

### Automatic Planning

Jörg Hoffmann

University of Innsbruck  
DERI

Masterstudium Informatik Vertiefungsmodul (WM 9)

- ▶ **Prelude**
- ▶ Ignoring Delete Lists
- ▶  $h^+$
- ▶ Approximating  $h^+$
- ▶ Implementation Issues
- ▶ A More General Viewpoint
- ▶ Multiple-Valued Variables
- ▶ Numeric Variables



Jörg Hoffmann

Automatic Planning

### Remark 1

We define heuristics  $h(I, G)$  that, given a task  $(P, A, I, G)$ , estimate the distance between  $I$  and  $G$ .

These can be used in search in the following ways:

1. Forward search: for each search state  $s$ , consider the task  $(P, A, s, G)$  and set  $h(s) := h(s, G)$
2. Backward search: for each search state  $s$ , consider the task  $(P, A, I, s)$  and set  $h(s) := h(I, s)$

Most of the time, we will assume  $(P, A, I, G)$  and write  $h$  without the “ $(I, G)$ ”



Jörg Hoffmann

Automatic Planning



Jörg Hoffmann

Automatic Planning

### Remark 2

Often, we have either of these:

1. Once  $h(I, G)$  is computed, adapting it for different  $I$  is easy
2. Once  $h(I, G)$  is computed, adapting it for different  $G$  is easy

**No:** pattern database –  $h(I, G)$  is abstract solution length and does not tell us anything about either  $h(I', G)$  or  $h(I, G')$

**Yes:**  $r^k$  and  $R^k$  have property 2 – compute table/planning graph for  $I$  and look up for different  $G$



Jörg Hoffmann

Automatic Planning

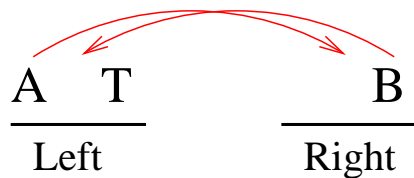
- ▶ Prelude
- ▶ Ignoring Delete Lists
- ▶  $h^+$
- ▶ Approximating  $h^+$
- ▶ Implementation Issues
- ▶ A More General Viewpoint
- ▶ Multiple-Valued Variables
- ▶ Numeric Variables

## Definition

Let  $(P, A, I, G)$  be a STRIPS task. For an action  $a \in A$ , the *relaxation* of  $a$  is  $a^+ = (pre(a), add(a), \emptyset)$ . We write  $A^+$  for  $\{a^+ \mid a \in A\}$ . A plan for  $(P, A^+, I, G)$  is called a *relaxed plan* for  $(P, A, I, G)$ .

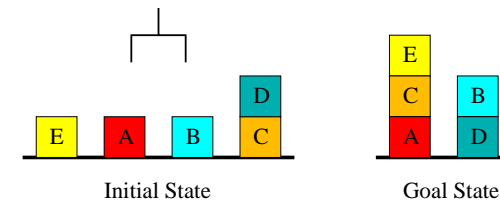
- ▶ All conditions are conjunctions of positive facts
- ▶ So the only effects that “hurt” are the delete effects
- ▶ We simplify the task by ignoring these
- ▶ Under relaxed actions, the set of true facts increases *monotonically*

## Example Logistics

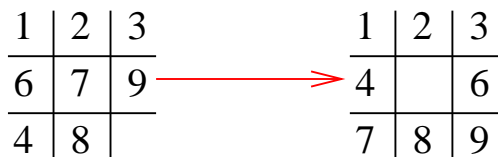


- ▶ Initial state:  $\{at(A, Left), at(B, Right), at(T, Left)\}$
- ▶  $result(I, \langle Drive(Left, Right)^+ \rangle) = ???$
- ▶  $result(I, \langle Drive(Left, Right)^+, Load(A, Left)^+ \rangle) = ???$

## Example Blocksworld



- ▶ Initial state:  $\{on(E, Table), clear(E), \dots, on(C, Table), on(D, C), clear(D), holding(NIL)\}$
- ▶  $result(I, \langle unstack(D, C)^+ \rangle) = \{on(E, Table), clear(E), \dots, on(C, Table), clear(C), holding(D), ???\}$
- ▶  $result(I, \langle unstack(D, C)^+, pickup(E)^+ \rangle) = \{holding(E), ???, \dots, on(C, Table), clear(C), holding(D), ???\}$



- ▶ Real: a tile can move from square A to square B if A is adjacent to B and B is blank
- ▶ Relaxed: a tile can move from square A to square B if A is adjacent to B and B is blank; in effect, ???

- ▶ For each variable  $v$  in the CNF, three facts  $v$ ,  $notv$ , and  $notsetv$ ; for each clause  $c$  in the CNF, one fact  $satc$
- ▶ Actions  $setvtrue : (\{notsetv\}, \{v\}, \{notsetv\})$  and  $setvfalse : (\{notsetv\}, \{notv\}, \{notsetv\})$
- ▶ Actions  $makesatc :$ 
  - ▶ Whenever  $v \in c : makesatc-byv : (\{v\}, \{satc\}, \emptyset)$
  - ▶ Whenever  $\neg v \in c : makesatc-bynotv : (\{notv\}, \{satc\}, \emptyset)$
- ▶ Initial state: ???
- ▶ Goal state: ???

Why does this encoding not work in the relaxation?



Definition

Let **PLANSAT<sup>+</sup>** denote the following decision problem. Given a STRIPS task  $(P, A, I, G)$ , is  $(P, A^+, I, G)$  solvable?

Proposition (Bylander, AI'94)

PLANSAT<sup>+</sup> is in **P**.

Proof.

