NAT-DL: Self-study questions with answers on GP — Set 3 (week 5)

1. Genetic programming (GP) is an evolutionary technique which attempts to evolve programs fit for some purpose. Describe a typical GP system: explain how programs are represented in the system; give examples of the genetic operators applied; and state the main steps of the evolutionary algorithm indicating where there are design choices to make.

   Answer: For an answer to this general question, please see the slides of the lecture in week 4 (GP) or to the "Field Guide" by R. Poli, W. B. Langdon, N. F. McPhee (2008) [http://www.gp-field-guide.org.uk].

2. Express the following functions in tree notation, using only "+", "−", "*", "/" as non-terminals and x, 0, 1, 2, 3, . . . as terminals.

   a) $y = 3x + 2x$

   b) $y = 5x^4 - 2x^2$

   c) $y = -0.25x^3 + 3.5$

   Which of these functions can you represent using only x and 1 as terminals?

   Answers: a) (+ (* 3 x) (* 2 x)) b) (- (* 5 (* x (* x (* x x)))) (* 2 (* x x)))

   c) First, rearrange: $y = 7/2 - [x^3]/4$ which leads to (- (/ 7 2) (/ (* x (* x x)) 4))

   All of them can be expressed using only x and 1 as terminals. Replace 2 with (+ 1 1), 3 with (+ (+ 1 1) 1), etc. There are few additional points:

   1. The trivial expression in a) may be needed as an intermediate step in the generation of an expression like b), so it should not be immediately removed in a running algorithms. This also indicates another form of redundancy (in addition to unused code) which may or may not be useful (i.e., either merely "bloat" or resources for neutral evolution and diversity etc.).
   2. We should note that the unary representation of a 7 as by additions of 1's, may be for the algorithm just as complex as the composition of any other formula from seven elements, such that is crucial to provide (or to develop) a set of operators that makes at least those steps easier that are easy.
   3. We need to make sure that the "-" is overloaded to work also as a unary node, because the adjustments as in c) are no always applicable, unless we want to write (- 0 x) to express a negative sign.

3. As it could be a bit more difficult to program a GP (compared to PSO or GA), you can start this computer exercise a notebook that is already available. It features the function x*x + 2*x + 1 which is to be reconstructed from a data set. The GO should have no problems with this simple function. What do you observe for a slightly more challenging function, such as (x+1)^3?

   Answer: This is a practical exercise, so observations may vary. The intentition was to point to two issue: The simple function should be easily found (where there is a factor 2 in front of the linear term or not as in the lecture example). As soon as the problem gets more complicated, there may be also many variants that explain the data equally well, but may not easily be recognised. Typically this issue is addressed by some form of *parsimony pressure*, a prinicple used everywhere in science to give preference to the simpler one among two theories. Here, for an GP which can be seen “automatic discovery machine”, we need something similar, i.e. fitness dedcution for complecity of the outcome. Among other solutions, the idea of fitness-complexity correlation seems to be reasonable, i.e. larger programs are acceptable only if the fit the data better.

4. What fitness function would GP use for solving symbolic regression problems? Can you think of any alternatives? How much domain specific knowledge about the problem is encoded in this fitness function?

   Answer: We have considered an example in the lecture, where the fitness function was the MSQE on a set of test cases given as a data set. This approach should include crossvalidation, but in principle crossvalidation would require for each individual a new set of test data. We can ignore this issue here, but may as well try to use another approach, leaving thus the cross validation for the final evaluation of the solution that is found by the algorithm. As an alternative, we could use a set of basis function (RBFs, Fourier components, principal components etc.) and simply fit the coefficients to the data, such that the algorithm consist only in deciding how many basis functions are needed. Although basis functions (by definition) can span the data space (if there are sufficiently many), we still would make an assumption whether a small set of them represents the data well. Likewise, we can ask what properties of the data are actually spanned by the basis. E.g., principal components span the data space but ignore higher-order correlations. RBFs are not good if the test-cases that are used in testing are not “within” the training test-cases, but require extrapolation.

5. Do local minima exist in Genetic Programming? What about "building blocks"?

   Answer: Local minima in other MHO algorithms can sometimes be circumvented, if the search space dimension is increased so that a new "escape route" is added. In GP the search space has not a predefined dimension, such that local optima may not exist in general. If, however, the depth or size of the evolving programs is limited then the "escape route" may be locked such that we will have local optimum in this case. For example, if we want to find the function x^4, then x^2 may appear as a local optimum, because x^3 is in between and would perform worse than both x^4 and x^2, if many of the fitness cases are near -1. Nevertheless, in this example, if solutions of the type $x^k + \varepsilon\, x^l$ are not already beyond the complexity limit, then the solution x^2 can be changed into $x^2+ \varepsilon\, x^4$, and evolve further to x^4 . In this way a steady improvement is possible. But not, if out bloat control disallows the branch $\varepsilon *x^4$.

