Since January 2020 Elsevier has created a COVID-19 resource centre with free information in English and Mandarin on the novel coronavirus COVID-19. The COVID-19 resource centre is hosted on Elsevier Connect, the company's public news and information website.

Genomics 112 (2020) 237–242

Contents lists available at ScienceDirect

Genomics

journal homepage: www.elsevier.com/locate/ygeno

Original Article

CodSeqGen: A tool for generating synonymous coding sequences with desired GC-contents

Abdulrakeeb M. Al-Ssulami[a,b], Aqil M. Azmi[a,*], Muhammad Hussain[a]

[a] Department of Computer Science, College of Computer & Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia
[b] Department of Computer Science, Faculty of Applied Sciences, Taiz University, Taiz, Yemen

ARTICLE INFO

Keywords:
Synonymous coding sequence
Sequence analysis
GC-content

ABSTRACT

Identification of regulatory elements is essential for understanding the mechanism behind regulating gene expression. These regulatory elements—located in or near gene—bind to proteins called transcription factors to initiate the transcription process. Their occurrences are influenced by the GC-content or nucleotide composition. For generating synthetic coding sequences with pre-specified amino acid sequence and desired GC-content, there exist two stochastic methods, multinomial and maximum entropy. Both methods rely on the probability of choosing the codon synonymous for usage in regard to a specific amino acid. In spite the latter exhibited unbiased manner, the produced sequences are not exactly obeying the GC-content constraint. In this paper, we present an algorithmic solution to produce coding sequences that follow exactly a primary amino acid sequence and a desired GC-content. The proposed tool, namely CodSeqGen, depends on random selection for smaller subsets to be traversed using the backtracking approach.

1. Introduction

The two regulatory elements, promoters which are near the coded area and enhancers which are further upstream or downstream, are DNA sequences located in the non-coding regions and bind to proteins called transcription factors. These interactions enable RNA polymerase from transcribing the related gene to produce mRNA which in turn is translated to a specific protein. Identifying such sequences is important to understand the mechanism of regulating gene expression. The occurrences of regulatory elements are highly influenced by common features such as GC-content [1], di-nucleotide profile [2], and codon bias [3,4]. Thus, identifying over/under-represented regulatory elements or genome-scale patterns relies on generating random sequences that obey the pre-specified amino acid sequence and GC-content constraints.

There are many tools that are used to generate random sequences with various constrains. The well-known ones include: SMS [5], FaBox [6], and GenRGenS [7]. SMS tool generates random coding sequences of specific length given the translation table. The primary goal of this tool is evaluating the results of sequence analysis. FaBox is used to construct random DNA sequences with a predefined nucleotide composition. GenRGenS creates random sequences with several models, such as Markov chains, hidden Markov models, weighted context-free grammars, and others. The sequences generated by GenRGenS are essentially used for structural motif evaluation. Although, these tools generate random DNA and coding sequences, none of them are capable of producing coding sequences given the amino acid sequence and GC-content.

Generating random coding sequences in response to complicated constraints is computationally expensive. This is because the number of codons are $1 \sim 6$ for each amino acid. Thus, for a particular protein sequence of length n, we have to test at most 6^n coding sequences to find those satisfying the desired amino acid sequence and GC-content constraints. That is, the time complexity is exponential with respect to the length of amino acid sequence. Currently, there are two solutions to solve this problem, multinomial [8,9] and NullSeq [10]. Multinomial method chooses the synonymous codon C of amino acid a with probability $P_a(C)$ given the nucleotide composition. The probability $P_a(C)$ is computed as a normalized product of probability distribution of the individual nucleotides within the codon C. As shown in [10], the multinomial method generates unbiased coding sequences. A more restricted method was presented recently, which the authors named NullSeq. NullSeq [10] uses the maximum entropy approach where the synonymous codon usage probability is derived from a strict function that expresses the expected GC-content in the reference amino acid sequence. However, majority of the resulting coding sequences defy the GC-content constraints.

In this paper, we present CodSeqGen, the first exact solution to produce synonymous coding sequences with the desired GC-contents accurately. The proposed method uses the backtracking approach which is smarter than an exhaustive search, where only promising candidate solutions are tracked instead of all possible combinations.

