

Developing JSequitur to Study the Hierarchical Structure of Biological Sequences in a Grammatical Inference Framework of String Compression Algorithms

Bulgan Galbadrakh, Kyung-Eun Lee, Hyun-Seok Park*

Department of Computer Science, Ewha Womans University, Seoul 120-750, Korea

Grammatical inference methods are expected to find grammatical structures hidden in biological sequences. One hopes that studies of grammar serve as an appropriate tool for theory formation. Thus, we have developed JSequitur for automatically generating the grammatical structure of biological sequences in an inference framework of string compression algorithms. Our original motivation was to find any grammatical traits of several cancer genes that can be detected by string compression algorithms. Through this research, we could not find any meaningful unique traits of the cancer genes yet, but we could observe some interesting traits in regards to the relationship among gene length, similarity of sequences, the patterns of the generated grammar, and compression rate.

Keywords: context-free grammar, formal language theory, natural language processing, stochastic modeling

Availability: JSequitur is freely available for academic purposes. Please contact neo@ewha.ac.kr.

Introduction

In formal language theory a language is simply a set of strings of characters drawn from some alphabet, where the alphabet (terminal) is a set of symbols. When we consider biological sequences simply as a language in the context of formal language theory (treating DNA, RNA, or protein sequences just as strings of alphabets of four nucleotides or 20 amino acids), a grammatical inference method based on formal language theory can be applied [1-3].

Nevill-Manning and Witten [4] pioneered the attempt to produce the context-free grammar of biological sequences in an automatic way. This task can be formalized as the problem of finding the smallest context-free grammar by recursively replacing the repeats by a new symbol. The algorithm builds a hierarchy of phrases by forming a new rule out of existing pairs of symbols, including non-terminal symbols.

For example, if we consider the string "atattattatt," the simplest way to represent the string by context-free gram-

mar is the following:

<Grammar 0>

$S \rightarrow atattattatt$

The most frequently occurring sequence in the string is "at," which occurs four times. Thus, introducing a new nonterminal symbol, 'A,' and creating a new rule for this yields the following modified grammar:

<Grammar 1>

$S \rightarrow AAtAtAt$

$A \rightarrow at,$

where the grammar consists of a start symbol (i.e., S), two terminal symbols (i.e., a, t) represented by lowercase letters, two non-terminal symbols (i.e., S, A) represented by uppercase letters, and two production rules (i.e., $S \rightarrow AAtAtAt$, $A \rightarrow at$) with a left- and a right-hand side consisting of a sequence of these symbols.

Repeatedly replacing the frequently occurring patterns "At," again to a new nonterminal symbol, B, gives the following modified grammar:

Received November 1, 2012; Revised November 14, 2012; Accepted November 16, 2012

*Corresponding author: Tel: +82-2-3277-2831, Fax: +82-2-3277-2306, E-mail: neo@ewha.ac.kr

Copyright © 2012 by the Korea Genome Organization

© It is identical to the Creative Commons Attribution Non-Commercial License (<http://creativecommons.org/licenses/by-nc/3.0/>).

<Grammar 2>

S → AB³B
 A → at
 B → At,

where the grammar consists of a start symbol (i.e., S), two terminal symbols (i.e., a, t), three non-terminal symbols (i.e., S, A, B), and three production rules (i.e., S → AB³B, A → at, B → At).

By applying the three production rules by replacing an occurrence of the nonterminals on the left-hand side of the production rule with those that appear on the right-hand side, the string “atattattatt” can be derived from the non-terminal S by constantly applying a series of derivations: S → AB³B → atB³B → atAtB²B → atattB²B → atattAtB →

atattattB → atattattAt → atattattatt.

Based on this concept, grammar-based compression algorithms have shown some success for various applications [4-7]. Especially, grammar can capture distant repetitions occurring far apart, which was a limitation of sliding window approaches. However, grammar-based compression algorithms at this moment do not show the best performance for compression itself [6]. Thus, our motivation of this study is not to develop a new algorithm or find the most efficient way to compress biological sequences for storage purposes. Our sole purpose of developing a new tool is to investigate any grammatical traits of biological sequences, based on formal language theory.

