Optimized and Cost Considering Huffman Code For Biological Data Transmission

Abstract

Keywords:

1. Introduction

2. Background

2.1. Issues on Biological Data Transmission

The size of biological data including DNA sequences increase with an ever expanding rate and will be bigger and bigger in the future. These Biological data are stored in biology database, the exponential growth of these database become a big problem to all biological data processing methods. Different operation will be applied to these data such as, searching [],e-mail attachment [], alignment [], and transmission on distributed computing []. Interestingly, biological data compression can play a key role in all biological data processing.

A recent deluge of interest in the development of new tools for biological data processing, these all algorithms needs an efficient methods for data compression. The main objective of data compression methods is minimizing the number of bits in the data representation. In [Marty C. Brandon] authors propose a new general data structure and data encoding approach for the efficient storage of genomic data. This method encode only the differences between a genome sequence and a reference sequence, the method use different encoding scheme from fixed codes such as Golomb and Elias codes, to variables codes, such as Huffman codes. Other methods based on same idea to encode only the difference between reference sequence and the target one, Authors in [Scott Christley1] uses Huffman code for encoding difference between sequence to sent it as an email attachment, but these methods suffer that they must sent the reference sequence for at least one time for each

species.
Wang and Zhang (2011) proposed a new scheme for referential compression of genomes based on the chromosome level. The Algorithm aim to search for longest common subsequence between matching parts and the differences encoded using Huffman coding.
All previous studies focus only on the differences and the relation between continuation of the sequence, and without improvement of the encoding scheme.

2.2. Rationale of Unequal Bit Cost Considering Encoding Approaches

In the recent years, application of battery-powered portable devices, e.g. laptop computers and mobile phones has increased significantly. Proper representation of digital data and their transmission efficiency has become a primary concern for digital community because it affects the performance, reliability, and the cost of computation in both portable and non-portable devices. CMOS technologies were developed in order to reduce the power consumption both in data processing and transmission. In order to increase transmission speed and reduce transmission cost, parallel data transmission methods are widely used. However, parallel transmission is limited to short distance communications, e.g. locally connected devices, internal buses. Ruling out the possible availability of parallel transmission links over long distance, we are left with its serial alternative only.

Data encoding techniques came into action to improve the data transmission efficiency over serial communication medium by compressing data before transmitting. Efficiency can be measured in terms of incurred cost, required storage space, consumed power, time spent and likewise. Data must be encoded to meet the purposes like: unambiguous retrieval of information, efficient storage, efficient transmission and etc. Let a message consist of sequences of characters taken from an alphabet Σ, where $\alpha_1, \alpha_2, \alpha_3 \ldots, \alpha_r$ are the elements that represent the characters in the source Σ. The length of α_i represents its cost or transmission time, i.e., $c(\alpha_i) = length(\alpha_i)$. A codeword w_i is a string of characters in Σ, i.e., $w_i \in \Sigma^+$. If a codeword is $w_i = \alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{in}$, then the length or cost of the codeword is the sum of the lengths of its constituent elements:

$$\text{cost}(w_i) = \sum_{j=1}^{n} c(\alpha_{ij}) \tag{1}$$

If all the elements of a codeword has unit cost or length then the cost of the codeword is equivalent to the length of the codeword. However, it is not necessary for the elements in the codeword to have equal length or cost. For example, in Morse Code all the ASCII characters are encoded as sequence of dots $(\cdot)$ and dashes$(-)$ where a dash is three times longer than a dot in duration [1]. However, the Morse code scheme suffers from the prefix problem [2]. Ignoring the prefix problem, Morse Code results in a tremendous savings of bits over ASCII representation. Using Morse Code, we can treat the binary bits differently; 0 as a dot and 1 as a dash. Even if we consider the voltage level to represent the binary digits then they are still different. Table 1 shows the logic level to represent binary digits in CMOS and TTL technologies.