* Corresponding author.
E-mail addresses: rakeeb_sulami@yahoo.com (A.M. Al-Ssulami), aqil@ksu.edu.sa (A.M. Azmi), mhussain@ksu.edu.sa (M. Hussain).

https://doi.org/10.1016/j.ygeno.2019.02.002
Received 30 October 2018; Received in revised form 29 January 2019; Accepted 2 February 2019
Available online 06 February 2019
0888-7543/

The rest of the paper is divided as follows. Section 2 covers the implementation details of CodSeqGen. The results of testing our method on different set of proteins and comparison against NullSeq is in Section 3. The last section concludes the paper.

2. Methods

Our proposed method utilizes the backtracking approach. Backtracking is more efficient than the exhaustive search when there is a very large search space where not all combinations of raw components are inspected. This technique is similar to tree depth-first search and depends on adding one component to the candidate solution at a time. Where, as soon as a violation for constraints is detected, the algorithm discards the current solution and backtracks to the parent to seek another candidate solution, see e.g., [11]. Since the reference amino acid sequence is long, it is infeasible to apply the backtracking approach directly. This is because a very large amount of memory will be required to generate the coding sequences. For tackling this problem, we devised a trick to permute the indices of the reference amino acid sequence. Thus, random sets are selected without interferences and backtracked to produce coding sequences with a random distribution for the GC-content over the whole sequence, Fig. 1 depicts our methodology.

Formally, let P refer to the reference amino acid sequence of length n, and $\widehat{P}$ refer to the corresponding DNA coding sequence of length $3n$. Assuming φ is the GC-content of $\widehat{P}$, then the problem is to generate a random set of synonymous coding sequences each of φ GC-content which in turn could be translated to P. Let Φ be the set of indices of P, where $\Phi = \{0, 1, \ldots, n-1\}$. By maintaining the set of indices, amino acids within the reference amino acid sequence can be accessed easily. According to the proposed methodology in Fig. 1, Φ is divided into smaller subsets each of size s, with the last subset being a fraction of s. Thus, the total number of subsets $\Gamma = \lceil n/s \rceil$. Each subset G_i is constructed by randomly selecting n indices, where the subsets are disjoint. In other words, the constructed subsets must satisfy three conditions.

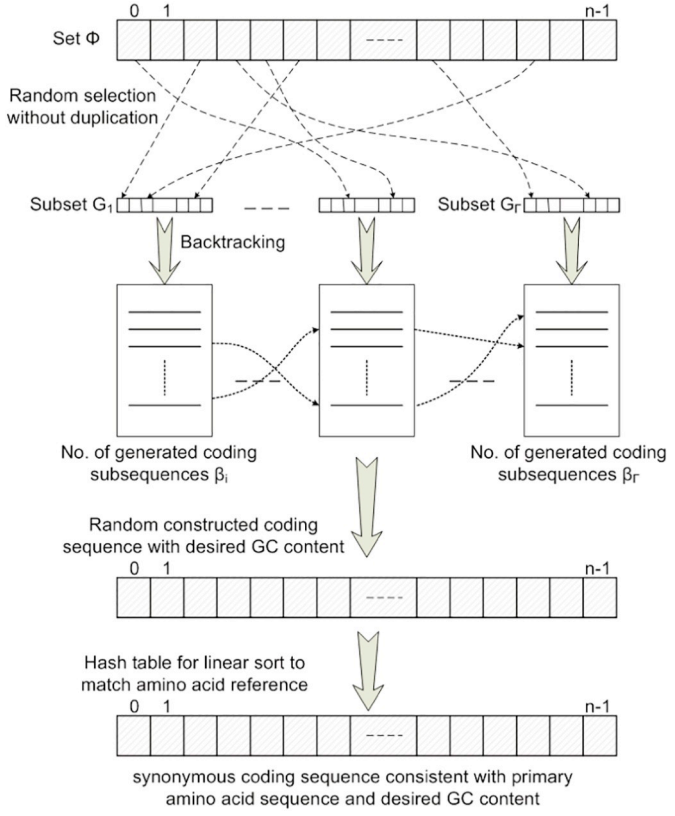

Fig. 1. The framework of our proposed method (CodSeqGen).