Implementation

We have developed a slightly modified version of Sequitur [4] called JSequitur for automatically creating hierarchical structures of sequences [8], as in Fig. 1. Our main contribution is to improve Sequitur to work better in a graphic user interface (GUI) environment, as our main interest was in studying the generated grammar, rather than enhancing the compression rate itself. JSequitur is implemented in Java and organized into six classes, as in Fig. 2: Sequitur, Symbol, Guard, Terminal, Nonterminal, and Rule. Sequitur class is called first and connects with all of the other classes. Symbol class is the connector class, which streams sequences of input to the system. Rule class accesses Terminal and NonTerminal classes in order to create rules. Guard class, which is based on digram uniqueness, is responsible for rule confirmation.

Thus, our string compression algorithm operates by



Fig. 1. User interface of JSequitur program.

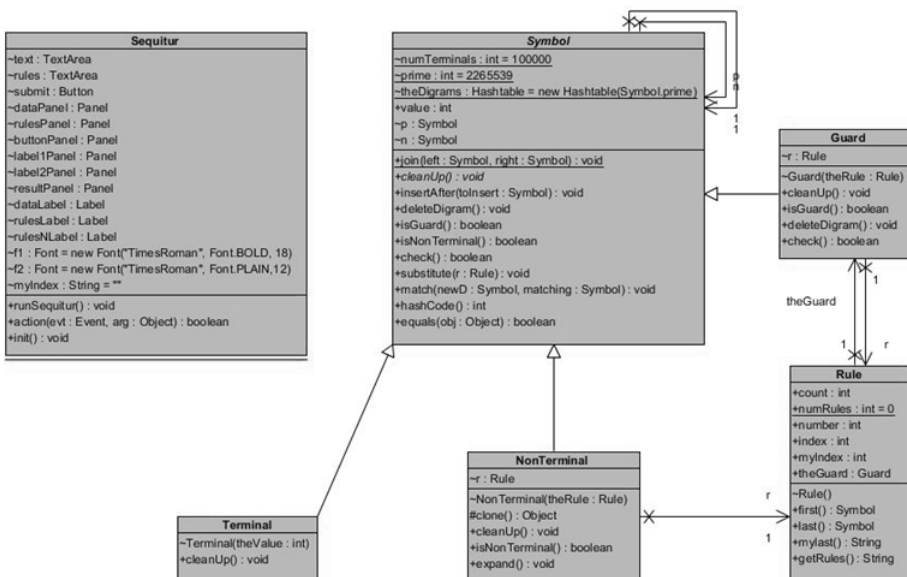


Fig. 2. JSequitur class diagram.

