COMPUTING APPLICATIONS ENRICHED BY SCIENTIFIC PHENOMENA *

FACULTY POSTER ABSTRACT

Vladimir V. Riabov
Department of Mathematics and Computer Science
Rivier University
Nashua, NH 03060-5086
603 897-8613
vriabov@rivier.edu

The merge of mathematics, electrical engineering, and electronics has successfully contributed to the development of computing devices and numerous practical applications. The objective of this paper is to review various computing applications that involve unique phenomena acquired from other sciences (e.g., linguistics, physiology, and psychology), as well as to explore some interdisciplinary applications with the developed computing tools.

Modern methods of deciphering use traditional concepts from the theory of numbers, Galois fields, and probability. For decades the Mono-Alphabetic substitution cipher approach with the letter frequency analysis [1] has been also effectively used. Students explored this approach, which is based on applying the linguistic properties of an original plaintext written in a natural language with known linguistic properties. As long as we know that there is a 1-to-1, unique mapping from plaintext to ciphertext, we can employ our knowledge of letter frequencies to crack a substitution cipher. A typical representation of the letter frequencies in English is "E, T, A, O, I, N, S …" [1]. The students examined the letter frequencies in the ciphertext with the created Java applets. If one of the characters has a 20% then the language may be German since it has a very high percentage of E. Italian has 3 letters with a frequency greater than 10% and 9 characters are less than 1% [1]. The linguistic analysis [1] shows that common pairs in English are consonants TH and vowels EA. Others are OF, TO, IN, IT, IS, BE, AS, AT, SO, WE, HE, BY, OR, ON, DO, IF, ME, MY, UP. Common pairs of repeated letters are SS, EE, TT, FF, LL, MM and OO. Common triplets of text are THE, EST, FOR, AND, HIS, ENT or THA.

In the first "simple" assignment for the *Computer Security* course, the Mono-Alphabetic Cipher Breaker Java applet was used for deciphering the structured ciphertext (620 words; 2,485 letters out of 3,533 characters), where the original word spacing, punctuation, and style were retained. The second ciphertext (25,955 words; 103,818 letters out of 129,772 characters) was organized in groups of four letters and word spacing and punctuation were removed. The absence of the content clues (word

spacing and punctuation) makes it more difficult to decipher the ciphertext, while the larger sample allows greater use of letter frequency analysis. To reduce the time of deciphering the unstructured ciphertext, students wrote the customized UNIX scripts.

In the second case study, the structured testing methodology [2] and graph-based metrics (cyclomatic complexity, essential complexity, and module design complexity) were applied by students in the *Software Engineering* course for studying the C-code complexity and estimating the number of possible errors and tests for the Carrier Networks Support (CNS) system. The data of Miller's psychological experiments [2] was used for the reduction of the programming-code complexity, suggesting that code modules approach zero defects when the cyclomatic comple-xity is within 7 ± 2. The CNS-code study confirmed a correlation between high cyclomatic complexity and error-prone software. Comparing different releases, students found that the reduction of the code complexity led to significant reduction of errors and maintainability efforts.

The third case study was designed to introduce Halstead's metrics [3] and explore the physiological phenomenon of human-brain restrictions in estimating the number of projected errors in codes. The concept of event-discriminations was introduced by J. M. Stroud, who defined a "moment" as "the time required by the human brain to perform the most elementary discrimination" [3, p. 48]. He reported that, for all waking, conscious time, these "moments" occurred at a rate of "from five to twenty or a little less" per second, which is known nowadays as the Stroud number, S [3]. His study was based on the analysis of the internal processing rate of the brain that correlates with the range of the number of frames per second which a motion picture should have to appear as a continuous picture rather than as single frames. The Stroud number, S, was used by Halstead in estimating the programming time, $\check{T} = E/S$, where E is the number of elementary mental discriminations required for the program implementation.

The linguistic complexity of various languages (English, PL/I, Algol, Assembly, etc.) was studied by Halstead [3, pp. 62-70] for estimating the language-level parameter ($\lambda = 2.16$) that was used in calculating the mean number of elementary discriminations between potential errors in programming ($E_0 = 3000$), and, finally, the number of "delivered" bugs, $B = E^{2/3}/E_0$. This concept was examined by students in the *Software Quality Assurance* course.

In the last case study, students in *Computer Science Fundamentals and Computer Graphics* courses explored how pictures resolved in human vision are represented on a computer screen. In nature, visible light is a continuous spectrum with wavelengths between 370 and 730 nanometers. But the human perception of light is limited by how human eye color sensors work [4]. Perception of color begins with specialized retinal "cone" cells containing pigments with different spectral sensitivities. In humans, there are three types of cones that are simplistically referred to as *red* (R, 620-740 nm), *green* (G, 495-570 nm), and *blue* (B, 450-495 nm). Based on the human perception of light, each pixel is encoded as a triplet of numbers that represent amounts of *red*, *green*, and *blue*. This concept is known as the RGB color model [4]. Each color channel in a pixel is typically represented with a single byte (1 byte = 8 bits). Therefore, three channels in a pixel represent $2^{24} = 16{,}777{,}216$ patterns of different colors.

Simple programs written in *Jython* were used for manipulating JPEG pictures by changing the pixel's *red*, *green*, and *blue* components in the picture. This approach was

used for changing color components, clearing any color component from a picture, lighten or darken the picture, creating a negative, converting to grayscale, mirroring the image, creating a collage, rotating a picture, reducing red-eye, blurring the image, etc. These exercises were effectively used in the introductory Computer Science courses that motivated non-CS majors to explore in depth the image-processing techniques and create simple programs and tools.

These theoretical concepts, case studies, interdisciplinary applications, and the developed computing tools and codes were thoroughly examined by students in *Discrete Mathematics, Algorithms, Computer Science Fundamentals, Computer Security, Computer Graphics, Software Engineering,* and other courses taught by the author. In the course evaluations, students stated that they became deeply engaged in course activities through examining the challenging problems related to the advanced concepts from the described theories and practical applications.

REFERENCES

[1]Frequency analysis. Online: http://www.richkni.co.uk/php/crypta/freq.php

[2]McCabe, T. A Complexity Measure. In *IEEE Trans. on Software Eng*., 2(4), 308-320, 1976.

[3]Halstead, M. H. *Elements of Software Science*. New York: Elsevier North Holland, 1977.

[4]Color Vision. Online: https://en.wikipedia.org/wiki/Color_vision