First, the union of all constructed subsets equals Φ, which is the set of indices of reference amino acid sequence P (Eq. (1)). Second, the sum of all subsets' lengths equals the same as the length of P (Eq. (2)). And finally, the subsets are disjoint (Eq. (3)),

$$\bigcup_{i=1}^{\Gamma} G_i = G_1 \cup G_2 \cup \cdots \cup G_\Gamma = \Phi, \tag{1}$$

$$\sum_{i=1}^{\Gamma} |G_i| = |G_1| + |G_2| + \cdots + |G_\Gamma| = n, \tag{2}$$

$$G_i \cap G_j = \emptyset, \quad \forall\, i \neq j. \tag{3}$$

Algorithm 1: The main algorithm.

Input: (1) γ: maximum number of synonymous coding subsequences; (2) θ: number of required coding sequences; (3) P: the amino acid sequence; and (4) [D: reference coding sequence, or φ: the desired GC-content]

Output: Synonymous coding sequences (θ)

begin
  $n \leftarrow |P|$ `// length of P`
  if *(input is coding sequence D)* **then**
    $\varphi \leftarrow$ GC-content(D)
  else
    `// input is desired GC-content` φ
    $\bar{D} \leftarrow$ coding sequence with first codon of each amino acid in P
    $\bar{\varphi} \leftarrow$ GC-content($\bar{D}$)
    if $\bar{\varphi} > \varphi$ **then**
      Adjust $\bar{D}$ by choosing the minimum GC-content codon from left of P towards right.
    else
      Adjust $\bar{D}$ by choosing the maximum GC-content codon from left of P towards right.
    end
    $D \leftarrow$ Adjusted $\bar{D}$
    $\varphi \leftarrow$ GC-content(D)
  end
  $s \leftarrow 10$ `// size of subset`
  $\Lambda \leftarrow \lfloor n/s \rfloor$; $\Gamma \leftarrow \lceil n/s \rceil$
  $\Phi \leftarrow \{0, 1, \ldots, n-1\}$
  $\bar{\Phi} \leftarrow$ permute(Φ)
  for $i \leftarrow 1, 2, \ldots, \Lambda$ **do**
    $\beta_i \leftarrow 0$ `// actual # subsequence synonyms`
    $G_i \leftarrow \bar{\Phi}[start_i \ldots end_i]$
    $C_i \leftarrow$ GC-content(G_i)
    $\widehat{C}_i \leftarrow 0$ `// GC-content so far for` G_i
    $(\bar{G}_i, \beta_i) \leftarrow$ Algorithm 2(P, *start*, *end*, β_i, γ, $\widehat{C}_i$, C_i)
    Add $\bar{G}_i$ of β_i coding subsequences to M right to $\bar{G}_{i-1}$
  end
  if ($\Gamma > \Lambda$) **then**
    $\beta_\Gamma \leftarrow 0$; $\widehat{C}_\Gamma \leftarrow 0$
    $G_\Gamma \leftarrow \bar{\Phi}[start_\Gamma \ldots n-1]$
    $C_\Gamma \leftarrow$ GC-content(G_Γ)
    $(\bar{G}_\Gamma, \beta_\Gamma) \leftarrow$ Algorithm 2(P, $start_\Gamma$, end_Γ, β_Γ, γ, $\widehat{C}_\Gamma$, C_Γ)
    Add $\bar{G}_\Gamma$ to M right to $\bar{G}_{\Gamma-1}$
  end
  for $k \leftarrow 1, 2, \ldots, \theta$ **do**
    Randomly select subsequence from each subset in M
    Hash selected indices on hash table of size n
    `// each value in the hash table stores an index for corresponding codon`
    Print complete synonymous coding sequence in order of amino acids in P
  end
end

Algorithm 2: Generating synonymous coding subsequences.