Table 1. One hundred four genes and their compression rates

No.	Gene	Length	Compressed length	No. of rules	Compression ratio
1	<i>TERC</i>	587	157	45	0.2675
2	<i>MIF</i>	1,099	265	79	0.2411
3	<i>HSPB1</i>	2,262	528	131	0.2334
4	<i>TNFRSF6B</i>	2,662	571	153	0.2145
5	<i>S100A4</i>	2,844	630	163	0.2215
6	<i>CDKN2D</i>	3,274	713	173	0.2178
7	<i>GSTP1</i>	3,977	847	200	0.2130
8	<i>HRAS</i>	4,301	869	206	0.2020
9	<i>EMS1</i>	4,741	978	235	0.2063
10	<i>TCL1A</i>	5,498	1,142	264	0.2077
11	<i>TFF1</i>	5,530	1,110	274	0.2007
12	<i>TCTA</i>	5,553	1,104	279	0.1988
13	<i>MUC1</i>	5,729	1,034	268	0.1805
14	<i>SNCG</i>	6,148	1,221	271	0.1986
15	<i>IL6</i>	6,312	1,283	287	0.2033
16	<i>CDKN1B</i>	6,504	1,293	301	0.1988
17	<i>MYC</i>	6,976	1,409	300	0.2020
18	<i>KLK3</i>	7,604	1,487	329	0.1956
19	<i>GSTM1</i>	7,734	1,525	325	0.1972
20	<i>CYP1A1</i>	7,793	1,527	352	0.1959
22	<i>KISS1</i>	7,997	1,526	342	0.1908
23	<i>ARHC</i>	8,159	1,587	349	0.1945
24	<i>PLAU</i>	8,318	1,624	346	0.1952
25	<i>MYCN</i>	8,381	1,655	350	0.1975
26	<i>MYCL1</i>	8,570	1,688	359	0.1970
27	<i>HSPCB</i>	8,796	1,685	380	0.1916
28	<i>CYP2A6</i>	8,982	1,701	395	0.1894
29	<i>BAX</i>	9,021	1,667	383	0.1848
30	<i>CYP17</i>	9,103	1,735	393	0.1906
31	<i>GSTT1</i>	10,590	1,941	439	0.1833
32	<i>ING1</i>	10,841	2,069	438	0.1909
33	<i>CYP1B1</i>	11,152	2,128	443	0.1908
34	<i>NAT2</i>	12,959	2,406	484	0.1857
35	<i>TFAP2C</i>	12,976	2,458	510	0.1894
36	<i>FGF8</i>	13,312	2,437	507	0.1831
37	<i>CDKN1A</i>	14,144	2,645	524	0.1870
38	<i>RASSF1</i>	14,497	2,621	533	0.1808
39	<i>CTSD</i>	14,613	2,558	532	0.1751
40	<i>BIRC5</i>	14,872	2,536	571	0.1705
41	<i>MMP11</i>	14,908	2,695	555	0.1808
42	<i>PCNA</i>	15,170	2,726	572	0.1797
43	<i>CYP2E</i>	15,280	2,750	582	0.1800
44	<i>RCA1</i>	15,646	2,660	611	0.1700
45	<i>BAG1</i>	15,979	2,967	572	0.1857
46	<i>CCNE1</i>	16,009	2,902	589	0.1813
47	<i>CCND1</i>	17,380	3,132	597	0.1802
48	<i>BCL1</i>	17,380	3,132	597	0.1802
49	<i>TAL1</i>	17,525	3,154	629	0.1800
50	<i>NAT1</i>	18,081	3,236	634	0.1790
51	<i>CEACAM8</i>	19,094	3,348	662	0.1753
52	<i>LIBC</i>	20,296	3,603	711	0.1775
53	<i>VEGF</i>	21,163	3,702	720	0.1749
54	<i>MPL</i>	21,659	3,807	723	0.1758
55	<i>SLC2A3</i>	22,189	3,853	748	0.1736

Table 1. Continued

No.	Gene	Length	Compressed length	No. of rules	Compression ratio
56	<i>STIP1</i>	23,964	4,022	825	0.1678
57	<i>TP53</i>	24,886	3,972	860	0.1596
58	<i>IGF2</i>	26,633	4,512	881	0.1694
59	<i>CSK</i>	27,449	4,662	875	0.1698
60	<i>STK11</i>	29,427	4,792	932	0.1628
61	<i>TFAP2A</i>	29,746	5,180	923	0.1741
62	<i>ERBB3</i>	30,527	5,021	961	0.1645
63	<i>MSH6</i>	31,034	5,038	973	0.1623
64	<i>MLLT6</i>	31,438	5,346	974	0.1700
65	<i>BCL6</i>	31,653	5,426	971	0.1714
66	<i>SLC22A1L</i>	33,184	5,440	1,010	0.1639
67	<i>PSEN2</i>	33,192	5,660	978	0.1705
68	<i>CDKN2A</i>	34,762	5,940	987	0.1709
69	<i>TPMT</i>	34,883	5,702	1,053	0.1635
70	<i>POU2AF1</i>	35,332	5,809	1,087	0.1644
71	<i>MMP2</i>	35,758	6,058	1,074	0.1694
72	<i>PI5</i>	36,627	6,098	1,079	0.1665
73	<i>COMT</i>	36,706	5,860	1,131	0.1596
74	<i>TFAP2B</i>	37,554	6,392	1,123	0.1702
75	<i>NOTCH4</i>	37,993	6,191	1,123	0.1630
76	<i>TOP2A</i>	38,258	6,027	1,159	0.1575
77	<i>MKI67</i>	38,410	6,258	1,152	0.1629
78	<i>SLC2A1</i>	43,942	7,254	1,178	0.1651
79	<i>MDM2</i>	48,414	7,466	1,409	0.1542
80	<i>CD9</i>	49,342	7,937	1,400	0.1609
82	<i>THBS2</i>	49,741	7,801	1,377	0.1568
83	<i>BCAR1</i>	50,730	8,015	1,397	0.1580
84	<i>PPP2R1B</i>	51,400	8,203	1,452	0.1596
85	<i>SH3GL1</i>	52,262	8,222	1,465	0.1573
86	<i>ERBB2</i>	52,679	8,151	1,504	0.1547
87	<i>TERT</i>	54,445	7,781	1,531	0.1429
88	<i>PDGFRB</i>	54,627	8,684	1,533	0.1590
89	<i>AXL</i>	55,332	8,369	1,608	0.1513
90	<i>GAS6</i>	56,583	8,476	1,551	0.1498
91	<i>EFNB2</i>	58,902	9,472	1,538	0.1608
92	<i>KRAS2</i>	59,377	9,280	1,623	0.1563
93	<i>TSG101</i>	60,640	9,516	1,640	0.1569
94	<i>EIF3S6</i>	61,084	9,600	1,574	0.1572
95	<i>WT1</i>	62,089	9,980	1,718	0.1607
96	<i>RARA</i>	63,015	9,783	1,681	0.1552
97	<i>TNFRSF10B</i>	63,771	9,896	1,720	0.1552
98	<i>NOTCH1</i>	66,745	8,476	1,551	0.1270
99	<i>LASP1</i>	67,486	10,220	1,816	0.1514
100	<i>EIF4E</i>	67,834	10,262	1,814	0.1513
101	<i>ARHA</i>	68,833	9,651	1,917	0.1402
102	<i>PML</i>	69,091	10,636	1,826	0.1539
103	<i>CHEK2</i>	70,320	10,317	1,921	0.1467
104	<i>COT</i>	70,410	10,930	1,797	0.1552

