http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

Information Technology Journal 13 (14): 2269-2276, 2014
ISSN 1812-5638 / DOI: 10.3923/itj.2014.2269.2276

A Self-configuration Compression Algorithm for Mass Data Processing

Zeng Jie and Wei Nie
College of Information Engineering, Shenzhen University, Shenzhen,
Guangdong, 518060, China

Abstract: This study presents a self-adaptive intelligent single particle optimizer (AdpISPO) for DNA sequence data compression codebook design. Featured with the crucial self-adaptive optimization process, AdpISPO is capable of attaining better fitness value than most existing PSO variants with no specific parameters required. A novel DNA sequence data compression algorithm, namely BioSqueezer, is proposed in this study. Introducing all the unique data features in constructing the compression codebook, BioSqueezer compresses DNA sequences by replacing similar fragments with the index of its corresponding code vector. For attaining higher compression ratio, the AdpISPO is employed in BioSqueezer for the codebook design. Experimental results on benchmark DNA sequences demonstrate that, BioSqueezer attains better performance than other state-of-the-art DNA compression algorithms.

Key words: Data compression, codebook design, PSO, ISPO

INTRODUCTION

Containing complete genetic information of organisms, the DNA sequence data is used in many frontier subjects, i.e., bioinformatics and molecular biology and have significant value for scientific research. To obtain the genetic information of various creatures, huge amount of DNA sequence data have been produced which has brought a lot pressure on storage and transmission (Cochrane *et al*., 2009). DNA can be expressed as an ultra-long string made of four symbols from the alphabet of {A, T, G, C}. General-purpose compression algorithms failed to attain substantial size reduction when applied on DNA sequence data due to its intrinsic biological nature (Srinivasa *et al*., 2006). Thus new techniques specific for DNA sequence data compression are introduced.

Many achievements have been made since the problem was proposed. Grumbach and Tahi (1994) presented a novel algorithm of BioCompress-2. The algorithm is capable of compressing the information contained in DNA and RNA sequences efficiently by detecting the unique regularities of the sequences such as complementary palindromes and similar fragments. Chen *et al*. (2000) improved the algorithm by introducing approximate repeats in sequence encoding. Significant performance improvement suggested that the approximate repeats are one of the main hidden regularities in DNA sequences. Matsumoto *et al*. (2000) proposed CTW+LZ algorithm by introducing improved context-tree weighting compression in characteristic structures of DNA sequences. Korodi *et al*. (2007) presented an efficient DNA compression algorithm called GeNML which combined specific strategies including Normalized Maximum Likelihood (NML) model in fragments compression. The algorithm was shown to achieve much higher compression ratio in a limit of time.

Most of the existing algorithms exploit statistical redundancy in such a way as to represent the DNA sequence data more concisely without information loss, in which a compression codebook is involved in duplications encoding. Codebooks of existing algorithms are usually constructed by using conventional text compression techniques, i.e., sliding window in BioCompress-2 and context-tree weighting algorithm in CTW+LZ. Performance of these algorithms is degraded for not taking the unique data features of DNA sequences, i.e., special repeat patterns and approximate duplications, into fully concern (Salomon *et al*., 2006). To overcome this drawback, a novel DNA data compression algorithm is proposed, namely BioSqueezer which employs all the unique data features in constructing the compression codebook. A self-adaptive intelligent single particle optimizer (AdpISPO) is introduced to optimize the codebook design.

Intelligent Single Particle Optimizer (ISPO) (Ji *et al*., 2010) is a variation of the Particle Swarm Optimization (PSO) algorithm (Kennedy and Eberhart, 1995) that has been shown to be much more effective than PSO in many optimization problems. Performance of ISPO relies heavily on the settings of input parameters which should be given manually for each independent problem. This makes the algorithm difficult to be used in many situations. To solve this problem, in AdpISPO the crucial parameters of ISPO

Corresponding Author: Zeng Jie, College of Information Engineering, Shenzhen University, Shenzhen, Guangdong, 518060, China

are optimized along with the process of evolution and adjusted dynamically. Experimental results on benchmark functions demonstrate that, AdpISPO, with no specific parameters required, is capable of achieving better performance than the counterpart ISPO and other conventional PSO variants.