Input: P, $start_i$, end_i, β, γ, $\widehat{C}_i$, C_i
Output: $\bar{G}_i$, β

```
1  begin
2      if (Ĉ_i > C_i) ∨ (start_i > end_i) then
3          Discard current path and backtrack.
4      else if (Ĉ_i = C_i) ∧ (start_i = end_i) ∧ (β < γ) then
5          Add subsequence (current path) to Ḡ_i.
6          β ← β + 1
7      else
8          q ← # of codons of amino acid P[start_i]
9          for k ← 1, 2, ..., q do
10             ϵ ← GC-content of codon k
11             Algorithm 2(P, start_i + 1, end_i, β, γ, Ĉ_i + ϵ, C_i)
12         end
13     end
14 end
```

Setting $s = 10$, means that an array of size 59,049 on average is required to hold the coding subsequences. The worst case is that when all the indices in the subset are pointing to Leucine, Serine, or Arginine. In this case, the search space will be large, 60,466,176 coding subsequences. But, with backtracking and GC-content constraint, this large search space will be pruned to about 100,000, which is feasible for memory limitations. With more complicated constraints such as nucleotide composition, this search space will be further pruned to about 10,000. Our algorithm reads amino acid sequence symbols one by one from left to right and is checked each time the amino acid is replaced with the corresponding codon synonyms and the GC-content constraint. If the total GC-content so far for subset G_i is greater than C_i and the end of sequence is not reached, the current path is discarded and the algorithm backtracks to find a more promising path. Fig. 2 illustrates the backtracking with a tree of height $s + 1$. The nodes at level 2 represent the codons of the first amino acid in the subset and the variable $\widehat{C}_i$ refers to the GC-content so far, where each node has its own $\widehat{C}_i$ that accumulates the GC-content of the path from the second level to the node. Algorithms 1 and 2 further elaborate our method. Note that the total

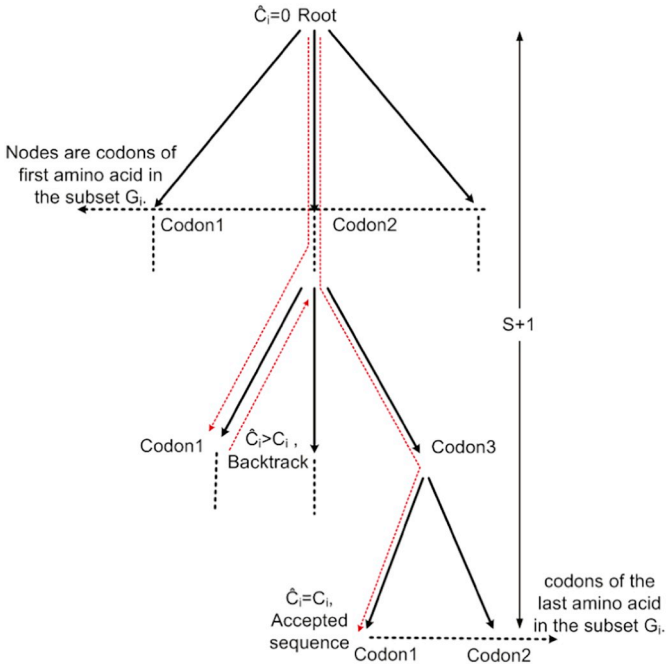

Fig. 2. Backtracking the coding subsequences for subset G_i.

GC-content of all subsets equal φ.

For lines 3–13 in Algorithm1, the GC-content is provided either by the input coding sequence that corresponds to the primary amino acid sequence or GC-content value in the interval (0,100). In the second case, initial coding sequence is created and adjusted to contain the desired GC-content.

Unlike stochastic methods, CodSeqGen is easy to modify for accommodating more complicated constraints such as nucleotide composition and di-nucleotide profile by simply altering the GC-content constraint to a new constraint.

3. Results and discussion

We tested CodSeqGen on a set of proteins in various species: Human, Saccharomyces cerevisiae S288c, Bovine popular stomatitis virus, Zika virus, SARS coronavirus, and Hantavirus. Table 1, lists these proteins with their NCBI accession numbers and their size, in term of the number of amino acids. For each protein, the GC-content is computed for the reference coding sequence. For instance, the reference coding sequence of human Titin protein has a GC-content of 44.04%, whereas Zika virus's reference coding sequence has 49.60%, and so on. In addition, we measured the possible range of the GC-content for each amino acid sequence. This range indicates the minimum and maximum amount of GC-content allowed to create coding sequences. As an example, the primary amino acid sequence of human Titin protein has the possible range 30.42–66.67% of GC-content. Therefore, it is infeasible to create coding sequences of GC-content less than 30.42% or over 66.67%.