reading in a new symbol and processing it by appending it to the top-level string and then examining the last symbols of that string; it then applies zero or more of the following transformations until none applies anywhere in the grammar;

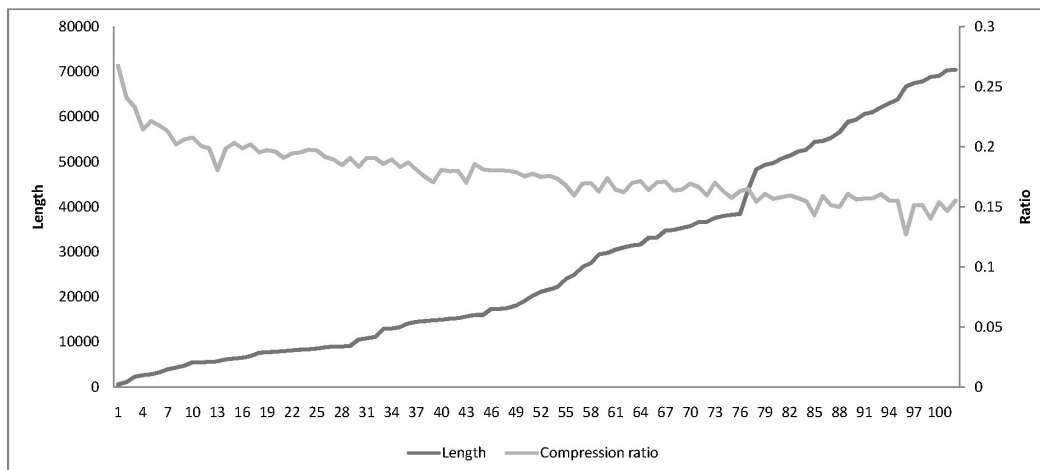


Fig. 3. Compression rates in relation to gene length.

it then repeats the cycle by reading in a new symbol.

The following production rules, which have been created automatically, are an exemplary output of applying JSequitur to the partial sequence of the *TERT* gene (175 bp, “gccccgggtgtccctgtcacgtgcagggtgagtgaggcgcggtccccgggtgtccctgtcacgtgcagggtgagtgaggcgcggtccccgggtgtccctgtcacgtgcagggtgagtgaggcgcggtcccc”):

R0 → g R1 R2 R3 R4 R5 R6 R7 R6 R8 R5 R9 R1 R3 R4 R10
R11 g a R4 a R12 R13 R14 R7 R15 R8 R16 R10 R14 c

R1 → c c

R2 → R1 c

R3 → R11 R17 R2 R18

R4 → R17 g

R5 → R16 R13 R19

R6 → t R2

R7 → R19 R4

R8 → R18 R4

R9 → t R1

R10 → a a

R11 → R12 R17

R12 → g g

R13 → c g

R14 → R19 R15

R15 → R9 c

R16 → R10 R11 g a R4 a R12

R17 → g t

R18 → t R17 R10 c

R19 → R13 g,

where the grammar consists of a start symbol (i.e., R0), four terminal symbols (i.e., a, t, g, c), 20 non-terminal symbols (i.e., R0-R19), and 20 production rules for each nonterminal. In summary, the partial sequence of 175 bp of the *TERT* gene could be compressed to 37 symbols with 20 rules.

