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

The size of biological data base increase with an expanding rate and it doesn't stop for increasing. In the last decade the use of biological data history increase which evolve a high request to data from this high volume biological data base. The exponential growth of these databases become a big problem to all biological data processing methods. Data compression methods is the most used methods to evolve the reduce the cost of treatment of high volume data bases. The main objective of data compression methods is minimising the number of bits in the data representation. Authors in [] proposed a new approach to genomic data compression based in suffix and common substring and code the difference with Huffman code. In [16], authors propose a general scheme for coding only the difference between the target genome and the referential one, the coding method used is Huffman. Recent work on data compression for biological data have been proposed to deal with transmission of genomic data as an email attachment [][][]. Cost of data transmission is based on two point, firstly the size of the data which has been improved considerably in the recent year with the improvement of Huffman code. Secondly, is the number of switches on the generated codes. The power consumption increase linear with the number of switches.

3. GA-SO : Genetic algorithm for Switches Optimising

Genetic algorithm (GA) is one the of the most popular bio-inspired metaheuristic 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 *node*1 (*node*2) and we replace the children of *node*1 (*node*2) by the children of *node*2 (*node*1) (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
Generate triplet frequencies and adjacent matrix
Generate the initial population of trees
repeat
    Select part of the population
    Generate children candidates via crossover
    Mutate children
    Add children to population
    Rank by fitness
    Remove worst solutions until population limit
