Artificial Intelligence Tutorial 7 - Answers

1. Suppose you have a variable, x, which can range over the numbers 1 to 20 and a variable, y, which can range over the numbers 5 to 20.

a) Express this constraint on x formally: $9 < x \leq 17$
b) How does this alter the domain of x?

1a) Really, this constraint should be seen as just a statement of x's domain, but to write it as a constraint formally, we write it is as a set of tuples of values which the variable can take. So in this case, the constraint would be:

$C_x = \{(10),(11),(12),(13),(14),(15),(16),(17)\}$

because x can take any value greater than 9 but less than or equal to 17. Note that the values are expressed as (10), (11), etc., rather than just 10, 11, etc., because constraints take on the form of sets of tuples of values. It just so happens that, in this case, the tuples contain only a single element.

1b) The domain of x becomes {10,11,12,13,14,15,16,17}.

c) Ignoring the constraint in part a) and reverting back to the domain for x being 1 to 20, express the following constraint on x and y formally: $x + y = 17$.

d) Now use the constraint in part a) to reduce the size of the constraint in (b)

1c) We must represent the constraint C_{xy} as a set of *pairs* of values that x and y can take simultaneously, as prescribed by the constraint. We must bare in mind the values that x and y are allowed to take independently (their domains), i.e., that x ranges between 1 and 20 and y ranges between 5 and 25. The constraint is therefore:

$C_{xy} = \{(1,16),(2,15),(3,14),(4,13),(5,12),(6,11),(7,10),(8,9),(9,8),(10,7),(11,6),(12,5)\}$

Note that we do not have the pair (13,4) in the constraint, because y cannot take value 4 (it is not in the domain of y).

1d) The domain of x has effectively been reduced in constraint (a) to just the values 10 to 17. So, wherever we see any other value for x in constraint (b), we can remove that pair from the constraint. So, for instance, we can remove the pair (1,16) because x takes value 1 in this solution, which we know it cannot do from constraint (a). Hence the altered constraint becomes:

$C'_{xy} = \{(10,7),(11,6),(12,5)\}$

e) Use your answer from (c) to make the CSP specification arc-consistent. Write down the final formal specification of the CSP.

1e) To make a CSP arc-consistent involves removing values from the domains of the variables which we know cannot possibly appear in a solution, because they are forbidden by a constraint. In binary CSPs, this is done by looking at each *arc* (*a,b*) between a pair of variables *a* and *b*, looking at the corresponding constraint between those variables and removing any values of *a* which do not appear anywhere in the constraints.

As we only have two variables, x and y, then we only need to worry about two arcs, namely (x,y) and (y,x). The constraint from part (c): {(10,7),(11,6),(12,5)} means that, to make the CSP arc-consistent with respect to x, we must remove all variables except 10, 11 and 12 from the domain of x. This makes sense, because any solution to the CSP which didn't have 10, 11 or 12 as the value for x wouldn't be a solution at all, because it would break the constraint from part (c). Hence the domain of x is reduced to: {10,11,12}.

We now have to do the same for the arc (y,x). To make this arc consistent, we need to reduce the domain of variable y to just: {5,6,7}. Hence, finally, we can state our CSP as follows:

Variables: x and y
Domains: d(x) = {10,11,12} and d(y) = {5,6,7}
Constraints: C_{xy} = {(10,7),(11,6),(12,5)}

Note that we do not state the constraint C_x again, as this has been taken care of in the domain of x.

f) Suppose we have the CSP stated in part (d), and this additional constraint on x and y:

$$C^*_{xy}\{(10,6),(10,7),(11,6),(12,6),(12,7)\}.$$

[Note that, in real implementations, constraints C_{xy} and C^*_{xy} would be combined into one constraint because they are both over the same variables, but for the purposes of this exercise, keep the constraints separate].

Suppose further that we have a CSP solving agent which is using the variable ordering: [x,y] and the value ordering [20,19,...,1] for the CSP.

What would be the first possible solution that the agent looked at? What would be the first actual solution that the agent found? What would be the first actual solution the agent found if the variable ordering was reversed?