We ran both tools, CodSeqGen and NullSeq [10], to generate 1000 coding sequences given the primary amino acid sequence and the target GC-content of the reference coding sequence. Results in Table 2 show that CodSeqGen is more accurate in generating 1000 sequences with exactly the desired GC-content of the reference coding sequence, while NullSeq produced coding sequences that do not exactly match the targeted GC-content. For example, the coding sequences that are generated for Titin protein to match the GC-content of its reference coding sequence (DNA nucleotides) do not match the target GC-content of 44.04% but rather vary in range 43.54–44.68%. Scatter plots in Fig. 3, and Fig. 4 for human Titin protein and Zika virus protein, clearly, show GC-contents of the 1000 generated random sequences by both tools. These figures evidently display that sequences generated by CodSeqGen tool fit with the GC-content constraint. All sequences have the precise GC-contents of 44.04% and 49.60%, as shown by the red vertical lines in Fig. 3 and Fig. 4, respectively.

Since CodSeqGen tool generates random synonymous coding sequences having the exact GC-content, it is important to test how GC-content is distributed over complete sequences. This was tested by dividing the complete sequence into subsets and then computing the GC-contents for all subsets. Thus, standard deviation (STD) and average (Avg) are computed for each generated sequence. Finally, a range is computed over 1000 random coding sequences. The same experiment was repeated for 1000 random coding sequences generated by NullSeq. Table 3 lists the ranges of STD and Avg for both tools over 1000 generated coding sequences in addition to the STD and Avg for the reference coding sequences. Since it is a single reference coding sequence for each protein, no ranges are shown. Smaller STD means that all subsets have similar GC-content and larger STD means that GC-content is non-uniformly distributed over subsets. Thus, larger STD and Avg ranges means that a diversity of random sequences are produced. Both tools generate sequences with a good STD and Avg ranges. However, random sequences by NullSeq originally have GC-content smaller or greater than targeted GC-contents which cause the ranges of STD and Avg for subsets GC-content to be a bit larger.

Graphically, Figs. 5-6 depict GC-content distribution for subsets over five randomly selected sequences on Titin protein by CodSeqGen (Fig. 5) and NullSeq (Fig. 6). As another example, Figs. 7-8 show

Table 1
List of proteins in our experiment. The Size refers to that of protein in term of the number of amino acids (aa).

Protein	Accession	Size	GC-content range	GC-content of ref coding seq (%)
Titin [*Homo sapiens*]	CAA62188.1	26,926	30.42–66.67%	44.04
Mitotic regulator LTE1 [*Saccharomyces cerevisiae* S288c]	NP_009378.1	1435	24.26–61.37%	37.07
RPO147 [Bovine papular stomatitis virus]	NP_957965.1	1289	27.16–64.39%	63.20
E protein, partial [Zika virus]	AIC06934.1	504	32.08–67.72%	49.60
S protein [SARS coronavirus]	ABF65836.1	1255	27.42–63.58%	38.83
Glycoprotein 1 [Hantavirus]	AAB20470.2	1134	28.16–64.10%	39.83

Table 2
The GC-content achieved by NullSeq [10] vs CodSeqGen (our approach) for 1000 generated synonymous coding sequences based on primary reference coding sequences.

Protein	Targeted GC-content	Achieved GC-content	
		NullSeq	CodSeqGen (%)
Titin *Homo sapiens*	44.04%	43.54–44.68%	44.04
Saccharomyces cerevisiae	37.07%	35.01–38.86%	37.07
Bovine papular stomatitis virus	63.20%	61.16–65.30%	63.20
Zika virus	49.60%	46.10–53.97%	49.60
SARS coronavirus	38.83%	36.33–41.20%	38.83
Hantavirus	39.83%	37.36–42.59%	39.83