With compression codebook fully considered the unique data features in DNA sequences and employing AdpISPO in optimizing the codebook design, the proposed BioSqueezer can attain higher compression ratio than other state-of-the-art DNA data compression algorithms.

DNA SEQUENCE DATA COMPRESSION

DNA is the double-helical biological polymer for long-term hereditary information storage in living creatures. DNA sequence data which is the bioinformatics abstraction of DNA molecules, can be expressed as strings over a four symbol (base) alphabet of Adenine (A), Thymine (T), Guanine (G) and Cytosine (C). Theoretically, if the appearance of bases in DNA sequences is totally random, two binary bits should be used for representing each base, i.e., 2 Bits Per Base (BPB). However, the DNA sequence data, containing enormous genetic information, actually is not random and exist numerous duplications.

In DNA sequence data compression algorithms, these duplications, also called as code vectors, are searched and constituted into a compression codebook. Substituting the duplicate fragments with short code vector indexes will considerably reduce the storage space of the sequence data. As shown in Fig. 1, assume that the original DNA sequence data contains duplicate fragments "ATCCG". These fragments are used as the ith code vector in constructing compression codebook. Thus the DNA data can be compressed by substituting all the duplications of "ATCCG" with its corresponding code vector index i which normally takes only few bits.

The process of DNA sequence data compression is similar as that on common text strings. Thus conversional string data compression techniques are usually employed in DNA sequences compression algorithms. However, unlike ordinary string data, duplications in DNA sequences are widely known to have unique patterns (Gupta *et al*., 2006). As illustrated in Fig. 2, besides the most common direct repeat pattern, the duplicate fragments could also in mirror repeat pattern and according to the base pairing rules of {A-T} and {G-C}, there also exist pairing repeat pattern and inverted repeat pattern. For the example in Fig. 2, given a reference code vector "ATCCG", the direct repeat is exactly the same as

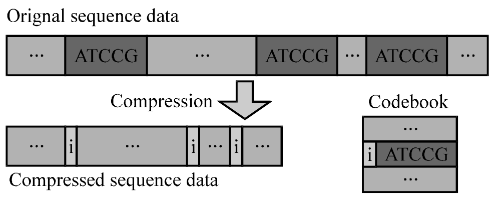

Fig. 1: DNA sequence data compression using a codebook

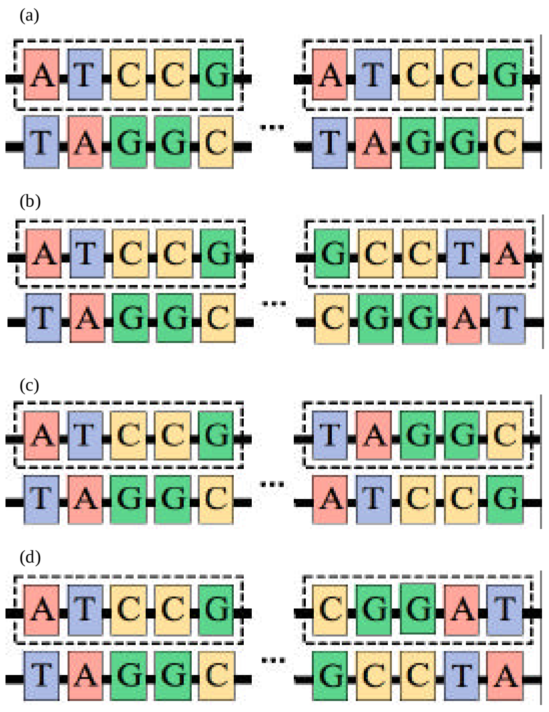

Fig. 2(a-d): Repeat patterns in DNA sequence data, (a) Direct, (b) Mirror, (c) Pairing and (d) Inverted repeat

"ATCCG" while the mirror repeat is the symbols reversed "GCCTA". The pairing repeat "TAGGC" is the complementary sequence of the code vector and the inverted repeat "CGGAT" is the reversion of the pairing repeat. In BioSqueezer all the repeat patterns are involved in codebook design.

