Function ReCognition

by

Karol Bucek

&

Peter Grilli

(as external advisor)

Introduction

Project F-ReC

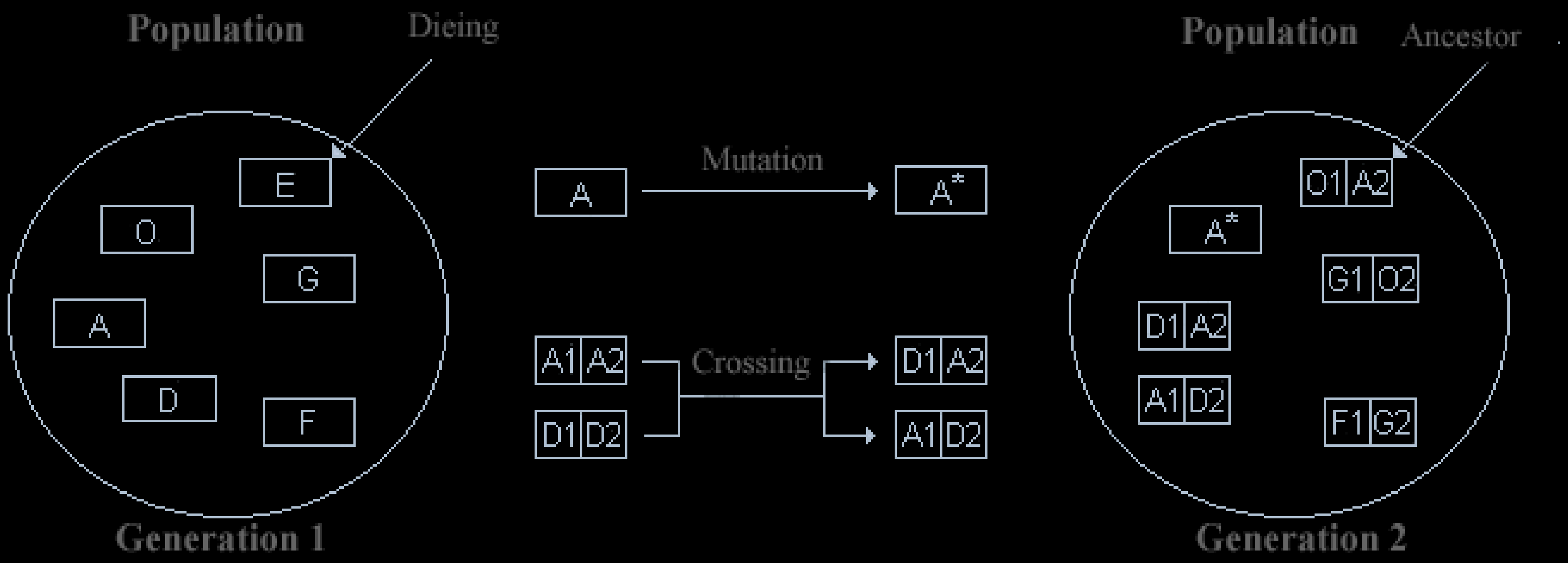
- recognizing 2D functions by a genetic algorithm (GA)
- using genetic programming (GP) schema
- input function provided as a sample graph
- training data computed from the (input) graph
- comparing standard models used in GA and GP
- possibility to easily define own computation models

What is GA ?

- data approximating algorithm
- inspired by the “real” evolution principle
- working with several results at one time
- a result = an individual with “genes”
- generation = several actual results
- fitness = suitability (goodness) of an individual (result)
- individuals applicably represented for genetic operations (mostly as a string) … thus we get a “genetic code”
- mutation = random (genetic) code change(s)
- crossing = code exchange between individuals

GA Schema

Basic GA scheme widely used:

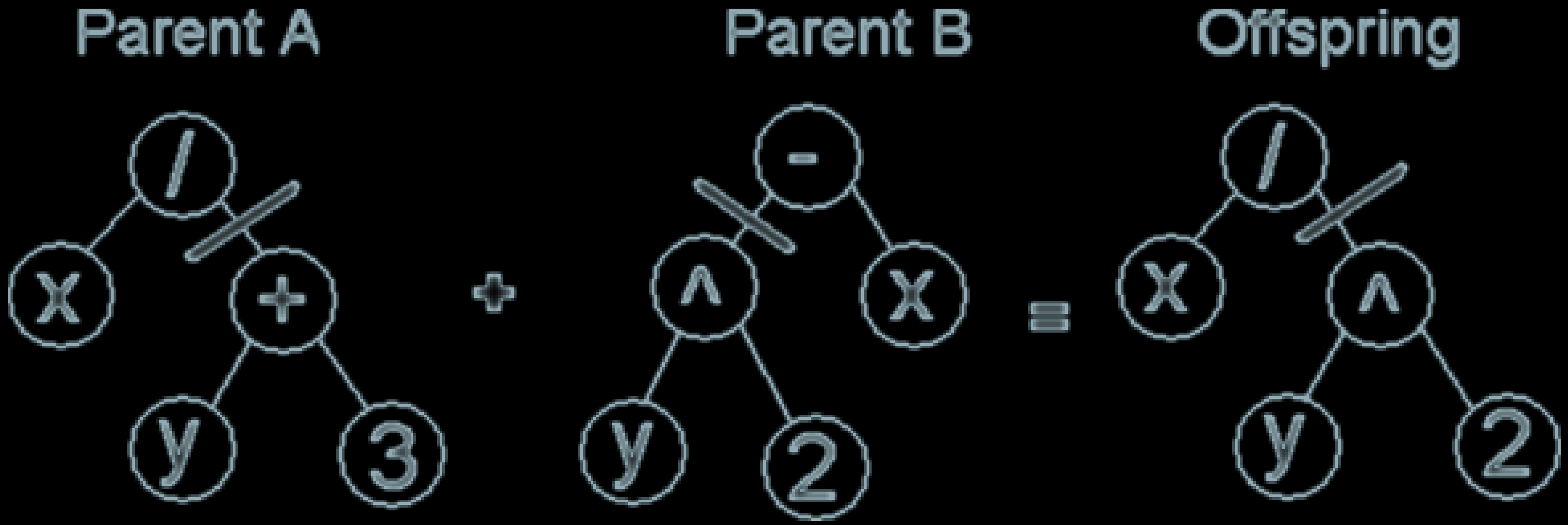


What is GP ?

- GA extension
- dynamic in representation of individuals
- non constant tree based representation
- individual = program that is being “executed” by going throught the tree vertices
- thus an individual is a syntax tree known from predicate calculus
- adaptation of genetic operators
- GA is rewriting the current generation thus getting a new one while GP is creating the new one besides the old one