$F := I$

while  $G \not\subseteq F$  do

$F' := F \cup \{p \in P \mid \exists a \in A : pre(a) \subseteq F, p \in add(a)\}$

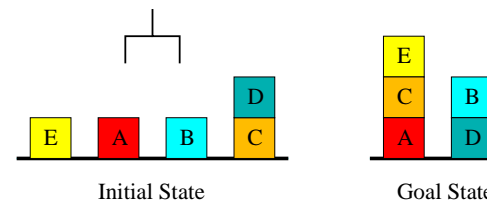
    if  $F' = F$  then output "Unsolvable" endif

$F := F'$

endwhile

output "Solvable"

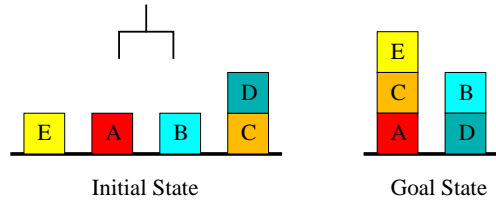
So what? See blackboard. □



A fraction:

1.  $\{on(E, Table), clear(E), on(A, Table), clear(A), on(B, Table), clear(B), on(C, Table), on(D, C), clear(D), holding(NIL)\}$
2.  $\{holding(E), holding(B), clear(C), \dots\}$
3.  $\{on(E, C), on(B, D), holding(C), \dots\}$
4.  $\{on(C, A), \dots\}$





Let's assume that *D* is not clear initially. The complete sets are:

1.  $\{on(E, Table), clear(E), on(A, Table), clear(A), on(B, Table), clear(B), on(C, Table), on(D, C), holding(NIL)\}$
2. ???
3. ???
4. ???

- ▶ Prelude
- ▶ Ignoring Delete Lists
- ▶  $h^+$
- ▶ Approximating  $h^+$
- ▶ Implementation Issues
- ▶ A More General Viewpoint
- ▶ Multiple-Valued Variables
- ▶ Numeric Variables



Really, what we want to know is the *length* of a relaxed plan!

**Definition**

Let  $(P, A, I, G)$  be a STRIPS task.  $h^+$  is the length of an optimal plan for  $(P, A^+, I, G)$ , or  $\infty$  if there is no such plan.

- ▶ We refer to an optimal plan for  $(P, A^+, I, G)$  as an **optimal relaxed plan** for  $(P, A, I, G)$

**Proposition**

Let  $(P, A, I, G)$  be a STRIPS task. If  $\langle a_1, \dots, a_n \rangle$  is a plan for  $(P, A, I, G)$ , then  $\langle a_1^+, \dots, a_n^+ \rangle$  is a plan for  $(P, A^+, I, G)$ .

**Proof.**

Exercise. □

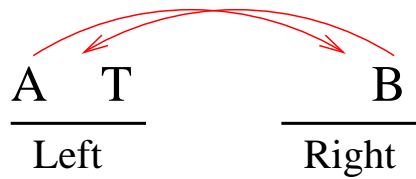
**Proposition**

Let  $(P, A, I, G)$  be a STRIPS task. Then  $h^+ \leq sd$ .

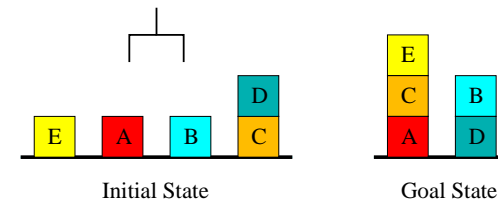
**Proof.**

???. □

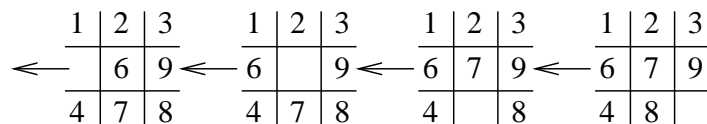
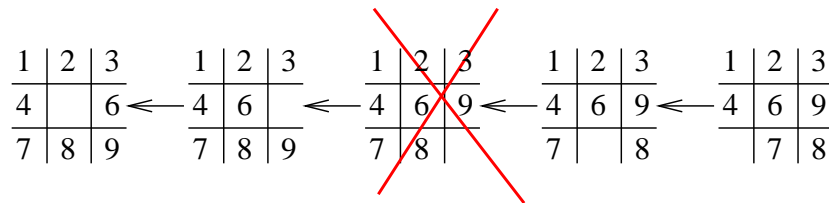




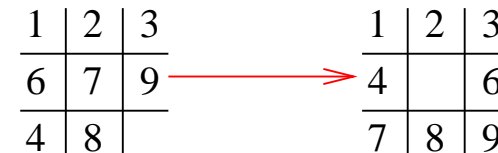
- ▶ Optimal plan:  $\langle \text{load}(A, T, \text{Left}), \text{drive}(T, \text{Left}, \text{Right}), \text{unload}(A, T, \text{Right}), \text{load}(B, T, \text{Right}), \text{drive}(T, \text{Right}, \text{Left}), \text{unload}(B, T, \text{Left}) \rangle$
- ▶ Optimal relaxed plan: ??? (is a sub-sequence of the above)
- ▶  $sd = 6, h^+ = ???$



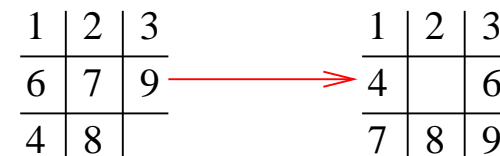
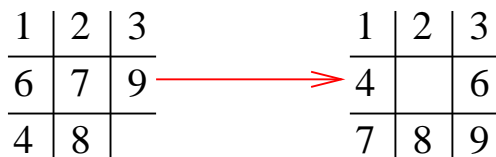
- ▶ Optimal plan:  $\langle \text{unstack}(D, C), \text{putdown}(D), \text{pickup}(B), \text{stack}(B, D), \text{pickup}(C), \text{stack}(C, A), \text{pickup}(E), \text{stack}(E, C) \rangle$
- ▶ Optimal relaxed plan: ??? (is a sub-sequence of the above)
- ▶  $sd = 8, h^+ = ???$



The skipped step moves tile 8 *back* to its position.



- ▶ “A tile can move from A to B if A is adjacent to B and B is blank”  $\Rightarrow$  solution distance
- ▶ “A tile can move from A to B if A is adjacent to B”  $\Rightarrow$  manhattan distance heuristic  $h^{MD}$
- ▶ “A tile can move from A to B if A is adjacent to B and B is blank; in effect, the tile is at both A and B, and both A and B are blank”  $\Rightarrow h^+$



- ▶ Optimal MD plan:  $\langle move(t_9, p_6, p_9), move(t_7, p_5, p_8), move(t_6, p_4, p_5), move(t_6, p_5, p_6), move(t_4, p_7, p_4), move(t_7, p_8, p_7) \rangle$
- ▶ Optimal no-deletes plan:  $\langle move(t_9, p_6, p_9), move(t_8, p_8, p_9), move(t_7, p_5, p_8), move(t_6, p_4, p_5), move(t_6, p_5, p_6), move(t_4, p_7, p_4), move(t_7, p_8, p_7) \rangle$
- ▶ So  $h^+ = 7$  and  $h^{MD} = 6$  ( $sd = 8$ )

$h^+$  dominates the Manhattan Distance heuristic:

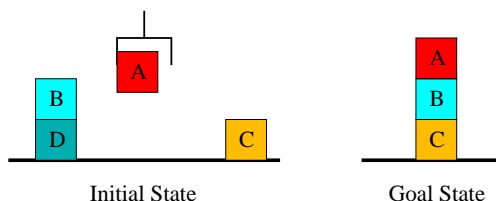
- ▶ The goal is given as the conjunction of the  $at(t_i, p_i)$  facts
- ▶ Achieving each single one of these facts takes at least as many steps as the respective tile's Manhattan Distance: we have to make the respective horizontal and vertical moves

We have just seen that  $h^+$  strictly dominates  $h^{MD}$ !