Table 1: Example of binary logic level

Technology	0	1	Notes
CMOS	$0\ V$ to $\frac{V_{DD}}{2}$	$\frac{V_{DD}}{2}$ to V_{DD}	V_{DD}= supply voltage
TTL	$0\ V$ to $0.8\ V$	$2\ V$ to V_{CC}	V_{CC} is $4.75\ V$ to $5.25\ V$

As the unequal letter cost problem is not new therefore it has been addressed by different researchers. The more general case where the costs of the letters as well as the probabilities of the words are arbitrarily specified was treated in [3]. A number of other researchers have focused on uniform sources and developed algorithm for the unequal letter costs encoding [4, 5, 6, 7, 8]. Let $p_1, p_2, \ldots, p_n$ be the probabilities with which the source symbols occur in a message and the codewords representing the source symbols are $w_1, w_2, \ldots, w_n$ then the cost of the code W is:

$$C(W) = \sum_{i=1}^{n} cost(w_i) .p_i \tag{2}$$

The aim of producing an optimal code with unequal letter cost is to find a codeword W that consists of n prefix code letters each with minimum cost c_i that produces the overall minimum cost $C(W)$, given that costs $0 < c_1 \leq c_2 \leq c_2 \ldots \leq c_n$, and probabilities $p_1 \geq p_2 \geq \ldots \geq p_n > 0$.

2.3. Huffman Codes

In computer science and information theory, Huffman code is an entropy encoding algorithm used for lossless data compression. It takes into account the probabilities at which different symbols are likely to occur and results into fewer data bits on the average. For any given set of symbols and associated occurrence probabilities, there is an optimal encoding rule that minimises the number of bits needed to represent the source. Encoding symbols in predefined fixed length code, does not attain an optimum performance, because every character consumes equal number of bits irrespective to their degree of contribution to the whole message. Huffman code tackles this by generating variable length codes, given a probability usage frequency for a set of symbols. It generates prefix-code to facilitate unambiguous retrieval of information. A scheme of prefix code assigns codes to letters in Σ to form codeword w_i such that none of them is a prefix to another. For example, the codes $\{1, 01, 001, 0001\}$ and $\{000, 001, 011, 111\}$ are prefix-free, whereas the code $\{1, 01, 100\}$ is not, because 1 is a prefix in 100.

Applications of Huffman code are pervasive throughout computer science. The algorithm to completely perform Huffman encoding and decoding is explained by [**?**]. It can be used effectively where there is a need for a compact code to represent a long series of a relatively small number of distinct bytes. For example, Table 1 shows 8 different ASCII characters, their frequencies, ASCII codes and the codewords generated for those symbols using Huffman code. It is seen from the table that the codeword to represent each character is compressed and the most frequent character gets the shortest code. In this example, the compression ratio obtained by Huffman code is 64.16%.

3. Approach

3.1. Proposed Scheme

3.2. Power Efficient Huffman code

3.3. Optimisation of the Codes

3.3.1. Problem formulation

The problem of finding the best allocation of codes to each symbol can be modelled as an Assignment Problems with Constraint, the problem is formulated as follows :

Table 2: Example of application of Huffman Code to compress ASCII characters

Symbols	Frequency	ASCII Code	Codewords using Huffman Code
A	50	01000001	00
B	35	01000010	101
C	42	01000011	110
D	22	01000100	1001
E	65	01000101	01
F	25	01000110	1111
G	9	01000111	1000
H	23	01001000	1110

Definition: Given a set of codes $C = \{C_1, C_2 ... C_n\}$, and a set of frequencies $C = \{Q_1, Q_2 ... Q_n\}$. For each code we have the length of the code $|C_i|$ (number of bits) and the cost of the code S_{C_i} (cost of ones and zeros), the objective is to assign to each frequency a code in order to get the minimum total number of bits, while respecting the initial assignment total cost S_t.
The Objective Function is :

