EBL and Automatic Planning

Javier Béjar

LSI-FIB-UPC

2010/2011
1. EBL and Automatic Planning

2. A linear planning algorithm

3. Example - Rooms

4. Example - Blocks world (1)

5. Example - Blocks world (2)

6. Effect of learning new operators
1. EBL and Automatic Planning

2. A linear planning algorithm

3. Example - Rooms

4. Example - Blocks world (1)

5. Example - Blocks world (2)

6. Effect of learning new operators
Automatic Planning - Linear Planning

- Solving problem methodology
- Problems are described using a predicate calculus logic formalism (situation calculus)
- A problem is composed by:
  - A description of the initial state
  - A description of the goals
  - A set of planning operators
- A search algorithm will be used to obtain a sequence of operators
Lineal Planning (STRIPS)

Operator

- An operator is defined by:
  - A set of preconditions
  - The predicates that will be true in the state (add list)
  - The predicates that will be false in the state (delete list)

- Preconditions and effects are expressed in predicate calculus

Plan

- A plan is defined by:
  - An initial state and a goal state
  - An ordered list of instantiated operators

- The search is performed in the state space

- A search strategy has to be defined (for instance: depth first, breadth first, means end analysis, ...)

Javier Béjar (LSI-FIB-UPC)
EBL and Automatic Planning

A linear planning algorithm

Example - Rooms

Example - Blocks world (1)

Example - Blocks world (2)

Effect of learning new operators
function LP (e_{initial}, e_{goal}, operators) returns plan

state ← e_{initial}

stack.push(G(e_{goal}))

while ¬stack.empty?() do
    goal ← stack.pop()

switch
case: goal is composed goal
    if ¬valid(goal, estate) then
        stack.push(goal)
        goals[] ← choose_new_order_for_subgoals(goal)
        for each g in goals[] do stack.push(G(g))
eif
ecase
Lineal Planning algorithm (II)

case: goal is simple goal
  if ¬valid(goal, state) then
    Choose an operator not used (op) that adds goal
    if that operator exists then
      Unify variables of the operator so goal holds
      stack.push(obj)
      stack.push(A(op))
      stack.push(G(preconditions(op)))
    eif
  eif
eecase
case: goal is apply operator
  if preconditions of goal hold in the current state then
    state ← apply(goal, state)
    plan.insert(goal)
  eif
ecase
eswitch
ewhile
return(plan)
efunction
1 EBL and Automatic Planning

2 A linear planning algorithm

3 Example - Rooms

4 Example - Blocks world (1)

5 Example - Blocks world (2)

6 Effect of learning new operators
Operators

- `go(robot, door, room1, room2)`
  - Precondition: `connected(door, room1, room2), in(room1, robot)`
  - Delete: `in(room1, robot)`
  - Add: `in(room2, robot)`

- `push(box, door, room1, room2)`
  - Precondition: `connected(door, room1, room2), in(room1, robot), in(room1, box)`
  - Delete: `in(room1, robot), in(room1, box)`
  - Add: `in(room2, robot), in(room2, box)`
Initial and goal state

\begin{align*}
\text{in}(R_1, \text{Robot}) \\
\text{in}(R_2, B_1) \\
\text{connected}(D_1, R_1, R_2) \\
\text{connected}(D_1, R_2, R_1) \\
\text{connected}(D_2, R_2, R_3) \\
\text{connected}(D_2, R_3, R_2)
\end{align*}
Resolution of the plan (1)

1. Goal = in(R1,B1)
   The operator that allows to move a box from one place to another is push. If we unify with the add list [in(R1,B1) = in(room2,obj)] → [room2=R1, obj=B1]
   The preconditions that we have are: connected(door,room1,R1), in(room1,R) and in(room1,B1), we look for an unification that gives the new goals (for instance: [room1=R2, door=D1])
   We add the operator push(B1,D1,R2,R1)

2. Goal = connected(D1,R2,R1), in(R2,B1), in(R2,R)
   We create a goal for each subgoal

3. Goal = connected(D1,R2,R1) → holds in the current state

4. Goal = in(R2,B1) → holds in the current state
5. Goal $= \text{in}(R2,R)$
   The operator that moves the robot is $\text{go}$. If we unify with the add list 
   $[\text{in}(R2,R) = \text{in}(\text{room2},R)] \rightarrow [\text{room2}=R2]$
   We will have the preconditions: $\text{connected}(\text{door},\text{room1},R2)$, $\text{in}(\text{room1},R)$, we look for an unification to obtain the new goals (for instance $[\text{room1}=R1, \text{door}=D1]$)
   We add the operator $\text{go}(D1,R1,R2)$

