Crittografia nel Caos

Umberto Cerruti

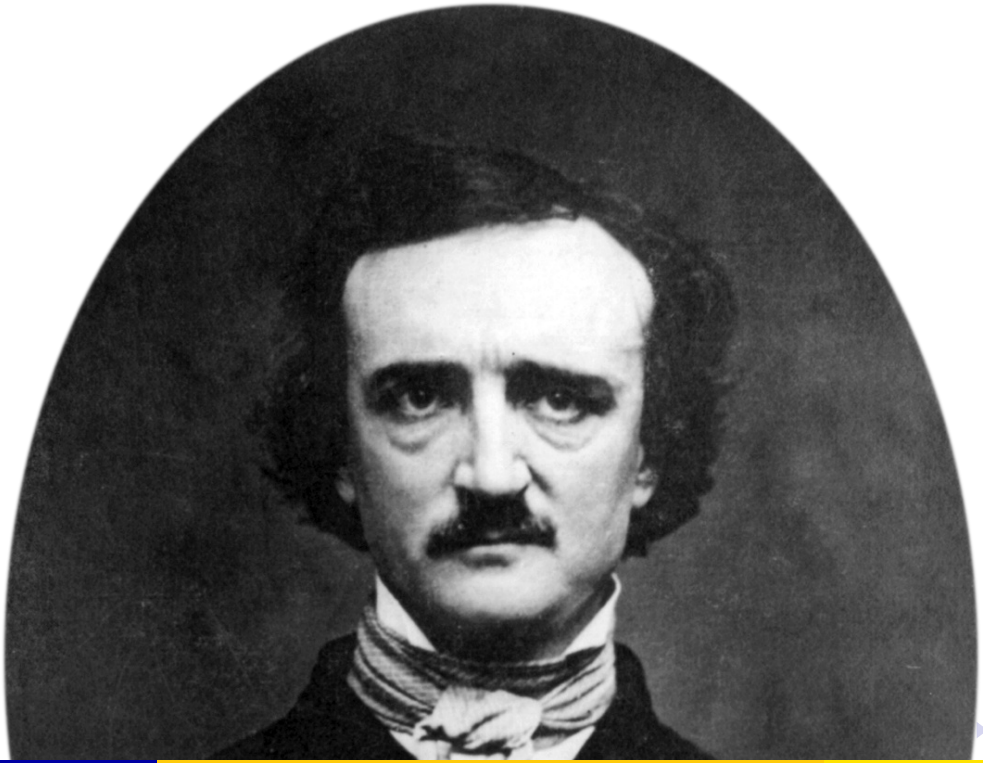
Università di Torino

Codici di Sostituzione Monoalfabetica

Chiamiamo *alfabeto* un insieme $\mathcal{A}$ di q simboli, che, quasi sempre, identificheremo con numeri interi.
Un *messaggio* è una successione finita di simboli. Dato un messaggio $M = x_1, x_2, ..., x_t$ siamo interessati ai metodi più efficaci per *crittare* M, in modo tale che risulti indecifrabile a chi non possieda la *chiave*. Diciamo $\mathcal{K}$ l'insieme delle chiavi. Questo insieme deve essere sufficientemente grande (rispetto alla potenza di calcolo dell'avversario) per far sì che non si possa utilizzare l'attacco più banale: quello di *esaustione*, si provano tutte le chiavi.
Il metodo più semplice è quello di applicare una *permutazione* σ ad $\mathcal{A}$.
Invece di inviare $x_1, x_2, ..., x_t$, inviamo $\sigma(x_1), \sigma(x_2), ..., \sigma(x_t)$
Per esempio, se $\mathcal{A} = (0, 1, 2, 3, 4)$, $M = (1, 1, 3, 4, 0, 2, 2, 3)$ e $\sigma = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 0 & 1 \end{pmatrix}$, allora, detto M' il messaggio cifrato, $M' = C_\sigma(M) = (4, 4, 0, 1, 2, 3, 3, 0)$, dove C_σ denota la funzione che mette in cifra, determinata dalla chiave σ.

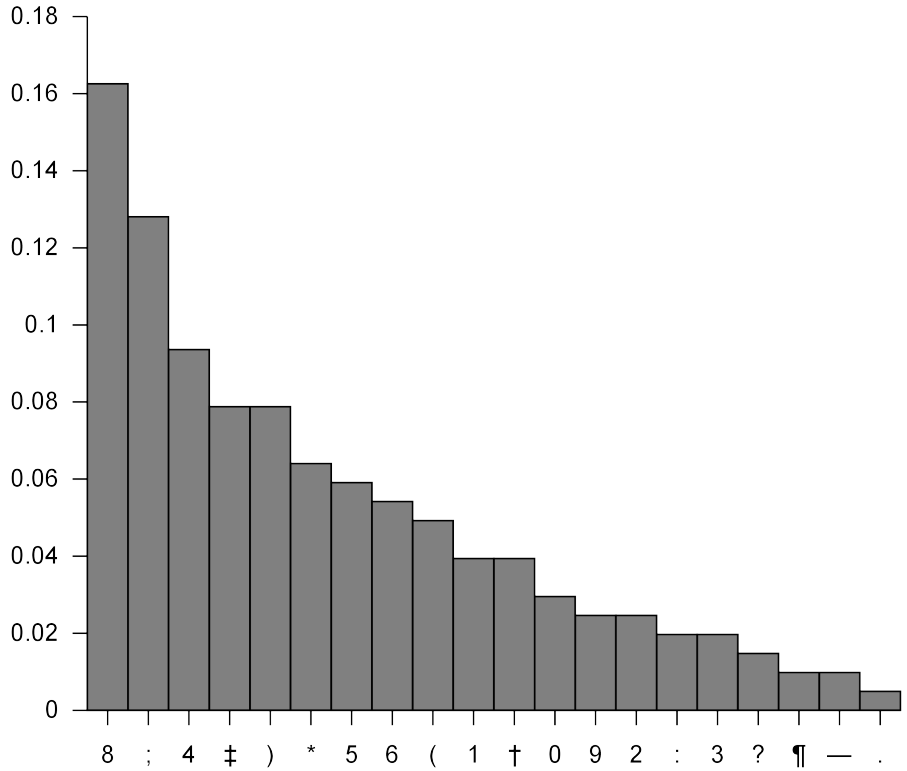
Debolezza della Sostituzione Monoalfabetica

La funzione di decodifica D_σ utilizza la permutazione inversa, $D_\sigma = C_{\sigma^{-1}}$.
Ovviamente, invece di utilizzare permutazioni di $\mathcal{A}$ possiamo utilizzare biezioni tra $\mathcal{A}$ e un alfabeto $\mathcal{A}'$, non cambia nulla.
Questi codici si possono rompere con un *attacco statistico*.
Un ottimo esempio di questo tipo di attacco è stato dato da Edgar Allan Poe nel racconto "The Gold-Bug" (1843).



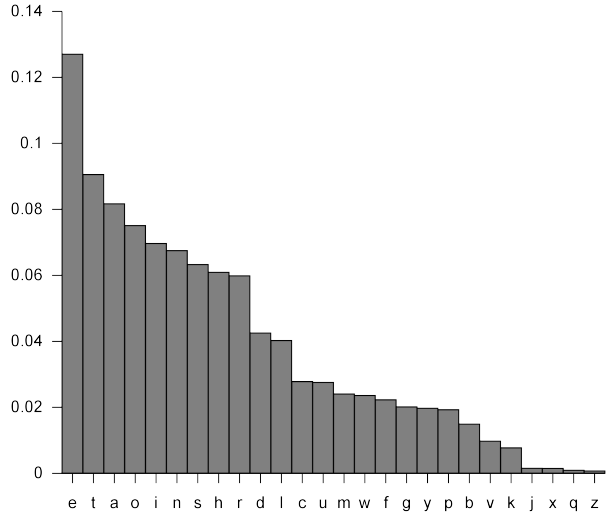
L'Attacco Statistico

Nel racconto di Poe la sostituzione monoalfabetica viene realizzata da una biezione tra l'alfebeto inglese e un insieme di 26 simboli.
Legrand scopre un messaggio cifrato con strani caratteri, e determina le loro frequenze nel documento che ha trovato, e le ordina.
Istogramma delle frequenze dei simboli nel testo:



Analisi delle frequenze

Legrand conosce anche le frequenze delle lettere nella lingua inglese:



La lettera più frequente è la *e*, e il simbolo più frequente è 8. Associamo allora 8 con *e*, ; con *t* e così via. Naturalmente non si otterrà subito la biezione esatta. Sostituendo le lettere ai simboli emergeranno alcune parti che hanno senso. Inoltre si utilizza la ridondanza della lingua (per esempio dopo la *t* c'è assai spesso una *h*) per correggere quello che non va. Procedendo con pazienza Legrand determinò la corrispondenza tra l'alfabeto inglese e i simboli.

I codici di sostituzione polialfabetica

Si può dire che la crittografia moderna sia nata nel 1586, quando il diplomatico francese Blaise de Vigenère (1523-1596) pubblicò il cifrario che porta il suo nome.

L'idea di Vigenere è quella di utilizzare più di una permutazione dell'alfabeto. La chiave σ è una stringa di k permutazioni $\sigma_1\,\sigma_2\ldots\sigma_k$: il messaggio M verrà cifrato così:

$$D_\sigma(M) = \sigma_1(x_1)\,\sigma_2(x_2)\ldots\sigma_k(x_k)\,\sigma_1(x_{k+1})\,\sigma_2(x_{k+2})\ldots$$

Come rompere i codici di sostituzione polialfabetica

L'idea di Vigenère è molto efficace. La statistica del messaggio viene completamente distrutta. Risulta inutile una indagine basata sulle frequenze relative delle lettere. Però nel 1863 Friedrich Kasiski comprese che, se è nota la lunghezza k della parola chiave, il problema di rompere un codice di sostituzione polialfabetica si può ridurre a quello di decifrare k codici monoalfabetici.

Infatti i simboli $x_1, x_{k+1}, x_{2k+1}, \ldots, x_{hk+1}$ saranno cifrati dalla permutazione σ_1, i simboli $x_2, x_{k+2}, x_{2k+2}, \ldots, x_{hk+2}$ saranno cifrati dalla permutazione σ_2, e così via.

Si tratta di una idea veramente *algebrica*: un codice complesso viene *spezzato* nella somma di k codici semplici.

Kasiski stesso descrisse un metodo per trovare la lunghezza della parola chiave, basato sulla distanza tra segmenti uguali nel messaggio cifrato. Se, per esempio, *xyz* appare più volte a distanze 10 e 15, è probabile che la lunghezza sia 5.

Trovare la lunghezza della chiave

Esistono metodi molto efficienti per trovare la lunghezza, come *l'indice di coincidenza* $I(T)$ di Friedman (1922). $I(T)$ è la frequenza delle coppie di simboli uguali nel testo T. Supponiamo che ci siano q simboli, che il messaggio sia lungo n, e poniamo n_i uguale al numero di occorrenze dell'$i-$esimo simbolo in T. Allora

$$I(T) = \sum_{i=1}^{q} \frac{n_i(n_i - 1)}{n(n-1)} \tag{1}$$

Ogni linguaggio L ha un suo indice (medio) che denotiamo con P_L.
$Q = 1/q$ è l'indice del linguaggio casuale.
Se T vene messo in cifra con cifrario polialfabetico e chiave di lunghezza λ, si avrà $I \sim P_L$ se $\lambda = 1$ mentre I si avvicinerà sempre più a Q all'aumentare di λ. Da queste considerazioni si ricava una formula che dà λ con ottima approssimazione. Precisamente, detta n la lunghezza di T

$$\lambda = \frac{(P_L - Q)n}{(P_L - I) + (I - Q)n} \tag{2}$$

Codifica dei Byte alla Vigenere

Attualizziamo il cifrario di Vigenère.

I simboli sono i *byte*, ovvero numeri di 8 bit, compresi tra 0 e 255.
Un testo T è una stringa di byte di lunghezza n.
La chiave k è una stringa di byte di lunghezza h.

La codifica avviene scrivendo ripetutamente k sotto T e sommando byte per byte modulo 256.

Per esempio vogliamo mettere in codice *La Terra ruota* con la chiave di lunghezza 3 $k = 181, 17, 200$. Trasformiamo la frase in una stringa T di byte con il codice *UTF8* e scriviamo sotto la chiave:

76	97	32	84	101	114	114	97	32	114	117	111	116	97
181	17	200	181	17	200	181	17	200	181	17	200	181	17

Sommando testo e chiave modulo 256 si ottiene

$$1, 114, 232, 9, 118, 58, 39, 114, 232, 39, 134, 55, 41, 114$$

Dal Friedman biologo ...