$$Minimise \sum(|C_i| \times Q_j) \tag{3}$$

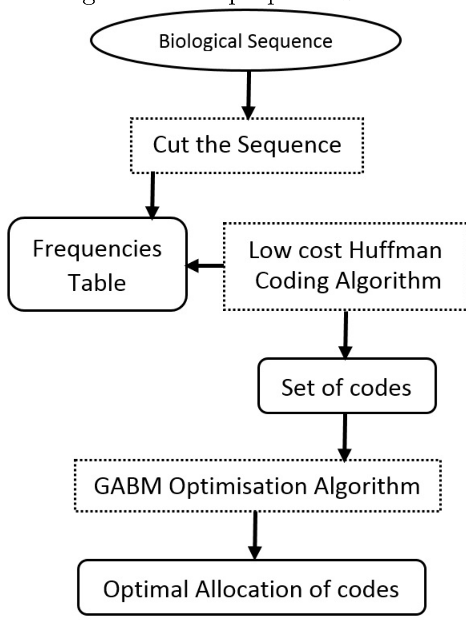
while :

$$\sum(|S_{C_i}| \times Q_j) \leqslant S_t \tag{4}$$

3.3.2. Basic Genetic Algorithm

Genetic Algorithm (GA) is a bio-inspired meta-heuristics algorithm developed by []. GA is a stochastic optimization algorithm imitate the natural evolution process of genomes. GA started by generate a population of random feasible solutions, the optimization process of GA is as follow, and we select two solution among the population, by one of the well-known selection techniques. This two selected solution will be considered as two parents, we generate two other new solutions from the two selected solution (Sons), this new solutions can be mutate according to a given mutation probability. The quality of each solution is computed with the fitness function which control

Figure 1: The proposed Scheme



the evolution of the GA population by the deletion of the worst solution and insertion of the good solutions among parents and sons. This processes is repeated until the stopped criteria is achieved which can be the number of generation or if the population is stabilized.

3.3.3. GA for Bits minimisation

The main objective of the GA optimisation algorithm for bits minimisation (GaBm) problem is to assign to each frequency a specific code. The GaBm population is generated randomly from the different codes, and the affectation of codes to different frequencies given by the low cost considering algorithm to initial population to ensure that the final solution is better or at least equal to the solution given by the low cost considering Huffman code algorithm (step 1). The optimisation process of the genetic algorithms start with the selection of two solution randomly from the population (step 3). After that the operations of genetic algorithms are apply for the initial population optimization (see figure 3). Firstly the crossover operation are applied to these two selected solution (considered as parents) to generate two new solutions (considered as sons)(step 4). These two children may contain conflict like finding a duplicated code allocated to two different frequencies in the solution, so a regulation step is done to ensure the correctness of the solution (see figure 3). Secondly these two new solutions are mutated according to a predefined probability (step 5), to ensure a good diversification on the space solutions. The next step is to add these two new solutions (children) to the population (step 6) (see figure 2). Finally the new population are ranked by fitness (step 7), and the worst solution are deleted until the initial size of the population are achieved (step 8). the whole process are repeated until the max number of operation is achieved (step 9).

Algorithm 1 GA for bits minimisation

1: Population initialization (P).
2: **while** Max number of generation not achieved **do**
3: Select two solutions S_1, S_2 form P.
4: Crossover S_1, S_2 to generate S_{11}, S_{21}.
5: Mutate S_{11}, S_{21}.
6: Add children to population
7: Rank the population by fitness
8: Remove worst candidates until population limit
9: Return to 2;
10: Display the best solution from the population P;

