Comp 6721 - Artificial Intelligence - Project 2 project report

Federico O'Reilly Regueiro - 40012304

December 7th, 2016

1 Goal-stack planning

1.a problems with the representation?

$MOVE(B, A, Table)$ collides with $MOVE{-}TO{-}TABLE(B, A)$ and generates an inconsistent knowledge-base as well as having silly preconditions such as $Clear(Table)$ which could potentially make it impossible to move a block from the $Table$ and place it atop another block. Additionally, $MOVE(B, C, C)$ could display inconsistent behavior.

The table can hold many blocks and therefore should be treated differently from blocks. So another problem stems from the fact that the predicate $On(x)$ should not take $Table$, instead, we need another predicate $OnTable(x)$ as well as an operation to move a block from the table atop another empty block, a predicate $Block(x)$ that becomes a precondition to $Move(b, x, y)$ (added preconditions: $Block(x) \wedge Block(y)$ and $\neg(x = y)$).

1.b Demonstrate goal-stack planning

1.

goal-stack	popped	KB
$OnTable(A)$		$On(B, C)$
$On(B, A)$		$OnTable(A)$
$On(C, B)$		$Clear(B)$
		$Clear(A)$
		$OnTable(C)$

2.

goal-stack	popped	KB
	$OnTable(A)$	$On(B, C)$
$On(B, A)$		$OnTable(A)$
$On(C, B)$		$Clear(B)$
		$Clear(A)$
		$OnTable(C)$

3.

goal-stack	popped	KB
$MOVE(B, C, A)$		$On(B, C)$
$On(B, A)$		$OnTable(A)$
$On(C, B)$		$Clear(B)$
		$Clear(A)$
		$OnTable(C)$

4.

goal-stack	popped	KB
$On(B, C)$		$On(B, C)$
$Clear(A)$		$OnTable(A)$
$Clear(B)$		$Clear(B)$
$MOVE(B, C, A)$		$Clear(A)$
$On(B, A)$		$OnTable(C)$
$On(C, B)$		

5.

goal-stack	**popped**	**KB**
$Move(B,C,A)$	$On(B,C)$	$On(B,C)$
$On(B,A)$	$Clear(A)$	$OnTable(A)$
$On(C,B)$	$Clear(B)$	$Clear(B)$
		$Clear(A)$
		$OnTable(C)$

6.

goal-stack	**popped**	**KB**
	$MOVE(B,C,A)$	$\cancel{On(B,C)}$
$On(B,A)$		$OnTable(A)$
$On(C,B)$		$Clear(B)$
		$\cancel{Clear(A)}$
		$On(B,A)$
		$Clear(C)$
		$OnTable(C)$

7.

goal-stack	**popped**	**KB**
$MOVE(C,Table,B)$	$On(B,A)$	$Clear(C)$
$On(C,B)$		$OnTable(A)$
		$Clear(B)$
		$On(B,A)$
		$OnTable(C)$

8.

goal-stack	**popped**	**KB**
$OnTable(C)$		$Clear(C)$
$Clear(C)$		$OnTable(A)$
$Clear(B)$		$Clear(B)$
$MOVE(C,Table,A)$		$On(B,A)$
$On(C,B)$		$OnTable(C)$

9.

goal-stack	**popped**	**KB**
$MOVE(C,Table,B)$	$OnTable(C)$	$Clear(C)$
$On(C,B)$	$Clear(C)$	$OnTable(A)$
	$Clear(B)$	$Clear(B)$
		$On(B,A)$
		$OnTable(C)$

10.

goal-stack	**popped**	**KB**
$On(C,B)$	$MOVE(C,Table,B)$	$Clear(C)$
		$OnTable(A)$
		$\cancel{Clear(B)}$
		$On(B,A)$
		$\cancel{OnTable(C)}$
		$On(C,B)$
		$Clear(Table)$!