Duplications in DNA sequence data appear more in approximate form rather than exact repeats (Tran *et al*., 2004). Base differences including insertions, deletions and substitutions can be found in most of the duplicate fragments. As an example shown in Fig. 3, given a code vector "ATCCG", the first duplicate fragment is considered to have a base difference of symbol "G" inserted in position 2. Similarly, the second fragment has a base "T" deleted from position 2 and the third fragment

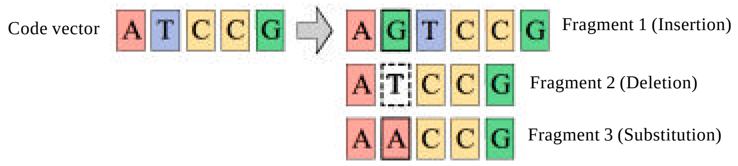

Fig. 3: Base differences in DNA sequence data

have a base "T" on position 2 substituted into "A". Number of base differences between code vector and its duplicate fragments is called edit distance. For the situation in Fig. 3, the edit distance of all three fragments is 1, for each of them can be translated from the code vector using only one symbol operation. Sequence data duplication with base differences, also known as the approximate repeat, could occur on fragments in all four repeat patterns.

In concerning the unique repeat patterns and base differences, duplications in DNA sequence data can be compressed by using following encoding format Eq. 1:

$$v = \{cid, pid, [basediff]\} \tag{1}$$

in which cid is the index of corresponding code vector and pid is the repeat pattern code. For the example in Fig. 1, duplications of "ATCCG" can be encoded as {i, "D"}, where i is the code vector index and "D" denotes direct repeat pattern. The basediff part encodes approximate repeat information which defines as follows in Eq. 2:

$$basediff = \{\{pos_1, eid_1, base_1\}, \{pos_2, eid_2, base_2\}, ...\} \tag{2}$$

where, pos_j is the position of the jth base difference, eid_j is the difference type code, i.e., "I" for insertion, "D" for deletion and "S" for substitution and $base_j$ is the correct base symbol in the duplication. Given the example in Fig. 3, the basediff part of the first fragment can be encoded as {2, "I", "G"}, indicating that there is a base "G" inserted (eid = "I") in position 2. Similarly, fragment 2 and 3 can be encoded as {2, "D"} and {2, "S", "A"}. More base differences the duplication contains, the edit distance is larger and more data bits should be used in recording basdiff information.

The performance of DNA compression algorithm is determined by the quality of its compression codebook. Constructing a codebook that more base symbols can be substituted with its corresponding code vector indexes and fewer base differences in the duplications enhance the compression ratio of the algorithm. In BioSqueezer, the codebook is optimized using AdpISPO algorithm.

SELF-ADAPTIVE INTELLIGENT SINGLE PARTICLE OPTIMIZER

AdpISPO is an effective improvement of ISPO algorithm. Unlike most of the existing PSO variants, ISPO uses only one particle in optima searching and each dimension of the particle position is optimized separately in every generation. The update equations of ISPO are shown as follows Eq. 3-5:

$$V_d^{n+1} = \frac{a}{n^p} \times r + b \times L_d^n \tag{3}$$

$$X_d^{n+1} = \begin{cases} X_d^n + V_d^{n+1} & \text{if fitness}(X^n) > \text{fitness}(X^{n+1}) \\ X_d^n & \text{if fitness}(X^n) \le \text{fitness}(X^{n+1}) \end{cases} \tag{4}$$

$$L_d^{n+1} = \begin{cases} V_d^{n+1} & \text{if fitness}(X^n) > \text{fitness}(X^{n+1} \\ \frac{L_d^n}{s} & \text{if fitness}(X^n) \le \text{fitness}(X^{n+1}) \end{cases} \tag{5}$$

where, V is the velocity and X is the position of the particle. Vector L, namely learning factor, is introduced in regulating the evolution process of each dimension. Variable r is a random number with uniform distribution in the interval of (-0.5, 0.5), n is the number of iterations and d denotes dimension of the vector. The update equations take four parameters: a is the diversity factor, b is the acceleration factor, p is the descend factor and s is the shrink factor. With little effect on the algorithm performance, factors b and s are usually set to b = 2 and s = 4. Experimental results demonstrate that the optimization performance of ISPO is promising with proper settings of the crucial parameters a and p. Yet these parameters could only be given manually for each independent problem.

