Master's Thesis

Adaptive Genetic Algorithms
with
Multiple Subpopulations
and
Multiple Parents

B.A. Thijssen

March 12, 1997

Contents

1 Introduction

In this master's thesis research on genetic algorithms with multiple subpopulations will be handled. In this project each subpopulation has its own crossover operation. Also multiple parents participated in the crossovers. The idea is that the genetic algorithm will adaptively select the subpopulation with the best crossover for the problem it is solving, by increasing its size. This adjusting of the sizes of the subpopulations is done with migration. The research has three goals:

- to examine if multi-parent crossover is better than the standard two parent crossover
- to examine if our adaptive genetic algorithm[1] performs better than a genetic algorithm with a fixed crossover operator
- to examine if our adaptive genetic algorithm is able to detect good crossovers

The genetic algorithm used in this project is the steady-state variant, in which the population size will be the same after each iteration. Although the sizes of the different subpopulations can vary, the sum of the sizes of the subpopulations will remain the same.

2 A genetic algorithm

A genetic algorithm is an algorithm that searches for a solution of a problem, using crossover, mutation and selection. The algorithm has a population of individuals (called chromosomes) which are bitstrings of some fixed length. If a bitstring has length l it will be numbered in the rest of this project from 0 to $(l-1)$. Each individual is a possible solution and has a fitness-value assigned to it. This fitness-value is to be minimized or maximized. In this project the fitness is to be maximized. The fitness-value tells us how close the individual is to a solution. The higher the fitness of an individual, the closer it is to a solution. The genetic algorithm has many variants, but the general form of the genetic algorithm is given in Figure 1. Now an explanation of the algorithm will be given. The variable `t` represents the number of the generation. In `initpopulation P(t)` a population is initialized, where `P(t)` is the population at generation `t`. This is done by randomly taking a number of bitstrings. This number of bitstrings is the size of the population and is a parameter of the genetic algorithm. Also the length of the bitstring

[1] The genetic algorithm selects the crossover operation it will use at run-time.

```
PSEUDO CODE FOR GA

begin GA
  t := 0
  initpopulation P(t)
  evaluate P(t)
  while not done do
    t := t + 1
    P' := selectparents P(t)
    recombine P'
    mutate P'
    evaluate P'
    P(t) := survive P(t), P'
  od
end
```

Figure 1: A Genetic Algorithm

is a parameter of the genetic algorithm. In `evaluate P(t)` the individuals in the population are assigned fitness-values. `while not done do ... od` checks if a solution has been found, and if not executes the body (here the dots) after which the test is done again until a solution is found. But if it takes too long to find a solution it is possible that the genetic algorithm will give up. The tests if a solution has been found or if it takes too long are implemented in the function `done`. In the body of the while a number of things will be done: `t` will be increased, because the algorithm will produce a next generation. Then parents are selected in `selectparents P(t)` and they will be stored in a temporary population `P'`. The selection mechanism used in this project is tournament selection, since that is the most used selection. The parents will recombine with crossover or will do nothing at all in `recombine P'`. This depends on a parameter of the genetic algorithm usually named p_c which is the chance that crossover will occur. Then in `mutate P'` the (un)altered population `P'` is mutated with a chance p_m which is also a parameter of the genetic algorithm. p_m is the chance that a bit will swap value. All the bits in all the individuals of population `P'` will be swapped with that chance p_m. After the crossover and mutation, the population is evaluated again. Now the individuals will be selected to go to the new generation in `survive P(t), P'`. The selection mechanism used for survival in this project, is the survival of the fittest mechanism.

There are of course many variants possible which still are of this structure. One thing which can vary is the crossover operation which will be used for the recombination. The crossovers used in this project were N-Point Crossover and Diagonal Crossover. N-Point Crossover uses two parents, and produces two children. And Diagonal Crossover uses N parents, and produces N children. In this project we also used a variant of N-Point Crossover and of Diagonal Crossover which produces only one child. These variants of the crossovers (with only one child) were used to make a fair comparison between the N-Point Crossover and the Diagonal Crossover.

2.1 Crossovers with multiple children

2.1.1 N-Point Crossover

The N-Point Crossover [DS92] uses two parents and creates two children just as the 'normal' 1-Point Crossover would do. In fact N-Point Crossover is a generalization from 1-Point Crossover. N-Point Crossover selects N crossover points. After that the children are build in the following manner: the first child gets the odd segments from the first parent, and the even segments from the second parent. For the second child the complement holds: it gets the even segments from the first parent, and the odd segments from the second parent. An example for $N = 2$ is given in Figure 2.

2.1.2 Diagonal Crossover

The Diagonal Crossover [ERR94] uses N parents and selects $(N-1)$ crossover points in the bitstring. After that N children are created from combining the segments from the parents. This is done in a diagonal manner. All segments from the parents will be in the same place in the children as they are in the parents. The i-th child has the first segment of the i-th parent, the second segment of the $(i+1)$-th parent, and so on. Each next segment will be taken from each next parent wrapping around at the last parent. This is done for $i = 1$ to $i = N$. An example for $N = 3$ is given in Figure 3.