11.

goal-stack	**popped**	**KB**
$Done$!	$On(C,B)$	$Clear(C)$
		$OnTable(A)$
		$On(C,B)$
		$On(B,A)$
		$Clear(Table)$

1.c Demonstrate the Sussman Anomaly

There are three ways of ordering the sub-goals $On(A,B), On(B,C), OnTable(C)$ in this scenario which could cause Sussman's Anomaly:

$On(B,C)$	$On(B,C)$	$OnTable(C)$
$OnTable(C)$	$On(A,B)$	$On(A,B)$
$On(A,B)$	$OnTable(C)$	$On(B,C)$

An illustration of the first of these stacks follows:

	goal-stack	**popped**	**KB**
1.	$On(B,C)$		$On(C,A)$
	$OnTable(C)$		$OnTable(A)$
	$On(A,B)$		$OnTable(B)$
			$Clear(B)$
			$Clear(C)$

	goal-stack	**popped**	**KB**
2.	$MOVE(B,Table,C)$		$On(C,A)$
	$On(B,C)$		$OnTable(A)$
	$OnTable(C)$		$OnTable(B)$
	$On(A,B)$		$Clear(B)$
			$Clear(C)$

	goal-stack	**popped**	**KB**
3.	$Clear(B)$		$On(C,A)$
	$Clear(C)$		$OnTable(A)$
	$OnTable(B)$		$OnTable(B)$
	$MOVE(B,Table,C)$		$Clear(B)$
	$On(B,C)$		$Clear(C)$
	$OnTable(C)$		
	$On(A,B)$		

	goal-stack	**popped**	**KB**
4.	$MOVE(B,Table,C)$	$OnTable(B)$	$On(C,A)$
	$On(B,C)$	$Clear(C)$	$OnTable(A)$
	$OnTable(C)$	$Clear(B)$	$OnTable(B)$
	$On(A,B)$		$Clear(B)$
			$Clear(C)$

	goal-stack	**popped**	**KB**
5.	$On(B,C)$	$MOVE(B,Table,C)$	$On(C,A)$
	$OnTable(C)$		$OnTable(A)$
	$On(A,B)$		$\cancel{OnTable(B)}$
			$Clear(B)$
			$\cancel{Clear(C)}$
			$On(B,C)$

	goal-stack	**popped**	**KB**
6.	$OnTable(C)$	$On(B,C)$	$On(C,A)$
	$On(A,B)$		$On(B,C)$
			$OnTable(A)$
			$Clear(B)$

	goal-stack	**popped**	**KB**
7.	$MOVE-TO-TABLE(C)$		$On(C,A)$
	$OnTable(C)$		$On(B,C)$
	$On(A,B)$		$OnTable(A)$
			$Clear(B)$

	goal-stack	**popped**	**KB**
8.	$Clear(C)$ $On(C, A)$ $MOVE-TO-TABLE(C, A)$ $OnTable(C)$ $On(A, B)$		$On(C, A)$ $On(B, C)$ $OnTable(A)$ $Clear(B)$

And now we can't proceed with the second sub-goal without undoing the first (which is no longer on the stack)!

2 Context free grammars for English

2.a sentences parsed by the given grammar

For the proposed grammar, a noun can be composed in two ways and is included twice in a sentence. Thus, the given grammar could parse/generate $2 \times 2 = 4$ sentences:

- the computer crashes the computer
- the computer crashes the program
- the program crashes the computer
- the program crashes the program

2.b enhance the grammar to parses/generates NPs with modifiers

By modifying rules 1 and 2, the grammar could parse sentences such as *the bad program that crashes the computer*. The necessary modifications are listed below.

i	sentence	$\longrightarrow$ np vp | np compl vp
ii	np	$\longrightarrow$ det noun | det adj noun
iii	vp	$\longrightarrow$ verb np
iv	noun	$\longrightarrow$ computer | program
v	verb	$\longrightarrow$ crashes
vi	det	$\longrightarrow$ the
vii	adj	$\longrightarrow$ fast | bad
viii	compl	$\longrightarrow$ that

