CS 687 Fall'21 – Project 3 Report

Chien Wei, Hsiung
Sai Kishore, Salaka

Introduction:

This project contains a High-level Evolutionary Computation framework, which is later used for performing GA on boolean and floating-point vectors, and GP for Symbolic Regression and Artificial Ant problems.

Implementation and Experimentation:

Note:

- EM refers to "Essentials of Metaheuristics" by Dr. Sean Luke.
- Almost all the utility functions were made use of, and the code was written as short and meaningful as possible while functioning as expected.
- Almost all the runs executed in about 10-15 seconds, with the specified parameters.

Top-level Evolutionary Computation:

The `tournament-select-one()`, `tournament-selector()`, and `evolve()` functions were implemented based on the contextual instructions and the respective algorithms in EM. The `evolve()` function runs for a specified number of `generations`, finding two individuals with the best fitness in the `population` mutating them, and creating a new population with better individuals at the end of each run. This function will be used in the later experiments.

Boolean Vector Genetic Algorithm:

The functions `boolean-uniform-crossover()` and `boolean-vector-modifier()` were implemented from EM. In `mutation-boolean()`, the Boolean vector is mutated by randomly shifting the bits from 0 to 1 and vice-versa.

The testing problems were implemented from EM. The parameters chosen and results of the testing problems with those parameters is as follows:

Parameter	Value
`*boolean-vector-length*`	100
`*tournament-size*`	7
`*boolean-crossover-probability*`	0.2
`*boolean-mutation-probability*`	0.01
`generations`	100
`pop-size`	100
`Result`	Best of 50 runs

Problem	Results	Comments
`max-ones-f()`	Almost 1 in every 2 runs finds the 100% optimal solution.	As the fitness is the number of 1s in the vector, the problem easily converges to an optimal solution.
`trap-f()`	Rarely results in a single 1 and rest 0s vector, but mostly all 0s vector.	The function performs poorly as the probability of finding the individual with the best fitness, i.e., (n+1) is almost impossible. It is almost unaffected by the parameters.
`leading-ones-f()`	The best result was around 96, i.e., 96 1s in the beginning of the vector.	The higher the generations is, the better it converges to an optimal solution.
`leading-ones-blocks-f()`	The best result was around 28 for b=3, i.e., 28 strings with 3 1s at the beginning of the vector.	The same goes with this as previous, the higher the generations the better is the chance to find the best individual.

Floating-point Vector Genetic Algorithm:

The functions `gaussian-random()`, `float-uniform-crossover()`, `gaussian-convolution()`, `float-vector-modifier()` were implemented from EM, and the testing problems too. A detailed analysis of the parameters chosen, and the results is below:

Parameter	**Value**
`*tournament-size*`	7
`*float-vector-length*`	20
`*float-min*`	-5.12
`*float-max*`	5.12
`*float-crossover-probability*`	0.1
`*float-mutation-probability*`	0.1
`*float-mutation-variance*`	0.02
`generations`	
`Pop-size`	100

Problem	**Results**	**Comments**
`sum-f()`	Almost reaches the optimal solution. All the values are near to the `*float-max*`	As the fitness is the sum of the floating-points in the individual, it reaches an optimal easily.
`step-f()`	It performs well, but not as good as the `sum-f()`.	The values are in the range of (4.2, 5.1).
`sphere-f()`	Fitness almost reaches zero every time.	Considering how the function looks (in EM), it almost always results in a fitness near to 0.
`rosenbrock-f()`	The fitness tends to be a negative value around -90.	Considering how the function looks (in EM), most of the values lie in the negative axes.
`rastrigin-f()`	The fitness tends to be a negative value around -20.	Very few values reach the maximum float value. Increasing the population provides better results.
`schwefel-f()`	The fitness is around 8000, as the values are multiplied by 100.	Performs better than the previous two problems, and most of the values are near the max value.

Symbolic Regression:

The functions `gp-symbolic-regression-evaluator()` was implemented as suggested, by handling any errors in the calculations. In the `gp-modifier()` when the crossed-over trees go beyond he `*size-limit*` they are replaced by random new trees.

With a `pop-size=500` and `generations=50`, one of the runs provided a 100% optimal solution. For a value of `x = 0.9697573`, the individual $(+(x)(*(-(+(x)(x))(x))(+(x)(*(x)(+(*(x)(x))(x))))))(+(x)(*(-(+(x)(x))(x))(+(x)(*(x)(+(*(x)(x))(x))))))$ provided a fitness of 1.

Verification:

```
CL-USER> (defun x () 0.9697573)
WARNING: redefining COMMON-LISP-USER::X in DEFUN
X
CL-USER> (+ (* (x) (* (+ (x) (* (x) (x))) (x))) (* (+ (x) (cos (- (x) (x)))) (x)))
3.7065818
CL-USER> (+ (x) (* (- (+ (x) (x)) (x)) (+ (x) (* (x) (+ (* (x) (x)) (x)))))) (+ (x) (* (- (+ (x) (x)) (x)) (+ (x) (* (x) (+ (* (x) (x)) (x))))))
3.7065818
```

Artificial Ant:

The functions were implemented as instructed without deviating much. In the resultant `*map*`, a 1 is placed in every location where the ant had gone. One out of 10s of runs provided the 100% optimal solution, with `generations=50` and `pop-size=500`:

```
(PROGN3
 (IF-FOOD-AHEAD (MOVE)
                (IF-FOOD-AHEAD (IF-FOOD-AHEAD (PROGN2 (LEFT) (LEFT)) (MOVE))
                               (PROGN3 (LEFT) (LEFT)
                                       (IF-FOOD-AHEAD (MOVE)
                                                      (IF-FOOD-AHEAD (LEFT)
                                                                     (LEFT))))))
 (MOVE) (LEFT))
CL-USER> (print-map *map*)

ABCD............................
...E............................
...F....................ABCDEF..
...G....................Z....G..
...H....................Y....H..
...IJKLMNOPQR.......TUVWX....I..
............S.......S........J..
............T.......R........K..
............U.......Q........L..
............V.......P........M..
............W.......O........N..
............X.......N........O..
............Y.......M........P..
............Z.......L........Q..
............A.......K..XWVUTSR..
............B...FGHIJ..Y........
............C...E......Z........
............D...D......A........
............E...C......BCDEF....
............F...B..........G....
............G...A..........H....
............H...Z..........I....
............I...Y......NMLKJ....
............J...X......O........
.VUTSRQPONMLK...W...............
.W..............V...............
.X..............U...............
.Y....KLMNOPQRST................
.Z....J.........................
.A....I.........................
.BCDEFGH........................
................................
```