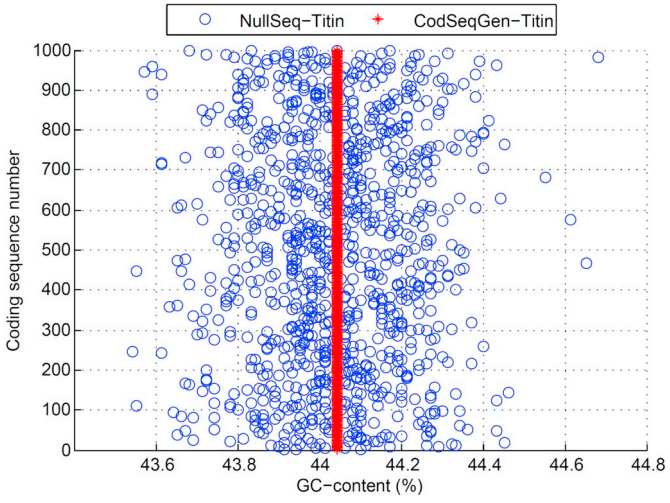

Fig. 3. The GC-contents of 1000 random coding sequences generated by NullSeq [10], and CodSeqGen on Titin Homo sapiens protein. All the random sequences generated by CodSeqGen fall exactly with the same GC-content of 44.04% (red vertical line). The distorted random sequences generated by NullSeq have GC-contents oscillate between 43.54% and 44.68% (scattered circles in blue). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

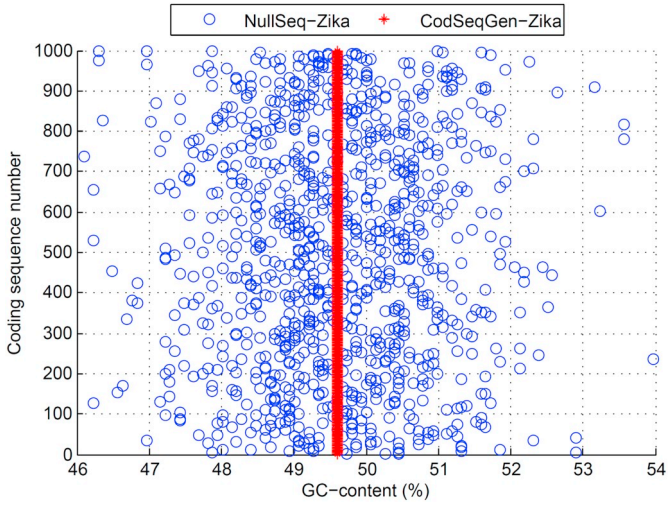

Fig. 4. The GC-contents of 1000 random coding sequences generated by NullSeq and CodSeqGen on Zika virus protein. Again, all the random sequences generated by CodSeqGen fall with the same GC-content of 49.60% (vertical line), while NullSeq generated random sequences with GC-contents that oscillate between 46.10% and 53.97% (scattered circles).

subsets' GC-content distribution on Zika virus by CodSeqGen and NullSeq, respectively. The Figures show that the produced sequences have different GC-content distributions over the whole sequences as the sizes of the four quartiles in box plots differ among the five sequences.

As has been shown from results, CodSeqGen is a powerful tool to achieve the exact target GC-content with all generated synonymous coding sequences where GC-content is distributed more randomly over whole generated sequences. Moreover, our tool can be adjusted easily when we have more complicated constraints and it can be useful in generating random RNA structures with pre-specified constraints.

4. Conclusion

In this paper, we presented a tool called CodSeqGen. The proposed tool produces synonymous coding sequences following pre-specified amino acid sequence and desired GC-content. Our approach uses the backtracking technique to produce exact coding subsequences and these subsequences are aggregated to produce the desired synonymous coding sequences. The proposed tool will help researchers for identifying accurate protein-DNA binding sites and producing sequences with more complicated constraints.

Table 3
Standard deviation (STD) and average (Avg) of GC-contents over subsets for 1000 randomly generated coding sequences using NullSeq and CodSeqGen. The Table lists ranges, minimum and maximum, of STD and Avg. The subset size is measured in term of the number of aa. The Ref. Seq. denotes the reference coding sequence.