The series of parsed/generated sentences grows considerably, since we can now generate sentences in two different ways and nouns in $2 \times 3 = 6$ ways. Since we have two nouns in the sentence then we have $2 \times 2 \times 3 \times 2 \times 3 = 72$ sentences:

the computer crashes the computer
the computer crashes the program
the program crashes the computer
the program crashes the program

the computer that crashes the computer
the computer that crashes the program
the program that crashes the computer
the program that crashes the program

the fast computer crashes the computer
the fast computer crashes the program
the fast program crashes the computer
the fast program crashes the program

the fast computer that crashes the computer
the fast computer that crashes the program
the fast program that crashes the computer
the fast program that crashes the program

the bad computer crashes the computer
the bad computer crashes the program
the bad program crashes the computer
the bad program crashes the program

the bad computer that crashes the computer
the bad computer that crashes the program
the bad program that crashes the computer
the bad program that crashes the program

the computer crashes the fast computer
the computer crashes the fast program
the program crashes the fast computer
the program crashes the fast program

the computer that crashes the fast computer
the computer that crashes the fast program
the program that crashes the fast computer
the program that crashes the fast program

the fast computer crashes the fast computer
the fast computer crashes the fast program
the fast program crashes the fast computer
the fast program crashes the fast program

the fast computer that crashes the fast computer
the fast computer that crashes the fast program
the fast program that crashes the fast computer
the fast program that crashes the fast program

the bad computer crashes the fast computer
the bad computer crashes the fast program
the bad program crashes the fast computer
the bad program crashes the fast program

the bad computer that crashes the fast computer
the bad computer that crashes the fast program
the bad program that crashes the fast computer
the bad program that crashes the fast program

the computer crashes the bad computer
the computer crashes the bad program
the program crashes the bad computer
the program crashes the bad program

the computer that crashes the bad computer
the computer that crashes the bad program
the program that crashes the bad computer
the program that crashes the bad program

the fast computer crashes the bad computer
the fast computer crashes the bad program
the fast program crashes the bad computer
the fast program crashes the bad program

the fast computer that crashes the bad computer
the fast computer that crashes the bad program
the fast program that crashes the bad computer
the fast program that crashes the bad program

the bad computer crashes the bad computer
the bad computer crashes the bad program
the bad program crashes the bad computer
the bad program crashes the bad program

the bad computer that crashes the bad computer
the bad computer that crashes the bad program
the bad program that crashes the bad computer
the bad program that crashes the bad program

Out of all these syntactically correct sentences, there are several which make little sense. Although a faulty computer might cause a program to crash, this is not generally understood to be the case. Also, potentially one computer might crash another computer[1] in some form of networked situation, in general it is programs that crash computers or maybe even other programs running synchronously. However, in the case of a program crashing *another* program, we would generally require some specifier to distinguish between programs (eg 'this program crashed the other program').

Another thing that makes little sense is qualifying a computer as 'bad', where it makes perfect sense for a program (specially one that causes computers to crash.

This leaves only a small subset of the language that really makes sense:

the program crashes the computer
the program crashes the fast computer
the program that crashes the computer
the program that crashes the fast computer
the bad program crashes the computer
the bad program crashes the fast computer
the bad program that crashes the computer
the bad program that crashes the fast computer
the fast program crashes the computer
the fast program crashes the fast computer
the fast program that crashes the computer
the fast program that crashes the fast computer

[1] . . . by running a red light. Sorry, couldn't help it.