The variable ordering means that the agent will fix a value for x first and then range over values for y, only changing the value for x if it exhausts all possibilities for y. The value ordering means that it will try the largest value of x in its domain first, namely x=12. Similarly, it will try y=7, then y=6, then y=5 for a solution. If it can't find a solution which appears in both constraints, then it will change x to be x=11, etc. So, the first *possible* solution it will try is x=12 and y=7. As (12,7) is found in neither C_{xy} nor C^*_{xy}, this will not be an actual solution. The solution with x being as large as possible is actually (11,6), so this will be the first found with the variable ordering of [x,y]. With the variable ordering [y,x], the first solution found will be (10,7), because this appears in both constraints (hence is actually a solution), and it is the solution with the largest value of y.

g) Suppose that the agent was equipped with a forward checking mechanism. If the agent set x to be value 10, what would the forward checking mechanism do (temporarily) to the domain of y?

Remember what the forward checking mechanism does: whenever the agent fixes a value, v, for one variable, it removes values temporarily from the domains of the variables which have yet to be fixed. The values removed are those which are forbidden any constraint given that v has been fixed. In our case, with x fixed to 10, then C_{xy} contains only (10,7) and C^*_{xy} contains (10,6) and (10,7). This means that C_{xy} rules out all values for y except 7. Hence the domain of y temporarily gets reduced to {7} by the forward checking mechanism

2. [Taken from 2003 MSc. exam in AI –worth around 10%]

In a cryptoarithmetic puzzle, the variables A, B, C, D, E and F can take values 1 to 7. The variables must all be different and, when taken as digits, they must satisfy the following sum:

```
  A B
+ C D
-----
  E F
```

In addition, $A + C \leq 7$, $B + D \leq 7$, $A < C$ and $B < D$.

a) Model this problem as a CSP with two constraints: one constraint on the variables (A,C,E) and one constraint on the variables (B,D,F). Write your model as a formal CSP.

b) In light of the constraints, what variables can you remove from the domains of variables A to F? Write down the domains after the removals.

c) Suppose the domains were as in ii) and the CSP solving agent was using a variable ordering of {A,C,E,B,D,F} and a value ordering of {1,2,3,4,5,6,7}. Suppose also that the agent was using forward checking. When would the first back-track occur during the search and why?

d) Given A = 1, find a solution to the puzzle.

We may go through this question in the exam lecture next week.

3. Imagine we have a search problem where the solution space is three dimensional, with each dimension being integer valued and ranging from 0 to 100. That is, solutions to the problem are triples of integers (a, b, c) such that 0 <= a, b, c <= 100.

a) How big is the search space for this problem?
b) Represent the number 17 as a bit string.
c) How many bits will solutions to the problem require?
d) How many possible strings are there of the length specified in part 1c?

3a) There are 100 possibilities for each of a, b and c in the solution triple. Hence there are 100*100*100 = 1,000,000 possible solutions.

1b) Reading from right to left, as we do with digits, tens, hundreds, etc. in base ten, the bit string for 17 is: 10001, i.e., 1 lot of 16 + 0 lots of 8 + 0 lots of 4 + 0 lots of 2 + 1 lot of 1 = 17.

3c) We will need to represent three integers which could be as large as 100 each. Now, $2^6 = 64$ and $2^7 = 128$, so to cover integers up to 100, we will need 7 bits. As we will need 3 of these, we will need a total of 21 bits for each possible solution.

3d) Given that each bit can be either on or off (1 or 0), there are $2^{21} = 2{,}097{,}152$ possible bit strings of length 21. As there are only 1,000,000 possible solutions to our problem, it appears that we will be searching a space twice as big as we need. Hence, it would be advisable in this scenario to come up with a more sophisticated bit-string representation, as we could lose an entire bit from our solutions and still have $2^{20} = 1{,}048{,}576$ solutions in the bit string space (enough to cover all possibilities).

4. Some problems in coding theory require the construction of binary code words of minimal weight, i.e., containing as few ones as possible (with certain constraints). Letting w(n) = number of ones in the binary representation of integer n, suppose we define the following evaluation function for integers:

f(n) = w(n) if n is *evil* but not *pernicious*
= w(n)*2 if n is *odious* but not *pernicious*
= w(n)*3 if n is *pernicious*

[where a number is *evil* if it has an even number of ones in its binary representation, a number is *odious* if it has an odd number of ones, and a number is *pernicious* if it has a prime number of ones]. For those of you without a comprehensive school education, prime numbers are defined as having exactly two divisors, which means that 1 is *not* a prime (and hasn't been considered so for over a hundred years). Note that 2 is a prime (the only even one).

a) Which of these numbers is pernicious? Which is evil? What would they score using the evaluation function?