To overcome the drawback of overly dependence on the settings of crucial parameters, in AdpISPO factors a and p are modified dynamically along with the process of evolution. Realizing that selecting crucial factors values for ISPO can also be treated as an optimization problem, update equations of ordinary PSO algorithm are employed in searching the optimal parameters settings.

As shown in Fig. 4, in AdpISPO parameters a and p are encoded into 2-dimension particles and constituted into a particles swarm, namely Crucial Factors Swarm (CFS) which is updated using following Eq. 6-7:

$$U^{t+1}_{k,d} = w\times U^{t}_{k,d}+c_1\times r_1\times(Pbest_{k,d}-Y^{t}_{d})+c_2\times r_2\times(Gbest_{d}-Y^{t}_{d}) \tag{6}$$

$$Y^{t+1}_{k,d} = Y^{t}_{k,d}+U^{t+1}_{k,d} \tag{7}$$

where, $U_{k,d}$ and $Y_{k,d}$ are the velocity and position value of the kth CFS particle on dimension d. Variable t is the number of CFS generations. Position vector Y_k is encoded from a pair settings of parameters a_k and p_k (Eq. 8):

$$\begin{cases} Y_{k,0} = a_k \\ Y_{k,1} = p_k \end{cases} \tag{8}$$

Variable w is the inertia weight, c_1 and c_2 are the learning factors, r_1 and r_2 are two random numbers with uniform distribution in interval (0, 1). Pbest is the best position a particle ever reaches and Gbest is the global optimal position of the particles swarm. Fitness value of CFS on the tth iteration is denoted by the optimization result of ISPO searching using particle position Y as parameters input Eq. 9:

$$\text{Fitness\_CFS} = \text{ISPO}(X, a = Y_0, p = Y_1) \tag{9}$$

where, ISPO(X, a, p) is the ISPO searching process with initial position vector X using a and p as parameters settings. By introducing CFS in optimizing the factors a and p and using ISPO evolution results as the fitness evaluation, AdpISPO can attain better performance with no crucial parameters settings required.

Procedure of AdpISPO is illustrated in Table 1, in which ps is the particles size of the swarm, M is the maximum generation of ISPO evolution, D is the number of dimension and N is the iteration times for updating each dimension. Termination condition of AdpISPO is usually set as the maximum number of fitness function calls (FEs) being reached.

COMPRESSION CODEBOOK DESIGN USING AdpISPO

In BioSqueezer, the AdpISPO algorithm is employed to search the optimal compression codebook. As shown

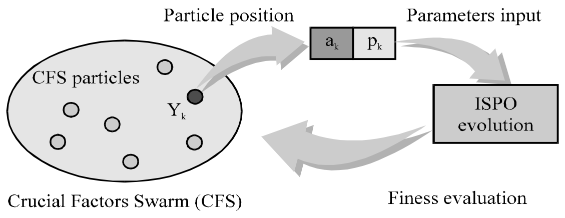

Fig. 4: Crucial parameters selection in AdpISPO

Table 1: Procedure of AdpISPO algorithm

```
1   BEGIN
2   Generate Crucial Factors Swarm (CFS) randomly with ps particles;
    Each particle is encoded by a pair settings of parameters a and p;
3   Initialize pbest position for each CFS particle and gbest position for the particles swarm;
4   Randomly initialize ISPO particle position X;
5   Do until termination condition reached:
6     For k = 1 to ps:
7       Update CFS particle position $Y_k$ based on Eq. 6-7;
      ********************* ISPO Evolution Process *********************
8       Initialize crucial parameters a = $Y_{k,0}$ and p = $Y_{k,1}$;
9       Initialize the maximum number of generations M and the maximum iteration N for updating each dimension;
10      For m = 1 to M:
11        For d = 1 to D:
12          Initialize learning factor L = 0;
13          For n = 1 to N:
14            Update ISPO particle position X based on Eq. 3*4;
15            Update learning factor L based on Eq. 5;
16          End for
17        End for
18      End for
      ******************************************************************
19      Evaluate CFS fitness value on position $Y_k$ based on Eq. 9;
20      Update $pbest_k$ and gbest if necessary;
21    End for
22  Loop
23  END
```

