Pruning bad quality causal links in sequential satisfying planning

ICAPS 2013 - Workshop on Planning and Learning

Sergio Jiménez [1] Patrik Haslum [2] Sylvie Thiebaux [2]

[1]Planning and Learning Group
Universidad Carlos III de Madrid

[2]Optimisation Group, NICTA

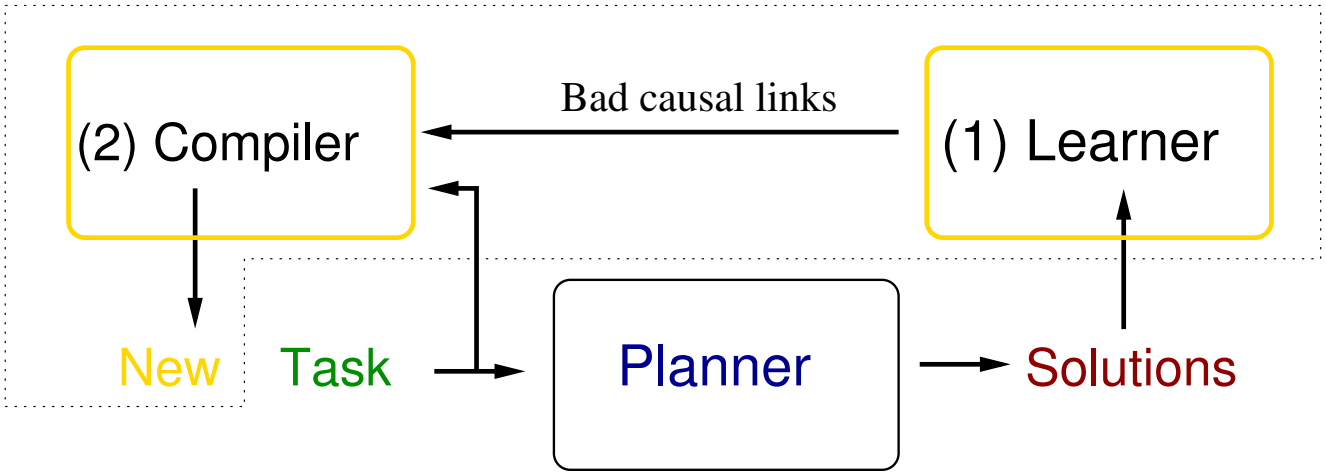
Sequential satisfycing planning

- **Problem**: initial state $s_0 \in S$ and goals $G \subseteq S$
- **Domain**: actions $a \in A$, $a = [pre(a), del(a), add(a), \mathbf{cost(a)}]$
- **Plan**: sequence of ground actions $\Pi = (a_1, a_2, ..., a_n)$
- **Optimization metric**: plan cost, $cost(\Pi) = \sum_{a_i \in \Pi} cost(a_i)$

Pruning bad quality causal links in seq. sat. planning

- **Lama-2011**: cost-bound restarts, [BFS, WA* (W=[5, 3, 2, 1])]
- **Restarts** are an intraproblem **learning opportunity**
- Plans with **different qualities** present **different causal links**
- Bad **causal links** represent bad **choices** made by the search process
- Bad causal links can be **pruned** in next planning episodes

Pruning bad quality causal links in seq. sat. planning



- **Lama-2011**: cost-bound restarts, [BFS, WA* (W=[5, 3, 2, 1])]
- **Restarts** are an intraproblem **learning opportunity**
- Plans with **different qualities** present **different causal links**
- Bad **causal links** represent bad **choices** made by the search process
- Bad causal links can be **pruned** in next planning episodes

