Reducing Power Consumption in Huffman Coding

Abstract

Keywords:

1. Introduction

2. Background

2.1. Huffman Coding

2.2. Power consumption

2.3. Issues in Biological data transmission

3. GA-SO : Genetic algorithm for Switches Optimising

Genetic algorithm (GA) is one the of the most popular bio-inspired meta-heuristic algorithm inspired from the natural evolution of species []. It is a population based algorithm starts with the generation of a random initial population. The population contain a set of feasible solutions called individuals, that are usually far from the optimal. The GA optimisation process uses a set of natural genetic operators such as selection, crossover and mutation to converge to the optimal solution.

3.1. Preparing data and population generation

Huffman algorithm read the whole message to compute frequencies. The proposed algorithm read the whole genome to generate the frequencies of the triplet and by the way to generate the adjacent matrix which represents the frequencies precedence of between triplet. The initial population is generated randomly, each solution on the population represents a tree that generates a set of codes equal to the total triplets number.

3.2. Selection

The first operator of the genetic algorithm is the selection []. The main objective of the selection is to choose the part of the current population to be a candidate for the different genetic operators in order to breed the next generation population. Many selection techniques have been proposed in the literature []. In this approach the process of natural selection is maintained. First we generate a random pair numbers *rand* between 2 and the population size, this number represent how many parents will be processed. After that, an unbiased random selection of *rand* individuals (solutions) from the population is maid.

3.3. Crossover

The main operator of the GA is the crossover, allows to construct new solutions from the selected part of the population. The selected solutions are ranked by fitness and crossover two by two from high fitness to low. The main objective of the crossover is to benefits from the two good solutions in order to generate a better solution. An internal node for each tree (solution) is selected ($node1$,$node2$), these two nodes must have the same number of leaf nodes (contain the same number of codes). The crossover operation will create two new trees (solutions), the first child (second) contain the nodes that are not children of the $node1$ ($node2$) and we replace the children of $node1$ ($node2$) by the children of $node2$ ($node1$) (see fig.1).

3.4. Mutation

The mutation operator changes the positions of two leaf nodes of the generated children (result of the crossover); this introduce the diversity in the search process, this diversification strategy allow the algorithm do conference to the global optimum. The algorithm for this operator goes through the leaf nodes and change the position of two leaf nodes from a parent to another and selects the ones to change according to a fixed mutation rate (see fig.1).

3.5. Population update

The crossover operation aims to generate new solutions from the current solution to built the second generation of the population. Furthermore, the mutation provide the diversity on the search space solution by changing the position of nodes on the same tree. After these two operations, the next step of the genetic algorithm is two to breed the new population. Firstly the new solutions (children) are added to the current population, after that the

population is ranked by fitness. The algorithm remove the worst solutions until the initial size of the population is achieved. The algorithm genetic repeat these operators until the stopped criteria is achieved, in this algorithm the stopped criteria is when the objective function (number of switches) stop decreasing.

Algorithm 1 Switches optimising Huffman codes

Require: Textual representation of a Genome
Ensure: Low switches Huffman codes
1: Generate triplet frequencies and adjacent matrix
2: Generate the initial population of trees
3: **repeat**
4: Select part of the population
5: Generate children candidates via crossover
6: Mutate children
7: Add children to population
8: Rank by fitness
9: Remove worst solutions until population limit
10: **until** The switches number stop decreasing

4. Experiments and Comparison

4.1. Datasets Description

4.2. Results

5. Conclusion

[1]

Table 1: Comparison of performance among classical Huffman code, CCA, and OCCA without penalty

	Huffman Algorithm	P	
Genome 1	18.16	10.05	44.65
Genome 2	24.66	15.63	36.61
Genome 3	18.46	10.66	42.25
Genome 4	25.18	16.26	35.42
Genome 5	19.24	16.08	16.42
Genome 6	19.61	14.96	23.71
Genome 7	22.40	13.21	41.02
Genome 8	23.87		
Genome 9			
Genome 10			
Genome 11			
Genome 12			
Genome 13			
Genome 14			
Genome 15			
Genome 16			
Genome 17			
Genome 18			