And, by the way...

An optimal relaxed plan can *not* always be obtained by skipping actions from the optimal (real) plan:



- ▶ Optimal plan:  $\langle putdown(A), unstack(B, D), stack(B, C), pickup(A), stack(A, B) \rangle$
- ▶ Optimal relaxed plan: ???



And, by the way...

An artificial example:

- ▶  $P = \{x, y, z\}, I = \{z\}, G = \{y, z\}$
- ▶  $A = \{a_1 : (\emptyset, \{x\}, \emptyset), a_2 : (\{x\}, \{y\}, \emptyset), a_3 : (\emptyset, \{y\}, \{z\})\}$
- ▶ Optimal plan: ???
- ▶ Optimal relaxed plan: ???

No heuristic is fail-safe! What counts is the behaviour in “the kind of examples we want to solve in practice”

Ideally, we want to know precisely in what classes of examples the heuristic is good  $\Rightarrow$  we'll see later some such results for  $h^+$



## Definition

Let *Bounded-PLANSAT<sup>+</sup>* denote the following decision problem. Given a STRIPS task  $(P, A, I, G)$  and an integer  $b$ . Is there a relaxed plan with at most  $b$  actions?

By computing optimal relaxed plans, we would solve Bounded-PLANSAT<sup>+</sup>.

## Theorem (Bylander, AI'94)

Bounded-PLANSAT<sup>+</sup> is **NP**-complete.

## Proof.

Blackboard. □

- ▶ Prelude
- ▶ Ignoring Delete Lists
- ▶  $h^+$
- ▶ **Approximating  $h^+$**
- ▶ Implementation Issues
- ▶ A More General Viewpoint
- ▶ Multiple-Valued Variables
- ▶ Numeric Variables



*Use the information obtained by our previous solvability algorithm.*

```

F := I
while G ⊄ F do
  F' := F ∪ {p ∈ P | ∃a ∈ A : pre(a) ⊆ F, p ∈ add(a)}
  if F' = F then output “Unsolvable”
  F := F'
endwhile
output “Solvable”

```



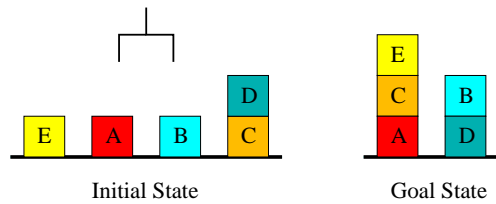
*Approximate  $h^+$  by the number of iterations until we have  $G ⊆ F$ .*

- ▶ Admissible: always  $\leq h^+$  because, if there is a relaxed plan of length  $n$ , then  $G ⊆ F$  in iteration  $n$  (this was a proof argument)
- ▶ *Far too optimistic*: it applies all actions “in parallel”
- ▶ Actually this is the same as: ??? (see also later)

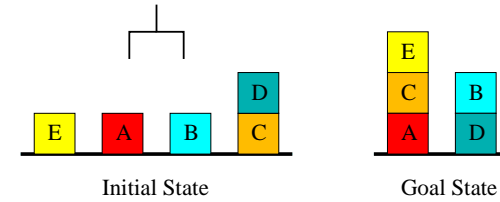
*Approximate  $h^+$  by the number of actions that appeared until we had  $G ⊆ F$ .*

- ▶ *Far too pessimistic*: it applies all actions that can possibly be applied





- ▶  $sd = 8, h^+ = 7, \text{estimate} = 3$
- ▶  $\langle \text{pickup}(E) - \text{pickup}(B) - \text{unstack}(D, C) - \dots, \text{stack}(E, C) - \text{stack}(B, D) - \text{pickup}(C) - \dots, \text{stack}(C, A) - \dots \rangle$



- ▶  $sd = 8, h^+ = 7, \text{estimate} = 46$
- ▶  $\langle \text{pickup}(E), \text{pickup}(A), \text{pickup}(B), \text{unstack}(D, C), \text{putdown}(E), \text{putdown}(A), \text{putdown}(B), \text{putdown}(D), \text{stack}(E, A), \text{stack}(E, B), \dots, \text{stack}(D, C), \text{pickup}(C), \text{unstack}(E, A), \text{unstack}(E, B), \dots, \text{unstack}(D, B), \text{putdown}(C), \text{stack}(C, E), \dots, \text{stack}(C, D) \rangle$

Approximating  $h^+$ , Idea Nr. 2

Assume that there are no positive interactions, i.e. that all facts have to be achieved separately [Bonet et al, AAAI'97]:

$$r^{add}(s) = \begin{cases} 0 & s \subseteq I \\ 1 + \min_{a \in A, p \in \text{add}(a)} r^{add}(\text{pre}(a)) & s = \{p\} \\ \sum_{p \in s} r^{add}(\{p\}) & |s| > 1 \end{cases}$$

- ▶ For STRIPS task  $(P, A, I, G)$ , we set  $r^{add} = r^{add}(G)$
- ▶ This is a *pessimistic* simplification; it simplifies the computation of the heuristic, *not* the solving of the task
- ▶ This is identical to ??? except that ???