William Friedman (1891 – 1969) era un genetista delle piante, di origine moldava. Dopo la laurea presso la Cornell University, fu assunto, nel complesso di Riverbank, dall'eccentrico milionario George Fabyan, per studiare alcuni metodi di ibridazione e selezione in particolari coltivazioni. Friedman però venne catturato da un progetto secondario di Fabyan, la congettura che Francis Bacon sia il reale autore delle opere di Shakespeare.

... al Friedman crittografo

Si impegnò dunque nello studio del cosiddetto codice di Bacon, con il quale, secondo alcuni autori, Bacon avrebbe crittografato prove della sua paternità shakespeariana.

Durante la Prima Guerra Mondiale, Riverbank divenne un vero e proprio laboratorio - utilizzato dai militari - dove venivano decifrati, con notevole successo, i messaggi segreti inviati dal nemico.
Nel 1919 Friedman scrisse l'importante articolo *The Index of Coincidence and its Applications in Cryptography* ([Friedman 1919]), nel quale introduceva, tra l'altro, il famoso indice di coincidenza (1).

Ad un attento studio questo indice si rivela estremamente interessante e assai flessibile, tanto da adattarsi alla nostra epoca digitale.
Noi di solito abbiamo a che fare con file. Sospettiamo che uno di essi contenga un testo crittografato, in quanto non ci capiamo nulla.
Vogliamo vedere se è stato crittografato con Vigenère, e, in questo caso, vogliamo trovare la lunghezza della chiave.

L'indice in azione

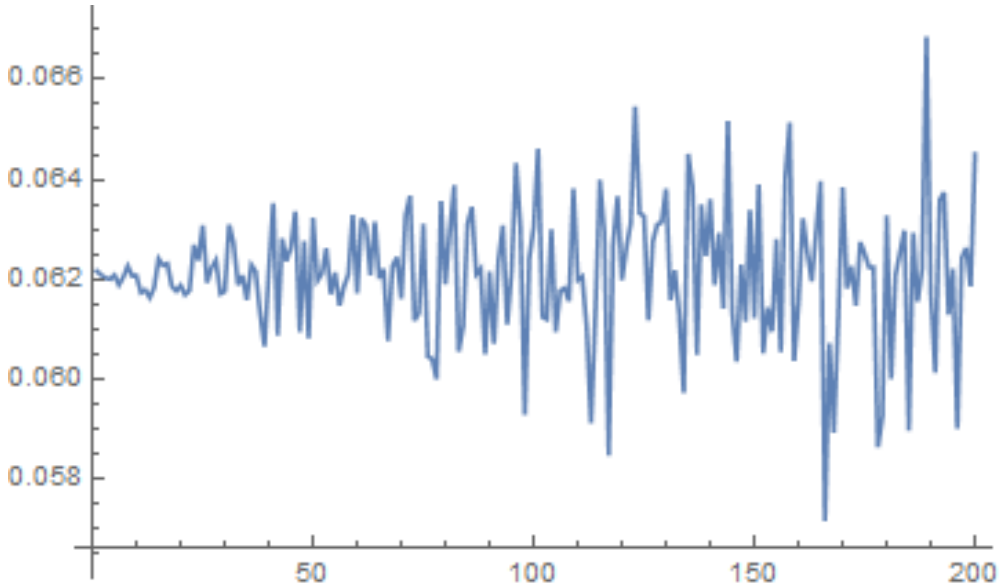
La formula (2) che ci dà la lunghezza della chiave richiede un dato che in genere non conosciamo: l'indice della lingua P_L.

In quale lingua o linguaggio o formato è stato scritto il testo?
E' Italiano o Tedesco? E' un programma in C? E' un file .exe o .pdf, o ...?
E chi conosce l'indice del linguagio C, per esempio?
Inoltre possono essere mescolati caratteri che non c'entrano con il contenuto, ma sono dovuti ad una particolare (e sconosciuta) formattazione.

Si può utilizzare l'indice di Friedman senza sapere nulla sulla lingua usata nel testo e sulla formattazione, in moltissimi casi.
La tecnica è questa: data una lunghezza d dividiamo il testo T di lunghezza n in $m = \lfloor \frac{n}{d} \rfloor$ parti, ottenendo una matrice $m \times d$ di interi (byte).
Diciamo $I(T, d)$ l'indice di coincidenza della $d-$esima colonna di M.
Se facciamo un grafico di $I(T, d)$ per $1 \leq d \leq m$, nel caso in cui la lunghezza della parola usata per codificare T sia $\leq m$ vedremo uno o più picchi nel grafico.

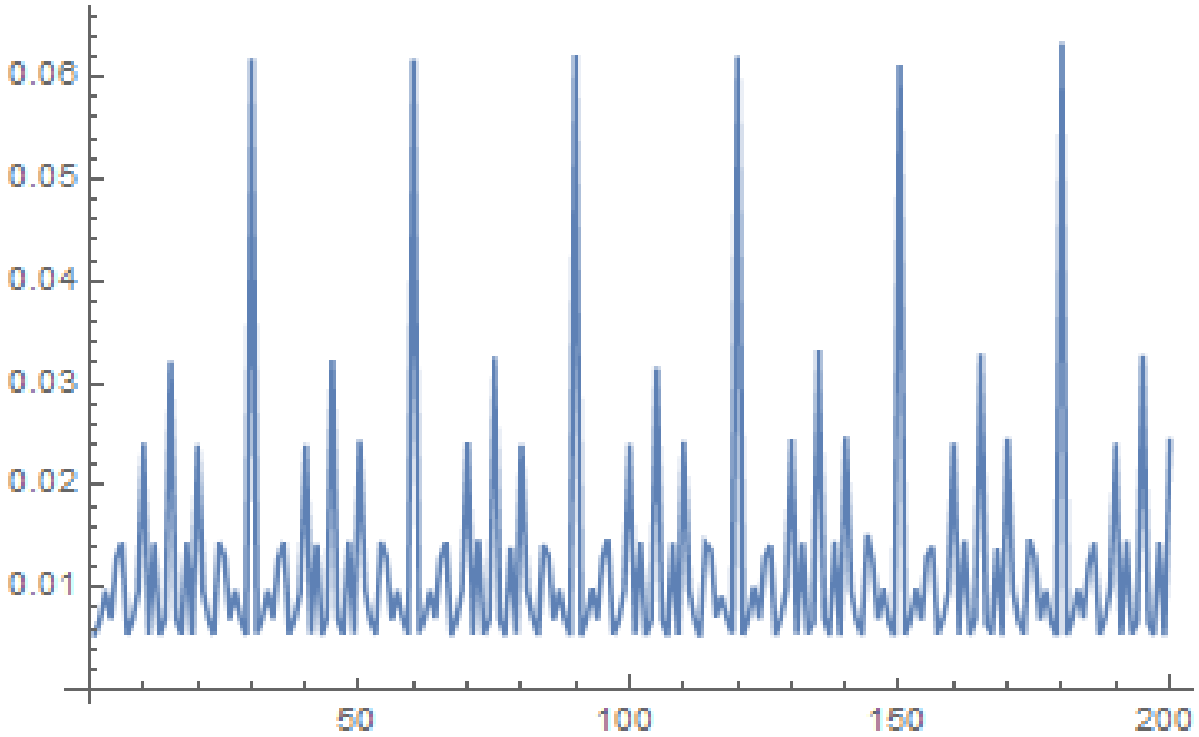
Esempio 1A

Il testo T considerato è il file dante.txt, il testo completo della Divina Commedia. Contiene 573753 caratteri.
Calcoliamo l'indice di coincidenza di T con la formula (1). In questo caso $q = 82$, ci sono 82 byte differenti.
Otteniamo $I(T) = 0.06217$, mentre l'indice di un testo casuale con 82 simboli è $1/82 = 0.01219$.
Il grafo della funzione $I(T, d)$, con $1 \leq d \leq 200$ è



Esempio 1B

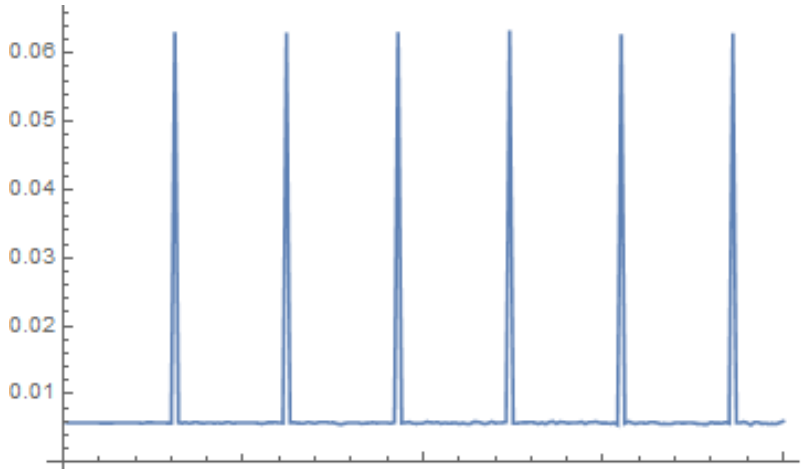
Ora codifichiamo T con Vigenère e chiave casuale lunga 30.
Abbiamo $T' = \mathit{Vigenere}(T, 30)$.
Il grafo della funzione $I(T', d)$, con $1 \leq d \leq 200$ è



In T' ci sono $q' = 256$ caratteri, $I(T') = 0.00590$ e $1/256 = 0.00390$.

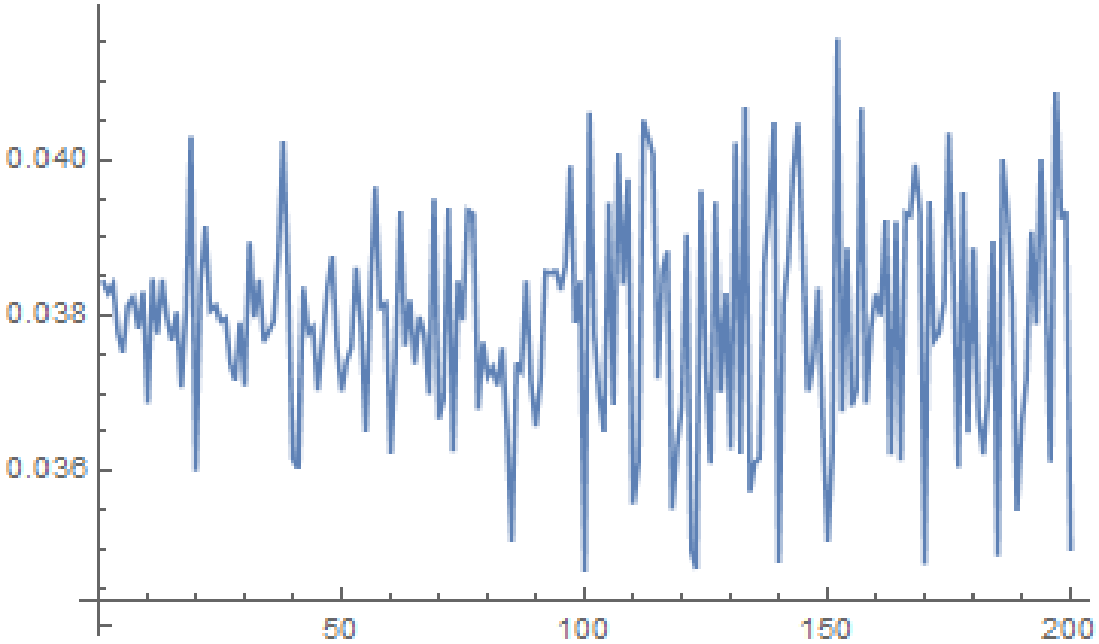
Esempio 1C

Si vede chiaramente che i picchi più alti sono in corrispondenza dei multipli di 30.
I picchi minori corrispondono ai divisori della lunghezza! Questo è facilmente comprensibile. Quando $d = 30$ la funzione $I(T', 30)$ restituisce l'indice di coincidenza della stringa dei byte di posto $30, 60, 90, ...$Quando $d = 15$, $I(T', 15)$ restituisce l'indice di coincidenza della stringa dei byte di posto $15, 30, 45, 60, ..$, che è per metà uguale alla colonna $30-$esima, e così via.
Dunque se la lunghezza è un numero primo, non ci saranno picchi secondari (dovuti ai divisori). Con $d = 31$ otteniamo infatti:



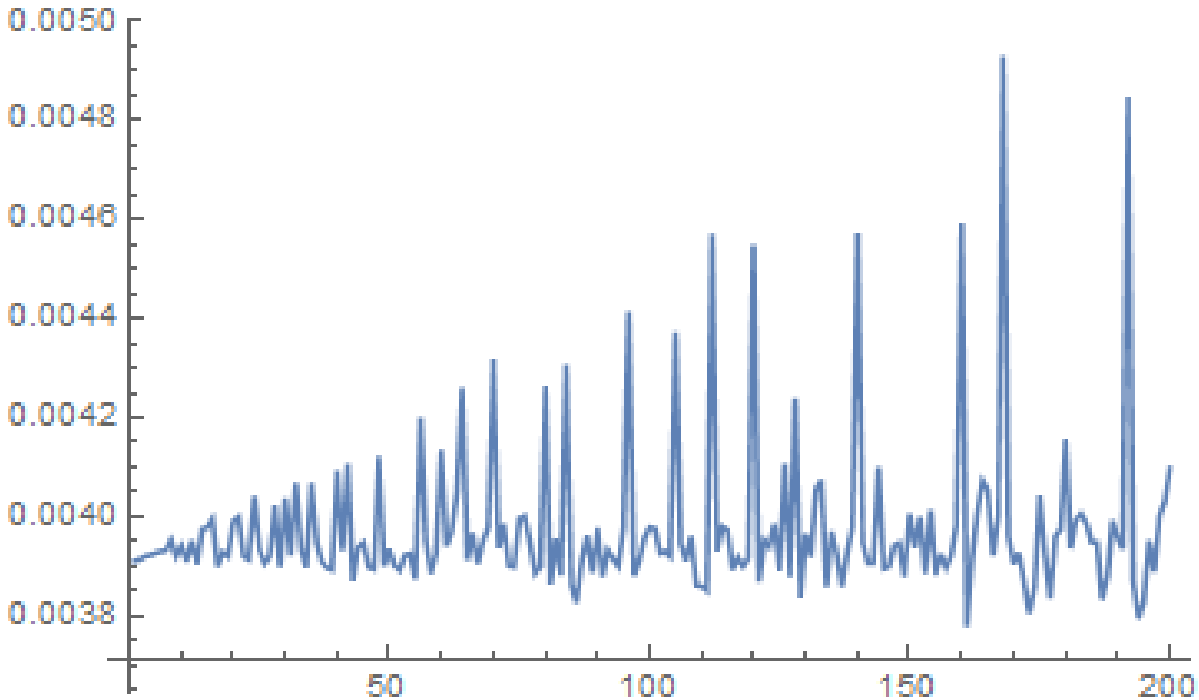
Esempio 2A

Il testo T che ora consideriamo è il file lip.c, il file principale di una nota libreria per il calcolo con interi in multiprecisione. Contiene 209747 caratteri.
In questo caso $q = 95$, ci sono 95 byte differenti.
Otteniamo $I(T) = 0.03842$, mentre l'indice di un testo casuale con 95 simboli è $1/95 = 0.01052$.
Il grafo della funzione $I(T, d)$, con $1 \leq d \leq 200$ è



Esempio 2B

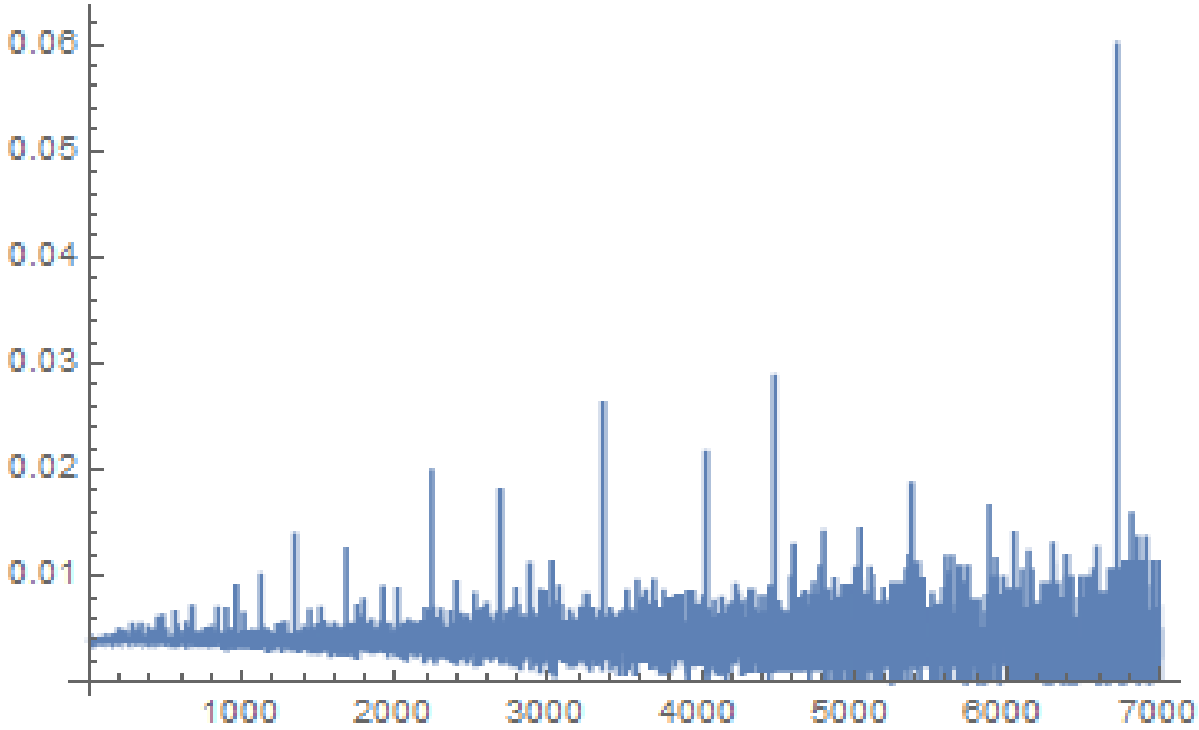
Supponiamo che T sia stato crittografato mediante Vigenère, con una chiave casuale di lunghezza ignota, ottenendo un testo T' che noi possiamo esaminare.
Vediamo che T' possiede 256 caratteri. Risulta che $I(T') = 0.00391$, molto vicino all'indice del testo casuale (0.00390). Questo è il grafo di I(T',d) con d che varia tra 1 e 200:



Esempio 2C

Il picco più alto si trova per $d = 168 = 2^3 \times 3 \times 7$. Il picco successivo corrisponde a $d = 192 = 2^6 \times 3$. Il picco precedente lo troviamo per $d = 160$, divisibile per 5. Tutto ciò ci porta a pensare che la chiave abbia una lunghezza multipla di $2^6 \times 3 \times 5 \times 7 \times 7 = 6720$.

L'ipotesi è confermata dal grafo di I(T',d), con d che va da 1 a 7000.



Qualità della Chiave

Lavorando e riflettendo sul metodo di Vigenère e l'indice di Friedman, ci si accorge presto che la chiave utilizzata, per essere sicura, deve possedere due qualità

- Deve essere lunga
- Deve essere casuale

Ricordiamo che la chiave è una sequenza di permutazioni σ_i dei q simboli dell'alfabeto utilizzato.

Negli esempi appena visti una permutazione si identifica con un byte, un intero compreso tra 0 e 255, quindi l'insieme delle permutazioni è in corrispondenza biunivoca con l'alfabeto $\mathcal{A} = (0, 1, 2, ..., 255)$.
Abbiamo utilizzato permutazioni particolari: sia $0 \leq m \leq 255$, denotiamo con σ^m la permutazione di $\mathcal{A}$ indotta da m. Si ha

$$\forall b \in \mathcal{A} \quad \sigma^m(b) = m + b \mod 256$$

Tutto è fatto di bit

Passiamo ora dai byte ai bit, riducendo tutto all'essenziale.
L'alfabeto è $\mathcal{A} = (0, 1)$. Un messaggio M è semplicemente una stringa di bit 0011101010...
Quali sono le permutazioni dell'alfabeto? Sono solamente due, quella identica, che lascia invariato il bit, chiamiamola σ_0, e quella che scambia 0 con 1, diciamola σ_1
Una possibile chiave k, di lunghezza 5, è $\sigma_1\sigma_1\sigma_0\sigma_1\sigma_0$. Allora 0011101010... codificato con k diventa 1110110000.

Denotiamo con $\oplus$ lo *XOR* dei bit, ovvero la somma modulo 2.
Evidentemente per ogni bit b, $\sigma_0(b) = b \oplus 0$ e $\sigma_1(b) = b \oplus 1$.
Concludendo, a livello di bit, la codifica alla Vigenère di M è semplicemente

$$C = M \oplus K$$

dove K è la chiave k opportunamente ripetuta. Ricordiamo che anche K è una stringa di bit, abbiamo sostituito b a σ_b.

One Time Pad

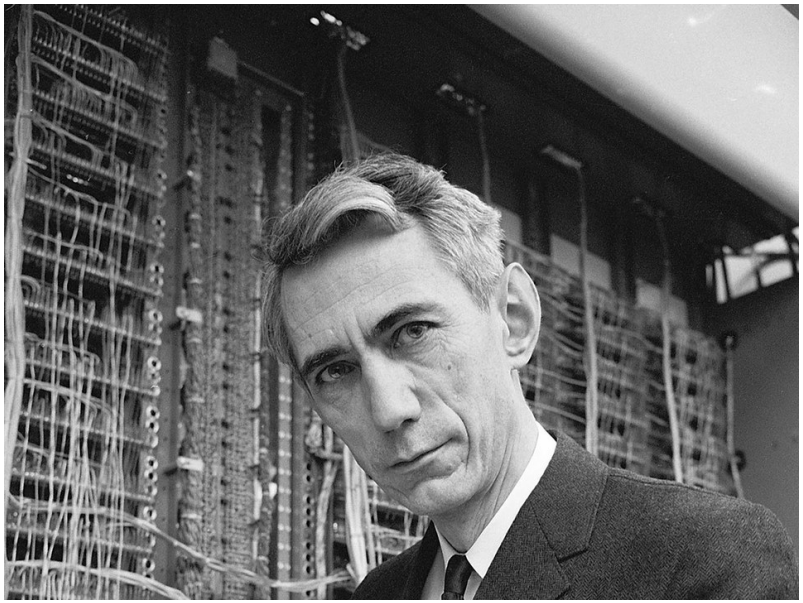
Infine, la chiave più sicura sarà una stringa di bit *lunga come il messaggio* e *casuale*. Si arriva così al metodo crittografico detto *One Time Pad*, che descriviamo:
A e B condividono una stringa casuale di bit K, lunga n. Se A vuole inviare a B un messaggio M di n bit, A calcola $C = M \oplus K$ e lo invia sul canale insicuro. B riceve C e lo decritta calcolando $C \oplus K = M$.
Chi intercetta C, non può ricavare, dal solo esame di C, alcuna informazione sul testo M, in quanto (posto che K sia casuale) C può provenire con la stessa probabilità da *qualsiasi* stringa di n bit

Questo metodo venne descritto da F. Miller nel 1882 ([Bellovin 2011]).

E' perfetto!

Venne riscoperto e brevettato da G. Vernam nel 1919.
Nel 1949 Shannon ([Shannon 1949]) dimostrò che questo codice è (l'unico) *codice perfetto*: come abbiamo detto, da C non si può ottenere alcuna informazione su M. Nessuna potenza di calcolo può rompere il One Time Pad!.

Gilbert Vernam
1890-1960



Claude Shannon
1916-2001

Solo in apparenza inutile

Il One Time Pad ha, evidentemente, un grosso problema: gli utenti devono condividere n bit di chiave per scambiarsi n bit di messaggio. Tutti pensiamo, di primo acchito, che sarebbe più semplice condividere direttamente il messaggio!

Ma non è così. Il metodo OTP è stato utilizzato anche a livello militare, si vedano, per una storia dettagliata, ([Bauer 2013]) e ([Kahn 1967]).

Al giorno d'oggi le sorgenti di bit sono ovunque, e non è difficile concordare con l'altro utente una canzone, una immagine, una applicazione, tra le miriadi che si trovano in rete. Dopo di che A e B avrannno a disposizione giga e giga di bit, che saranno poi adattati per utilizzarli come chiavi.

L'unica precauzione è cambiare ogni volta la chiave.

In molti casi però si richiedono precisi protocolli. Il più utilizzato è questo: A e B producono, ognumo per conto proprio, la *stessa* sequenza di bit.

Generatori di Numeri Casuali

Cominciamo con l'osservare che queste sequenze di numeri (o bit) sono prodotte da programmi che girano su ordinari computer, e sono pertanto del tutto deterministiche.

Chiamarle casuali sembra azzardato. Infatti vengono chiamate *pseudo casuali*. I loro generatori vengono detti *PRNG*, *Pseudo Random Number Generators*.

La generazione di numeri casuali è indispensabile in Crittografia, ma anche in molti importanti settori, dai modelli matematici, alle simulazioni ai giochi. Infatti sono stati scritti fiumi di articoli e decine di libri sull'argomento. Si veda la bibliografia ragionata, redatta da Beebe ([Beebe 2019]): sono 701 pagine che contengono 3978 citazioni.

Ma che cosa generano questi *PRNG*? Che cosa è una sequenza di interi, o di bit, casuali?

Tre visioni della casualità di sequenze

In ([Goldreich 2019]) Goldreich riassume molto bene i concetti basilari delle tra principali teorie.