Protein	Subset size	Ref. Seq.		NullSeq		CodSeqGen	
		STD	Avg	STD	Avg	STD	Avg
Titin *Homo sapiens*	1000	1.68	44.09	0.57–1.36	43.54–44.69	1.16–1.75	44.03–44.04
Saccharomyces cerevisiae	100	2.11	36.91	0.92–4.68	34.69–39.01	1.22–4.05	36.54–37.28
Bovine papular stomatitis virus	100	2.52	63.20	1.05–4.37	61.17–65.33	2.04–3.13	63.19–63.21
Zika virus	50	6.01	51.03	2.03–13.96	44.21–54.67	2.01–11.08	49.64–52.43
SARS coronavirus	100	3.55	39.04	1.26–4.64	36.48–41.18	2.06–4.40	38.66–39.10
Hantavirus	100	1.54	39.63	1.36–5.75	36.98–42.73	1.59–5.37	38.99–39.96

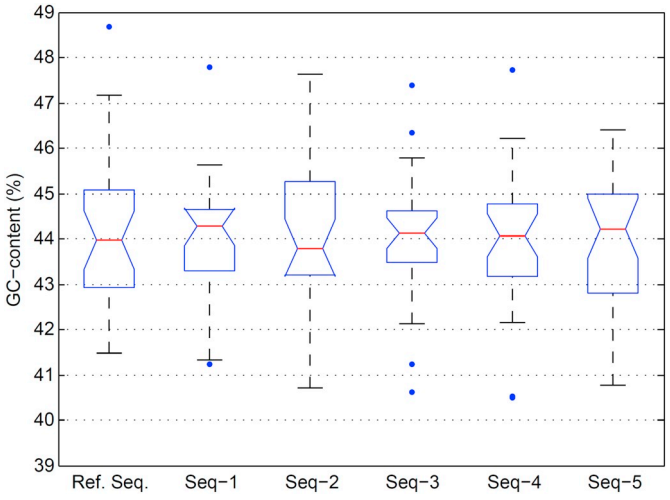

Fig. 5. Box plot indicates the GC-content distribution for 5 Titin synonymous coding sequences (Seq-1, …, Seq-5) generated by CodSeqGen over subsets of size 1000 amino acids (aa) versus GC-content of the reference coding sequence (the leftmost box). In each box, the central line marks the median, while the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points that are not considered outliers. The latter are marked using solid dots.

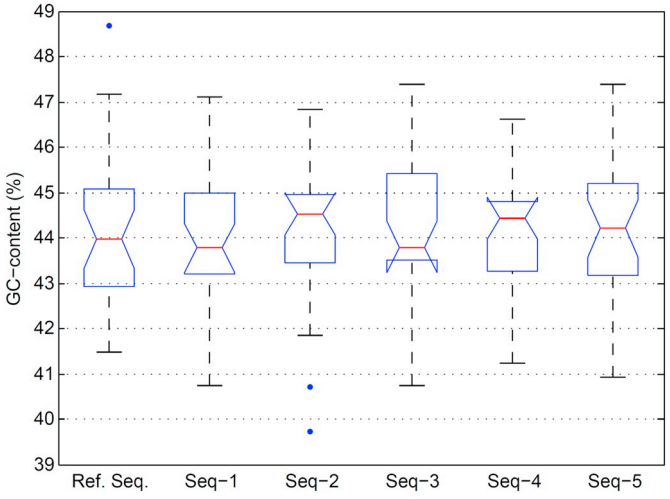

Fig. 6. The GC-content distribution for 5 Titin synonymous coding sequences (Seq-1, …, Seq-5) generated by NullSeq [10] over subsets of size 1000 aa versus GC-content of the reference coding sequence.

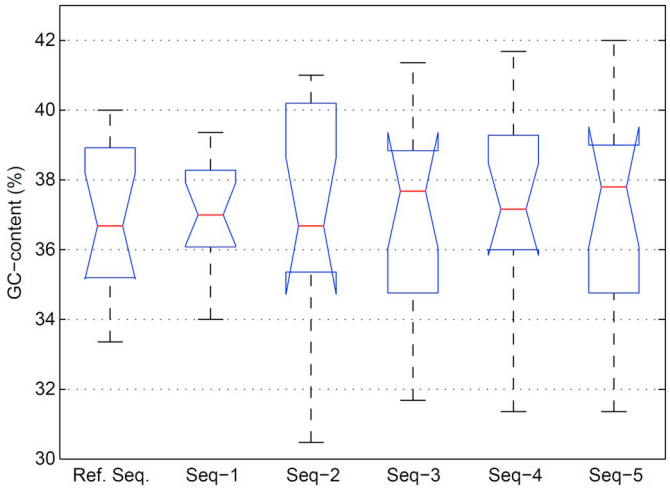

Fig. 7. The GC-content distribution for 5 Saccharomyces cerevisiae synonymous coding sequences (Seq-1, …, Seq-5) generated by CodSeqGen over subsets of size 100 aa versus GC-content of the reference coding sequence.