The necessary modifications could easily be made by using a context-sensitive grammar but are not as natural to a context-free grammar.

i	sentence	$\longrightarrow$ np vp | np compl vp
ii	np	$\longrightarrow$ det noun | det adj noun
iii	vp	$\longrightarrow$ verb np
iv	noun vp	$\longrightarrow$ program vp
v	adj noun vp	$\longrightarrow$ bad program vp | fast program vp
vi	verb det noun	$\longrightarrow$ verb det computer
vii	verb det adj noun	$\longrightarrow$ verb det fast computer
viii	verb	$\longrightarrow$ crashes
ix	det	$\longrightarrow$ the
x	compl	$\longrightarrow$ that

Where in a context-free grammar we would have to use something like the following grammar, which can quickly become unwieldy:

i	sentence	$\longrightarrow$ np vp | np compl vp
ii	np	$\longrightarrow$ det noun | det adj noun
iii	vp	$\longrightarrow$ verb det obj | verb det obj-adj obj
iv	noun	$\longrightarrow$ program
v	obj	$\longrightarrow$ computer
vi	verb	$\longrightarrow$ crashes
vii	det	$\longrightarrow$ the
viii	adj	$\longrightarrow$ fast | bad
ix	obj-adj	$\longrightarrow$ fast
x	compl	$\longrightarrow$ that

3 A*

3.a BFS expansion

closed list	**open list**
	S-11
S	D-8.9, A-10.4
SD	E-6.9, A-10.4
SDE	F-3, B-6.7, A-10.4
SDEF	G-0, B-6.7, A-10.4
SDEFG	B-6.7, A-10.4 — **Done!**

3.b A* expansion

closed list	**accrued**	**open list**
	0	S-11+0
S	0	D-8.9+4, A-10.4+3
SD	4	E-6.9+6, A-10.4+3
SDE	6	F-3+10, A-10.4+3, B-6.7+11
SDEF	10	G-0+13, A-10.4+3, B-6.7+11
SDEFG	13	A-10.4+3, B-6.7+11 — **Done!**

4 Decission tree

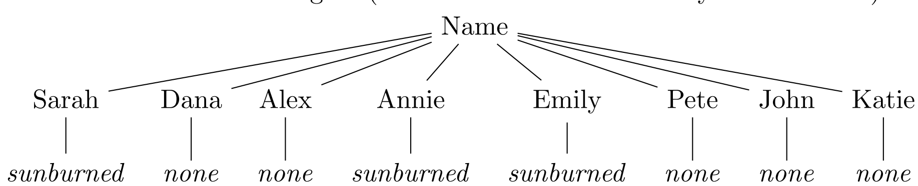
From the table we are given, we can derive the entropy of our observations for the two possible outcomes $sunburnt = \{0, 1\}$.

$$H[sunburnt] = -\frac{3}{8}log_2(\frac{3}{8}) - \frac{5}{8}log_2(\frac{5}{8}) = 0.954434002924965$$

Information gain, $IG(x, y) = H[x] - \sum_y p(y)H[x|y]$ requires calculating conditional entropies given each one of the features. For names, since we have no repeated names, each name is associated with a single outcome, which implies that the entropy of sunburnt *given* a certain name will be 0 for these observations.

$$\begin{aligned} H[sunburnt|Name] =& \sum_n p(sunburnt|Name = n)H[sunburnt|Name = n] \\ =& \sum_n \frac{1}{8} \cdot 0 \\ IG(sunburnt, Name) =& H[sunburnt] - 0 = 0.954434002924965 \end{aligned}$$

Which would make $Name$ an obvious choice for the tree, given the sole IG criterion for deciding, since we have maximal information gain (which is not the case for any other feature).



It must be noted, however, that yielding one leaf per observation is generally due to a poor choice of feature leading to overfitting, and representative of the high variance typical of decision trees. This decision tree does not generalize well.

5 Genetic Algorithms

5.a defining a gene representation

Use a string of 4 hexadecimal digits, a sign and another hexadecimal digit with it's own sign as an exponent. Placing the exponent on one side and the sign on the other would give these two elements some positional independence

5.b fitness function

5.c crossover and mutation - 2 generations for a small initial population of 3

5.d explain the state space - convergence?

5.e how might GA's solve this? Preferable to brute force search?