6. Goal $= \text{connected}(D1,R1,R2), \text{in}(R1,R)$
   We create a goal for each subgoal

7. Goal $= \text{connected}(D1,R1,R2) \rightarrow \text{holds in the current state}$

8. Goal $= \text{in}(R1,R) \rightarrow \text{holds in the current state}$
Resolution of the plan (and 3)

9. We execute the action go(D1,R1,R2) modifying the state, now in(R2,R) will be true in the state and in(R1,R) will be no longer true

10. Goal = in(R2,R) → holds in the current state

11. We execute the action push(B1,D1,R2,R1) modifying the state, now in(R1,R) and in(R1,B1) will be true in the state and in(R2,R) and in(R2,B1) will be no longer true

12. Goal = in(R1,B1) → holds in the current state

Now the goal stack is empty, so we have arrived to the goal state with the plan:

\[ E_{initial} \rightarrow go(D1,R1,R2) \rightarrow push(B1,D1,R2,R1) \rightarrow E_{goal} \]
Building the triangular table

- First column, first row = initial state
- For the rest
  - (column \(i+1\), row \(i\)) = Operator \(i\) of the plan
  - (column 1 to \(i\), row \(i+1\)) = (column 1 to \(i\), row \(i\)) except the predicates the operator has deleted
  - column \(i+1\), row \(i+1\) = The predicates that the operator adds
### Plan - Triangular table

<table>
<thead>
<tr>
<th>in(R1,R)</th>
<th>in(R2,B1)</th>
<th>connected(D1,R1,R2)</th>
<th>connected(D1,R2,R1)</th>
<th>go(D1,R1,R2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in(R2,B1)</td>
<td>in(R1,R)</td>
<td>connected(D1,R1,R2)</td>
<td>connected(D1,R2,R1)</td>
<td>in(R2,R)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>push(B1,D1,R2,R1)</td>
<td></td>
<td>push(B1,D1,R2,R1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>connected(D1,R1,R2)</td>
<td>in(R1,B1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>connected(D1,R2,R1)</td>
<td>in(R1,R)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We delete from the table all predicates that are not used to prove the preconditions of the operators.
Variabilization of the triangular table

- We assign different variables to each constant in the predicate from the first column.
- We variabilize the parameters of the operators and propagate the substitutions accordingly in the corresponding column.
<table>
<thead>
<tr>
<th>in(x1,x2)</th>
<th>go(x11,x12,x13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>connected(x3,x4,x5)</td>
<td></td>
</tr>
<tr>
<td>in(x6,x7)</td>
<td>in(x13,R)</td>
</tr>
<tr>
<td>connected(x8,x9,x10)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>push(x14,x15,x16,x17)</td>
</tr>
<tr>
<td></td>
<td>in(x17,R)</td>
</tr>
<tr>
<td></td>
<td>in(x17,x14)</td>
</tr>
</tbody>
</table>
We unify the variables of the operators using the preconditions as guide.

The restriction has to be performed so the original proofs of the precondition still hold.
It has to be the robot

in(x1,x2)  We have to begin from the room of the robot
connected(x3,x4,x5)  Preconditions

in(x6,x7)  We move what is in the room
in(x13,R)  Everythin in the same room
connected(x8,x9,x10)  push(x14,x15,x16,x17)

x1=x4=x12  X2=R  x3=x11  x5=x13=x6=x9=x16
x7=x14  x8=x15  x10=x17
### Plan - Final table

<table>
<thead>
<tr>
<th>in(x1,R)</th>
<th>connected(x3,x1,x5)</th>
<th>go(x3,x1,x5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in(x5,x7)</td>
<td>connected(x8,x5,x10)</td>
<td>in(x5,R)</td>
</tr>
<tr>
<td>push(x7,x8,x5,x10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>in(x10,R)</td>
<td>in(x10,x7)</td>
<td></td>
</tr>
</tbody>
</table>

---

**Example - Rooms**

Javier Béjar (LSI-FIB-UPC)

EBL and Automatic Planning 2010/2011 23 / 44
New operator

\[ \text{op: } \text{move}(x_1, x_3, x_5, x_7, x_8, x_{10}) \]
\[ \text{prec: } \text{in}(x_1, R), \text{connected}(x_3, x_1, x_5), \text{in}(x_5, x_7), \text{connected}(x_8, x_5, x_{10}) \]
\[ \text{add: } \text{in}(x_{10}, R), \text{in}(x_{10}, x_7) \]
\[ \text{del: } \text{in}(x_1, R), \text{in}(x_5, x_7) \]

