding the cost of its producer and its consumer. Formally, $cost(cl_{id}) = cost(a_i) + cost(a_j)$.

III. Planning rejecting bad causal links.

(a) Run LAMA-2011 on the new planning task for 1400 seconds.

(b) The anytime strategy of LAMA-2011 is initiated with the cost bound of the best plan found in the learning phase.

The proposed method is evaluated in the *openstacks*, *parking*, *nomystery* and *visitall* domains from the IPC-2011. These domains are selected because they are hard for optimization. In particular all of them present symmetries and transpositions that cause a rapid growth of their search space.

Phase I: Learning causal links to reject

LAMA-2011 implements an anytime strategy that uses the cost of the last plan found to prune, in the next search episodes, the nodes exceeding this cost. The anytime strategy of LAMA-2011 comprises the following search episodes: two greedy best first searches, the first one quality blind, followed by a sequence of weighted A* searches with decreasing weights 5, 3, 2 and 1. The Table 1 illustrates the outcome of the learning phase in the evaluation domains showing, for each problem, two data: *plans*, the number of solution plans found by LAMA-2011 during the learning phase and *clinks*, the number of causal links learned from these solution plans.

	Openstacks		**Parking**		**Nomystery**		**VisitAll**	
Prob	plans	clinks						
000	8	19	2	88	3	36	4	100
001	11	43	7	100	1	0	2	62
002	7	26	5	100	5	92	1	0
003	14	41	3	37	2	15	2	100
004	5	15	1	0	5	100	1	0
005	10	55	5	100	1	0	3	100
006	12	31	6	100	1	0	2	100
007	8	100	2	100	-	-	3	100
008	4	13	1	0	1	0	3	100
009	12	86	1	0	3	49	1	0
010	15	86	2	41	2	21	3	100
011	5	1	4	100	1	0	2	100
012	5	25	4	100	-	-	2	100
013	5	3	1	0	1	0	2	100
014	5	29	5	100	-	-	1	0
015	4	1	50	100	-	-	2	100
016	5	41	1	0	-	-	2	100
017	4	0	1	0	-	-	1	0
018	5	32	5	100	-	-	1	0
019	5	2	3	69	2	17	1	0

Table 1: Number of solution plans found and number of causal links to reject extracted from these solution plans. Dashed lines indicates that no solution plan was found for this problem during the learning phase.

The reported data show that the number of causal links to reject is not directly related with the number of solutions. A good example are problems `000` and `007` from the *openstack* domain. In the learning phase of problem `000` eight solutions were found producing 19 causal links while in problem `007`, with the same number of solutions, the learning phase produced more than 100 causal links. Precisely, the number of learned causal links for a given problem strongly depends on how different is the best plan found with respect to the rest of generated plans and the length of all these plans.

We can also observe that the learning time bound of 300 seconds is not enough for extracting a set of causal links to reject for every problem in the evaluation domains. In particular, the cases marked with a dashed line, for example problem `007` from the *nomystery* domain, indicate that no solution plan was found for this problem during the learning phase. Likewise when the number of solutions is 1 no causal link to reject can be extracted because our learning algorithm requires at least two plans to compute the set of causal links to reject, an example is problem `002` from the *visitall* domain.

Phase II: Compilation of the causal links to reject

At this point we evaluate the drawbacks of the new planning task resulting from the compilation of the set of causal links CL_{rej}. In particular we evaluate the increase in the size of the planning task resulting from the introduction of the causal links to reject. Note that this increase is somehow limited since only a maximum of 100 causal links is

compiled into the new planning task to deal with the *utility problem* (Minton 1988). For each problem the Table 2 illustrates the increase in memory and instantiation time caused by the use of the new planning task.