30, 31, 32, 33, 34

b) Suppose an agent is using a GA-style search for bit strings of length 6, and the fitness function divides the above evaluation function, f, for an integer by the average of the evaluation function over all the population [as we described in the GA lecture]. Suppose that the bit strings for the numbers 30 to 34 make up the entire population (with an extra zero put on the LEFT hand side for integers which can be represented with 5 bits). What number would the fitness functions assign each integer? For each integer, how many copies of each are *guaranteed* to go into the intermediate population?

4a) In binary, the numbers are as follows:

30 = 011110 31 = 011111 32 = 100000 33 = 100001 34 = 100100

Note that there is an extra 0 added to the left hand side of 30 and 31. This basically says that there are zero 32's, which is fine, as it does not change the value of the number. If the zero was put on the right hand side, it would change the integer being represented (effectively by multiplying by two).

Hence, 30 has four ones in its binary representation, so is evil (even number of ones) but not pernicious (prime number of ones). 31 has five ones in its binary representation, so is both odious and pernicious. 32 has a single one in its binary representation, so it is odious. 33 and 34 have two ones in their binary representations, so they are both evil and pernicious (because 2 is a prime number). With this information, they will score the following in the evaluation function:

f(30) = w(30) = 4
f(31) = w(31)*3 = 5*3 = 15
f(32) = w(32)*2 = 1*2 = 2
f(33) = w(33)*3 = 2*3 = 6
f(34) = w(34)*3 = 2*3 = 6

Hence, the number 31 scores the most for the evaluation function.

4b) The average score for the population of 5 individuals is (4+15+2+6+6)/5 = 6.6. Hence the fitness of the individuals will be calculated as:

fitness(30) = 4/6.6 = 0.61
fitness(31) = 15/6.6 = 2.27
fitness(32) = 2/6.6 = 0.30

fitness(33) = 6/6.6 = 0.91
fitness(34) = 6/6.6 = 0.91

The integer parts of the fitness values correspond to how many copies of the individual are guaranteed to go into the intermediate population. This means that number 31 will be put into the intermediate population twice for sure, and another copy will be put in with probability 0.27. All of the other integers have a fitness value less than one, so they are not guaranteed to have any copy go into the intermediate population (although 33 and 34 have a high probability of getting a copy in).

c) Perform one-point crossover on the bit-strings for 31 and 32, with the crossover point after the second bit (counting bits from the LEFT in their representation. What numbers are represented by the bit-strings for the two offspring? Calculate the value of the evaluation function for the two offspring.

d) Perform two-point crossover on the bit-strings for 30 and 33, with the crossover points after the first and the fourth bits (counting bits from the LEFT). What integers are represented by the bit-strings of the two offspring? Calculate the value of the evaluation function for the two offspring.

e) Why can there be no other individual which scores higher on the evaluation function than the individual 31?

f) [Optional – for the mathletes amongst you]. Show that *even* perfect numbers (such as 6, 28 and 496, which are equal to half the sum of their divisors) are pernicious. As an aside – an AI program called NumbersWithNames made this discovery.

4c) The bit strings for 31 and 32 are: 011111 and 100000 respectively. To perform one point crossover, we must split the strings into a left hand and a right hand side at the crossover point, which, in this case, is after the second bit. So, this gives us:

LHS(011111) = 01, RHS(011111) = 1111, LHS(100000) = 10, RHS(100000) = 0000

To recombine these using crossover, we put together the left hand and right hand sides of the different strings:

Offspring 1 = LHS(011111)RHS(100000) = 010000
Offspring 2 = LHS(100000)RHS(011111) = 101111

The first offspring is therefore 16 and scores 2*1 = 2 for the evaluation function, as it is odious but not pernicious, and has binary weight 1. The second offspring is 32 + 8 + 4 + 2 + 1 = 47, and scores 5*3=15, as it is pernicious with binary weight 5.

4d) The bit strings for 30 and 33 are 011110 and 100001 respectively. To combine these bit strings using two point crossover, we need to split each string into three: a LHS part, a MIDDLE part and a RHS part. To do this, we split them at the two crossover points, which in this case is after the first bit and after the fourth bit. Hence, after splitting, we get:

LHS(011110) = 0, MIDDLE(011110) = 111, RHS(011110) = 10
LHS(100001) = 1, MIDDLE(100001) = 000, RHS(100001) = 01

To combine these into two offspring, we swap over the middle parts, to give us:

Offspring 1 = LHS(011110)MIDDLE(100001)RHS(011110) = 000010
Offspring 2 = LHS(100001)MIDDLE(011110)RHS(111110) = 111110

Thus, the first offspring is the bit string for the number 2, which scores 1 for the evaluation function, as it is evil but not pernicious. The second offspring is 60, which has evaluation score 15, as it is a pernicious number of weight 5.

