Complexidade de Kolmogorov e Caracterização da Hierarquia de Classes de Linguagens Formais – Uma Introdução*

Carlos A. P. Campani[1] e Paulo Blauth Menezes[2]

[1] IFM/ UFPel - CP 354, 96010-900, Pelotas/RS, Brasil
campani@ufpel.tche.br
[2] II/ UFRGS - CP 15064, 91501-970, Porto Alegre/RS, Brasil
blauth@inf.ufrgs.br

Resumo Este trabalho é uma revisão bibliográfica selecionada (limitada ao espaço disponível e concentrando-se em aplicações em Ciência da Computação) da área de complexidade de Kolmogorov. O principal objetivo do trabalho é apresentar, de uma forma concisa e didática, a definição da complexidade de Kolmogorov e suas principais propriedades. Ao final é mostrada uma aplicação selecionada, que constitui-se em uma nova visão da hierarquia de Chomsky, com o objetivo de ilustrar as potencialidades de aplicação da área em Ciência da Computação. A vantagem dos lemas de bombeamento apresentados sobre os da teoria tradicional são sua maior expressividade e flexibilidade. Além disto, são indicadas outras aplicações em Ciência da Computação, apresentadas em outros artigos, incluindo: Inteligência Artificial; Complexidade Computacional; e Engenharia de Software. Apesar da área ser reconhecida internacionalmente, no Brasil ela é relativamente desconhecida. Assim, a importância deste trabalho reside no fato de ele ser a primeira visão geral da área de complexidade de Kolmogorov publicada no Brasil ou mesmo em português, o que, espera-se, propiciará sua maior divulgação.

1 Introdução

Este texto é uma revisão bibliográfica selecionada da área de complexidade de Kolmogorov. A principal contribuição dos autores é a apresentação concisa e didática de uma visão geral da área em português, assim como de uma aplicação desta que ilustra o seu potencial em Ciência da Computação, objetivando sua divulgação no Brasil.

A complexidade de Kolmogorov, proposta por Kolmogorov como uma teoria algorítmica da aleatoriedade, é uma teoria profunda e sofisticada que trata da quantidade de informação de objetos individuais, medida através do tamanho de sua descrição algorítmica. Ela é uma noção moderna de aleatoriedade, e refere-se

* Este trabalho é parcialmente financiado por: CNPq (Projetos HoVer-CAM, MEFIA), UFRGS (Projeto Hyper-Automaton), FAPERGS (Projeto QaP-For) e CAPES/UFPel.

a um conceito pontual de aleatoriedade, ao invés de uma aleatoriedade média como o faz a teoria das probabilidades.

Selecionamos uma aplicação da teoria para ser apresentada ao final do artigo, com o objetivo de ilustrar as possíveis aplicações do método da incompressividade. A aplicação apresentada constitui-se em lemas de bombeamento para linguagens regulares e livres de contexto que são mais expressivos e flexíveis que as técnicas tradicionais. Três assuntos tradicionais da área, probabilidade algorítmica, complexidade de prefixo e seqüências aleatórias de Martin-Löf, sem os quais uma revisão bibliográfica de complexidade de Kolmogorov não seria completa, serão tratados em um outro artigo devido ao espaço disponível.

2 Complexidade de Kolmogorov

2.1 Complexidade Básica

Assumimos que existe um *método de especificação* f que associa no máximo um objeto x com uma descrição y [9]. Seja A o conjunto dos objetos e seja B o conjunto das descrições, então $f(y) = x$, para $x \in A$ e $y \in B$. Chamamos f^{-1} de *método de codificação*. Assumimos alguma enumeração padrão dos objetos $x \in A$ por números naturais $n(x)$. Podemos entender tal método de especificação como uma função parcial sobre o conjunto dos naturais definida como $f(p) = n(x)$, onde p representa uma descrição de x com relação a f. Por enquanto não faremos nenhuma restrição sobre a classe dos métodos de especificação e assumiremos que f pode ser qualquer função parcial arbitrariamente escolhida.