	Openstacks		**Parking**		**Nomystery**		**VisitAll**	
Prob	Mem	Time						
000	0.96	0.26	0.91	0.03	0.95	0.07	0.74	0.03
001	0.94	0.09	-	-	*	*	0.86	0.05
002	0.96	0.20	0.92	0.03	0.96	0.03	*	*
003	0.95	0.10	0.96	0.07	0.99	0.19	0.86	0.03
004	0.98	0.32	*	*	0.97	0.03	*	*
005	0.94	0.07	0.90	0.03	*	*	0.90	0.03
006	0.95	0.12	0.91	0.03	*	*	0.92	0.03
007	0.93	0.05	-	-	*	*	0.93	0.03
008	0.99	0.28	*	*	*	*	0.94	0.03
009	0.93	0.05	*	*	0.99	0.06	*	*
010	0.94	0.05	0.96	0.07	0.95	0.11	0.95	0.04
011	0.99	0.9	0.93	0.03	*	*	0.95	0.03
012	0.99	0.26	0.93	0.03	*	*	-	-
013	0.98	0.76	*	*	*	*	0.96	0.04
014	0.99	0.16	0.93	0.03	*	*	*	*
015	0.99	0.87	0.93	0.03	*	*	-	-
016	0.98	0.12	*	*	*	*	0.97	0.04
017	-	-	*	*	*	*	*	*
018	-	-	0.94	0.03	*	*	*	*
019	0.99	0.82	0.96	0.04	0.99	0.17	*	*

Table 2: Ratios of the memory and instantiation time required by the original and the new planning task resulting from the compilation. Stars indicate that no causal link was learned for this problem. Dashed lines indicate that the instantiation of the new planning task exceeded memory or time bounds.

The increase is shown by the ratio of these two values, memory and instantiation time, in the original planning task with respect to the new one. The values of the ratio are obtained with the prepossessing tools of the FAST DOWNWARD planning system (Helmert 2006). A value of 1 means that there is no increase caused by using the new planning task while a ratio under 1 means that the new task is more expensive, in terms of memory or instantiation time. The lower the ratio the more expensive the new task is. Please note that only the ratio of problems in which the learning phase extracted causal links is shown. Stars indicate that no causal link was learned for this problem because less than two solutions were found for this problem in the learning phase. Dashed lines indicate that new planning task was not instantiated successfully because it exceeded the memory or time bounds.

As expected the increase in the size of the planning task and the corresponding instantiation time depends on the number of causal links compiled into the new task. This effect can be observed looking again to problems `000` and `007` from the *openstack* domain. The reported data also shows that while the memory required by the new planning task is not far from the required by the original one, the instantiation time easily blows up growing, in most cases, in one order of magnitude. This observation suggest the study of further compilations of the learned causal links that hold better the instantiation time in the new planning task.

Phase III: Planning rejecting bad causal links

Finally we compare the planning performance of the winner of the last IPC, LAMA-2011 and our method for planning with the task resulting from compiling the learned causal links. Table 3 shows, for each problem, the quality of the best plan found by the two approaches in the following format: LAMA-2011/our proposed method. The dashed lines indicate that no causal link was learned for this particular problem so no comparison is shown.

Prob	**Openstacks**	**Parking**	**Nomystery**	**VisitAll**
000	7/7	61/37	18/18	181/179
001	15/12	31/31	-/-	260/260
002	9/9	42/44	25/25	-/-
003	20/19	62/60	29/29	410/410
004	17/14	-/-	34/34	-/-
005	20/22	41/42	-/-	604/604
006	11/10	44/45	-/-	763/706
007	34/31	75/75	-/-	860/860
008	30/34	-/-	-/-	1027/1027
009	32/33	-/-	52/50	-/-
010	32/31	67/67	18/18	1300/1300
011	100/100	50/66	-/-	1455/1455
012	48/47	62/62	-/-	1725/1769
013	128/128	-/-	-/-	1887/1895
014	76/74	52/54	-/-	-/-
015	155/155	47/50	-/-	2168/2168
016	107/105	-/-	-/-	2387/2387
017	189/190	-/-	-/-	-/-
018	137/138	50/51	-/-	-/-
019	221/221	74/74	48/48	-/-
Total	1388/1380	758/758	224/222	15027/15020

Table 3: Total cost of the best solution found by LAMA-2011 and our method, in the problems from the IPC-2011. Dashed lines indicate that no causal link was learned for this problem.