4e) 31 is pernicious and has a weight of 5. As the biggest multiplication factor (of 3) is given out by the evaluation function for pernicious numbers, any number with lower weight will score less than 31. Hence, a higher scoring integer must have higher weight. As the strings are of length 6, this means it will have to have weight 6, but then it won't be pernicious, and will score 6 only as it is evil. So, 31 scores the highest possible in the search space.

4f) Perfect numbers are so called because the first two are 6 and 28, and, apparently, God created the earth in 6 days, and made the lunar cycle to be 28 days. No-one could find another perfect number until Euler came along and found 496, which doesn't appear to have any divine inspiration. It was very nice when my NumbersWithNames program told me that perfect numbers are actually pernicious, both for the irony factor and because my friend Dr. Jeremy Gow of University College London invented pernicious numbers. When you write even perfect numbers (we don't know yet whether or not there are any odd perfect numbers) in binary, surprisingly, you get a prime number of ones followed by one fewer zeros, which obviously makes them odious and, more to the point, pernicious. For a proof of why this is always so, see the paper here:

http://www.doc.ic.ac.uk/~sgc/papers/colton_aim02_1.pdf

5a) Represent the following functions as graphs

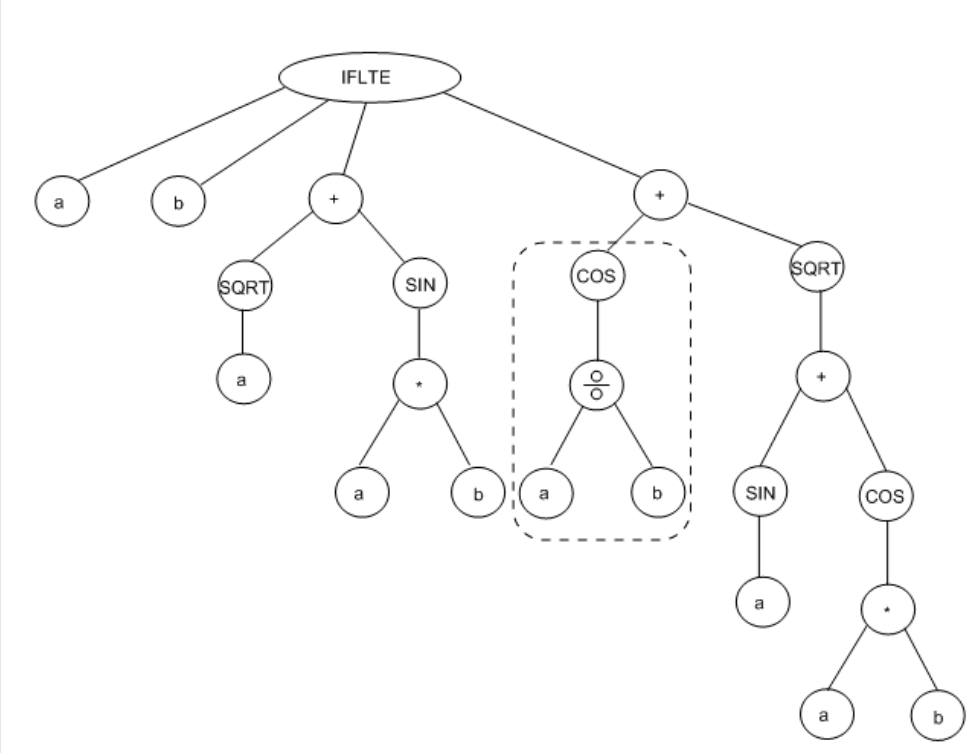
f(a,b) = sqrt(a) + sin(a*b) if a < b
= cos(a / b) + sqrt(sin(a)+cos(a*b)) otherwise

g(a,b) = cos(a / sqrt(sin(b-cos(a))))

b) Perform a crossover, with the crossover fragment for f(a,b) being the subtree representing cos(a/b), and the crossover fragment for g(a,b) being the subtree representing sqrt(sin(b-cos(a))).