Computing  $r^{add}$ 

$r^{add}(\{p\}) := 0$  for  $p \in I$ ,  $r^{add}(\{p\}) := \infty$  for  $p \notin I$   
changes := TRUE

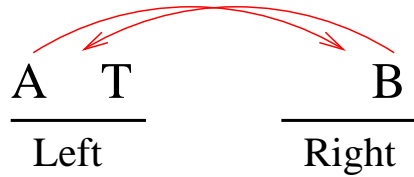
```

while changes do
  changes := FALSE
  for all a in A, p in add(a) do
    if 1 + sum_{p' in pre(a)} r^{add}({p'}) < r^{add}({p}) then
      r^{add}({p}) := 1 + sum_{p' in pre(a)} r^{add}({p'})
      changes := TRUE
    endif
  endfor
endwhile

```

Note:  $r^{add}(p)$  may change more than once! (earlier action may have more preconditions than later one: exercise)





Blackboard.

- ▶  $r^{add} = 6 > h^+ = 5$
- ▶  $r^{add} = sd$ , but for the wrong reason:  $drive(Left, Right)$  is counted twice
- ▶ If we had 100 packages, it would be counted 100 times

An artificial example:

- ▶  $P = \{x_1, \dots, x_n\}, I = \emptyset, G = \{x_1, \dots, x_n\}$
- ▶  $A = \{(\emptyset, \{x_1, \dots, x_n\}, \emptyset)\}$
- ▶  $r^{add}(\{x_i\}) = ?, r^{add}(G) = ?$
- ▶ Optimal plan length =  $h^+ = ?$

As said, every heuristic can be fooled ...

But  $r^{add}$  over-estimates in many natural examples, and we can easily do better!

## Approximating $h^+$ , Idea Nr. 3

“Relaxed Planning:”

*Find some, not necessarily optimal, relaxed plan for  $(P, A, I, G)$ , and take the length of that relaxed plan as the estimate for  $h^+$  [Hoffmann&Nebel,JAIR'01].*

- ▶ The relaxed plan may take account of positive interactions

Two-step process to compute  $h^{FF}$ :

1. Chain forward to build a relaxed planning graph (RPG)
2. Chain backward to extract a relaxed plan from the RPG

## Relaxed Planning Graphs

### Proposition

Let  $(P, A, I, G)$  be a STRIPS task. For any  $k$ , in a  $k$ -planning graph to  $(P, A^+, I, G)$  all sets  $EF_t$  and  $EA_t$  are empty.

### Proof.

Exercise. □

So we use:

```

 $F_0 := I, t := 0$ 
while TRUE do
   $A_t := \{a \in A^N \mid pre(a) \subseteq F_t\}$ 
   $F_{t+1} := \bigcup_{a \in A_t} add(a)$ 
  if  $F_{t+1} = F_t$  then stop; endif
   $t := t + 1$ 
endwhile
    
```

- ▶ Yes, this is a 1-planning graph
- ▶ Intuitive reason: if you care about single facts only, then delete lists don't matter; if you don't have delete lists, then there are no harmful interactions between facts
- ▶ It is also basically the same as Bylander's algorithm for deciding relaxed solvability
- ▶ To extract a relaxed plan, it would suffice to stop as soon as  $G \subseteq F_t$

We obtain from the RPG:

- ▶ For  $p \in P$ ,  $level(p) := \min\{t \mid p \in F_t\}$
- ▶ For  $a \in A$ ,  $level(a) := \min\{t \mid a \in A_t\}$
- ▶  $maxlevel := \min\{t \mid G \subseteq F_t\}$  where the minimum over an empty set is  $\infty$

That is:  $level(p) = R^1(\{p\})$ ,  $level(a) = R^1(pre(a))$ ,  
 $maxlevel = R^1(G)$

## Extracting a Relaxed Plan, II

## Example Logistics

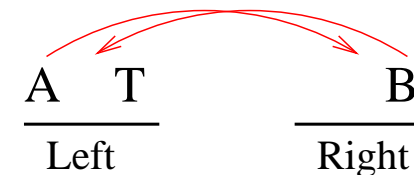
If  $maxlevel < \infty$ :

```

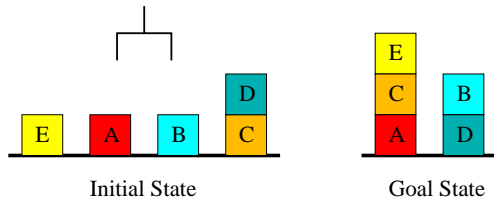
for  $t := 0, \dots, maxlevel$  do
   $G_t := \{g \in G \mid level(g) = t\}$ 
endfor
for  $t := maxlevel, \dots, 1$  do
  for all  $g \in G_t$  do
    select  $a$ ,  $level(a) = t - 1$ ,  $g \in add(a)$ 
    for all  $p \in pre(a)$  do
       $G_{level(p)} \cup = \{p\}$ 
    endfor
  endfor
endfor

```

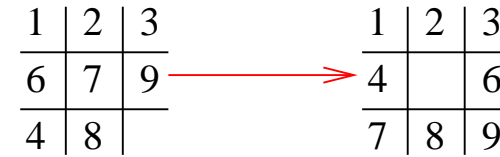
== Parallel regression with relaxed actions; no backtracking since using the RPG we are sure to reach the initial state



Blackboard.



- 0.  $on(B, Table), clear(B), \dots, on(C, Table), on(D, C), clear(D), holding(NIL)$   
 $pickup(E), pickup(B), unstack(D, C)$
- 1.  $holding(E), holding(B), clear(C), \dots$   
 $stack(E, C), stack(B, D), pickup(C)$
- 2.  $on(E, C), on(B, D), holding(C), \dots$   
 $stack(C, A)$
- 3.  $on(C, A), \dots$



- 0.  $at(t_9, p_6), at(t_8, p_8), blank(p_9), \dots$   
 $move(t_9, p_6, p_9), move(t_8, p_8, p_9)$
- 1.  $at(t_9, p_9), blank(p_6), blank(p_8), \dots$   
 $move(t_7, p_5, p_8)$
- 2.  $at(t_7, p_8), blank(p_5), \dots$   
 $move(t_6, p_4, p_5)$
- 3.  $at(t_6, p_5), blank(p_4), \dots$   
 $move(t_6, p_5, p_6), move(t_4, p_7, p_4)$
- 4.  $at(t_6, p_6), at(t_4, p_4), blank(p_7), \dots$   
 $move(t_7, p_8, p_7)$
- 5.  $at(t_7, p_7), \dots$



## Fooling $h^{FF}$

$h^{FF}$  can still over-estimate even the real solution distance:

- ▶  $P = \{x_1, \dots, x_n\}, I = \emptyset, G = \{x_1, \dots, x_n\}$
- ▶  $A = \{a_{all} : (\emptyset, \{x_1, \dots, x_n\}, \emptyset), a_1 : (\emptyset, \{x_1\}, \emptyset), \dots, a_n : (\emptyset, \{x_n\}, \emptyset)\}$
- ▶ All actions appear in  $A_0$
- ▶ Relaxed plan extraction might choose to ???
- ▶ Well we might design some simple selection rules to avoid that...