- **Precondition**: Relevant predicate from the first column
- **Add**: Predicates from the last row except the ones from the first column
- **Delete**: Differences among the initial state and the predicates from the last row of the first column
1. EBL and Automatic Planning
2. A linear planning algorithm
3. Example - Rooms
4. Example - Blocks world (1)
5. Example - Blocks world (2)
6. Effect of learning new operators
### Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precondition</th>
<th>Delete</th>
<th>Add</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pick(x)</code></td>
<td>on_table(x), clear(x), emptyhand</td>
<td>on_table(x), clear(x), emptyhand</td>
<td>on_hand(x)</td>
</tr>
<tr>
<td><code>unpick(x)</code></td>
<td>on_hand(x)</td>
<td>on_hand(x)</td>
<td>on_table(x), clear(x), emptyhand</td>
</tr>
<tr>
<td><code>stack(x,y)</code></td>
<td>on_hand(x), clear(y)</td>
<td>on_hand(x), clear(y)</td>
<td>emptyhand, on(x,y), clear(x)</td>
</tr>
<tr>
<td><code>unstack(x,y)</code></td>
<td>emptyhand, clear(x), on(x,y)</td>
<td>emptyhand, clear(x), on(x,y)</td>
<td>on_hand(x), clear(y)</td>
</tr>
</tbody>
</table>
Initial and goal states

Plan: \text{pick}(A) \rightarrow \text{stack}(A,B)
Plan - Triangular table

<table>
<thead>
<tr>
<th>clear(A) clear(B) on_table(A) on_table(B) emptyhand</th>
<th>pick(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear(B) on_table(B)</td>
<td>on_hand(A)</td>
</tr>
<tr>
<td>on_table(B)</td>
<td>stack(A,B)</td>
</tr>
<tr>
<td></td>
<td>clear(A) emptyhand on(A,B)</td>
</tr>
</tbody>
</table>
### Plan - Generalized table

<table>
<thead>
<tr>
<th>Preconditions</th>
<th>Clear(x1)</th>
<th>On_table(x2)</th>
<th>Emptyhand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>Clear(x3)</td>
<td>On_hand(x4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Clear(x5)</td>
<td></td>
<td>On(x5,x6)</td>
</tr>
</tbody>
</table>

\[ x_1 = x_2 = x_4 = x_5 \quad x_3 = x_6 \]
### Plan - Constrained table

<table>
<thead>
<tr>
<th>clear(x1)</th>
<th>pick(x1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_table(x1)</td>
<td></td>
</tr>
<tr>
<td>emptyhand</td>
<td></td>
</tr>
<tr>
<td>clear(x3)</td>
<td>on_hand(x1)</td>
</tr>
<tr>
<td></td>
<td>stack(x1,x3)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clear(x1)</td>
</tr>
<tr>
<td></td>
<td>emptyhand</td>
</tr>
<tr>
<td></td>
<td>on(x1,x3)</td>
</tr>
</tbody>
</table>
New operator

\[
\text{op: } \text{on-block}(x_1,x_3) \\
\text{prec: } \text{clear}(x_1), \text{on-table}(x_1), \text{emptyhand}, \text{clear}(x_3) \\
\text{add: } \text{on}(x_1,x_3), \text{clear}(x_1), \text{emptyhand} \\
\text{del: } \text{clear}(x_3), \text{clear}(x_1), \text{on-table}(x_1), \text{emptyhand}
\]

Sometimes the operator can be simplified

\[
\text{op: } \text{on-block}(x_1,x_3) \\
\text{prec: } \text{clear}(x_1), \text{on-table}(x_1), \text{emptyhand}, \text{clear}(x_3) \\
\text{add: } \text{on}(x_1,x_3) \\
\text{del: } \text{clear}(x_3), \text{on-table}(x_1)
\]
EBL and Automatic Planning

A linear planning algorithm

Example - Rooms

Example - Blocks world (1)

Example - Blocks world (2)

Effect of learning new operators
Initial and goal states

Plan: unstack(C,A) → unpick(C) → pick(A) → stack(A,B)
**Plan - Triangular table**

<table>
<thead>
<tr>
<th>clear(C) clear(B) on_table(A) emptyhand</th>
<th>unstack(C,A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear(B) on_table(B) on_table(A)</td>
<td>on_hand(C)</td>
</tr>
<tr>
<td></td>
<td>clear(A)</td>
</tr>
<tr>
<td>clear(B) on_table(B) on_table(A)</td>
<td>clear(A)</td>
</tr>
<tr>
<td></td>
<td>emptyhand</td>
</tr>
<tr>
<td>on_table(B) clear(B)</td>
<td>on_table(C)</td>
</tr>
<tr>
<td></td>
<td>clear(C)</td>
</tr>
<tr>
<td>on_table(B)</td>
<td>on_hand(A)</td>
</tr>
<tr>
<td></td>
<td>stack(A,B)</td>
</tr>
<tr>
<td>on_table(B)</td>
<td>on_table(C)</td>
</tr>
<tr>
<td></td>
<td>clear(C)</td>
</tr>
<tr>
<td>on_table(B)</td>
<td>on(A,B)</td>
</tr>
<tr>
<td></td>
<td>emptyhand</td>
</tr>
<tr>
<td></td>
<td>clear(A)</td>
</tr>
</tbody>
</table>
### Plan - Table - relevant predicates