Outline

Causal links

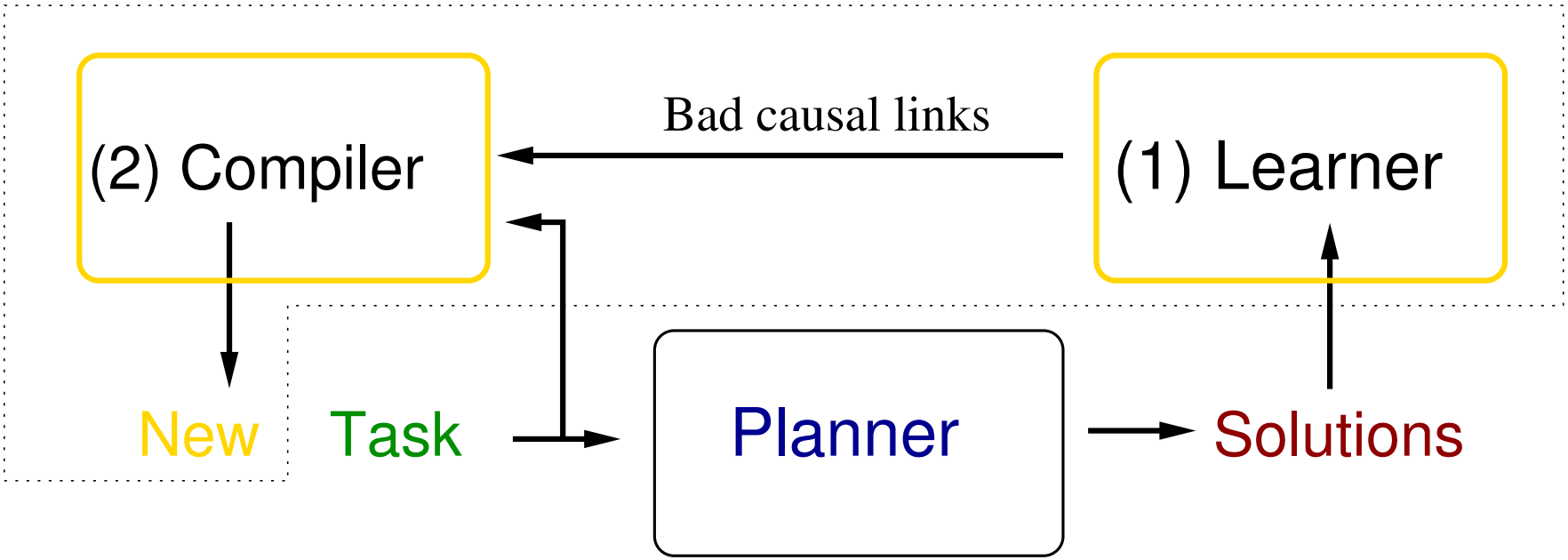
A **causal link** of a sequential satisf. plan Π is a tuple $<\mathbf{a_i}, \mathbf{a_j}, \mathbf{p}>$,

1. $\mathbf{a_i}$ (the **producer**) precedes $\mathbf{a_j}$ (the **consumer**) in the plan
   $a_i \in \Pi,\ a_j \in \Pi,\ i<j$

2. $\mathbf{p}$, **proposition** added by the producer and required by the consumer
   $p \in add(a_i), p \in pre(a_j)$

3. $\mathbf{p}$ is not threatened by some other action before the consumer occurs
   $\nexists k,\ a_k \in \Pi,\ i<k<j,\ p \in (del(a_k) \cup add(a_k))$

(1) Learning bad quality causal links



(1) We learn as **bad** causal links the **causal links** of the plans that are **not present** in the plan with the **best cost** found.

(1) Learning bad quality causal links

Input: planning task $\mathbf{\Pi}$ and planner **P** able to find different solution plans
Output: set of learned bad causal links $\mathbf{CL_{rej}}$