until The switches number stop decreasing
```

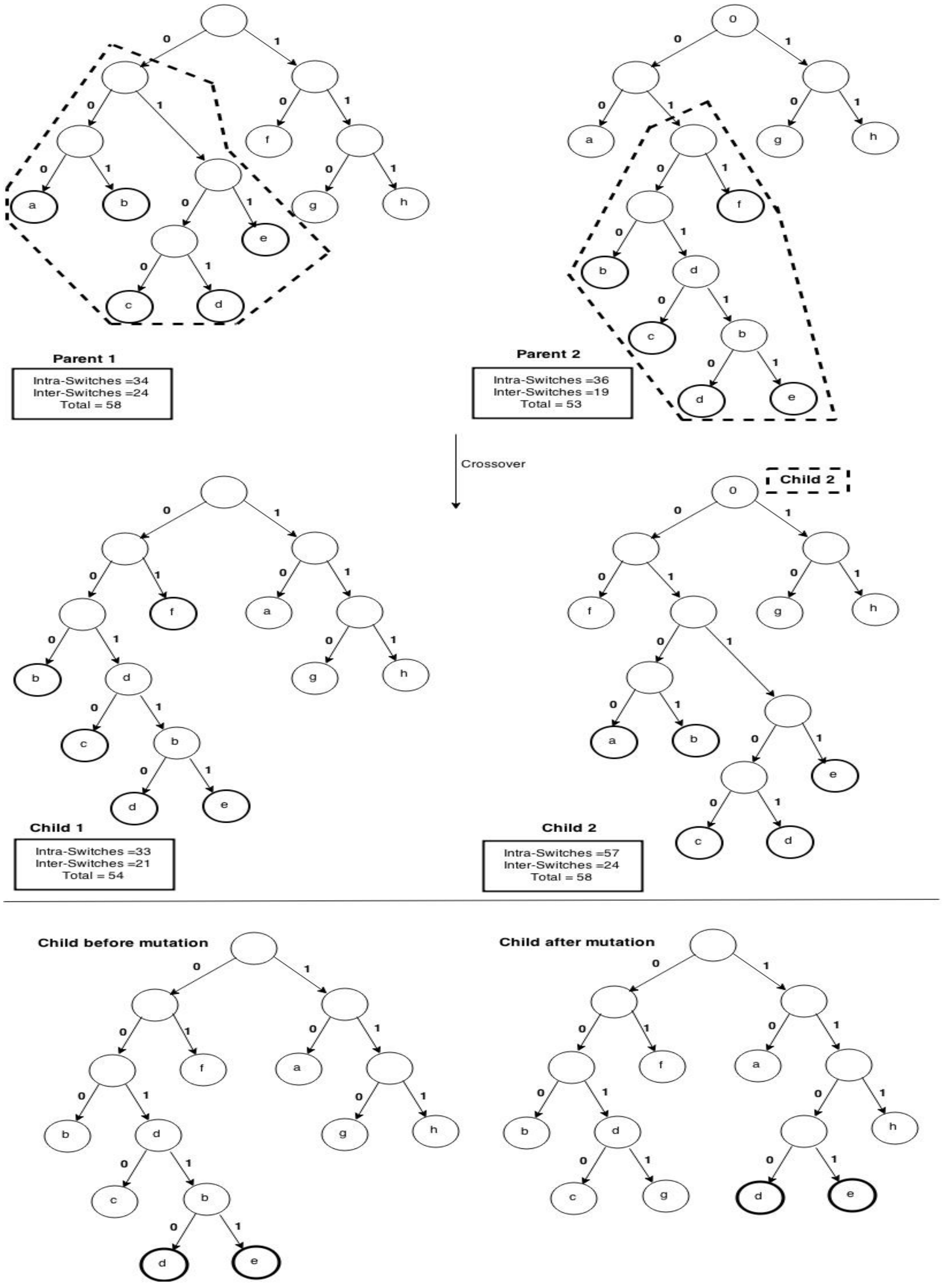

Figure 1: The Crossover operator

4. Experiments and Comparison

The effectiveness of the approach has been evaluated with different real genomic biological data, these genomes were downloaded from a recent version of The National Center for Biotechnology Information (NCBI) available on ($http : //www.ncbi.nlm.nih.gov$) [**?**]. We focused on the sequences alone, ignoring any header and any other exogenous information. In table 3, the different data sets are described with the size of each of them in megabytes (MB) and the references on the biological data bank.

5. Conclusion

[1]

Table 1: Datasets description

Data sets	**Name**	**Size (MB)**	**Reference**
Genome 1	Mycobacterium smegmatis	6.66	CP009496
Genome 2	Amycolatopsis benzoatilytica	8.30	NZ_KB912942
Genome 3	Mycobacterium rhodesiae NBB3	6.11	CP003169
Genome 4	Streptomyces bottropensis ATCC 25435	8.54	NZ_KB911581
Genome 5	Mycobacterium smegmatis str. MC2 155	6.66	CP009494
Genome 6	Mycobacterium smegmatis MKD8	6.76	NZ_KI421511
Genome 7	Bradyrhizobium WSM471	7.42	NZ_CM001442
Genome 8	Amycolatopsis thermoflava N1165	8.27	NZ_CM001442
Genome 9	Bacillus thuringiensis Bt407	5.74	NZ_CM000747
Genome 10	Bacillus thuringiensis serovar thuringiensis	6.03	NZ_CM000748
Genome 11	Pseudomonas aeruginosa 9BR	6.48	NZ_AFXI01000001
Genome 12	Bacillus thuringiensis serovar berliner ATCC	5.97	NZ_CM000753
Genome 13	Bacillus thuringiensis serovar pakistani	5.75	NZ_CM000750
Genome 14	Pseudomonas aeruginosa LES400	6.28	CP006982
Genome 15	Mus musculus chromosome 1	25.58	GL456087
Genome 16	Danio rerio chromosome 1	56.14	CM002885
Genome 17	Homo sapiens chromosome 18	76.64	CM000680
Genome 18	Homo sapiens chromosome 22	99.94	CM000684

Table 2: 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	17.70	25.84
Genome 9	17.66	10.04	43.14
Genome 10	18.14	9.89	45.47
Genome 11	19.63	11.48	41.51
Genome 12	17.59	11.86	32.57
Genome 13			
Genome 14			
Genome 15			
Genome 16			
Genome 17			
Genome 18			