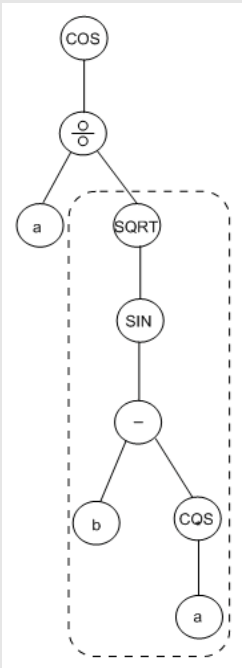
c) If an evaluation function calculated the output value of these functions for the values a=10, b=20, which of the four programs (parents and children) would be the most fit individual? [Calculate the trigonometric functions in degrees, not radians].

5a) The following is a graphical representation for f(a,b). The subtree in the box is the crossover fragment (for part b).

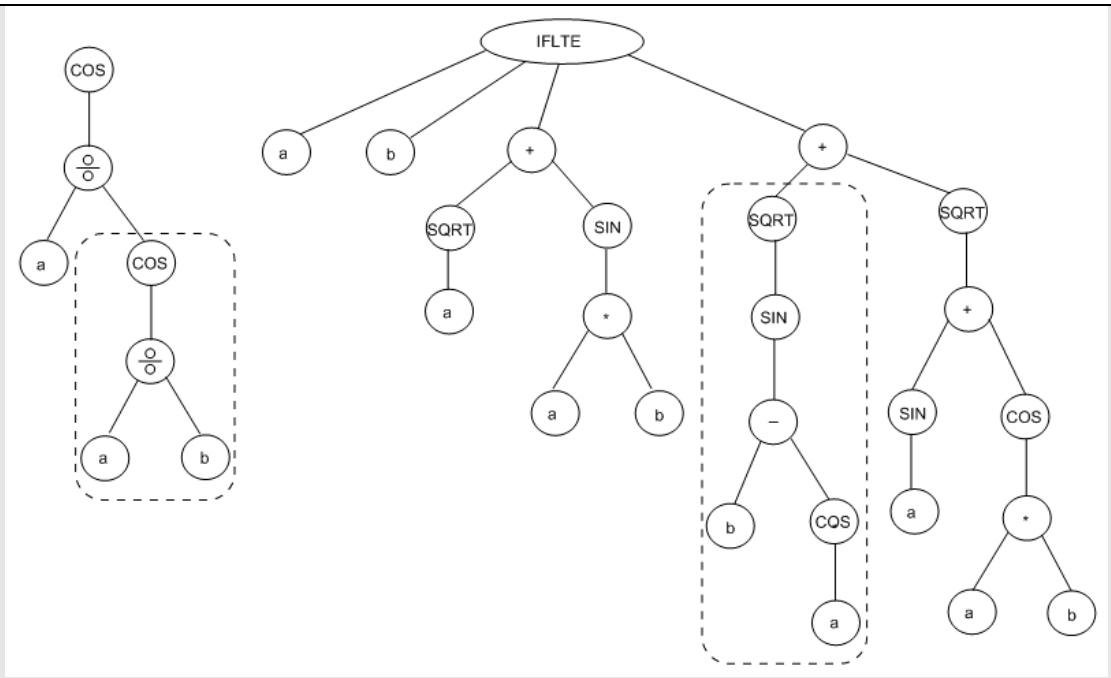


Remember how the IFLTE node works: if the first function below it (on the left) is less than the second function below it, then return the third function below it, otherwise return the fourth function below it.

b) The following is a graphical representation for g(a,b). The subtree in the box is the crossover fragment (for part b).



5b) When we swap the crossover fragments, the following two offspring are produced:



These represent the following two functions

x(a,b) = cos(a / cos(a / b))
y(a,b) = sqrt(a) + sin(a * b) if a < b
= sqrt(sin(b-cos(a))) + sqrt(sin(a) + cos(a*b)) otherwise

5c) To determine which is the most fit individual program, we need to plug the values a = 10 and b = 20 into f(a,b), g(a,b), x(a,b) and y(a,b) as follows:

f(10,20) = sqrt(10) + sin(10*20) [this part is used because 10 < 20]
= 3.662
g(10,20) = cos(10 / sqrt(sin(20-cos(10))))
= 0.954
x(10,20) = cos(10 / cos(10□20))
= 0.984
y(10,20) = sqrt(10) + sin(10*20) [this part is used because 10 < 20]
= 3.662

Hence the two most fit individuals are f and y. We see here that a fit part of the program (in terms of scoring highly for the evaluation function) has been passed on from parent to offspring, which is a good thing for crossover.