1. Nella sua Teoria dell'Informazione (1948), Shannon caratterizza la casualità perfetta come il caso estremo nel quale la stringa di simboli non contenga ridondanza alcuna, ovvero *sia massima la informazione*.
2. Solomonov (1960), Kolmogorov (1963) e Chaitin (1965), fondarono la seconda teoria, di stampo computazionale. La complessità di una stringa è essenzialmente la lunghezza del più piccolo programma che può generarla. In sostanza, se una stringa è veramente casuale per poterla esprimere un programma deve contenerla.
3. Blum, Goldwasser, Micali e Yao iniziarono, negli anni $1982 - 84$, la terza teoria che presta attenzione al calcolo effettivo. Una sequenza è casuale se non disponiamo di procedure di calcolo che riescano a distinguerla da una distribuzione uniforme.

Kolmogorov

La teoria che mi ha sempre affascinato è quella di Kolmogorov. Malgrado si dimostri che la complessità di Kolmogorov non è computabile, possiede molte applicazioni ([Li 2008]) e fornisce indicazioni, anche pratiche.

Senza entrare nei dettagli teorici possiamo ricorrere a concetti intuitivi e dire che, dalla teoria di Kolmogorov segue questo fatto

Casualità e Incompressibilità delle stringhe sono equivalenti

Si noti bene che questa equivalenza è vera a livello teorico. In pratica se non riusciamo a comprimere un testo M, questo non implica che M sia casuale.
Si tratta comunque di una buona indicazione, l'indice di compressione fa sempre parte di una batteria di test di casualità. Inoltre, se M si comprime in modo significativo, evidentemente non è casuale.
Mentre la teoria di Shannon si fonda sulla idea di distribuzione uniforme, con Kolmogorov possiamo parlare di casualità di una sequenza senza fare riferimento ad alcuna distribuzione.

Quasi tutto è incomprimibile I

Consideriamo l'insieme B_n delle 2^n stringhe di n bit.

Se una stringa $x \in B_n$ è comprimibile di almeno un bit, deve essere espressa da una stringa $y \in B_j$ con $j \leq n-1$.
Poichè

$$\sum_{j=0}^{n-1} 2^j = 2^n - 1$$

esiste almeno una stringa di lunghezza n che è del tutto *incomprimibile*, tutti i suoi bit sono *necessari* per identificarla.

Proseguendo con il semplice calcolo scopriamo che il numero delle stringhe in B_n comprimibili di almeno c bit è $2^{n-c+1} - 1$.

Quindi quelle non comprimibili più di $c-1$ bit sono $2^n - 2^{n-c+1} + 1$.

Quasi tutto è incomprimibile II

Pertanto, approssimando, possiamo dire che almeno la metà delle stringhe non è comprimibile più di 1 bit, almeno i 3/4 non sono comprimibili più di 2 bit e così via (vedi ([Li 2008]) p.117).

Come mai allora abbiamo programmi che comprimono i nostri file anche della metà o più?

I file che più si comprimono sono quelli di testo. I linguaggi sono estremamente *ridondanti*. La ridondanza è necessaria per facilitare la comprensione e la memorizzazione.

Vi sono però, anche sul nostro hard disk file incomprimibili. Se comprimiamo un file .mp3, otteniamo a volte un file più grande dell'originale!

Lo stato *normale* della informazione è dunque quello di essere assai poco comprimibile. Questo fatto può avere conseguenze?

La ragione per cui esistono infiniti primi ...

Dalla famosa dimostrazione di Euclide della infinità dei numeri primi, molte altre sono state date. Ce n'è anche una nostra (con Nadir Murru ([Cerruti 2017])), dove si prova che se i numeri primi fossero finiti, il reciproco di un intero sarebbe ancora intero!

Una della prove che più amo è quella di Chaitin ([Chaitin 2006]).

Supponiamo che esistano soltanto k primi, $p_1, p_2, ..., p_k$. Dato un *qualsiasi* intero $N > 1$, per il teorema di fattorizzazione unica, avremo

$$N = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$$

Chiaramente abbiamo $k \leq \log_2 N$ e $\forall i,\ e_i \leq \log_2 N$.

Il numero N è identificato precisamente dalla stringa degli esponenti $e_1, ..., e_K$. Questi esponenti sono interi $\leq \log_2 N$ e quindi si possono esprimere (ognuno) con $\log_2 \log_2 N$ bit.

... è l'esistenza di testi incomprimibili!

In conclusione, ogni intero N può essere espresso mediante

$$k \log_2 \log_2 N \tag{3}$$

bit.

Prendiamo una stringa M che contenga m bit. A questa corisponde in modo univoco un intero N di m bit, dell'ordine di 2^m.

Per la (3) N, e quindi M, è determinato da una stringa di $k \log_2 m$ bit.

Esiste un m_0 tale che

$$\forall m > m_0 \quad m > k \log_2 m$$

Da questo si deduce che *tutte le stringhe di bit abbastanza lunghe sono comprimibili!*
Ma sappiamo bene che questo non è vero.
Pertanto esistono infiniti primi!

Vigenère e la Compressione

Una della richieste della crittografia moderna è che il messaggio crittografato sia indistinguibile da una stringa casuale di bit. Conseguentemente dovrebbe essere incomprimibile.

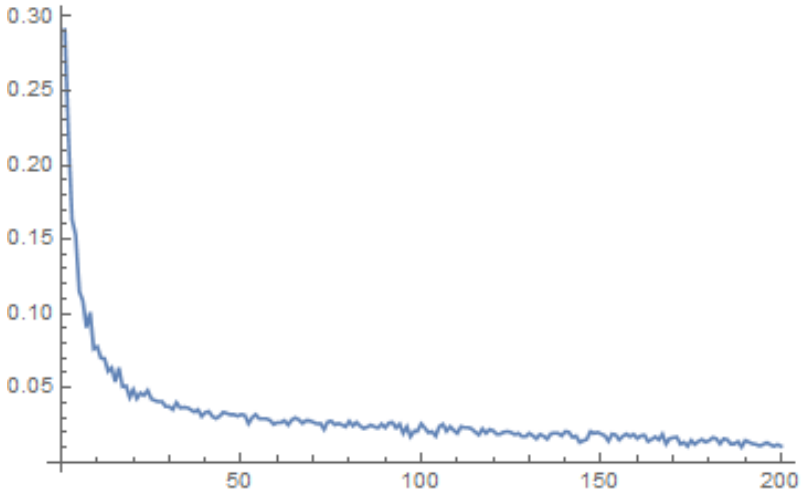
Solitamente un file viene prima compresso e poi crittografato, ma un buon metodo crittografico dovrebbe esso stesso restituire un messaggio poco comprimibile.

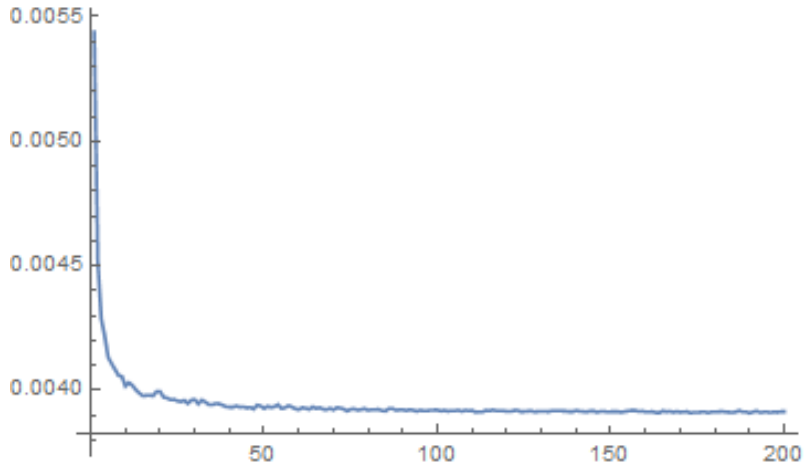
Diciamo $|s|$ la lunghezza di una stringa, e indice di compressione (relativamente ad un certo metodo utilizzato) il numero $1 - |s'|/|s|$. Negli esperimenti abbiamo usato il compressore 7 – *zip*.

Ci aspettiamo che all'aumentare della lunghezza della chiave un testo crittato con Vigenère diventi sempre meno comprimibile. Al tempo stesso scenderà l'indice di Friedman, tendendo all'indice del linguaggio casuale.

E' molto interessante paragonare i due grafi.

Grafici





Il testo T utilizzato contiene circa 55000 caratteri (sono due lezioni di Galileo).

T è stato crittografato con Vigenère con chiavi casuali di lunghezza $50k$, con $1 \leq k \leq 200$. I valori di k sono riportati sulle ascisse.

Sul grafico a sinistra si vede l'andamento dell'indice di compressione.

L'indice di Friedman è rappresentato a destra.

Compressione irreversibile e illimitata: le impronte digitali

$B_n = \mathbb{Z}_2^n$ sia l'insieme di tutte le stringhe binarie di lunghezza n.
Ovviamente se $n > m$ e $f : B_n \longrightarrow B_m$ è una funzione, f non può essere iniettiva. Esistono certamente in B_n x, x' tali che $f(x) = f(x')$. Diciamo che x, x' sono una *collisione*. Diciamo che f è resistente alla collisione se è computazionalmente difficile trovare una collisione.
Merkle e Damgard hanno trovato un metodo, a cascata, che permette di costruire, partendo da una funzione $h : B_{2m} \longrightarrow B_m$, una funzione $H : B^* \longrightarrow B_m$, dove B^* è l'insieme di *tutte* le stringhe binarie. Inoltre se h è resistente alla collisioni anche H lo è.
Una funzione $H : B^* \longrightarrow B_m$ resistente alle collisioni viene detta funzione *hash*.
Le funzioni hash sono molto importanti in crittografia, per esempio per gestire le password. In questo ambito abbiamo brevettato un sistema che contiene una funzione hash piuttosto interessante ([Perna 2017]).
Tra l'altro le funzioni hash sono parte essenziale della procedura del sistema Bitcoin ([Nakamoto 2002]).

Impredicibilità e automi

Intuitivamente una sequenza casuale dovrebbe essere *impredicibile*.

Nei problemi di predizione, risulta naturale utilizzare tecniche di Soft Computing o Intelligenza Artificiale.

Qualche anno fa, con Mario Giacobini e Pierre Liardet ([Cerruti 2002]), abbiamo studiato le capacità di predizione di automi a stati finiti, evolvendo popolazioni di automi mediante Algoritmi Genetici.
I risultati sono stati incoraggianti.

Il numero di stati dell'automa necessari per ottenere una certa soglia di successo aumenta con la casualità delle stringhe. Si potrebbero considerare automi più complessi e introdurre una nuova misura di casualità.

Per esempio in ([Smith 2018]) T. Smith considera automi di diverso tipo per la predizione di stringhe infinite con vari tipi di periodicità.
Queste macchine possono avere più testine che ad ogni passo si muovono lungo la sequenza infinita assegnata, leggono il simbolo, cambiano stato e fanno una ipotesi sul simbolo seguente.

Impredicibilità e reti neurali

In ([Fan 2018]) Fenglei Fa e Ge Wang utilizzano Reti Neurali per fare predizioni sui bit di π. Si ritiene che π sia un numero *normale* (vedi ([Cerruti 2019])).

I numeri normali sono spesso utilizzati come *PRNG* mediante la sequenza delle loro cifre.

Sorprendono dunque i risultati di Fa e Wang, le lori reti neurali predicono il bit successivo (di stringhe di 6 bit) con probabilità $> 1/2$ (con un altissimo grado di confidenza).

Questo contrasta con la ipotesi di normalità di π in base 2.
Gli Autori applicano il metodo ad altri irrazionali, e a stringhe generate da *PRNG*. Concludono che reti neurali, anche molto semplici, possono trarre dai dati (se non siamo in presenza di massima entropia, ovvero totale disordine) informazioni utili alla predizione.
Gli autori intendono affrontare con il loro metodo anche i *QRNG* (*Q* sta per quantum) per vedere se *Dio gioca con pseudo dadi*!

Blum Blum Shub

In un articolo veramente magistrale ([Blum 1986]) L. Blum, M.Blum e M. Shub hanno esaminato la predicibilità di due *PRNG*. Uno di essi è il *generatore quadratico*.
Sia $N = pq$, dove p, q sono primi congrui a 3 modulo 4. Scegliamo un intero x_0, il *seme*.
Fabbrichiamo la successione

$$x_{n+1} = x_n^2 \mod N$$

Infine abbiamo la sequenza di bit generata: $\mathcal{B}_n = x_n \mod 2$.
Diciamo che una successione di bit (b_n) è impredicibile se dato un segmento iniziale privato di un suo qualsiasi bit b una macchina di Turing probabilistica che agisca in tempo polinomiale (nella lunghezza del seme) non può indovinare il bit mancante con probabilità $> 1/2$.
Basandosi sulla ipotesi di difficoltà della fattorizzazione di interi (cosa per me certissima) gli autori provano che la sequenza $\mathcal{B}_n$ è impredicibile.