For testing purposes, 104 cancerous genes from 6 cancer

types (bladder, breast, endometrial, leukemia, lung, and melanoma) were initially chosen, and JSequitur was applied. Table 1 shows the result of applying one of the string compression algorithms of JSequitur to these genes.

The rule column in Table 1 shows the number of generated rules from the context-free grammar, while the ratio column shows the ratios of the compressed sequences vs. the original sequences.

Fig. 3 is a sorted diagram in the order of the length of the original sequence. In this specific case, it shows that the length of the original sequence influences the compression rate of the target sequence, even though there are many other factors that influence compression rate. For example, compression rate can also be influenced by the algorithm itself, depending on whether we replace the longest pattern first or the most frequently occurring pattern first.

We also compared some mouse genes to find any homologous traits in regards to compression rate and hierarchical structure of the grammar. For example, we compared human *MUC1* (*Homo sapiens*, 5,279 bp) with mouse *MUC1* (*Mus musculus*, 5,614 bp), and the compression rates for these two sequences were 0.180 and 0.195, respectively. For the *ARHA* genes, the compression rate for human *ARHA* (68,833 bp) was 0.140, whereas that for mouse *ARHA* (41,255 bp) was 0.157. Thus, the distance on the evolutionary tree can be measured by compression algorithms, to a certain extent.

Conclusion and Future Direction

We have developed a GUI-based JSequitur, based on string compression algorithms, to examine grammatical traits of biological sequences. On top of compression capacity, a string compression algorithm is appealing for studying biological sequences, because it can give insights into the

structure of these sequences. Precisely constructed models for linguistic structure can play a vital role in the process of discovery itself.

We also applied JSequitur to analyze 104 cancer genes for testing purposes only. Even though there are some interesting results in regards to the relationship among gene length, similarity of sequences, the patterns of the generated grammar, and compression rate, our test samples were too small to conclude anything. Thus, our result should be regarded as preliminary for future research. We should consider various factors other than grammatical structures and compression rates.

As our main purpose of developing the tool was to examine any grammatical traits of biological sequences, the graphical user interface was important for a semiautomatic screening process. However, we still need to implement various features to compare gene structures to summarize statistics in regards to grammatical structures and to combine evolutionary trees. Hopefully, these features will be implemented in the next version of JSequitur. We also hope to enhance the algorithm more elaborately to handle reversal, translocation, and shuffling.

References

1. Sakakibara Y. Grammatical inference in bioinformatics. *IEEE Trans Pattern Anal Mach Intell* 2005;27:1051-1062.
2. Coste F. Modelling biological sequences by grammatical inference. In: ICGI 2010 Tutorial Day. Valencia: International Conference on Grammatical Inference, 2010. Accessed 2012 Nov 1. Available from: http://www.irisa.fr/symbiose/people/fcoste/pub/biblio_tutoICGI2010_coste.pdf.
3. Park HS, Galbadrakh B, Kim YM. Recent progresses in the linguistic modeling of biological sequences based on formal language theory. *Genomics Inform* 2011;9:5-11.
4. Nevill-Manning CG, Witten IH. Compression and explanation using hierarchical grammars. *Comput J* 1997;40:103-116.
5. Lanctot JK, Li M, Yang E. Estimating DNA sequence entropy. In: Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms, 2000 Jan 9-11, San Francisco, CA. Philadelphia: Society for Industrial and Applied Mathematics, 2000. pp. 409-418.
6. Cherniavsky N, Ladner R. Grammar-based compression of DNA sequences. UW CSE Technical Report (TR2007-05-02). In: DIMACS Working Group on the Burrows-Wheeler Transform, 2004 Aug 19-20, Piscataway, NJ.
7. Carrascosa R, Coste F, Gallé M, Infante-Lopez G. Searching for smallest grammars on large sequences and application to DNA. *J Discrete Algorithms* 2012;11:62-72.
8. Galbadrakh B. Identifying hierarchical structure in biological sequences based on context-free grammars. M.S. Thesis. Seoul: Ewha Womans University, 2011.

1. Sakakibara Y. Grammatical inference in bioinformatics. *IEEE*