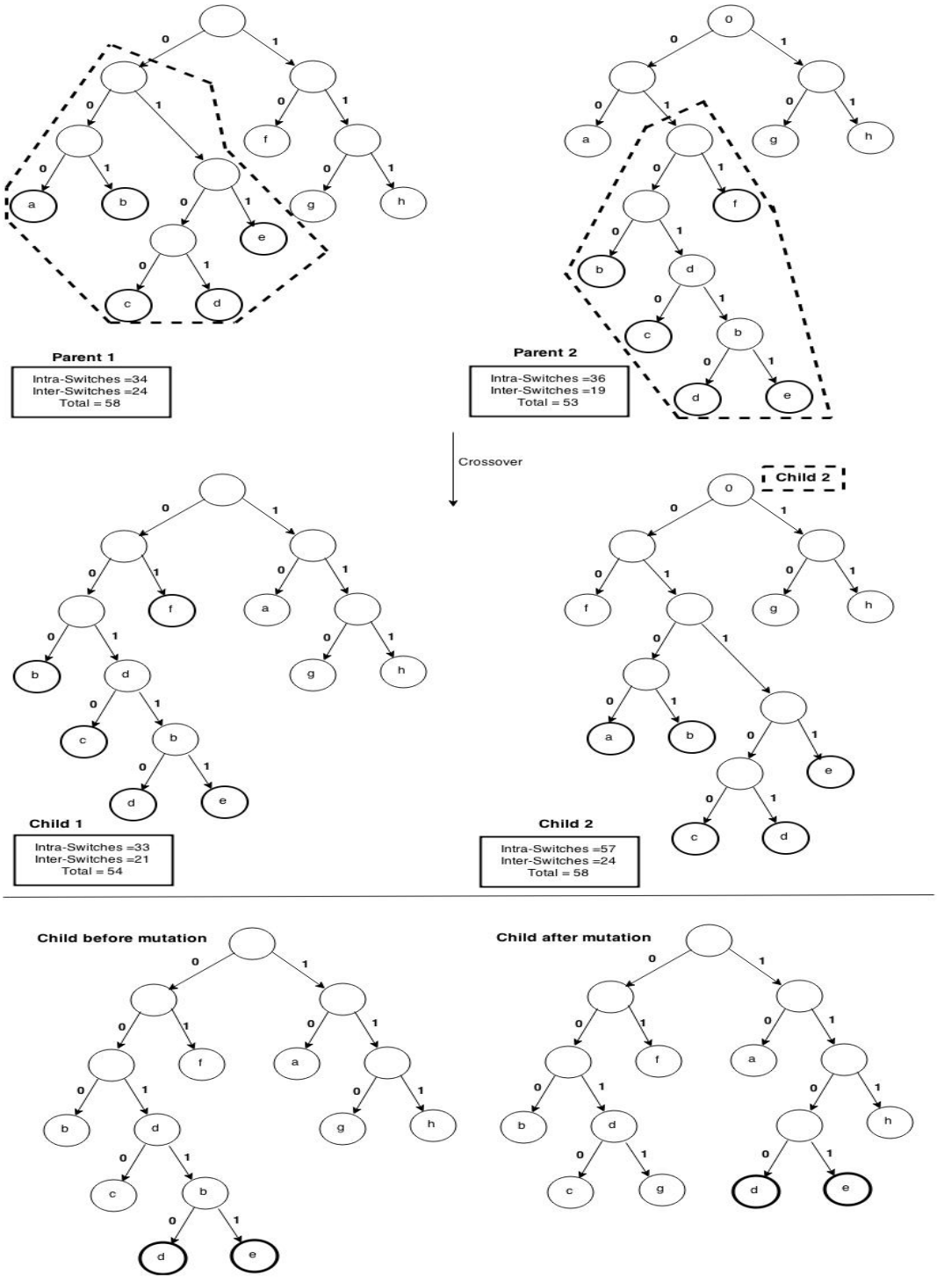

Figure 1: The Crossover operator

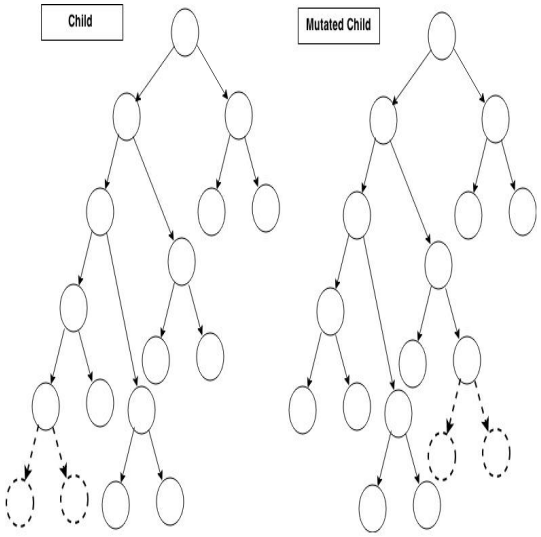

Figure 2: The Mutation operator