E' davvero casuale? I

Nel loro articolo *BBS* studiarono un secondo generatore, oltre quello quadratico, il generatore $1/P$, che è veramente notevole.

P è un numero primo, e b è una base. Se b genera il gruppo moltiplicativo modulo P, l'espressione di $1/P$ in base b ha periodo $P-1$.

Se b genera $\mathbb{Z}_p^*$ si ottengono lunghe stringhe di $b-$cifre, che passate ai test statistici danno buoni risultati.

Però *BBS* dimostrano che il generatore è facilmente predicibile, infatti è sufficiente conoscere $2l$ cifre consecutive (dove l è la lunghezza di p in base b) per dedurre l'intero periodo.

Vediamo come si fa con un esempio,il metodo è poi facilmente applicabile a tutti i casi.

Prendiamo $p = 541$. 10 genera $\mathbb{Z}_{541}^*$ e $l = 3$. Il periodo è lungo 540.

E' davvero casuale? II

Supponiamo di conoscere 6 cifre consecutive del periodo, per esempio 256931.
Dividiamo 256931 per 10^6 ottenendo $z = \frac{256931}{1000000}$.
Ora sviluppiamo z in frazione continua e scriviamo i convergenti. In questo caso ci sono 15 convergenti. Il primo è 0 e l'ultimo è z. Calcoliamo le espressioni decimali di essi con più di 6 cifre esatte.
Prendiamo il primo convergente la cui epressione decimale inizia con 0.256931...
Il denominatore di c è p. Conoscendo p si ricava tutto il periodo, ovvero la sequenza segreta.
Nel nostro caso i convergenti sono

$$0, \frac{1}{3}, \frac{1}{4}, \frac{9}{35}, \frac{28}{109}, \frac{37}{144}, \frac{102}{397}, \frac{139}{541}, \frac{658}{2561}, \frac{797}{3102}, \frac{6237}{24275}, \frac{7034}{27377}, \frac{13271}{51652}, \frac{20305}{79029}, \frac{256931}{1000000}$$

Le espressioni decimali dei convergenti sono

$$0, 0.3333333, 0.2500000, 0.2571429, 0.2568807, 0.2569444,$$
$$0.2569270, 0.2569316, 0.2569309, 0.2569310, 0.2569310, ...$$

E' davvero casuale? III

Il primo posto dove appare 0.256931 è l'ottavo, che corrisponde al convergente $\frac{139}{541}$. Il denominatore è il primo cercato!

Ma, se non ce lo avessero detto *BBS* avremmo capito da soli che il generatore $1/p$ è così predicibile?

In genere non ci sono teoremi che provino la predicibilità e la casualità (come nel caso del generatore quadratico). Occorre affidarsi a batterie di test come Ent, DieHard, NIST Test, TestU01, che sono i più utilizzati.

Però ci può trovare in un caso analogo a quello del *Mersenne Twister*, che ha un periodo enorme, $2^{19937} - 1$ ma, come del resto dichiarato dagli stessi autori (vedi ([Matsumoto 1988])), non è adatto a fini crittografici.

Vengono creati sempre nuovi generatori e si oppongono ad essi nuovi test.

Sembra una contesa senza fine!

Alla ricerca del Generatore Perfetto

In un articolo del Luglio 2019 ([Haramoto 2019]) Makoto Matsumoto, uno degli autori del Mersenne Twister, con due colleghi, ha criticato il generatore *xorshift*128+ (introdotto da Sebastiano Vigna nel 2017) molto utilizzato (per esempio in Google Chrome). Questo *PRNG* passa i TestU01, considerati molto severi, però presenta diversi difetti.

Gli autori concludono che non ci deve fidare troppo dei test, anche perché contengono molti parametri, determinati con grande accuratezza, ma comunque poco rappresentativi rispetto alle infinite possibilità esistenti.

Sul mercato poi un fattore determinante è la velocità, che non sempre va d'accordo con la qualità.

La situazione dei *PRNG* sembra caotica, e se ci rivolgessimo direttamente al Caos?

Non parliamo del Caos Primordiale ...

"Prima del mare, della terra e del cielo che tutto copre, unico era il volto della natura in tutto l'universo, quello che è detto Caos, mole informe e confusa, non più che materia inerte, una congerie di cose mal combinate tra loro (...)

Niente aveva forma stabile, ogni cosa si opponeva all'altra perché, in un corpo solo, il freddo lottava con il caldo, l'umido con il secco, il molle con il duro, il peso con l'assenza di peso.

Un dio, col favore della natura, sanò questi contrasti: dal cielo separò la terra, dalla terra il mare e dall'aria densa distinse il cielo limpido.

E, districati gli elementi fuori dell'ammasso informe, riunì quelli dispersi nello spazio informe in concorde armonia"

(Ovidio, Le Metamorfosi, Libro I)

Sistemi Dinamici Caotici

Utilizziamo la definizione di Devaney ([Devaney 2003])

Sia definita una funzione $f : J \longrightarrow J$ dove J è contenuto in $\mathbb{R}^n$.
Siamo interessati a questo problema: dato un punto iniziale $x_0 \in J$ che cosa succede iterando la funzione f a partire da x_0?

Chiamiamo *orbita* di x_0 l'insieme $(x_0, x_1 = f(x_0), ..., x_{n+1} = f(x_n), \dots)$.
Poniamo $f^{(n)} = \underbrace{f \circ f \circ f \dots \circ f}_{n \text{ volte}}$. Pertanto $x_n = f^{(n)}(x_0)$.

Un punto z si dice *periodico*, di periodo n, se $f^{(n)}(z) = z$.

Un sistema dinamico (J, f) viene detto *caotico* se valgono tre proprietà:

1. E' sensibile alle condizioni iniziali.
2. E' topologicamente transitivo.
3. L'insieme dei punti periodici è denso in J.

Sistema Dinamico Caotico: spiegazioni

Sensibilità alle condizioni iniziali:

Esiste $\delta > 0$ tale che per tutti gli $x \in J$ e ogni intorno N di x esistono un $y \in N$ e un intero $n \geq 0$ tali che $|f^{(n)}(x) - f^{(n)}(y)| > \delta$.

Topologicamente transitivo:

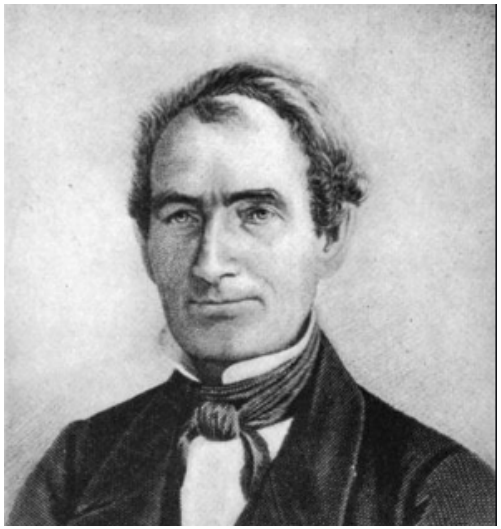
Dati due aperti $U, V \subseteq J$, esiste n tale che $f^{(n)}(U) \cap V \neq \varnothing$.

L'insieme dei punti periodici è denso in J

Per ogni $x \in J$ e per ogni intorno N di x, esiste un $y \in N$ periodico.

Come dice Devaney queste proprietà garantiscono che un sistema caotico è *impredicibile*, *indecomponibile* e, al tempo stesso, è inseparabile da una sottostruttura *regolare*.

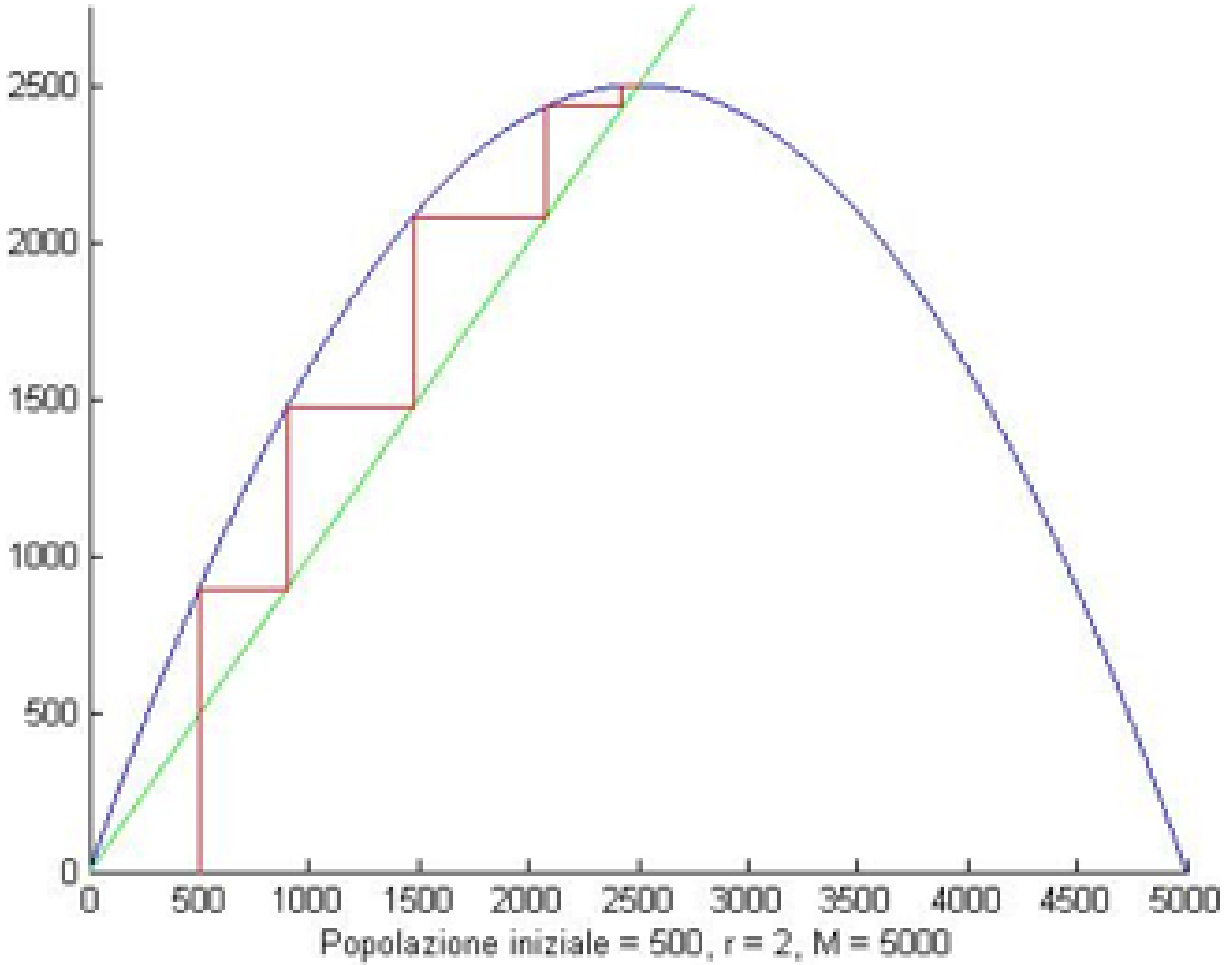
Un Modello di Crescita di una Popolazione