## Fooling it Some More

Here,  $h^{FF}$  will over-estimate the real solution distance:

- ▶  $P = \{x_1, \dots, x_n, y\}, I = \emptyset, G = \{x_1, \dots, x_n\}$
- ▶  $A = \{a_{all} : (\{y\}, \{x_1, \dots, x_n\}, \emptyset), a_y : (\emptyset, \{y\}, \emptyset), a_1 : (\emptyset, \{x_1\}, \emptyset), \dots, a_n : (\emptyset, \{x_n\}, \emptyset)\}$
- ▶  $a_{all}$  appears the first time in  $A_{???}$ , but the  $a_i$  appear in  $A_{???}$
- ▶  $h^+(I) = sd(I) = ?, h^{FF}(I) = ?$

*In practical examples, over-estimation by  $h^{FF}$  is extremely seldom!*



- $h^+$  Length of an optimal relaxed plan; admissible; NP-hard to compute
- $r^{add}$  Assume that all facts must be achieved completely independently; not admissible, over-estimates often; compute by forward chaining over values for single facts
- $h^{FF}$  Approximate  $h^+$  by length of some relaxed plan; not admissible, over-estimates rarely; compute by forward chaining (RPG) and backward chaining (relaxed plan extraction)

Use  $h^{FF}$  to find (hopefully) fast a plan without optimality guarantee; most currently successful “sub-optimal” systems do variations of this.

- ▶ Prelude
- ▶ Ignoring Delete Lists
- ▶  $h^+$
- ▶ Approximating  $h^+$
- ▶ **Implementation Issues**
- ▶ A More General Viewpoint
- ▶ Multiple-Valued Variables
- ▶ Numeric Variables



Both  $h^{FF}$  and  $r^{add}$  have “property 2”: once computed for  $I$ , computing them for different  $G$  is easy.

- ▶ Build the RPG just once
- ▶ For any backward search state  $s$ , ???
- ▶ Compute  $r^{add}$  for the single facts just once
- ▶ For any backward search state  $s$ , ??? [Bonet & Geffner, ECP-1999]



Turn this upside-down [Refanidis & Vlahavas, ECP'99]:

- ▶ Backwards from the goals, compute an  $r^{add}$ -style estimate of *goal* distance for the single facts just once
- ▶ Annotate the distance estimates with sets of “related facts”
- ▶ For any forward search state  $s$ , use sum of  $p \in s$  estimate minus some bonus given by related facts

Yes, it's a hack, and somewhat questionable outside the set of domains that Refanidis & Vlahavas used for testing ...

Even more generally, systems (searching forward and) re-computing  $r^{add}$ /the RPG tend to be faster



1. Schedule all  $p \in I$  for activation
  2. If all  $p \in G$  are activated or scheduled for activation, stop
  3. Activate scheduled facts; when activating  $p$ , increment  $cnt(a)$  for all  $a$  with  $p \in pre(a)$ ; if  $cnt(a) = |pre(a)|$ , schedule  $a$  for activation
  4. Activate scheduled actions; when activating  $a$ , schedule all yet non-activated  $p \in add(a)$  for activation
  5. Goto 2
- ▶ Keep pointers, e.g. from  $p$  to all  $a$  with  $p \in pre(a)$
  - ▶ Main bottleneck: 10000s of actions in large tasks
  - ▶ Therefore it is essential to not ask “ $pre(a) \subseteq F_t$ ?” for all  $a$  and  $t$

- ▶ Prelude
- ▶ Ignoring Delete Lists
- ▶  $h^+$
- ▶ Approximating  $h^+$
- ▶ Implementation Issues
- ▶ **A More General Viewpoint**
- ▶ Multiple-Valued Variables
- ▶ Numeric Variables



### Definition

Let  $P$  be a set of facts,  $L = P \cup \{\neg p \mid p \in P\}$ . A *negSTRIPS* action  $a$  is a triple  $a = (pre(a), add(a), del(a))$ , where  $pre(a) \subseteq L$ ,  $add(a), del(a) \subseteq P$ , and  $add(a) \cap del(a) = \emptyset$ . A *negSTRIPS* task is a tuple  $(P, A, I, G)$  where  $A$  is a (finite) set of STRIPS actions,  $I \subseteq P$ , and  $G \subseteq L$ .

- ▶ We allow negation in preconditions and goal
- ▶ Means: we can now require that certain facts are *false*
- ▶ “negSTRIPS” is not a word used in the literature



- ▶ For each variable  $v$  in the CNF, fact  $v$ ; for each clause  $c$  in the CNF, one fact  $satc$
- ▶ Actions  $setvtrue : (\{\neg v\}, \{v\}, \emptyset)$
- ▶ Actions  $makesatc$  :
  - ▶ Whenever  $v \in c$ :  $makesatc-byv : (\{v\}, \{satc\}, \emptyset)$
  - ▶ Whenever  $\neg v \in c$ :  $makesatc-bynotv : (\{\neg v\}, \{satc\}, \emptyset)$
- ▶ Initial state:  $\emptyset$
- ▶ Goal state:  $\{satc \mid \text{for all } c\}$

*What does this tell us about ignoring deletes in the presence of negations?*





With negation, ignoring delete lists does not necessarily simplify the task:

- ▶  $P = \{x, y\}, I = \{x\}, G = \{y\}$
- ▶  $A = \{a_{notx} : (\emptyset, \emptyset, \{x\}), a_y : (\{-x\}, \{y\}, \emptyset)\}$
- ▶ Plan: ???
- ▶ Relaxed plan: ???

Compilation:

- ▶  $P = \{x, y\}, I = \{x\}, G = \{y\},$   
 $A = \{a_{notx} : (\emptyset, \emptyset, \{x\}), a_y : (\{-x\}, \{y\}, \emptyset)\}$
- ▶  $P = ???, I = ???, G = ???,$   
 $A = \{a_{notx} : (???, ???, ???), a_y : (???, ???, ???)\}$
- ▶ Plan for compiled task: ???
- ▶ Relaxed plan for compiled task: ???