Figure 2: Population Update for genetic algorithm

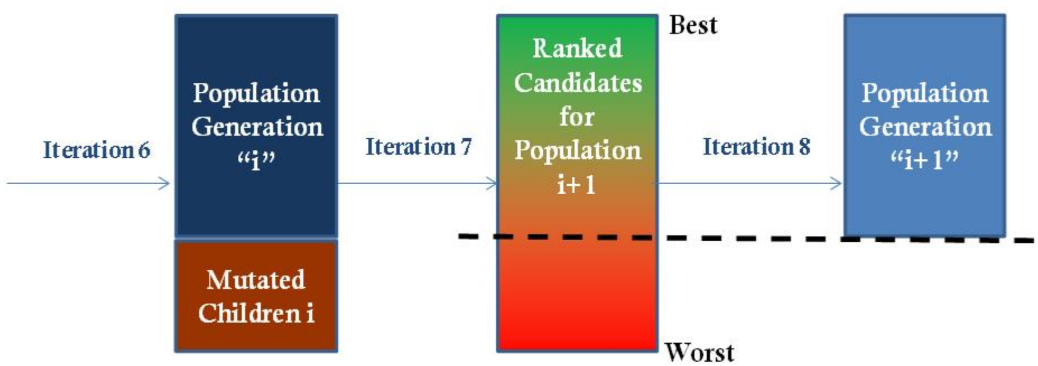
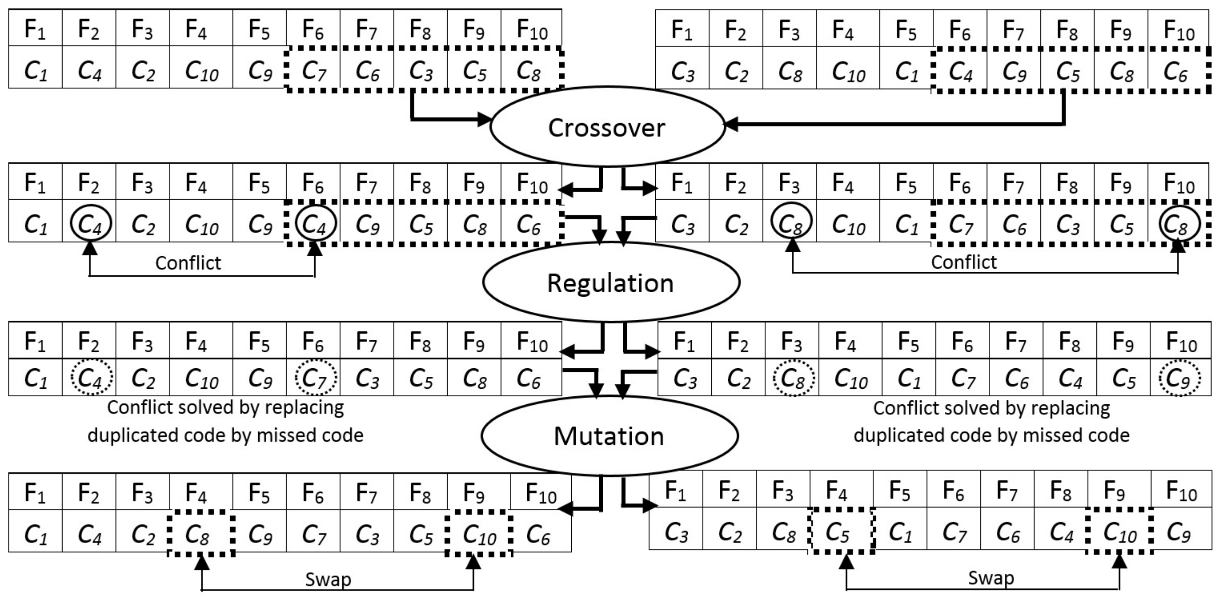


Figure 3: Operations of genetic algorithm



4. Results And Discussion

5. Conclusion

[1] W. A. Redmond, International morse code, Microsoft Encarta 2009 [DVD] (1964) 275–278.

[2] P. D. Grunwald, P. M. B. Vitany, Kolmogorov complexity and information theory, Journal of Logic, Language and Information 12 (2003) 497–529.

[3] R. Karp, Minimum-redundancy coding for the discrete noiseless channel, IRE Transactions on Information Theory 7 (1) (1961) 27–38.

[4] E. N. Gilbert, Coding with digits of unequal costs, IEEE Transactions on Information Theory 41.

[5] R. M. Krause, Channels which transmit letters of unequal duration, Information Control 5 (1962) 13–24.

[6] B. Varn, Optimal variable length codes -Arbitrary symbol cost and equal code word probability, Information Control (19) (1971) 289–301.

[7] D. Altenkamp, K. Mehlhorn, Codes: Unequal probabilities, unequal letter costs, Journal of the Association for Computing Machinery 27 (3) (1980) 412–427.

[8] Y. Perl, M. R. Garey, S. Even, Efficient generation of optimal prefix code: Equiprobable words using unequal cost letters, Journal of the ACM (JACM) 22 (2) (1975) 202–214.