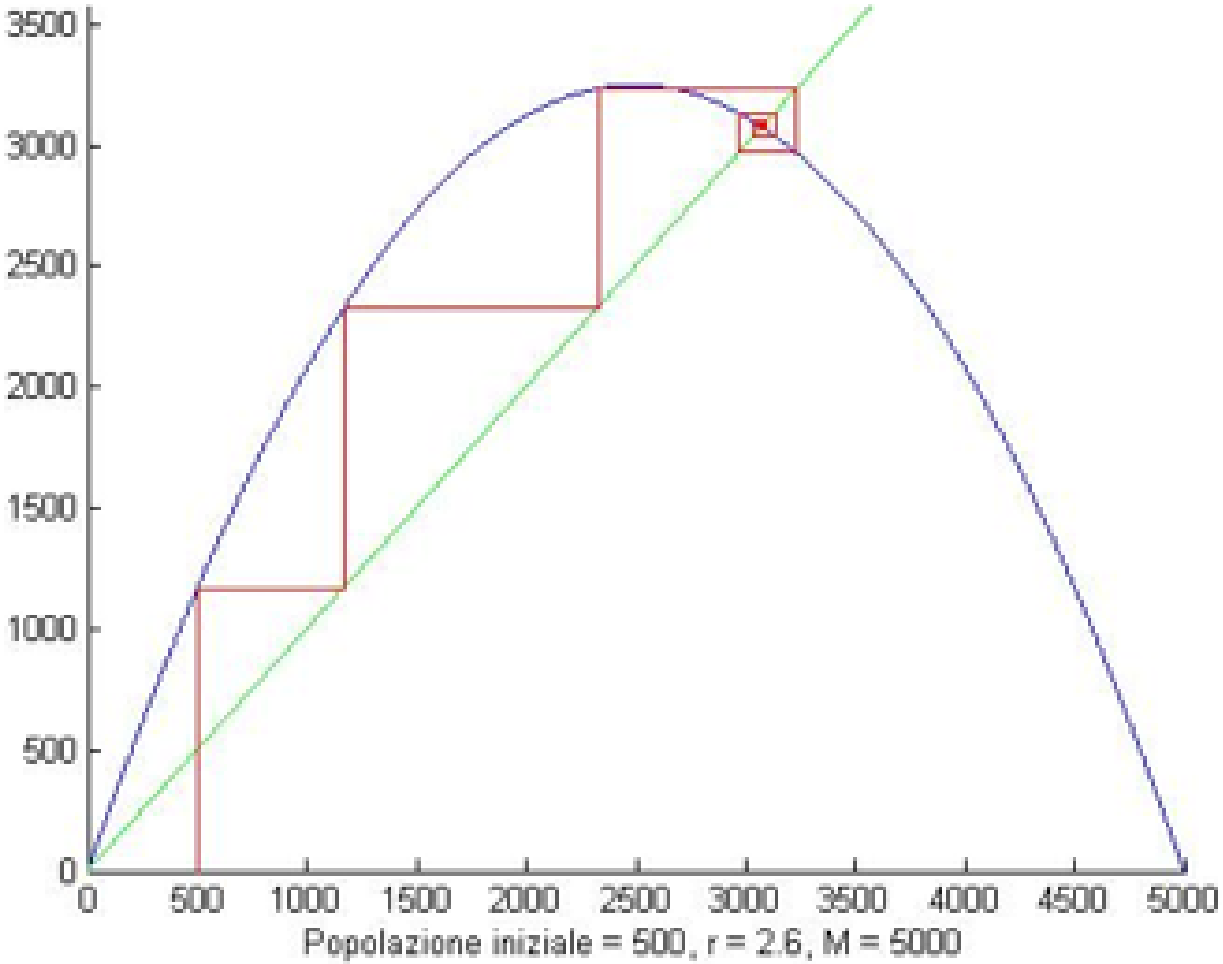
Un importante modello di crescita è quello della *logistica discreta*, introdotto e studiato da P. F. Verhulst.
Supponiamo che, in certo ambiente, non possano vivere più di M individui, e che il tempo sia contato in periodi $t = 0, 1, 2, \ldots$.
Denotiamo con $P(t)$ il numero di individui presenti al tempo t.

