SubProject 2B - Genetic Programming

Tao Zhang
CS 472

March 17, 2014

Abstract

For this subproject, I write the conditional function as steady state, a Mutation function, Crossover function with two trees, and a Tournament selection function. Every function works well.

1 Description

1.1 Conditional

My conditional function is steady state, and here's the brief description.

```
public void steady_state(){
    // initialize all the index for parents and losers
    // initialize son and dau as individual

    int count = 0;
    while(count < pop_size*100){
        father = tournament(good);
        while((mother = tournament(good)) == father){
            // do nothing
            // just make sure father != mother
        }

        // crossover to get children
        crossover(son, dau);

        // mutate
        son.Mutation();
        dau.Mutation();

        // find two losers
        loser1 = tournament(bad);
        while((loser2 = tournament(bad)) == loser1){
            // do nothing
            // just make sure loser1 != loser2
        }
        // replace two losers by chilren

        // refresh the data: size, fitness
    }
}
```

1.2 Selection

I choose tournament selection method, which the function will return the winner index.

```
public int tournament(boolean good){
    int winner = randomGenerator.nextInt(pop_size);
```

```
        double winner_fitness = fitness[winner];
        int tmp;

        int i;
        int N = pop_size/3;
        for(i = 0; i < N; i++){
            tmp = randomGenerator.nextInt(pop_size);
            if(good == true && fitness[tmp] > winner_fitness){
                winner_fitness = fitness[tmp];
                winner = tmp;
            }else if(good == false && fitness[tmp] < winner_fitness){
                winner_fitness = fitness[tmp];
                winner = tmp;
            }
        }

        return winner;
    }
```

1.3 Mutation

My mutation function will generate a random number (0 size), and track the tree while counting until the counter = random number. Delete that subtree's left and right, then use full method to regenerate the subtree.

```
    public void Mutation(){
        int r = randomGenerator.nextInt(terms+non_terms);

        Node sub;
        resetCount();
        // the TrackRoot function will return the node we point to
        sub = TrackRoot(root, r);
        //delete the left and right sub trees

        int max = randomGenerator.nextInt(5);
        // regenerate the subtree with another random max depth full
        full(0, max, sub);
    }
```

1.4 Crossover

My crossover function will read individuals: son and dau, which are the copy of parents. Inside the function:

```
    public void crossover(Individual son, Individual dau){
        Node gene_of_son = new Node();
        Node gene_of_dau = new Node();

        // set random gene position for parents
```

```
        int r_son, r_dau;

        boolean rule = false;

        while(rule != true){
            // generate random position of father's gene
            r_son = randomGenerator.nextInt(son.terms + son.non_terms);
            // point to random gene
            son.resetCount();
            gene_of_son = son.TrackRoot(son.root, r_son);
            if apply 90/10 rule
                rule = true
        }

        rule = false;
        // same way to generate gene_of_dau

        // put father's gene into another tmp individual
        Individual tmp_gene = new Individual();
        tmp_gene.copyNode(tmp_gene.root, gene_of_son);

        // delete father's subgene

        // copy mother's gene onto father
        son.copyNode(gene_of_son, gene_of_dau);

        // delete mother's subgene

        // copy tmp individual (father's gene) onto mother
        dau.copyNode(gene_of_dau, tmp_gene.root);
    }
```

2 Results

I tested all the functions seperately.
The mutation function does generate a new subtree.

- Before mutation: X + X + X / X - 0.00 * 6.00 - X / X = ?
- After Mutation: X + X + X / X - 0.00 * X * X - X + X - X / X = ?

The crossover function does swap the subtree of son and dau

- Before Crossover:
  father: 9.00 + 1.00 - 1.00 * X / 8.00 * X / X * X - X - X - X / X * 3.00 + X + 4.00 - 0.00 = ?
  mother: 2.00 - X * X / 1.00 + 7.00 * 3.00 / X * X = ?

- After Crossover:
  son: 9.00 + 1.00 - 1.00 * X / 8.00 * X / X * X - X - X - X / X * X = ?
  dau: 2.00 - X * X / 1.00 + 7.00 * 3.00 / 3.00 + X + 4.00 - 0.00 * X = ?

The Conditional function and tournament function also works well.

3 Conclusion

Every function works well, but I will come to have some questions after break.