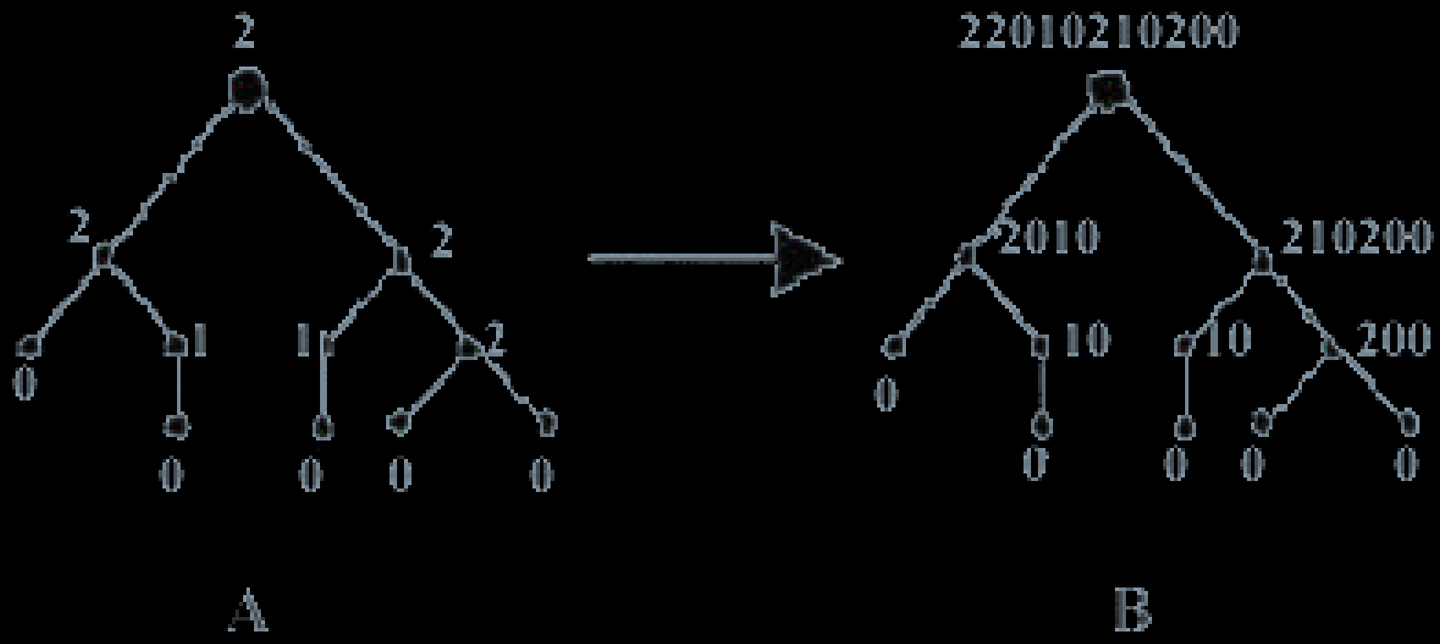
GP Operators

Tree crossing example:



Tree coding

Read’s linear tree coding (very effective for GA):



Problems with GA/GP

- GA problem of early convergence
- individual feasibility (typical for functions)
- !!! crossing does never guarantee a better result !!!
    - the ancestors may moreover be invalid
        - the crossing operator is thus yet another mutation
    - the convergence is very slow
- ??? what shall we do ???
    - accelerating the convergence
    - modifying the crossing operator

Custom Crossing

- extension of standard crossing
- let the operation go through the whole code
- all possible ancestors will be created
- eliminating the invalid ones
- the new results will be the best ones created
- higher probability of getting better results
- further improvements:
  - find code critical points by comparing sub-codes of the parents (backpropagation)
  - might be implemented using a neural network

GA Model

```
initialize G(0);
checkFitnessErrors G(0);
while (notEnoughtValids) addTo G(0);
while (t < max)
        mutate G(t);
        cross G(t);
        checkFitnessErrors G(t);
        checkPopulationErrors G(t);
        selectBest();
        addTo G(t);
        G(t+1) = G(t);
        t++;
```

GP Model

```
initialize G(0);
checkFitnessErrors G(0);
while (notEnoughtValids) addTo G(0);
while (t < max)
        cross G(t) → G(t+1);
        mutate G(t) → G(t+1);
        reproduct G(t) → G(t+1);
        addTo G(t+1);
        checkFitnessErrors G(t+1);
        checkPopulationErrors G(t+1);
        selectBest();
        t++;
```

GY (Custom) Model

```
initialize_adv G(0);
checkFitnessErrors G(0);
while (tooManyValids) removeFrom G(0);
while (t < max)
        mutate G(t) → G(t+1);
        reproduct G(t) → G(t+1);
        cross_my G(t) → G(t+1);
        checkFitnessErrors G(t+1);
        checkPopulationErrors G(t+1);
        addTo G(t+1);
        selectBest();
        t++;
```

Summary

- current version: 1.5, applet version available
- GAModel, GPModel and GYModel implemented
- extensible application core
- results are not sufficient enough yet
- further optimalizations needed
- project site: www.F-ReC.szm.com

Questions

???