<table>
<thead>
<tr>
<th>clear(C)</th>
<th>on(A,C)</th>
<th>emptyhand</th>
<th>unstack(C,A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_hand(C)</td>
<td>clear(A)</td>
<td>emptyhand</td>
<td>unpick(C)</td>
</tr>
<tr>
<td>on_table(A)</td>
<td>clear(A)</td>
<td>emptyhand</td>
<td>pick(A)</td>
</tr>
<tr>
<td>clear(B)</td>
<td></td>
<td>on_hand(A)</td>
<td>stack(A,B)</td>
</tr>
<tr>
<td></td>
<td>on_table(C)</td>
<td></td>
<td>on(A,B)</td>
</tr>
<tr>
<td></td>
<td>clear(C)</td>
<td></td>
<td>emptyhand</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clear(A)</td>
<td>Not deleted by any operator</td>
</tr>
</tbody>
</table>

Not deleted by any operator:

- on(A,B)
- emptyhand
- clear(A)
Example - Blocks world (2)

Plan - Generalized table

\[
x_1 = x_2 = x_6 = x_8 \quad x_3 = x_7 = x_4 = x_9 = x_{10} \quad x_5 = x_{11}
\]
<table>
<thead>
<tr>
<th>clear(x1)</th>
<th>on(x1,x3)</th>
<th>emptyhand</th>
<th>unstack(x1,x3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>on_hand(x1)</td>
</tr>
<tr>
<td>on_table(x3)</td>
<td>clear(x3)</td>
<td>emptyhand</td>
<td>unpick(x1)</td>
</tr>
<tr>
<td>clear(x5)</td>
<td></td>
<td></td>
<td>pick(x3)</td>
</tr>
<tr>
<td>on_table(x1)</td>
<td>clear(x1)</td>
<td></td>
<td>on_hand(x3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stack(x3,x5)</td>
</tr>
<tr>
<td></td>
<td>on(x4,x5)</td>
<td>emptyhand</td>
<td>clear(x4)</td>
</tr>
</tbody>
</table>
New operator

\[
\text{op: clear-put}(x_1, x_3, x_5) \\
\text{prec: clear}(x_1), on\_table(x_5), empty\_hand, clear(x_5), on(x_1, x_3) \\
\text{add: on}(x_3, x_5), clear(x_3), empty\_hand, clear(x_1), on\_table(x_1) \\
\text{del: clear}(x_5), on(x_1, x_3), on\_table(x_3), empty\_hand, clear(x_1)
\]
1. EBL and Automatic Planning

2. A linear planning algorithm

3. Example - Rooms

4. Example - Blocks world (1)

5. Example - Blocks world (2)

6. Effect of learning new operators
Effect of learning new operators

- To learn new operators from simple plans can improve the resolution time.
- The problems used to learn the new operators have to be selected carefully.
- The cost of having to check the new operators during the resolution have to be compensated by the resolution time saved.
In the previous example we can add to the initial operators the one learned **on-block** and another with the opposite effect (**off-block**).

We can measure the time we save solving different problems of increasing difficulty with the original operators, adding the new operators and only using the new operators.
Effect of learning new operators

Easy problems

Strips - Easy Problems

<table>
<thead>
<tr>
<th>Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>65000</td>
</tr>
<tr>
<td>60000</td>
</tr>
<tr>
<td>55000</td>
</tr>
<tr>
<td>50000</td>
</tr>
<tr>
<td>45000</td>
</tr>
<tr>
<td>40000</td>
</tr>
<tr>
<td>35000</td>
</tr>
<tr>
<td>30000</td>
</tr>
<tr>
<td>25000</td>
</tr>
<tr>
<td>20000</td>
</tr>
<tr>
<td>15000</td>
</tr>
<tr>
<td>10000</td>
</tr>
<tr>
<td>5000</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Num. Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

- F-Op3
- F-Op2
- F-Op1
Medium problems

Strips - Med. Problems

Milliseconds

Num. blocks

M-Op3

M-Op2

M-Op1
Effect of learning new operators

Difficult problems

Strips - Diff Problems

Milliseconds

num Blocks

D-Op3
D-Op2
D-Op1

Javier Béjar (LSI-FIB-UPC)