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

$\pi_{best} = \underset{\pi_i \in plans}{\text{argmin}}\, cost(\pi_i)$

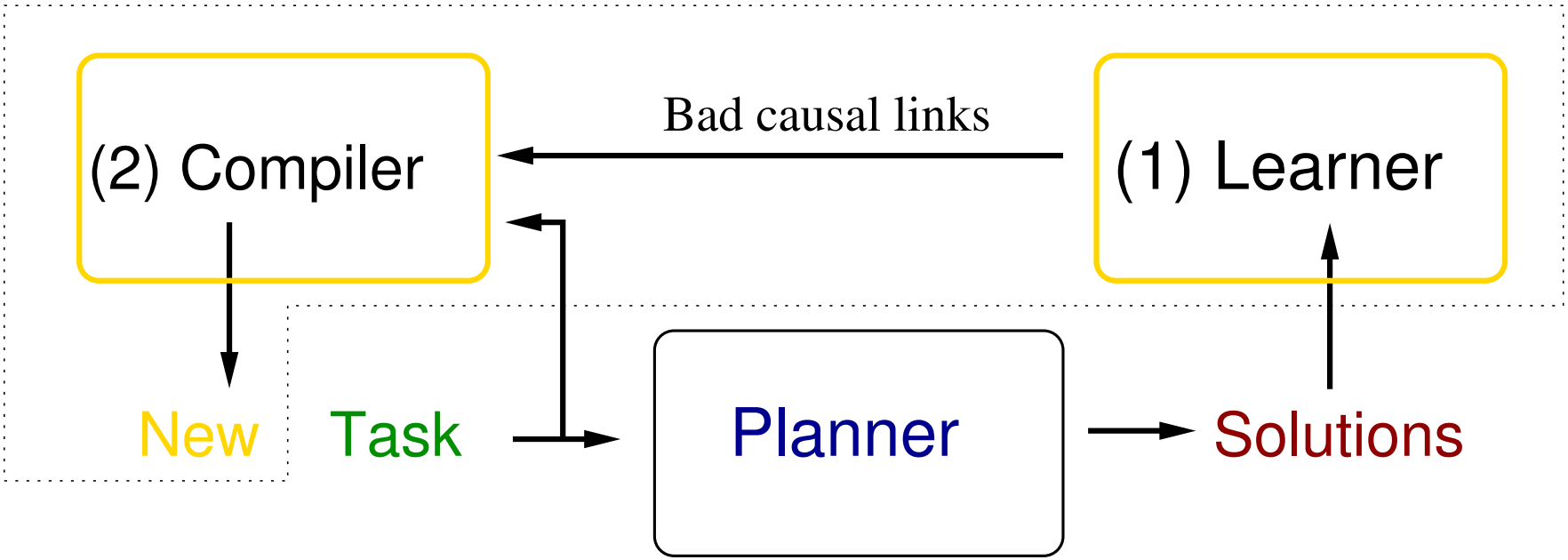
for $\pi_i \in plans$ and $\pi_i \neq \pi_{best}$ **do**
    $CL'_{\pi_i} = filter(CL_{\pi_i}, \pi_{best})$
end for

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

Figure: Algorithm for learning the *bad* causal links for a given task.

Outline

(2) Compilation of the learned knowledge



(2) The original planning **task** and the **bad causal links** are **compiled** into a new PDDL **problem** and a new PDDL **domain** that prune the plans containing the bad causal links.

(2) Compilation of the learned knowledge

For each causal link to prune $cl_{id} = <\mathbf{a_i}, \mathbf{a_j}, \mathbf{p}>$, $cl_{id} \in CL_{rej}$

1. A new object is included in the problem,

   `cLinkid - causalLink`

2. The initial state is extended with:

   2.1 a **fact** describing the **producer** of the causal link,

   `(isproducer-<`$name(a_i)$`>-of-<`$name(p)$`> cLinkid <args(`a_i`)>)`

   2.2 a **fact** describing the **consumer** of the causal link,

   `(isconsumer-<`$name(a_j)$`>-of-<`$name(p)$`> cLinkid <args(`a_j`)>)`

   2.3 for each threat of the causal link, a_k s.t. $p \in (del(a_k) \cup add(a_k))$