in Fig. 5, connecting each code vectors in the codebook from end to end constitutes the particles position X involved in ISPO(X, a, p) evolution, in which X_d is the position value on dimension d.

For a codebook with h code vectors and each of them contains l base symbols, the particle X is in D = h×l dimensions. As the positions are continuous values while the code vectors are discrete symbol strings, a simple value mapping is performed when translating the particle into compression codebook. The mapping equation used in BioSqueezer is shown as follows in Eq. 10:

$$S_d = \begin{cases} \text{"A"} & \text{if } -\gamma \le X_d < -\frac{\gamma}{2} \\ \text{"G"} & \text{if } -\frac{\gamma}{2} \le X_d < 0 \\ \text{"C"} & \text{if } 0 \le X_d < \frac{\gamma}{2} \\ \text{"T"} & \text{if } \frac{\gamma}{2} \le X_d < \gamma \end{cases} \tag{10}$$

where, S_d is the mapped base symbol of position value X_d. Searching region of the particle positions is $(-\gamma, \gamma)$. Experimental results shown that the optimization performance between different γ values selection is not statically significant. In BioSqueezer γ is normally set to 10.

Translating particle position X into compression codebook, the fitness value is evaluated using following Eq. 11:

$$\text{Fitness\_BioSqueezer} = \frac{1}{\text{Covers} - \text{Errors}} \tag{11}$$

where covers is the overall base number in duplicate fragments that can be substituted by codebook indexes and errors denotes total symbol differences (edit distance) the duplications contain. Larger covers value and smaller errors value, leading to better fitness result, implies that more base symbols is encoded and fewer base differences need to be recorded and the compression ratio is higher. Thus better fitness value denotes better codebook design. In BioSqueezer, the AGREP fast fuzzy string search algorithm (Wu and Manber, 1992) is employed in finding code vectors' duplications and their base differences.

Procedure of BioSqueezer compression codebook design is illustrated in Fig. 6. In each fitness calculation of AdpISPO, particle position X is mapped and constituted into compression codebook. By searching approximate duplications of each code vectors using AGREP algorithm, the Covers and Errors are figured out and calculated into fitness value. Optimizing particle position

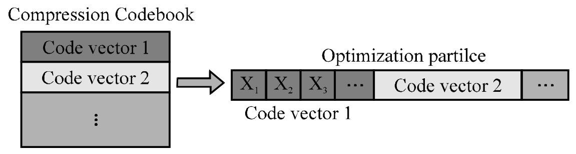

Fig. 5: BioSqueezer optimization particles structure

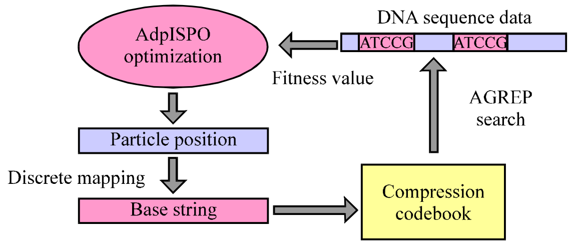

Fig. 6: Procedure of BioSqueezer compression codebook design

Table 2: Parameters settings for AdpISPO

ps	M	N	w	c_1	c_2
10	10	30	0.5	2	2

X gradually improves its fitness value and the compression ratio attained by using the translated codebook is enhanced. Encoding with the codebook can significantly reduce the size of DNA sequence data in BioSqueezer.

