Introduction to Intelligent, Knowledge-Based, and Cognitive Systems

89-674

Mor Vered

http://u.cs.biu.ac.il/~veredm/89-674/

mvered89230@gmail.com

Sunday 10-11, Building 202, Room 119
Prospective

• Solve an easy (but complex) problem
• Multiple operators
• Problem spaces
• Termination, search control, preferences
Water-jugs problem

• We have two water jugs:
  • One allowing 3 liters
  • One allowing 5 liters
  • No marks on the jugs

• We want to end up with exactly 1 liter in the 3-liter jug

• Allowed operations
  • Fill jug to completion from well
  • Pour water from jug to jug
Problem spaces

• Initial state
• Desired (and sometimes undesired) states
• Operators that move from state to state

For example:
• State: (3-liter full, 5-liter empty)
• State: (3-liter empty, 5-liter filled with 3 L.)
• Operation: Pour from 3 to 5.
Setting up problem solving

- An Initial operator for setting up initial state
- Operators for changing the state
  - To explore space of states
- Rules for discovering the desired state
  - “exit condition”
- Search control rules (guide solution process)
Init-water-jugs Operator

• Set up two objects (jugs)
• Each jug has:
  • Volume
  • Current water level
  • Current available volume (empty)

• Namespace control
  • (state <s> ^name water-jug)
  • Allows modularity when mixing rules
Implementing init-water-jugs

Propose:

```
sp {water-jug*propose*init-water-jugs
   (state <s> ^superstate nil)
   -(<s> ^name )
   -->
   (<s> ^operator <o> +)
   (<o> ^name init-water-jugs)
}
```

Naming convention: problem-space*phase*op.name*#
Init-water-jugs Operator

Application rule:

sp {water-jug*apply*init-water-jugs
  (state <s> ^operator.name init-water-jugs)
  -->
  ( <s> ^name water-jug
       ^jug <j1> ^jug <j2> )
  ( <j1> ^volume 5 ^contents 0 )
  ( <j2> ^volume 3 ^contents 0 )
}
Working with VisualSoar DataMap

- In VisualSoar Datamap – A representation of the structure of our working memory.
- check Datamap->Check all productions against the dataMap.
- Right click on project_name -> open datamap
- Add the jug structure to the data map. (right click on datamap -> add identifier)
- Then add integers for contents, empty, volume
Soar Syntax

• In Soar, all math is done using prefix notation where the math operations (+, -, *, /) come before the arguments, with parentheses surrounding the computation.
• <> means different
• (+ <v> <c>)
• (+ 2 (* <v> <c>))
State elaborations

State elaboration rules create i-supported working memory elements so that when the parts of the state they test change, they recompute their actions automatically.

When the contents of a jug changes through an operator application rule, this rule will retract the old value of ^empty and fire to compute a new value.

\[
\text{sp } \{ \text{water-jug}^* \text{elaborate}^* \text{empty} \}
\]

VisualSoar – right click elaborations, newFile, “empty” (template, elaborate state)
State elaborations

State elaboration rules create i-supported working memory elements so that when the parts of the state they test change, they recompute their actions automatically.

When the contents of a jug changes through an operator application rule, this rule will retract the old value of \(^\text{empty}\) and fire to compute a new value.

\[
\begin{align*}
\text{sp} \{ & \text{water-jug}^{\text{elaborate}^{\text{empty}}} \\
& (\text{state} <s> \ ^\text{name} \text{water-jug} \\
& \ ^\text{jug} <j>) \\
& (<j> \ ^\text{volume} <v> \ ^\text{contents} <c>) \\
& \rightarrow \\
& (<j> \ ^\text{empty} (- <v> <c>) ) \\
\}
\end{align*}
\]
Soar Cycle
A simulated run:
What will happen now?

• We create the agent
• The init-water-jugs proposal rule fires
  • State does not have a name
• Operator init-water-jugs is only candidate
• Operator init-water-jugs is selected
• Operator init-water-jugs is applied
  • Name, jugs, set-up
  • Then elaboration of empty fires
  • In parallel, proposal for init-water-jugs retracts
• Now what?
Create State-Transition Operators

What operations do we have?

Operators must change the state in some way so that another operator can be selected.