In general:

- ▶  $NOTx$  is initially true iff  $x$  is initially *FALSE*
- ▶  $NOTx$  is added whenever  $x$  is deleted
- ▶ So, in the relaxation,  $NOTx$  is *TRUE* iff we were able to delete  $x$  yet, or  $x$  was *FALSE* initially

*In the relaxation,  $x$  stores whether we reached  $x = 1$  yet;  $NOTx$  stores whether we reached  $x = 0$  yet!*

- ▶ Ignoring delete lists in a task with compiled negation means to:
  - Relax the task by assuming that each state variable, once it had a value  $v$ , keeps that value (amongst other values) forever.*
- ▶ The value of a variable  $v$ , in the relaxation, is the **accumulated subset**  $AS(v)$  of its domain
- ▶ Represented by  $x$  and  $NOTx$  (so far)
- ▶ State transitions only ever *enlarge* the accumulated subsets: e.g. in the compiled example  $a_{notx}$  sets  $AS(x)$  from ??? to ???
- ▶ That's why we call this the **monotonicity** relaxation

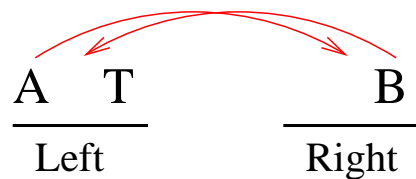
- ▶ Prelude
- ▶ Ignoring Delete Lists
- ▶  $h^+$
- ▶ Approximating  $h^+$
- ▶ Implementation Issues
- ▶ A More General Viewpoint
- ▶ **Multiple-Valued Variables**
- ▶ Numeric Variables

- ▶ Variables  $v$  with finite domain  $dom(v)$ , actions  $v_1 = c_1 \wedge \dots \wedge v_k = c_k \longrightarrow v_1 := c_1, \dots, v_l := c_l$

- ▶ Relaxation:

$$c_1 \in AS(v_1) \wedge \dots \wedge c_k \in AS(v_k) \longrightarrow AS(v_1) := AS(v_1) \cup \{c_1\}, \dots, AS(v_l) := AS(v_l) \cup \{c_l\}$$

- ▶ In STRIPS with negation, this is *ignore-deletes(compilation(task))*
- ▶ Without negation, the  $AS(v) := AS(v) \cup \{0\}$  operations don't make any difference



- ▶ Variables:  $at(A), at(B) : \{Left, Right, T\}, at(T) : \{Left, Right\}$
- ▶ Actions:
  - $drive(t, l, l') : at(t) = l \longrightarrow at(t) := l'$
  - $load(p, t, l) : at(p) = l \wedge at(t) = l \longrightarrow at(p) := t$
  - $unload(p, t, l) : at(p) = t \wedge at(t) = l \longrightarrow at(p) := l$

Relaxed plan: blackboard.

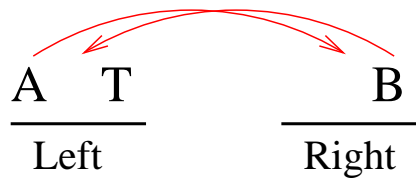


Denote:  $I(v)$  initial value of  $v$ ;  $G(v)$  set of goal requirements on  $v$  (one or none);  $pre(a)(v)$  set of precondition requirements made by  $a$  on  $v$  (one or none);  $eff(a)(v)$  set of effect assignments made by  $a$  to  $v$  (one or none);  $NOOP(v, c) : v = c \longrightarrow v := c$ .

```

For all  $v, AS_0(v) := \{I(v)\}$ 
 $t := 0$ 
while TRUE do
   $A_t := \{a \in A^N \mid \forall v : pre(a)(v) \subseteq AS_t(v)\}$ 
  For all  $v, AS_{t+1}(v) := \bigcup_{a \in A_t} eff(a)(v)$ 
  if  $AS_{t+1} = AS_t$  then stop endif
endwhile
    
```



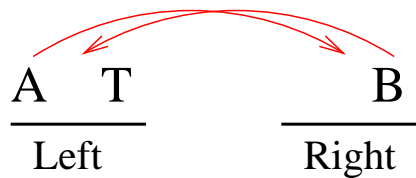


Blackboard.

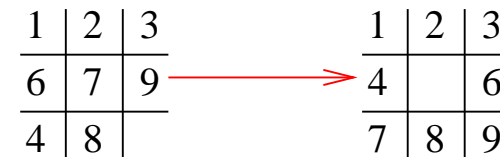
$level(v, c) := \min\{t \mid c \in AS_t(v)\}$ ,  $level(a) := \min\{t \mid a \in A_t\}$ ,  
 $maxlevel := \min\{t \mid \forall v : G(v) \subseteq AS_t(v)\}$

```

for  $t := 0, \dots, maxlevel$  do
     $G_t(v) := \{c\}$  if  $c \in G(v)$  and  $level(v, c) = t$ ,  $G_t(v) := \emptyset$  otherwise
endfor
for  $t := maxlevel, \dots, 1$  do
    for all  $v$  and  $c \in G_t(v)$  do
        select  $a$ ,  $level(a) = t - 1$ ,  $c \in eff(a)(v)$ 
        for all  $v'$  and  $c' \in pre(a)(v')$  do
             $G_{level(v', c')}(v') \cup = \{c'\}$ 
        endfor
    endfor
endfor
    
```



Blackboard.



- ▶  $t_i, blank : \{p_1, \dots, p_9\}$
- ▶  $move(t_i, p_j, p_k) : p_j \in AS(t_i) \wedge p_k \in AS(blank) \longrightarrow$   
 $AS(t_i) := AS(t_i) \cup \{p_k\}$ ,  $AS(blank) := AS(blank) \cup \{p_j\}$

Relaxation, here and in Logistics, effectively the same as before (in STRIPS). *Does this hold in general?*

In a linear (STRIPS) encoding of a Multiple-Valued STRIPS task, there is a one-to-one relationship between optimal relaxed plans in both tasks.

- ▶ Original task:  $v = c?$ ,  $v := c'$
- ▶ Linear encoded STRIPS task:  $(v = c)?$ ,  $\text{ADD } (v = c')$
- ▶ Adding new variable values corresponds to adding new facts

Is this also true in a logarithmic encoding?