$$P(t+1) = rP(t)\left(1 - \frac{P(t)}{M}\right) \tag{4}$$

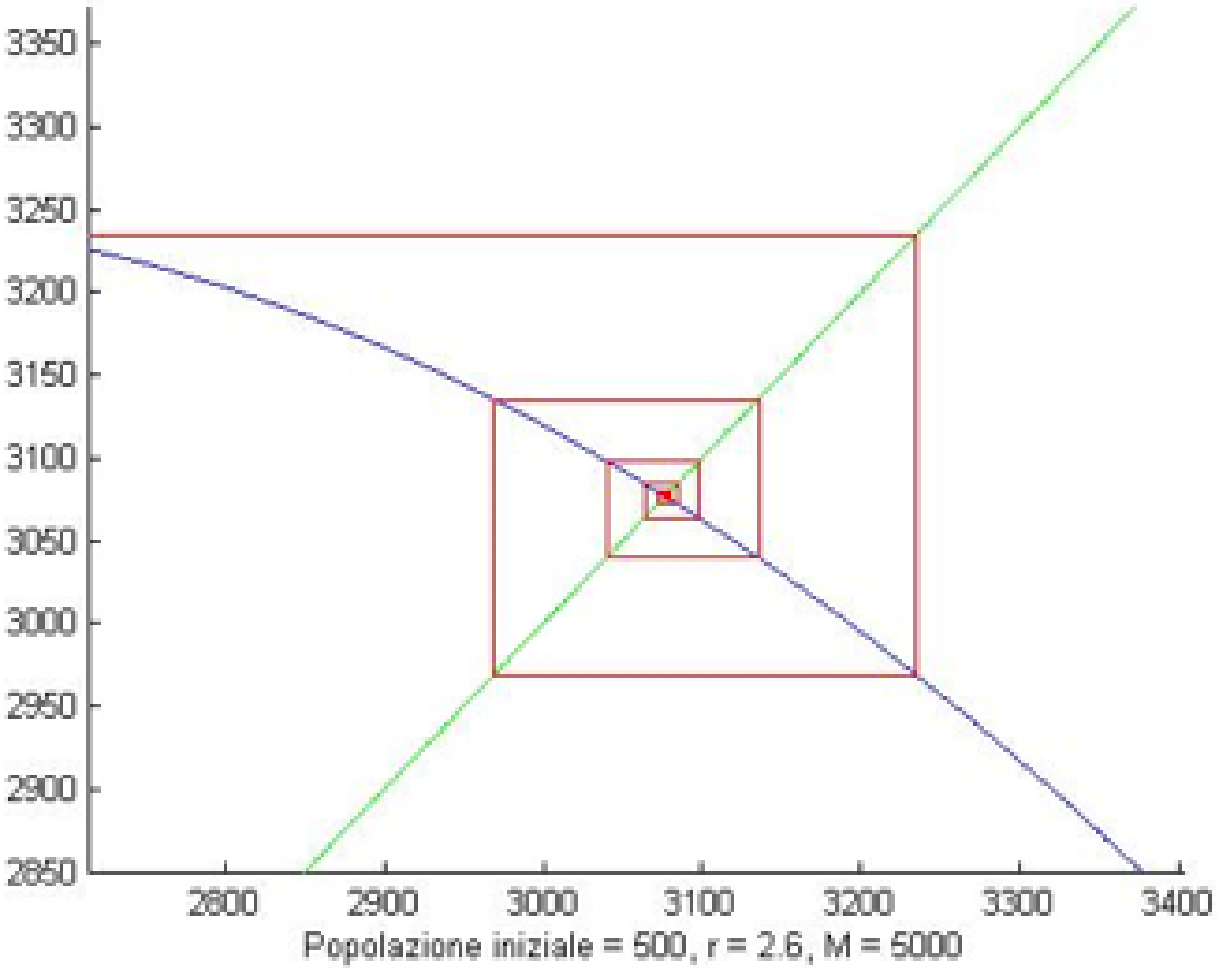
Evoluzione della Popolazione con r=2



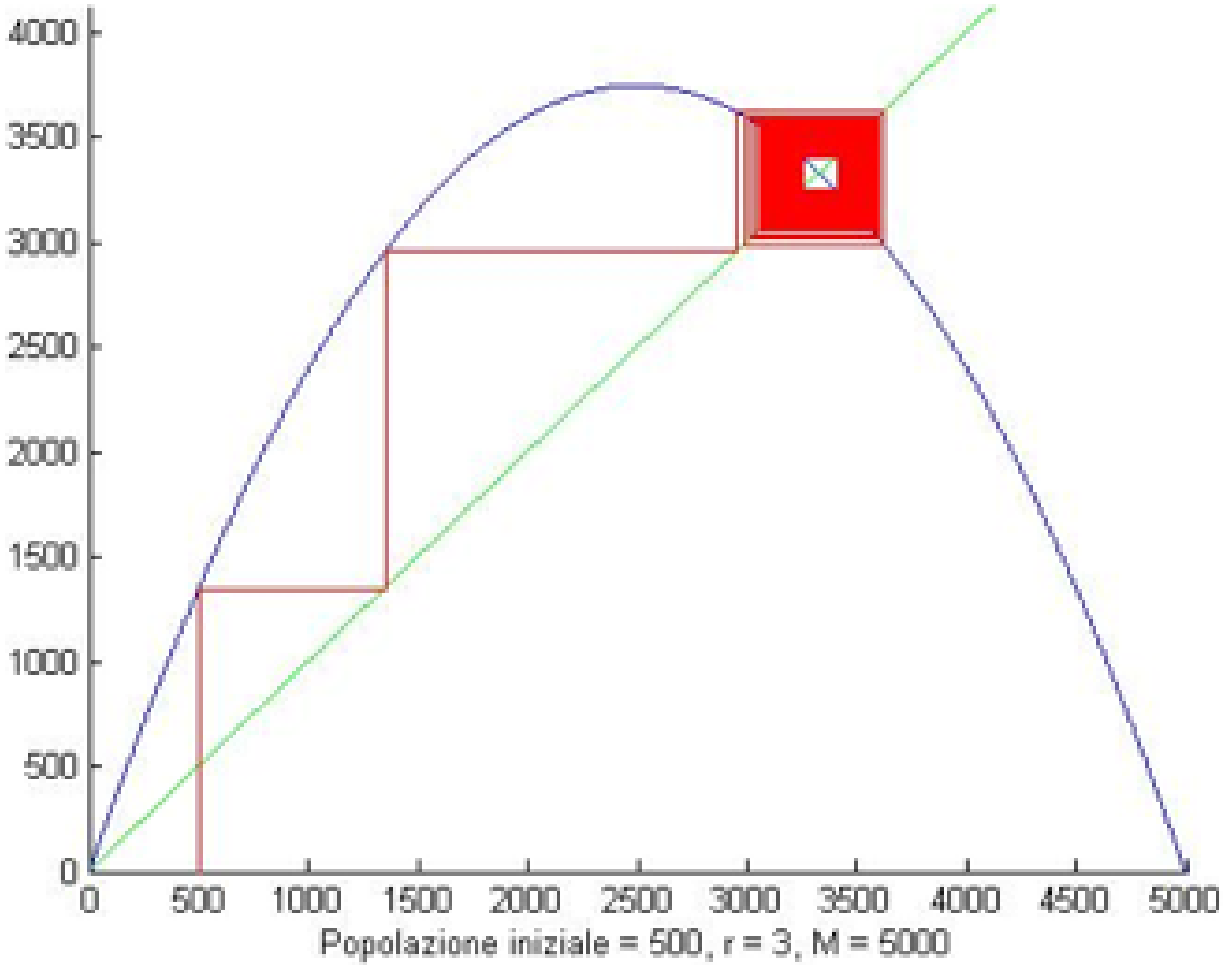
Evoluzione della Popolazione con r=2.6



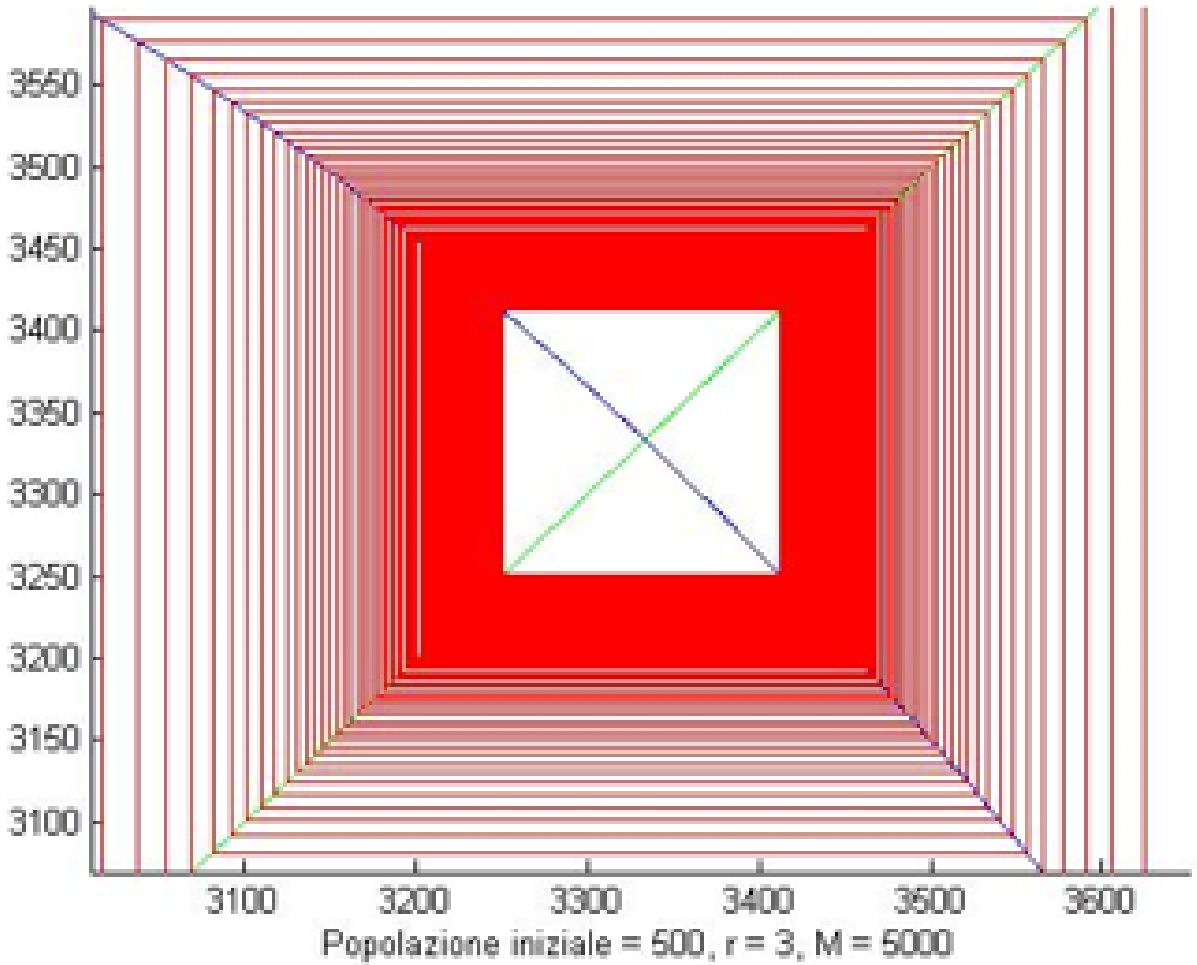
Particolare con r=2.6



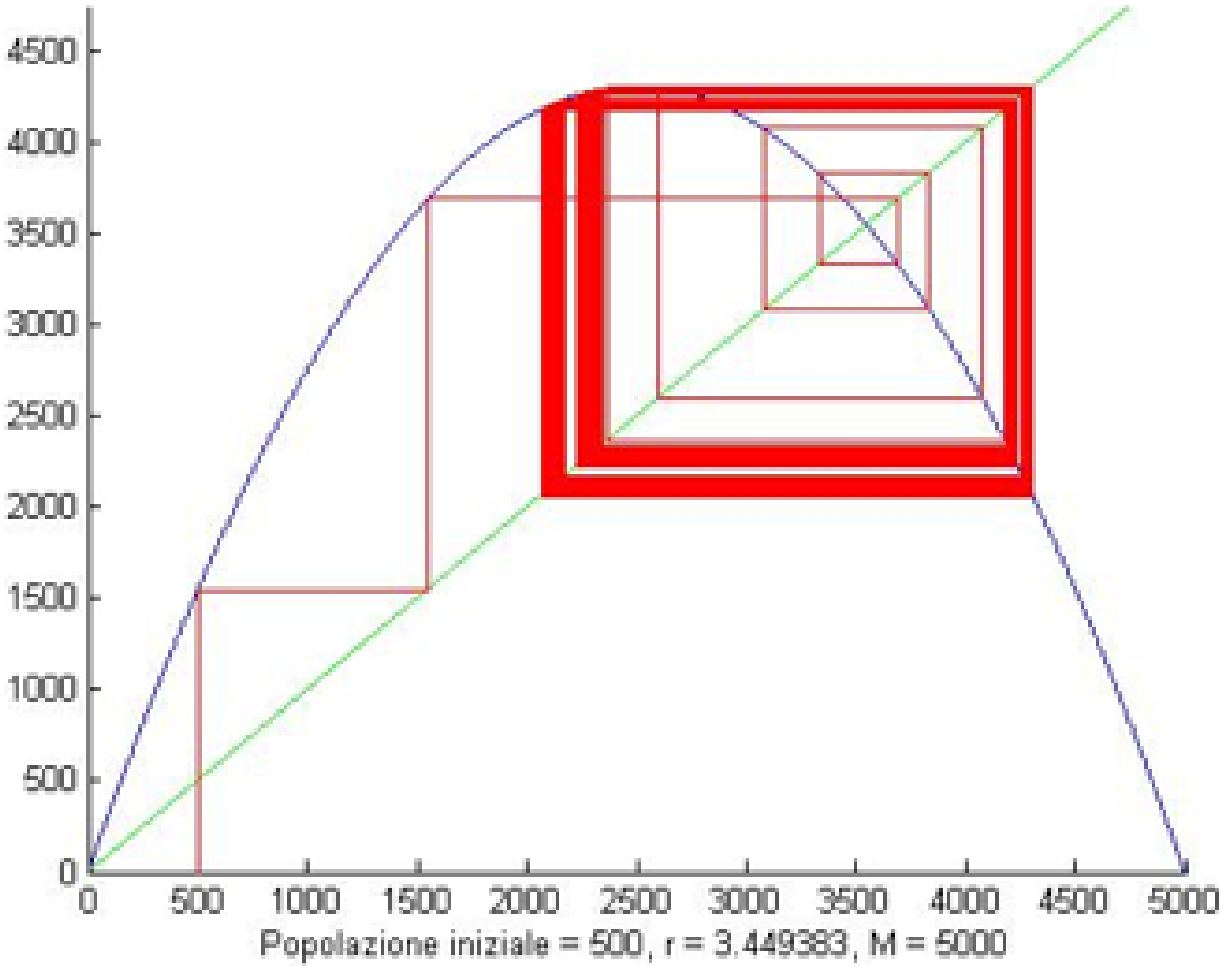
Evoluzione della Popolazione con r=3



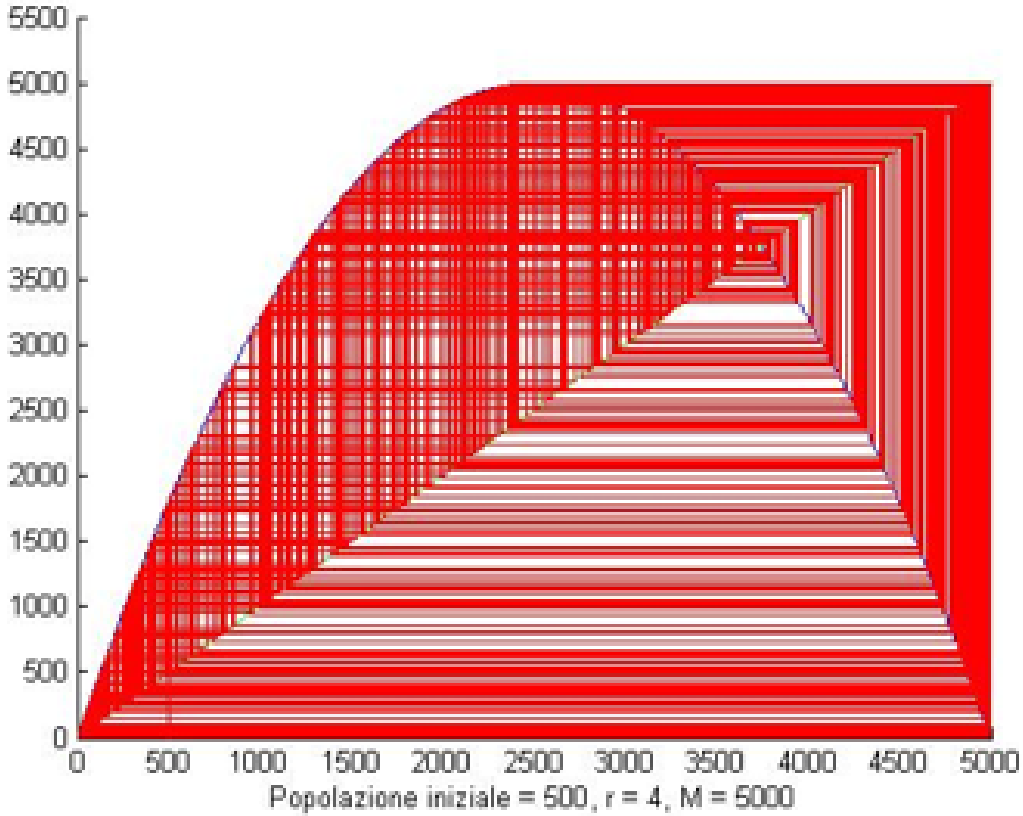
r=3 Prima Biforcazione



r=3.5 Seconda Biforcazione

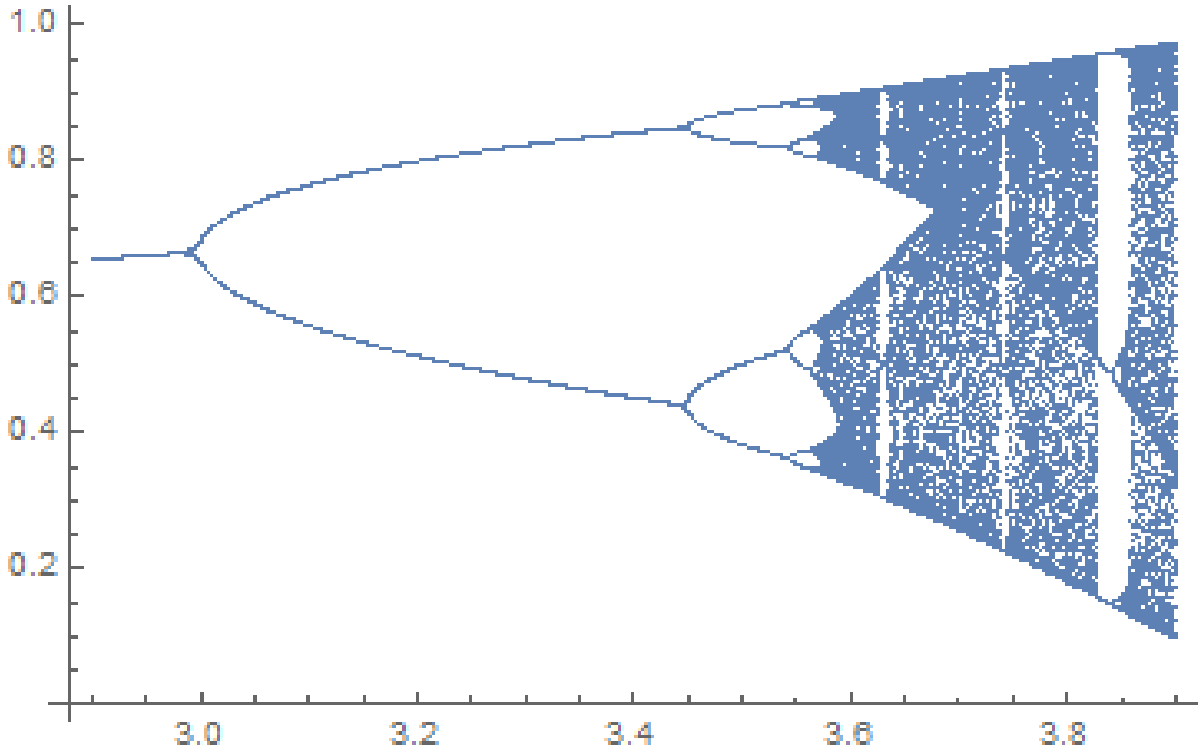


r=4 Caos Deterministico



Quando $r = 4$ abbiamo un sitema caotico, ovvero valgono le tre proprietà che abbiamo visto prima ▸ Sistema Caotico

Biforcazioni



Questo grafo rappresenta il procedere della logistica continua $f(x) = rx(1-x)$ verso il Caos, mostrando la sequenza delle biforcazioni. Si parte da $x_0 = 0.1$, si fa variare il parametro r da $r = 2.9$ a $r = 3.9$ con passo 0.001, e per ogni valore di r considerato si mappano i valori $f^{(n)}(x_0)$ per $200 \leq n \leq 300$.

Una nuova Crittografia basata sulla Logistica

Nel 1998 il fisico M.S. Baptista pubblicò un articolo dal titolo *Cryptography with chaos* ([Baptista 1998]). Questo lavoro ha conseguito un grande successo, con circa un migliaio di citazioni. Si tratta in effetti di un'idea nuova e assai interessante.
L'alfabeto possiede 256 simboli, numerati da 0 a 255.
Si tratta di un metodo di crittografia simmetrica, per cui gli utenti A, B devono condividere una chiave.
L'intervallo $J = [0, 1]$ viene suddiviso in 256 parti che, mediante una biezione σ sono associate ai simboli dell'alfabeto. Chiamiamo B_i il segmento assegnato da σ al simbolo i. Vengono scelti $x_0 \in J$ e un parametro $3.8 \leq b < 4$, e si fissa un intero N_0.
I due utenti condividono σ, x_0, b, N_0.
Hanno anche concordato (questo è noto) che utilizzeranno la funzione logistica $f(x) = bx(1 - x)$.

Cifratura del primo simbolo

Gli utenti devono anche condividere una precisione di calcolo, per esempio 16, che è quella utilizzata nei nostri esempi. Con questo intendiamo che ci sono sempre almeno 16 cifre esatte dopo la virgola.

Se A vuole inviare a B il testo $a_1, a_2, \ldots$ agisce così.