Results show that the *openstacks* is the more promising domain for the proposed method. In this domain our method actually improves the best plan cost in 9 out of 20 problems getting worse only in 5. There is no guarantee to always achieve the quality of the plans found with the original planning task. In some cases, like problem `005` from this domain, the quality achieved by the proposed method is worse that the one achieved by LAMA-2011. As we already observed the size of the planning task is increased by the introduction of the causal links so if the learned causal links are not effective for a particular problem this will cause larger planning times that might prevent the planner to reach the best plan cost achieved by the original planning task.

In the *parking* and *visitall* domains certain sets of learned causal links are able to significantly deteriorate the quality of the solutions found. In particular this is observed in problems in which the number of learned causal links is over 100. This observation suggests that better strategies for selecting the causal links to compile should be studied. Finally, despite the accumulated total cost is slightly better in the *nomystery* domain, in most of the problems of this domain the

cost of the best plan found by the two planning approaches is the same.

Related Work

Machine learning has been extensively used in planning to learn heuristics for faster search, to learn rules for selecting which planner from a given set to apply to a problem, and to learn domain models, but very rarely with the explicit aim of improving plan quality. The most significant work to our method is related to the learning of control rules for guiding the search of the PRODIGY planner towards good quality solutions (Pérez 1996).

There are recent work that introduces control knowledge into the domain model of a planning task. In particular (Baier and McIlraith 2008) shows how to compile procedural control knowledge, described in temporal logic, into a classical planning domain model. However, unlike the cited work, we propose a method not only to compile useful knowledge but to learn it from examples of solutions to the same problem. There are previous approaches that also succeed performing intra-problem learning for example, to automatically select the heuristic to compute (Domshlak, Karpas, and Markovitch 2012) or for creating macros to escape from plateaus (Coles and Smith 2007). Nevertheless these works modify the planner to exploit the learned knowledge while our method, based on a compilation of the planning task, does not need to modify the planner algorithms.

Finally, we also find previous works that enforce causal links to generate justified plans, in the case of automatic story telling (Haslum 2012) but again this knowledge is not automatically learned from examples.

Conclusions and future work

In this paper we have proposed a method that exploits the solutions found by an anytime planner to improve the quality of the subsequent ones. The method extracts a set of causal links from the first plans, the plans with worse quality, and compiles them into a more constrained definition of the planning task that rejects the creation of these causal links.

Despite the reported results show slight improvements of the quality of plans further research is needed to achieve more conclusive results. In particular several aspects of the three phases of the proposed method can be refined in order to improve the results. We observed that in some problems the proposed method was not able to learn anything because the anytime strategy was not able to find more than one solution. However, anytime strategies are not the only mechanism to produce a base of multiple plans with different quality. In practice solution plans with different quality can also be generated by introducing some randomization to the planner.

In addition the proposed compilation still causes high instantiation times when the number of learned causal links is high, examples are various problems from the evaluated *openstack*, *parking* and *visitall* domains. Further compilations have to be studied in order to hold instantiation time in these cases. Moreover, when the number of causal links is high, algorithms for choosing an effective subset of causal links to reject are necessary to improve the reported results.

References

Baier, J. A., and McIlraith, S. A. 2008. Planning with preferences. *AI Magazine* 29(4):25–36.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence* 69:165–204.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with online macro-action learning. *J. Artif. Intell. Res. (JAIR)* 28:119–156.

Coles, A. J.; Coles, A.; Olaya, A. G.; Jiménez, S.; López, C. L.; Sanner, S.; and Yoon, S. 2012. A survey of the seventh international planning competition. *AI Magazine* 33(1).

Domshlak, C.; Karpas, E.; and Markovitch, S. 2012. Online speedup learning for optimal planning. *J. Artif. Intell. Res. (JAIR)* 44:709–755.

Haslum, P. 2012. Narrative planning: Compilations to classical planning. *J. Artif. Intell. Res. (JAIR)* 44:383–395.

Helmert, M., and Röger, G. 2008. How good is almost perfect? In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, 944–949.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artif. Intell.* 143(2):219–262.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Boston, MA: Kluwer Academic Publishers.

Pérez, M. A. 1996. Representing and learning quality-improving search control knowledge. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML '96)*, 382–390.

Porteous, J., and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In *Proceedings of the ICAPS*.