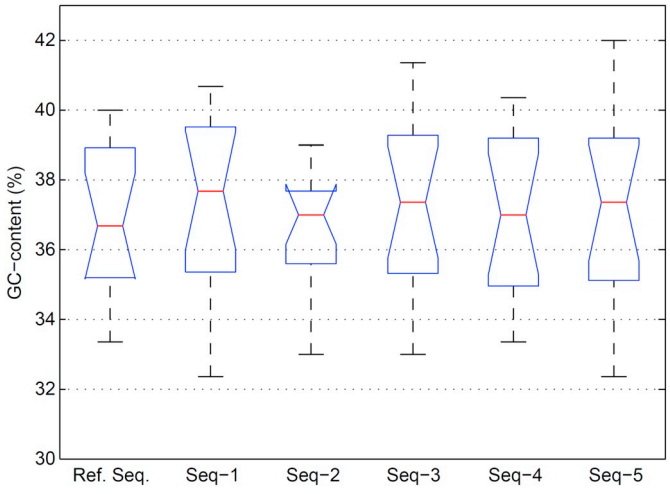

Fig. 8. The GC-content distribution for 5 Saccharomyces cerevisiae synonymous coding sequences (Seq-1, …, Seq-5) generated by NullSeq over subsets of size 100 aa versus GC-content of the reference coding sequence.

Availability

CodSeqGen executable is available for free download at: https://github.com/Abdulrakeeb/CodSeqGen

Acknowledgments

The authors extend their appreciation to the Deanship of Scientific Research at King Saud University for funding this work through research group no. RG-1439-067.

References

[1] F. Lassalle, S. Périan, T. Bataillon, X. Nesme, L. Duret, V. Daubin, GC-content evolution in bacterial genomes: the biased gene conversion hypothesis expands, PLoS Genet. 11 (2) (2015) 1–20.
[2] D. Kunec, N. Osterrieder, Codon pair bias is a direct consequence of dinucleotide bias, Cell Rep. 14 (1) (2016) 55–67.
[3] G. Boël, R. Letso, H. Neely, W.N. Price, K.-H. Wong, M. Su, J.D. Luff, M. Valecha, J.K. Everett, T.B. Acton, R. Xiao, G.T. Montelione, D.P. Aalberts, J.F. Hunt, Codon influence on protein expression in *E. coli* correlates with mRNA levels, Nature 529 (7586) (2016) 358–363.
[4] J. Fu, K.A. Murphy, M. Zhou, Y.H. Li, V.H. Lam, C.A. Tabuloc, J.C. Chiu, Y. Liu, Codon usage affects the structure and function of the drosophila circadian clock protein PERIOD, Genes Dev. 30 (15) (2016) 1761–1775.
[5] P. Stothard, The sequence manipulation suite: JavaScript programs for analyzing and formatting protein and DNA sequences, BioTechniques 28 (6) (2000) 1102–1104.
[6] P. Villesen, FaBox: an online toolbox for FASTA sequences, Mol. Ecol. Notes 7 (6) (2007) 965–968.
[7] Y. Ponty, M. Termier, A. Denise, GenRGenS: software for generating random genomic sequences and structures, Bioinformatics 22 (12) (2006) 1534–1535.
[8] F. Wright, The 'effective number of codons' used in a gene, Gene 87 (1) (1990) 23–29.
[9] J.A. Novembre, Accounting for background nucleotide composition when measuring Codon usage bias, Mol. Biol. Evol. 19 (8) (2002) 1390–1394.
[10] S. Liu, A. Hockenberry, A. Lancichinetti, M. Jewett, L. Amaral, NullSeq: a tool for generating random coding sequences with desired amino acid and GC contents, PLoS Comput. Biol. 12 (11) (2016) e1005184.
[11] A. Levitin, Introduction to the Design and Analysis of Algorithms, 3e, Addison-Wesley, 2012.