Per trasmettere a_1 A parte da x_0 e itera la funzione f $2, 3, \ldots, M$ volte, con $M > N_0$, abbastanza grande (nell'articolo originale $M = 65532$) e ottiene un insieme di interi T_1 tali che

$$\forall t \in T_1 \quad f^{(t)}(x_0) \in B_{a_1}$$

A sceglie a caso un $s_1 \in T_1$, $s_1 > N_0$ calcola $x_1 = f^{(s_1)}(x_0)$ e invia s_1 (che è la cifratura di a_1).
Si noti che, per le proprietà caotiche della logistica, partendo da un qualsiasi x_0, esistono diversi k (che variano a seconda della precisione usata) tali che $f^{(k)}(x_0) \in B_{a_1}$.

Cifratura dei simboli successivi

L'unica differenza tra la cifratura di a_1 e quella di a_2 è che, per cifrare a_2 l'utente A parte da x_1 invece che da x_0. Come sopra determina un insieme di interi T_2 tale che

$$\forall t \in T_2 \quad f^{(t)}(x_1) \in B_{a_2}$$

A sceglie un s_2 qualsiasi in T_2 e lo invia, è la cifratura di a_2. Si procede in questo modo per cifrare tutti gli altri simboli.
La tecnica di decifratura è evidente. L'utente B riceve la sequenza $s_1, s_2, \ldots$. L'utente B calcola $x_1 = f^{(s_1)}(x_0)$ Il punto x_1 appartiene ad un solo intervallino e precisamente sta in B_{a_1}. B decifra s_1 come a_1. Calcola poi $f^{(s_2)}(x_1)$ e così via.
Si noti che se nel messaggio ci sono k simboli e diciamo m_i l'ordine di T_i, il messaggio può essere cifrato in $m = \prod_{i=1}^{k} m_i$ modi diversi, *senza dover comunicare nulla a B* .

Esempio di utilizzo del cifrario di Baptista

Poniamo $b = 3.901665$, $x_0 = 0.2772643$. Fissiamo $N = 250$ e il numero di iterazioni $M = 20000$.
La biezione tra caratteri (qui prendiamo gli interi 1...256) e intervalli sia quella ovvia, determinata da $\epsilon = 1/256$. $k \leftrightarrow [(k-1)\epsilon, k\epsilon[$
Il messaggio sia *no*, corrispondente alla coppia $a_1 = 110, a_2 = 111$.
In T_1 ci sono 101 elementi. Scegliamo 11589.
Verifichiamo, per esercizio, che $s_1 = 11589$ va bene.
Calcoliamo $f^{(11589)}$. Si ha $f^{(11589)}(x_0) = 0.428386684062087 7$.
Poichè $B_{110} = [0.425781250000000 0, 0.429687500000000 0[$ concludiamo che, in effetti, $x_0 \in B_{110}$.
Per codificare a_2 poniamo $x_1 = 0.428386684062087 7$ e, partendo da x_1, iteriamo 20000 volte la f. In T_2 ci sono 75 elementi. Scegliamo 7705.
Pertanto *no* crittato diventa $11589, 7705$.
Osserviamo che *no* può essere codificato in $101 \times 75 = 7575$ modi diversi!

Un Caos più uniforme

Molti saggi sono stati scritti sul cifrario di Baptista, per attaccarlo, correggerlo e migliorarlo. Nel Maggio del 2019 è stato pubblicato un interessante articolo ([Machicao 2019]), nel quale si propone di usare una nuova tecnica per rendere più efficace la sua utilizzazione. Questa tecnica viene detta *deep-zoom*, e venne introdotta nel 2017 in ([Machicao 2017]) da due degli autori, Jeaneth Machicao e Odemir Bruno, della Università di San Paolo.

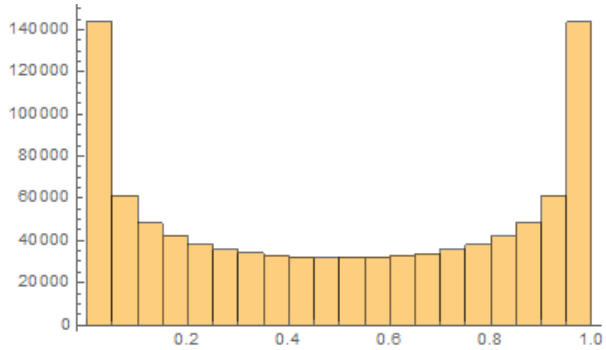
Lidea di ([Machicao 2019]) è questa:
data l'orbita di x_0 mediante la logistica $O(x_0) = (x_0, x_1, ...x_n, ...)$ la $k - orbita$ di x_0 è $O^k(x_0) = (x_0^k, x_1^k, ...x_n^k, ...)$, dove $x_t^k = 10^k x_t - \lfloor 10^k x_t \rfloor$.

Uno dei problemi che presenta la utilizzazione della logistica in crittografia è la non uniforme distribuzione di un'orbita nell'intervallo $[0, 1]$, per cui certi caratteri si raggiungono molto più sovente di altri.

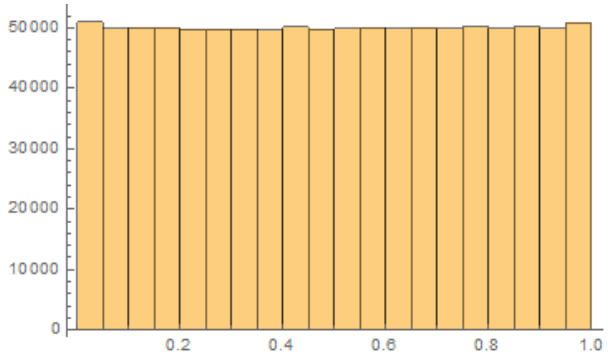
Impiegare la k-logistica risolve il problema.

Diverse Distribuzioni

Gli istogrammi si riferiscono all'orbita di 0.23456 (10^6 iterazioni), con $r = 4$



Orbita della Logistica



k-Orbita, con $k = 4$

Sequenze Pseudo Casuali dal Caos

Possiamo ricavare sequenze pseudocasuali da un sistema dinamico caotico definito sull'intervallo unitario.

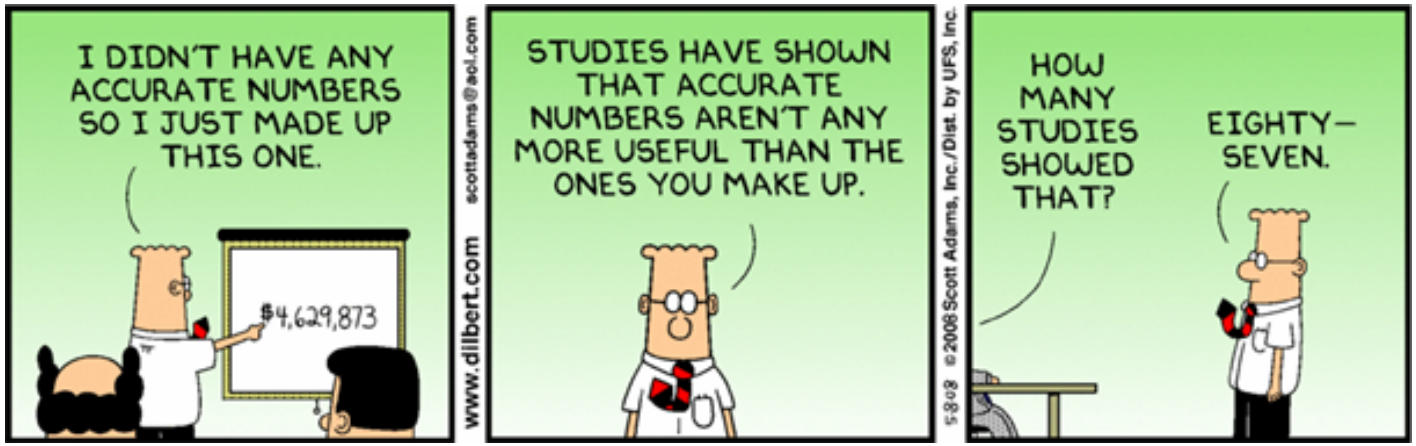
Il metodo introdotto in ([Flores 2019]), dove si usano mappe caotiche bidimensionali, può essere generalizzato a tutti i sistemi dinamici caotici.

Si calcola l'orbita di x_0 per N passi, con alta presione, diciamo di v cifre decimali, ottenendo N numeri $X_i = 0.x_{i1}x_{i2}....$
Gli X_i sono convertiti in numeri interi grandi q_i mediante
$q_i = Round(10^v X_i)$.
Ogni q_i viene scritto in binario, ottenendo una stringa di bit B_i.
Infine si costruisce una stringa pseudocasuale di bit B concatenando tutti i B_i.

Ho fatto molti esperimenti con precisione $v = 100$ e la logistica con $r = 4$.

Le successioni risultanti passano molto bene i test Ent, DieHard e NIST.
Per altri metodi si veda ([Linfeng 2016]).

Il parere di Dilbert



Numeri a caso

Numeri casuali

Bibliografia I

M.S. Baptista, *Cryptography with chaos*
Physics Letters A, Vol. 240 p. 50 – 54, 1998

Craig P. Bauer. *Secret History: The Story of Cryptology*
CRC Press 2013

Nelson F. H. Beebe, *A Bibliography of Pseudorandom Number Generation, Sampling, Selection, Distribution, and Testing*
University of Utah, Department of Mathematics, *September*2019

Steven M. Bellovin, *Frank Miller: Inventor of the One-Time Pad*
Cryptologia, Vol. 35, p. 203 – 222, 2011

L. Blum, M. Blum, e M. Shub, *A Simple Unpredictable Pseudo-Random Number Generator*
SIAM J. Comput., Vol. 15, p. 364 – 383, 1986

Bibliografia II

Umberto Cerruti, Mario Giacobini e Pierre Liardet, *Prediction of Binary Sequences by Evolving Finite State Machines*
In: Collet P., Fonlupt C., Hao JK., Lutton E., Schoenauer M. (eds) Artificial Evolution. EA 2001. Lecture Notes in Computer Science, vol 2310. Springer, Berlin, Heidelberg, 2002

Umberto Cerruti e Nadir Murru, *If the Primes are Finite, Then All of Them Divide the Number One*
American Mathematical Monthly, Vol.124, p. 969, 2017

Umberto Cerruti, *Le Successioni di Interi*
Conferenze e Seminari dell'Associazione Subalpina Mathesis 2018 – 2019, p. 127 – 154, 2019

Gregory Chaitin *Meta Math!: The Quest for Omega*
Vintage Books, New York, 2006

Bibliografia III

Robert L. Devaney, *An Introduction to Chaotic Dynamical Systems* Westview Press 2003

Fenglet Fan, Ge Wang, *Learning from Pseudo-Randomness With an Artificial Neural Network - Does God Play Pseudo-Dice?* IEEE Access, 2018

Flores-Vergara, A., García-Guerrero, E.E., Inzunza-González, E. et al., *Implementing a chaotic cryptosystem in a 64-bit embedded system by using multiple-precision arithmetic* Nonlinear Dynnamics, Vol. 96, p. 497 – 516, 2019

William Friedman, The Index of Coincidence and its Applications in Cryptography, 1919

Bibliografia IV

Oded Goldreich, *On the Impact of Cryptography on Complexity Theory*
Department of Computer Science, Weizmann Institute of Science
*May*2019

H. Haramoto, M. Matsumoto, M. Saito, *Pseudo random number generators: attention for a newly proposed generator*
arXiv:1907.03251v1 2019

David Khan, *The Codebrakers: The Story of Secret Writings*
The Macmillan Company, 1967

Ming Li, Paul Vitàni, *An Introduction to Kolmogorov Complexity and Its Applications*
Springer 2008

Bibliografia V

L. Lingfeng, M. Suoxia, *The complexity of binary sequences using logistic chaotic maps*
Complexity, Vol. 21, p. 121 – 129, 2016

Jeaneth Machicao, Odemir M. Bruno *Improving the pseudo-randomness properties of chaotic maps using deep-zoom*
Chaos Vol. 27 (2017)

Jeaneth Machicao, Marcela Alves, Murilo S. Baptista and Odemir M. Bruno, *Exploiting ergodicity of the logistic map using deep-zoom to improve security of chaos-based cryptosystems*
International Journal of Modern Physics C, Vol. 30 : 05, 2019

Bibliografia VI

M. Matsumoto, T. Nishimura *Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*
ACM Transactions on Modeling and Computer Simulation, Vol. 8, p. 3 – 30, 1988

Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System

Amedeo Perna, Marco Abrate, Stefano Barbero, Umberto Cerruti, Nadir Murru, Method for the management of virtual objects corresponding to real objects, corresponding system and computer program product

Claude Shannon, *Communication Theory of Secrecy Systems*
Bell System Technical Journal, Vol. 28, p. 656 – 715, 1949

Tim Smith, *Prediction of infinite words with automata*
Theory of Computing Systems, Vol. 62, p. 653 – 681, 2018

Indice I

Indice II

Indice III

Indice IV