- Fill a jug with water from the well, if that the jug is not full.
- Empty the water from a jug to the well, if there is water in the jug.
- Pour water from one jug to another jug, if there is water in the source jug and the destination jug is not full.
Fill Operator

water-jug*propose*fill

If the task is water-jug and there is a jug that is not full, then propose filling that jug.

need parameters that specify which jugs are being filled/emptied

• The name of the operator: ^name fill/empty/pour.
• The jug that is being filled by the fill operator: ^fill-jug <j>.
• The jug that is being emptied by the empty operator: ^empty-jug <j>.
• The jug that is being poured out of by pour: ^empty-jug <j1>.
• The jug being poured into by pour: ^fill-jug <j2>.
Fill Operator - propose

sp { water-jug*propose*fill
    (state <s> ^name water-jug
     ^jug <j>)
    (<j> ^empty > 0)
  } →

  ( <s> ^operator <o> +)
  (<o> ^name fill
     ^fill-jug <j> )

Test that the jug is not full

The “+” signifies that the operator is proposed and not selected

In VisualSoar, right-click on project_name and add supoperator. Insert template propose/apply
Preferences

• Nothing every gets moved from preference to working memory unless it has an acceptable (or a require) preference. The general effect of the other rules is reflected in the preference names

~ prohibit (note that this preference is almost never used)
- reject (and wipe out any other preferences)
+ acceptable
! require (note that this preference is almost never used)
= indifferent (ones as good as another)
& parallel (allows two or more values for an attribute)
> better (binary) or best (unary)
< worse (binary) or worst (unary)
@ reconsider (applies only to operators)
Must remove original WME and create a new one. No way to just replace the value. Cannot modify a WME once it is created. To remove a WME use "-".
Empty Operator

sp \{water-jug*propose*empty
  \[state <s> ^name water-jug
     ^jug <j>\)
  \(<j> ^contents > 0\)
→
  \(<s> ^operator <o> +\)
  \(<o> ^name empty
     ^empty-jug <j>\}\}

If the task is water-jug and there is a jug that is not empty, then propose emptying that jug.

sp \{water-jug*apply*empty
  \[state <s> ^name water-jug
     ^operator <o>
     ^jug <j>\)
  \(<o> ^name empty
     ^empty-jug <j>\)
  \(<j> ^volume <volume>
     ^contents <contents>\)
→
  \(<j> ^contents 0
     ^contents <contents> -\)
}
If the task is water-jug and there is a jug that is not full and the other jug is not empty, then propose pouring water from the second jug into the first jug.
Pour Operator Proposal

If the task is `water-jug` and there is a jug that is not full and the other jug is not empty, then propose pouring water from the second jug into the first jug.

Soar doesn’t explicitly prevent different conditions from matching the same working memory Element. Jug j different than i. Must have space before and after <>.

sp {water-jug*propose*pour
    (state <s> ^name water-jug
    ^jug <i>
    ^jug { <j> <> <i> })
    (<i> ^contents > 0)
    (<j> ^empty > 0)

→

    (<s> ^operator <o> +)
    (<o> ^name pour
    ^empty-jug <i>
    ^fill-jug <j>))}
Pour Operator Apply

If the task is water-jug and the pour operator is selected, and the contents of the jug being emptied are less than or equal to the empty amount of the jug being filled, then set the contents of the jug being emptied to 0; set the contents of the jug being filled to the sum of the two jugs.

If the task is water-jug and the pour operator is selected, and the contents of the jug being emptied are greater than the empty amount of the jug being filled, then set the contents of the jug being emptied to its contents minus the empty of the jug being filled; set the contents of the jug filled to its volume.
If the task is water-jug and the pour operator is selected, and the contents of the jug being emptied are less than or equal to the empty amount of the jug being filled, then set the contents of the jug being emptied to 0; set the contents of the jug being filled to the sum of the two jugs.
If the task is water-jug and the pour operator is selected, and the contents of the jug being emptied are greater than the empty amount of the jug being filled, then set the contents of the jug being emptied to its contents minus the empty of the jug being filled; set the contents of the jug filled to its volume.
Running – What will happen?

1) Init-water-jugs
2) Water-jug-propose-fill fires twice because matches both empty jugs.
3) What next? *Tie impasse* – doesn’t have enough knowledge to select between two operators

For now, select randomly using *indifferent preference.*
Running a search for a solution

• We have an initial state
• We have operators that cover the state-space
• What’s missing?