   2.3.1 a **fact** describing the **threat**

   `(isthreat-<`$name(a_k)$`>-for-<`$name(p)$`> cLinkid <args(p)>)`

(2) Compilation of the learned knowledge

```
(define (problem os-sequencedstrips-p10-1)
(:domain openstacks-sequencedstrips-adl)

(:objects
cLink1 - causalLink ;;; <(start-order o2 n1 n0),(open-new-stack n0 n1),(stacks-avail n0)>
cLink2 - causalLink ;;; <(open-new-stack n0 n1),(start-order o10 n1 n0),(stacks-avail n1)>
n0 - count n1 - count n2 - count n3 - count n4 ...)

(:init
;;; begin extension to the initial state - cLink1
(isproducer-start-order-of-stacks-avail clLink1 o2 n1 n0)
(isconsumer-open-new-stack-of-stacks-avail cLink1 n0 n1)
(isthreat-ship-order-for-stacks-avail cLink1 n0)
;;; end extension to the initial state - cLink1

;;; begin extension to the initial state - cLink2
(isproducer-open-new-stack-of-stacks-avail cLink2 n0 n1)
(isconsumer-start-order-of-stacks-avail cLink2 o10 n1 n0)
(isthreat-ship-order-for-stacks-avail cLink2 n1)
;;; end extension to the initial state - cLink2

(next-count n0 n1) (next-count n1 n2) (next-count n2 n3) (next-count n3 n4)...)

(:goal (and (shipped o1)(shipped o2)(shipped o3)(shipped o4)(shipped o5)(shipped
o6)(shipped o7)(shipped o8)(shipped o9)(shipped o10)))

(:metric minimize (total-cost)))
```

Figure: New problem from the *openstacks*, result of compiling 2 *bad* causal links.

(2) Compilation of the learned knowledge

1. New predicate for describing the **state** of the **causal links**

   `(clstarted ?clid - causalLink)`

2. For each causal link to prune $cl_{id} = <\mathbf{a_i}, \mathbf{a_j}, \mathbf{p}>$, $cl_{id} \in CL_{rej}$

   2.1 A quantified conditional **effect** is included in the **producer**

   ```
   (forall (?clid - causalLink)
    (when (and (isproducer-<name(a_i)>-of-<name(p)> ?clid <args(a_i)>))
     (and (clstarted ?clid))))
   ```

   2.2 A quantified **precondition** is included in the **consumer**

   ```
   (forall (?clid - causalLink)
    (or (not (isconsumer-<name(a_j)>-of-<name(p)> ?clid <args(a_j)>))
     (not (clstarted ?clid)))))
   ```

   2.3 For each threat a conditional **effect** is included in the **threat**

   ```
   (forall (?clid - causalLink)
    (when (and (isthreat-<name(a_k)>-for-<name(p)> ?clid <args(p)>))
     (and (not (clstarted ?clid)))))
   ```