SIMULATION RESULTS

In the first experiment, 6 composition benchmark functions (CF1-CF6) proposed by Liang *et al.* (2005) are used for evaluating the performance of the proposed AdpISPO algorithm. These high dimensional multimodal composition functions are considered to be more complex and more proximate to the real-world applications. The PSOw (Shi and Eberhart, 1998), Comprehensive Learning Particle Swarm Optimizer (CLPSO) (Liang *et al.*, 2006) and standard ISPO are selected for comparison. All algorithms are limited to the same maximum number of 1.5E+05 fitness function calls (FEs). In ISPO the crucial parameters a and p are set to a = 150 and p = 10. Parameters settings for AdpISPO are shown in Table 2.

The mean, variances, best and worst fitness values found by all algorithms over 100 runs on the composition functions are compared in Table 3.

The analysis of variance (ANOVA) results and paired t-test results of the best algorithm against other ones are shown in Table 4.

Results in Table 3 demonstrate that, the proposed AdpISPO can attain higher performance, including average, best and worst fitness value, on most of the composition benchmark functions than other algorithms. It is worth highlighting that AdpISPO outperforms standard ISPO with no crucial parameters settings required. The ANOVA and paired t-test results in Table 4 suggest that the performance differences between AdpISPO to other algorithms on most of the benchmark functions are statistically significant at level of α = 0.05. Paired t-test results on function CF3 and CF5 show that the differences between best algorithm CLSPO and AdpISPO is not statistically significant, indicating that the performance of these two algorithms is similar.

In the second experiment the compression performance of proposed BioSqueezer is compared with the BioCompress-2, GenCompress, CTW+LZ and GeNML algorithm base on 11 benchmark DNA sequences(Osborne, 2003). These most commonly used benchmark sequences, belonging to different organisms and containing various data features, are downloaded from the GenBank database (Benson *et al.*, 2008). Details of the benchmark sequences are tabulated in Table 5.

In BioSqueezer the codebook size is set to h = 30 and the code vector length is l = 10. Compression ratio achieved by BioSqueezer and other state-of-the-art

Table 3: Optimization results on 6 composition benchmark functions

	CF1				CF2			
Parameters	Mean	Var.	Best	Worst	Mean	Var.	Best	Worst
PSOw	1.85E+02	8.87E+01	0.00E+00	4.60E+02	2.29E+02	1.29E+02	2.11E+01	6.52E+02
CLPSO	2.70E+01	5.51E+01	1.02E-23	2.47E+02	5.13E+01	6.48E+01	1.63E-02	4.63E+02
ISPO	2.30E+02	1.79E+02	2.05E-32	5.33E+02	3.41E+02	1.67E+02	3.16E+01	8.41E+02
AdpISPO	1.75E+00	1.23E+01	0.00E+00	8.77E+01	4.77E+01	2.40E+01	2.00E-05	8.92E+01
	CF3				CF4			
	Mean	Var.	Best	Worst	Mean	Var.	Best	Worst
PSOw	3.21E+02	1.40E+02	1.19E+02	7.66E+02	4.94E+02	1.78E+02	2.99E+02	8.64E+02
CLPSO	1.53E+02	8.47E+01	7.77E+01	6.99E+02	3.30E+02	6.37E+01	2.76E+02	7.51E+02
ISPO	5.68E+02	2.72E+02	1.32E+02	1.18E+03	6.59E+02	1.38E+02	3.81E+02	1.10E+03
AdpISPO	1.59E+02	2.54E+01	8.92E+01	1.99E+02	4.31E+02	3.97E+01	3.05E+02	5.00E+02
	CF5				CF6			
	Mean	Var.	Best	Worst	Mean	Var.	Best	Worst
PSOw	2.37E+02	1.70E+02	6.24E+00	5.68E+02	8.52E+02	1.32E+02	4.87E+02	9.24E+02
CLPSO	4.52E+01	9.67E+01	9.77E-02	8.06E+02	5.93E+02	1.61E+02	4.52E+02	9.04E+02
ISPO	3.89E+02	2.42E+02	2.87E+01	1.07E+03	8.43E+02	1.40E+02	4.24E+02	9.17E+02
AdpISPO	5.17E+01	2.78E+01	8.34E-01	9.71E+01	5.55E+02	1.06E+02	4.04E+02	9.89E+02