- ▶ Multiple-Valued task:  $x : \{0, 1, \dots, 15\}$ ,  $\text{inc}_7 : x = 7 \rightarrow x := 8$  (15 actions of this kind, for  $0, 1, \dots, 14$ )
- ▶ Linear STRIPS encoding:  $(x = 0), (x = 1), \dots, (x = 15)$ ,  $\text{inc}_7 : (x = 7) \rightarrow \text{ADD } (x = 8) \text{ DEL } (x = 7)$
- ▶ Logarithmic STRIPS encoding:  $(Fbit_0), (Tbit_0), \dots, (Fbit_3), (Tbit_3)$ ,  $\text{inc}_7 : (Tbit_0) \wedge (Tbit_1) \wedge (Tbit_2) \wedge (Fbit_3) \rightarrow \text{ADD } (Fbit_0), (Fbit_1), (Fbit_2), (Tbit_3) \text{ DEL } (Tbit_0), (Tbit_1), (Tbit_2), (Fbit_3)$

Say  $x = 7$  initially, and  $x = 15$  is the goal. What happens in the different encodings, in the relaxation, if we execute  $\text{inc}_7$ ?

$x = 7$  initially, and  $x = 15$  is the goal; we execute  $\text{inc}_7^+$

- ▶ Multiple STRIPS:  $AS(x) = ???$
- ▶ Linear STRIPS:  $AS(x = 0) = ???$ ,  $AS(x = 7) = ???$ ,  $AS(x = 8) = ???$ ,  $AS(x = 15) = ???$
- ▶ Logarithmic STRIPS:  $AS(Tbit_0) = ???$ ,  $AS(Tbit_1) = ???$ ,  $AS(Tbit_2) = ???$ ,  $AS(Tbit_3) = ???$
- ▶ If  $x = 0$  initially, what is  $h^+$  in logarithmic? Exercise!
- ▶ With a logarithmic encoding, ignoring deletes can be exponentially less precise than with a linear one!

- ▶ The stuff we've seen for STRIPS generalizes very naturally to finite-domain variables
- ▶ Actually, at least in linear encodings, we were "already doing" the extended algorithms, in an awkward representation
- ▶ Doing the Multiple-Valued representation may (?) spare some runtime . . .
- ▶ (actually, it would have been better for us if the logicians hadn't infected us with their "facts" in the first place)
- ▶ . . . but it gets more exciting if we move on to numeric variables

- ▶ Prelude
- ▶ Ignoring Delete Lists
- ▶  $h^+$
- ▶ Approximating  $h^+$
- ▶ Implementation Issues
- ▶ A More General Viewpoint
- ▶ Multiple-Valued Variables
- ▶ **Numeric Variables**

- ▶ STRIPS extended with numeric (real-valued) state variables
- ▶ In addition to the facts  $P$  we have a set  $V$  of numeric variables
- ▶ An expression is formed by numeric variables  $v$ , constants  $c$ , and the  $+$ ,  $-$ ,  $*$ ,  $/$  connectives
- ▶ Expressions can be compared with  $<$ ,  $\leq$ ,  $=$ ,  $\geq$ ,  $>$
- ▶ Comparisons of expressions are allowed in action preconditions and the goal
- ▶ Action effects of the form  $v := \text{exp}(\bar{v})$  are allowed
- ▶ Even with finite-range numeric vars, this is more complicated than Multiple-Valued STRIPS due to the more complex condition formulas and effects

## A Very Simple Example

The relaxation is still that variables keep their old values:

- ▶  $V = \{x, y\}, I = \{x \mapsto 0, y \mapsto 0\}, G = \{y \geq 1\}$
- ▶  $a_x : \rightarrow x := x + 1, y := y - 1,$   
 $a_y : x \geq 2 \rightarrow y := y + 1$
- ▶ Real plan: ???
- ▶ Relaxed plan: ???

	AS(x)	AS(y)
$I$	{0}	{0}
$a_x$	{0, 1}	{0, -1}
$a_x$	???	???
$a_y$	???	???

## Numeric Relaxed Planning

In principle, things work as before:

- ▶ In conditions, a comparison  $\text{exp}(\bar{v}) \leq \text{exp}(\bar{v}')$  is relaxed to hold if there *exists* a value vector in  $AS(w_1) \times \dots \times AS(w_n)$  s.t. the comparison holds, where  $w_1, \dots, w_n$  are all variables used in  $\bar{v}$  or  $\bar{v}'$
- ▶ In effects,  $v := \text{exp}(\bar{v})$  is relaxed to  $AS(v) := AS(v) \cup \{\text{exp}(\bar{c}) \mid \bar{c} \in AS(\bar{v})\}$

- ▶ The domain subsets can grow arbitrarily large
- ▶ Answering “Does there exist a value vector s.t. ...” may be arbitrarily complicated (e.g. **NP**-complete: exercise)
- ▶ We can either approximate the necessary operations (there exist first tentative methods by Stefan Edelkamp)
- ▶ Or we can consider a sub-class of the language that is easier to handle: see next
- ▶ (Actually, we could also go and see if or if not the real thing explodes in practice)
- ▶ (Actually, I’ve gone there [Hoffmann et al, IJCAI’07] and the answer is: sometimes it does, sometimes it doesn’t)

- ▶ Expressions are restricted to the form  $exp(\vec{v}) = \sum_{v \in \vec{v}} c_v * v + c$
- ▶ Comparisons are restricted to  $exp(\vec{v}) > [\geq, <, \leq] 0$
- ▶ Less restrictive than it looks at first sight: transform expressions/comparisons (e.g., “=” turns to “ $\leq$ ” and “ $\geq$ ”)

Observe:

- ▶ There exists a value vector satisfying  $\sum_{v \in \vec{v}} c_v * v > [\geq] c$  iff the vector of the current *max* ( $c_v > 0$ ) respectively *min* ( $c_v < 0$ ) values in the  $AS(v)$  does the job; likewise for  $<$  and  $\leq$
- ▶ For this, it suffices to remember the current *min* and *max* element of each  $AS(v)$

## Linear Numeric STRIPS, Part II

Another problem:

- ▶  $\sum_{v \in \vec{v}} c_v * v - c \leq 0 \wedge \sum_{v \in \vec{v}} c_v * v - c \geq 0$
- ▶ Relaxed:  
 $\exists \vec{c} \in AS(\vec{v}) : (\sum_{v \in \vec{v}} c_v * c(v) - c \leq 0 \wedge \sum_{v \in \vec{v}} c_v * c(v) - c \geq 0)$
- ▶ Is there a value vector making a linear function 0?
- ▶ **NP**-hard with restrictions on values: “Knapsack” is modelled with natural numbers  $c, c_v$ , and  $AS(v) = \{0, 1\}$  for all  $v$

Its solution:

- ▶ Quantify separately:  $\exists \vec{c} \in AS(\vec{v}) : (\sum_{v \in \vec{v}} c_v * c(v) - c \leq 0) \wedge \exists \vec{c}' \in AS(\vec{v}) : (\sum_{v \in \vec{v}} c_v * c'(v) - c \geq 0)$
- ▶ That is, relax “=  $c$ ” to “ $max \geq c \wedge min \leq c$ ”

## An Example

- ▶ Multiple-valued task:  
 $at : \{KUF, MUC, IBK\}, I : at \mapsto IBK, G : at = MUC, at = IBK \rightarrow at := KUF, at = KUF \rightarrow at := MUC$
- ▶ Encoding in linear numeric STRIPS:  
 $at : \{0, 1, 2\}, I : at \mapsto 2, G : \{at \geq 1, at \leq 1\}, at \geq 2 \wedge at \leq 2 \rightarrow at := 0, at \geq 0 \wedge at \leq 0 \rightarrow at := 1$
- ▶ *What happens, in the relaxation of the linear numeric STRIPS encoding, if we drive from IBK to KUF?*

Sketch:

For all  $v$ :  $min_0(v) := I(v)$ ,  $max_0(v) := I(v)$

$t := 0$

**while**  $\exists e \in G : max_t(e) \not\geq 0$  **do**

$A_t := \{a \in A^N \mid \forall e \in pre(a) : max_t(e) > 0\}$

For all  $v$ :  $min_{t+1}(v) := \{min_t(e) \mid a \in A_t, (v := e) \in eff(a)\}$

For all  $v$ :  $max_{t+1}(v) := \{max_t(e) \mid a \in A_t, (v := e) \in eff(a)\}$

**if**  $min_{t+1} = min_t$  and  $max_{t+1} = max_t$  **then stop** **endif**

**endwhile**

- ▶  $min_{t+1} = min_t$  and  $max_{t+1} = max_t$  may never happen
- ▶ If we increment a var  $N$  times then the number of iterations is exponential in input encoding size

▶  $V = \{x, y\}$ ,  $I = \{x \mapsto 0, y \mapsto 0\}$ ,  $G = \{y \geq 1\}$

▶  $a_x : \rightarrow x := x + 1, y := y - 1,$   
 $a_y : x \geq 2 \rightarrow y := y + 1$

▶ RPG:

	$x$	$y$
$I$	$\{0\}$	$\{0\}$
$\{a_x\}$	$\{0, 1\}$	$\{0, -1\}$
???	???	???
???	???	???

▶  $min/max(x)$ : 0/0, 0/1, ???, ???

▶  $min/max(y)$ : 0/0, -1/0, ???, ???

Sketch:

**for all**  $e \in G$  **do**

for  $v \in e$  set  $min/max_{lev(e)}(v)$  to  $min/max_{lev(e)}(v)$  as required

**endfor**

**for**  $t := maxlevel, \dots, 1$  **do**

**for all**  $v$ ,  $minG_t(v) \neq -\infty$  **do**

select  $a$ ,  $level(a) = t - 1$ , and a minimizing  $(v := e) \in eff(a)$

For  $v' \in e$ , set  $min/max_{lev(e)}(v')$  to  $min/max_{lev(e)}(v')$  as required

**for all**  $e' \in pre(a)$  **do**

/\* same thing for  $e'$  \*/

**endfor**

**endfor**

/\* similarly for  $max$  goals \*/

**endfor**

▶  $V = \{x, y\}$ ,  $I = \{x \mapsto 0, y \mapsto 0\}$ ,  $G = \{y \geq 1\}$

▶  $a_x : \rightarrow x := x + 1, y := y - 1,$   
 $a_y : x \geq 2 \rightarrow y := y + 1$

Blackboard.

- ▶ I [Hoffmann, ECAI'02, JAIR'03] originally wrote this up from the compile-negations standpoint, so you'll hardly recognize it
- ▶ One can define negative RPG termination criteria that are guaranteed to hold after a finite number of steps (namely, when the *min/max* values have exceeded their respective *min/max needed* values)
- ▶ Relaxed solvability can be decided in polynomial time by introducing shortcuts ( $\max(x) := \infty$ ) for repeatable effect applications (e.g.  $x := x + 1$ )
- ▶ The *min/max* argument is valid for any expression *monotonic* in all variables, so linearity is more restrictive than necessary. *More general methods not implemented so far!*

- ▶ T. Bylander, *The Computational Complexity of Propositional STRIPS Planning*, Artificial Intelligence Journal, 1994
- ▶ Blai Bonet, Gabor Loerincs and Hector Geffner, *A Robust and Fast Action Selection Mechanism for Planning*, Proceedings AAAI 1997
- ▶ Jörg Hoffmann and Bernhard Nebel, *The FF Planning System: Fast Plan Generation Through Heuristic Search*, JAIR, vol. 14, 2001
- ▶ Blai Bonet and Hector Geffner, *Planning as Heuristic Search: New Results*, Proceedings ECP 1999
- ▶ Ioannis Refanidis and Ioannis Vlahavas, *GRT: A domain independent heuristic for STRIPS worlds based on greedy regression tables*, Proceedings ECP 1999

- ▶ Drew McDermott, *A Heuristic Estimator for Means-Ends Analysis in Planning*, Proceedings AIPS 1996
- ▶ Blai Bonet and Hector Geffner, *Planning as Heuristic Search*, AI, vol. 129, 2001
- ▶ B.Gazen and C.Knoblock, *Combining the Expressiveness of UCPOP with the Efficiency of Graphplan*, Proceedings ECP 1997
- ▶ Jörg Hoffmann, *Extending FF to Numerical State Variables*, Proceedings ECAI 2002
- ▶ Jörg Hoffmann, *The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables*, JAIR, vol. 20, 2003
- ▶ Jörg Hoffmann, Henry Kautz, Carla Gomes and Bart Selman, *SAT Encodings of State-Space Reachability Problems in Numeric Domains*, Proceedings IJCAI 2007