(2) Compilation of the learned knowledge

Example of producer action after compiling the causal link

cl_{id} = <(ship-order o9 n0 n1),(start-order o7 n1 n0), stacks-avail(n1)>

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

(2) Compilation of the learned knowledge

Example of consumer action after compiling the causal link

cl_{id} = <(ship-order o9 n0 n1),(start-order o7 n1 n0),stacks-avail(n1)>

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

(2) Compilation of the learned knowledge

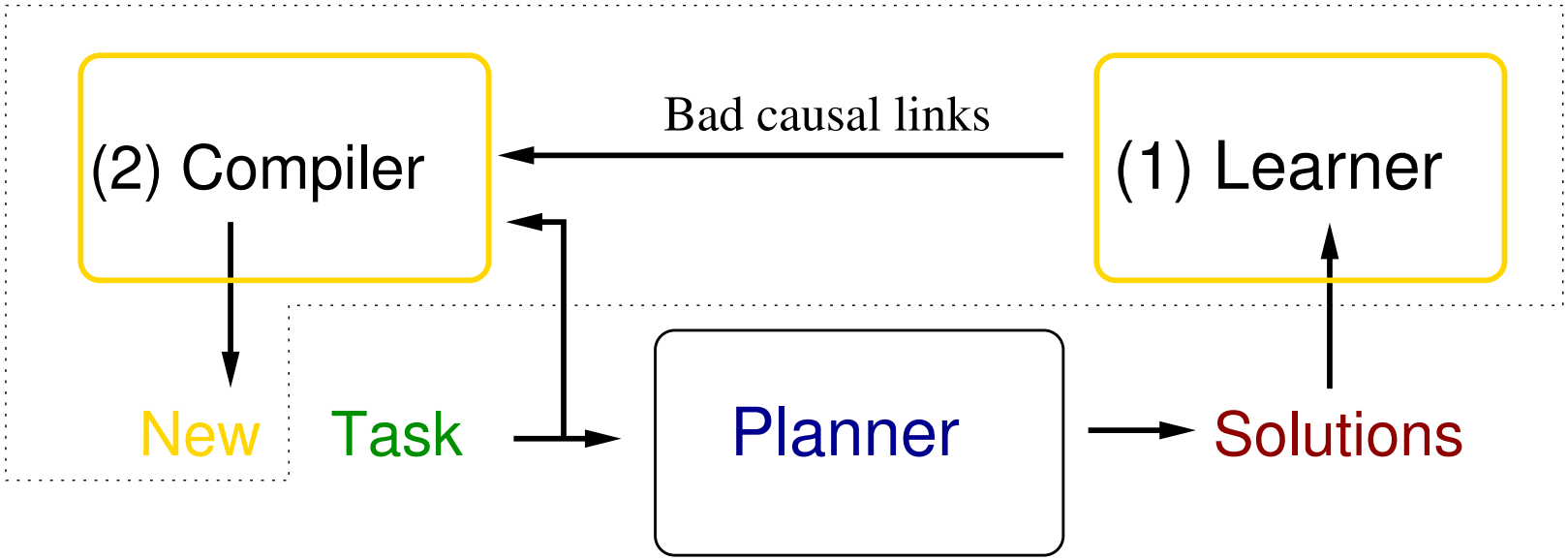
Example of threat action after compiling the causal link

cl_{id} = <(ship-order o9 n0 n1),(start-order o7 n1 n0),stacks-avail(n1)>

```
(: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)
  (forall (?clid - causalLink)
   (when (and (isthreat-open-new-stack-for-stacks-avail ?clid ?open))
    (and (not (clstarted ?clid))))))
(increase (total-cost) 1)))
```

Outline

(3) Planning with the learned knowledge



- Pruning bad quality causal links in seq-sat planning is **correct**
  - Initial state and goals from the original problem
  - New effects only affect the state of the causal links
  - New preconditions make action application more constrained
- . . . but is **not complete**
  - solutions better than the best one found could be pruned

Outline

Results - Experimental setup

lama-2011 (1800 secs) vs **our method** (300+1400 secs):

(1) Learning bad quality causal links
- Run LAMA-2011 (**300** secs)
- Compute the bad causal links

(2) Compilation of the bad causal links (maximum 100 per problem)
- No bad causal links above the best cost bound
- The most expensive preferred, $cost(cl_{id}) = cost(a_i) + cost(a_j)$

(3) Planning pruning the bad causal links
- Run LAMA-2011 (**1400** secs) on the new planning task
- Cost bound of the best plan found in the learning phase

Results - Effectiveness

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: Cost of the best plan found by LAMA-2011/our-method.

Results - Learning

	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: Plans found and bad causal links learned from them.

Results - Experimental compilation size

	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: Ratio $\frac{original}{new}$ planning task resulting from the compilation.

Conclusions

- The proposed method is correct but incomplete
- The compilation time is linear in the number of causal links
- The size of the new task grows exponential but keeps tractable
- Effectiveness of the method (preliminary results)
  1. Promising results in the *Openstacks* improving lama-2011 in 9 out of 20 problems
  2. More positive results conditioned to further research in the extraction of learning examples

Pruning bad quality causal links in sequential satisfying planning

ICAPS 2013 - Workshop on Planning and Learning

Sergio Jiménez [1] Patrik Haslum [2] Sylvie Thiebaux [2]

[1]Planning and Learning Group
Universidad Carlos III de Madrid

[2]Optimisation Group, NICTA


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


Huang, J. (2007).
The effect of restarts on the efficiency of clause learning.
In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 2318–2323.


Pipatsrisawat, K. and Darwiche, A. (2011).
On the power of clause-learning sat solvers as resolution engines.
Artif. Intell., 175(2):512–525.


Richter, S. and Westphal, M. (2010).
The lama planner: Guiding cost-based anytime planning with landmarks.
J. Artif. Intell. Res. (JAIR), 39:127–177.