Table 4: ANOVA and paired t-test results on 6 composition benchmark functions

	Paired t-test					
Parameters	Best algorithm	PSOw	CLPSO	ISPO	AdpISPO	ANOVA
CF1	AdpISPO	1.69E-02	3.98E-02	1.89E-02	-	3.51E-55
CF2	AdpISPO	6.96E-03	6.71E-03	6.56E-03	-	1.53E-140
CF3	CLPSO	2.11E-02	-	1.81E-02	5.38E-01	5.57E-65
CF4	CLPSO	2.28E-02	-	1.68E-02	1.83E-02	4.46E-60
CF5	CLPSO	2.02E-02	-	1.82E-02	5.85E-01	2.32E-52
CF6	AdpISPO	2.31E-02	1.80E-02	2.31E-02	-	4.41E-40

Table 5: Benchmark DNA sequence data

Sequence	Length	Summary
CHMPXX	121024	Marchantia polymorpha chloroplast genome DNA
CHNTXX	155943	Nicotiana tabacum chloroplast genome DNA
HEHCMVCG	229354	Human cytomegalovirus strain AD169 complete genome
HUMDYSTROP	38770	*Homo sapiens* dystrophin (DMD) gene, intron 44
HUMGHCSA	66495	Human growth hormone (GH-1 and GH-2) and chorionic somatomammotropin (CS-1, CS-2 and CS-5) genes, complete cds
HUMHPRTB	56737	Human hypoxanthine phosphoribosyltransferase (HPRT) gene, complete cds
HUMHDABCD	58864	Human DNA sequence of contig comprising 3 cosmids (HDAB, HDAC, HDAD)
HUMHBB	73308	Human beta globin region on chromosome 11
MPOMTCG	186609	Marchantia polymorpha mitochondrion, complete genome
SCCHRIII	316613	*S. cerevisiae* chromosome III complete DNA sequence
VACCG	194711	*Vaccinia virus*, complete genome

Table 6: Compression ratio on 11 benchmark DNA sequence data

Sequence	Bio2	Gen	CTW	GeNML	Biosqueezer
CHMPXX	1.684	1.673	1.669	1.661	1.582
CHNTXX	1.617	1.614	1.612	1.613	1.628
HEHCMVCG	1.848	1.847	1.841	1.839	1.611
HUMDYSTROP	1.926	1.923	1.916	1.912	1.683
HUMGHCSA	1.307	1.097	1.097	1.012	1.527
HUMHPRTB	1.913	1.846	1.844	1.758	1.689
HUMHDABCD	1.882	1.821	1.823	1.713	1.752
HUMHBB	1.881	1.819	1.808	1.796	1.668
MPOMTCG	1.942	1.913	1.907	1.883	1.784
SCCHRIII	1.948	1.948	1.945	1.937	1.712
VACCG	1.764	1.763	1.739	1.763	1.532
Average	1.816	1.794	1.787	1.772	1.652

algorithms are compared in Table 6, where Bio2 is abbreviation of BioCompress-2, Gen represents GenCompress and CTW denotes the CTW+LZ algorithm. The result values are given in the form of BPB.

From Table 6, it can be figured that, the BioSqueezer achieves smaller BPB than other DNA compression algorithms on most of the benchmark sequences. Average BPB value of BioSqueezer is the best among all the techniques. The compression ratio is observed to maintain stable among all the benchmark sequences which indicates that BioSqueezer algorithm have high robustness over different types of DNA data. The improvement of compression ratio will definitely save data space for storing and transmitting the highly expending DNA sequence data. Based on the average BPB value 1.652 achieved by BioSqueezer, a DNA sequences file in 1 GB data size can be compressed to about 211.5 MB.

CONCLUSION

A novel DNA sequence data compression algorithm, namely BioSqueezer, is proposed in this study. By introducing unique data features of DNA sequences in compression codebook's construction, BioSqueezer compresses the sequence data by replacing approximate duplications with the index of its corresponding code vector. For attaining higher compression ratio, a PSO algorithm improvement called AdpISPO is proposed and employed to optimize the codebook design. Experimental results on 11 benchmark sequences demonstrate that, BioSqueezer achieves better performance than other state-of-the-art DNA compression algorithms.