Stopping when we reached the target
Generating a rule that recognizes when a desired state has been achieved
Detecting goal state

If the task is water-jug and there is a jug with volume three and contents one, write that the problem has been solved and halt.

sp {water-jug*detect*goal*achieved
  (state <s> ^name water-jug
   ^jug <j>)
  (<j> ^volume 3
   ^contents 1)
→
  (write (crlf) |The problem has been solved.|)
  (halt)}

Linefeed before writing out any text, using the (crlf) command, which stands for carriage-return and linefeed
Now we have a running system

• To add monitoring rules (Soar tutorial 1 page 38)
  • Keep track of current state
  • Keep track of selected operator

• Short exercise:
  • Run the system ten times
  • Look at how many decisions cycles it takes
  • Average over 10 trials.
  • Use the log to save trial outputs.
Search control

- Many selected operators don’t make sense
  - Why Fill, then empty, same operator?
  - Why Pour from A to B, only to empty B?
  - Avoid undoing the last operator that was applied
- We can add rules that control the search
- Prefer or discourage certain selections
Adding simple search-control

Using last-operator, we can now add rules:

• If last=fill, and proposed empty, reject.
• If last=empty, and proposed=fill, reject.
• Don’t select operator twice in a row
• …
Search Control – Saving last operator

• To avoid undoing the last operator, the program must remember the operator in the state after it applies. In Soar, this memory is not automatic. The selected operator is retracted as soon as it applies. Must add rules that deliberately record the operator each time one is applied.

• A common technique
  • Useful in many settings

• To do this:
  • Test if we have a last-operator attribute
    • If not, add it (with O-Support)
  • Is current operator different from last-operator
    • If so, remove last-operator (with O-Support)
Saving last operator

• Recording an operator has two parts:
  • Creating a structure on the state that is the memory of the most recent operator
  • Remove any record of an older operator

• Must have one for each operator (overall 3) because each operator has different augmentations
If the task is water-jug and the pour operator is selected, then create an augmentation of the state (last-operator) with the name of the operator and a copy of the augmentations augmentation.

Pour Record Example

\[
\text{sp \{water-jug*propose*operator*record*last-operator*pour}
\]
\[
\text{(state <s> ^name water-jug}
\]
\[
\text{^operator <o>)}
\]
\[
\text{(o) ^name pour}
\]
\[
\text{^fill-jug <fj>}
\]
\[
\text{^empty-jug <ej>)}
\]
\[
\rightarrow
\]
\[
\text{(s) ^last-operator <last-op>)}
\]
\[
\text{(last-op) ^name pour}
\]
\[
\text{^fill-jug <fj>}
\]
\[
\text{^empty-jug <ej>)}
\]
Pour Record Example

If the task is water-jug and a pour operator is selected and last-operator does not have the same name and fill-jug, then remove the last-operator.

\[
\text{sp } \{\text{water-jug*apply*operator*remove*last-operator*pour} \\
\text{(state } <s> \text{ } ^\text{name water-jug} \\
\text{ } ^\text{operator } <o> \\
\text{ } ^\text{last-operator } <\text{last-op}>\}
\]

\[
\text{(}<o> \text{ } ^\text{name pour} \\
\text{ } ^\text{fill-jug } <fj> \\
\text{ } ^\text{empty-jug } <ej>\)
\]

\[
\text{-(<last-op> } \text{ } ^\text{name pour} \\
\text{ } ^\text{fill-jug } <fj> \\
\text{ } ^\text{empty-jug } <ej>\)
\]

\[
\rightarrow
\]

\[
\text{(}<s> \text{ } ^\text{last-operator } <\text{last-op} \text{ } -})\}
\]
Avoid Applying Apposing Operators

sp \{water-jug*select*operator*avoid*inverse*fill

\text{(state \(s\) \^name water-jug
\\^operator \(o\) +
\^last-operator \(lo\))
\text{(\(o\) \^name fill \^fill-jug \(i\))
\text{(\(lo\) \^name empty \^empty-jug \(i\))
\rightarrow
\text{(\(s\) \^operator \(o\) \})

If the task is water-jug and the last operator is empty then avoid a fill.
Homework

New take on the water jug problem, this time both jugs start out full and the goal is to get four gallons of water into the five gallon jug.

1) Implement the original water-jug example without search-control. How many decision cycles did it take? Run the system ten times and average.

2) Implement the take on the Water-Jug example without search-control. How many decision cycles did it take?

3) Write basic search control that refrains from choosing the same action and refrains from choosing opposing actions. What was the improvement in terms of decision cycles on both problems?

4) Write your own search control heuristic. Monitor the improvement.

Submit code and additional text explanation to mvered89230@gmail.com by Thursday 20.4.2017