   Another question in this context is the definition a local optimum, we can e.g. state that for a local optimum, a single mutation would always lead to a less fit state. So, in the above example, if adding a term like x^4 is a single mutation, then x^2 is not a local optimum. However, if we have to use several steps as in (*(* x x)(* x x)) then it may be a local optimum. In principle, we can allow to “cut” a sub-tree as single mutation in GP.

   The question of building blocks will be discussed later again (in the context of the schema theorem). Here it is already clear that different subtrees can be seen as building blocks that can exist in subtrees that belong to different individuals. By crossover, they can be combined into new individuals that are then more fit than their parents.

6. A division can be represented by a power series of the form $1/(1-x) = 1+x+x^2+ ...$ How would you implement a GP to learn to represent the right-hand side (i.e. without using a division operation)?

   Answer: For $|x|<1$, the expression on the r.h.s. does not have to be infinitely long in order to represent the result with any desired accuracy. The low order terms are more important in terms of the fitness (e.g. squared difference to true value), such that they are relatively stable and more higher-order terms can be added. Of course, it would be more efficient (for learning but possibly also for computation) to have an iteration operator. For example, an automatically defined subroutine that multiplies that last terminal of the "+" node by the second terminal value and appends the result as a new child to the "+" node. It is possible that such at routine emerges for program construction, but it could also appear in the code itself. Thus, we can ask whether the code the describes the operators that are used by the GP could be evolved just in the same way as the code of the evolving programs. However, we can also allow for the possibility that an iteration (including a counter) is available for the evolving programs.

7. Genetic programming use parse trees to encode solutions. Defining a distance between trees is not a trivial task, but may be useful to keep track of and control the diversity in the population. Propose such a distance to deal with tree-based representations. [Adapted from E. Talbi: Metaheuristics]

   Answer: As we are talking about diversity in metaheuristics here, we are interested in simple measures only, rather than in precise equality. E.g., we can count the size and depth of the trees and can apply some noisification to the population if most individuals coincide w.r.t. to these two criteria. If our algorithm works with trees of a specific size and depth, we could make use of somewhat more informative measures which, however, should avoid the full complexity of the tree-matching problem by using the earth-movers distance between the two:
   – histograms of the path lengths from the root to each of the terminals
   – distributions of degrees across the non-terminals
   – frequency over the functions used in the non-terminals
   or other criteria such as difference in the speed of computation implied by the tree or histograms of such speeds in the sub-trees from a certain level etc.

   A more systematic approach (which was proposed by a student in T06) would be to use the graph-edit distance (GED). Although, in principle, the determination of the GED is an NP-complete problem, there are linear-time approximations that are fully sufficient in metaheuristics (en.wikipedia.org/wiki/Graph_edit_distance).

8. Even-parity-4 problem using GP. In its general formulation, the Boolean even-parity *k* function of *k* Boolean arguments returns true if an even number of its Boolean arguments evaluates true, otherwise it returns false. For the special case where $k = 4$, 16 fitness cases must be checked to evaluate the fitness of an individual. Formulate the objective function for this problem. Our goal is to design a genetic programming approach to solve the problem. Propose a tree-like representation for this problem. For this purpose, the operators and the terminal set of the tree have to be defined. [From E. Talbi: Metaheuristics]

   Answer: Parity functions are always difficult, because every positive answer is right next to a negative one. Having four Boolean arguments, requires us to go through all inputs from (0,0,0,0), (0,0,0,1), … (1,1,1,1), and check whether any function in the population gives the (decimalised) numbers 0, 3, 5, 6, 9, 10, 12, 15

the output “1”, and the other ones (1, 2, 4, 7, 8, 11, 13, 14) a zero. A trivial solution would be to a single non-terminal node at the root and four leaves for the inputs, with the root calculating the desired function. This function may, however, not be present in the set of non-terminals we were starting with. With a XOR function and negation, we can represent the problem as NOT(XOR(XOR (x1, x2), XOR(x3, x4))). Having just AND, OR, and negation, the solution would be more complex, i.e., we would need more than four terminals (i.e., some or all bits enter at several terminals). We don’t have to worry how the solution looks like (or whether the XOR solution is correct), because we are just supposed to set up a GP:
Terminals: x1, x2, x3, x4
Non-terminals: AND, OR, NOT (for the latter case, which seems more interesting)
Mutations: Swapping AND and OR, cutting a branch and inserting NOT, or adding a branch and removing NOT, or putting NOT in between two connected nodes; replacing a terminal by a non-terminal plus appendages or vice versa.
Crossover: As in question 6 above.
Fitness: How many test cases (out of 16) are correctly dealt with.
In principle the description of the algorithm should include also probabilities for crossover and mutation (perhaps different probabilities for different operators?), but we can give here simple the standard values, e.g. p_c=0.7 and p_m=0.1 (p_m is quite large, but this is fine for a small scale problem where we should have at least one mutation per individual). We could also state what population size is reasonable (16?) and how many generations we would expect to need to find the optimal solution with a high probability (100 for the XOR case and 1000 for the AND-OR case?). We could also talk about limits to the individuals, e.g., for the XOR case we could try to fix the number of terminals to 4, which would clearly speed up the search.