ACKNOWLEDGMENTS

This study was supported by the National Natural Science Foundation of China (61103174, 61301182), Scientific and Technological Innovation Project of Department of Education of Guangdong Province (2013KJCX0162, 20134408120004), Natural Science Foundation of Guangdong Province (China) (S2013040016857), Fundamental Research General Program of Shenzhen City (JCYJ20120613113535357, JCYJ20130329105415965).

REFERENCES

Benson, D.A., I. Karsch-Mizrachi, D.J. Lipman, J. Ostell and D.L. Wheeler, 2008. GenBank. Nucleic Acids Res., 36: D25-D30.

Chen, X., S. Kwong and M. Li, 2000. A compression algorithm for DNA sequences and its applications in genome comparison. Proceedings of the 10th Workshop on Genome Informatics, December 14-15, 1999, Tokyo, Japan, pp: 51-61.

Cochrane, G., R. Akhtar, J. Bonfield, L. Bower and F. Demiralp *et al*., 2009. Petabyte-scale innovations at the European nucleotide archive. Nucleic Acids Res., 37: D19-D25.

Grumbach, S. and F. Tahi, 1994. A new challenge for compression algorithms: Genetic sequences. Inform. Process. Manage., 30: 875-886.

Gupta, R., A. Mittal and S. Gupta, 2006. An efficient algorithm to detect palindromes in DNA sequences using periodicity transform. Signal Process., 86: 2067-2073.

Ji, Z., J.R. Zhou, H.L. Liao and Q.H. Wu, 2010. A novel intelligent single particle optimizer. Chinese J. Comput., 33: 556-561.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. Proceedings of the International Conference on Neural Networks, Volume 4, November 27-December 1, 1995, Perth, WA., USA., pp: 1942-1948.

Korodi, G., I. Tabus, J. Rissanen and J. Astola, 2007. DNA sequence compression-based on the normalized maximum likelihood model. IEEE Signal Process. Magaz., 24: 47-53.

Liang, J.J., P.N. Suganthan and K. Deb, 2005. Novel composition test functions for numerical global optimization. Proceedings of the IEEE Swarm Intelligence Symposium, June 8-10, 2005, Pasadena, USA., pp: 68-75.

Liang, J.J., A.K. Qin, P.N. Suganthan and S. Baskar, 2006. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Tans. Evol. Comput., 10: 281-295.

Matsumoto, T., K. Sadakane and H. Imai, 2000. Biological sequence compression algorithms. Proceedings of the 10th Workshop on Genome Informatics, December 14-15, 1999, Tokyo, Japan, pp: 43-52.

Osborne, M., 2003. Predicting DNA sequences using a backoff language model. http://www.osti.gov/eprints/topicpages/documents/record/515/3862803.html

Salomon, D., G. Motta and D. Bryant, 2006. Data Compression: The Complete Reference. 4th Edn., Springer, USA.

Shi, Y. and R. Eberhart, 1998. A modified particle swarm optimizer. Proceedings of the World Congress on Computational Intelligence and IEEE International Conference on Evolutionary Computation, May 4-9, 1998, Anchorage, AK., pp: 69-73.

Srinivasa, K.G., M. Jagadish, K.R. Venugopal and L.M. Patnaik, 2006. Efficient compression of non-repetitive DNA sequences using dynamic programming. Proceedings of the International Conference on Advanced Computing and Communications, December 20-23, 2006, Surathkal, pp: 569-574.

Tran, T.T., V.A. Emanuele and G.T. Zhou, 2004. Techniques for detecting approximate tandem repeats in DNA. Proceedings of the International Conference on Acoustics, Speech and Signal Processing, May 17-21, 2004, Montreal, Quebec, Canada, pp: 449-452.

Wu, S. and U. Manber, 1992. Fast text searching: Allowing errors. Commun. ACM, 35: 83-91.