Representamos as descrições e objetos como seqüências de zeros e uns (strings binárias). Existe uma enumeração das strings binárias que segue a ordem lexicográfica,

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \\ \Lambda & 0 & 1 & 00 & 01 & 10 & 11 & 000 & 001 & \cdots \end{pmatrix}.$$

O número natural i representa a i-ésima string binária (a string s_i). Então, $|s_i| = \lfloor \log(i+1) \rfloor$, onde $|.|$ denota o tamanho em bits de uma string binária, e $\lfloor . \rfloor$ denota o maior número inteiro menor ou igual a um número dado. De agora em diante não faremos distinção entre strings binárias e números naturais. Assim, podemos definir o "tamanho" de um número natural, reescrevendo s_i como i. Além disto, podemos definir funções sobre o conjunto dos naturais $\phi : \mathbb{N} \to \mathbb{N}$ como funções sobre strings binárias $\phi : \{0,1\}^* \to \{0,1\}^*$ e vice-versa.

Definição 1. *Seja $x \in A$ um objeto qualquer. A complexidade do objeto x com respeito a um método de especificação $f : \mathbb{N} \to \mathbb{N}$ é definida como $C_f(x) = \min_{f(p)=n(x)} |p|$, onde* min *significa o valor mínimo de um conjunto. Se não existe uma descrição dizemos, por definição, que $C_f(x) = \infty$.*

Ou seja, a complexidade de x com respeito a um método de especificação f é o tamanho em bits de sua menor descrição.

Considere um conjunto de métodos de especificação distintos $f_0, f_1, f_2, \ldots, f_{r-1}$ que especificam objetos de A. É fácil construir um novo método f, não necessariamente pertencente a este conjunto, que atribui aos elementos de A uma complexidade que excede apenas por uma constante c a complexidade expressa por todos os outros métodos do conjunto. Nós dizemos que o método f *minora* (ou *minora aditivamente*) um método g se existe uma constante $c > 0$ tal que $C_f(x) \leq C_g(x) + c$, com $c \approx \log r$, pois precisamos apenas de $\log r$ bits para especificar as funções do conjunto.

Definição 2. *Seja $\mathcal{F}$ uma subclasse das funções parciais sobre o conjunto dos naturais. Uma função f é* universal *(ou* assintoticamente ótima*) para $\mathcal{F}$ se ela pertencer a $\mathcal{F}$ e para toda função $g \in \mathcal{F}$, existe uma constante $c_{f,g} > 0$ que depende apenas de f e g, tal que para todo x, $C_f(x) \leq C_g(x) + c_{f,g}$.*

A existência de um elemento universal no conjunto dos métodos de especificação permite que este seja usado como método de referência para a definição da complexidade, tornando a medida de complexidade independente do método de especificação escolhido, a não ser por uma constante. Permite assim que a medida da complexidade seja um atributo intrínseco do objeto, o que a torna um conceito objetivo e útil [1].

Definição 3. *Dois métodos f e g são* equivalentes *se para todo x,* $\text{abs}(C_f(x) - C_g(x)) \leq c_{f,g}$*, onde $c_{f,g}$ não depende de x.* abs(.) *denota o valor absoluto de um número.*

Teorema 1. *A classe das funções parciais não possui um elemento universal.*

Prova: Suponha que f é um elemento universal do conjunto das funções parciais. Tome uma seqüência infinita $p_1, p_2, p_3, \ldots$ de descrições, tal que $p_1 < p_2 < p_3 < \cdots$ e $f(p_i) = n(x_i)$, para $i \geq 1$, com p_i mínimo. Selecione uma subseqüência $q_1, q_2, q_3, \ldots$ da seqüência original, tal que $|p_i| < |q_i|/2$. Defina outra função g tal que $g(p_i) = f(q_i)$, para $i \geq 1$. Então, como p_i descreve, usando g, o mesmo objeto que q_i descreve usando f, $C_g(x) \leq C_f(x)/2$, o que é uma contradição com nossa hipótese de que f é um elemento universal.

Portanto, não podemos usar as funções parciais como método de especificação. A complexidade de Kolmogorov, ao contrário, usa como método de especificação as funções parciais recursivas (procedimentos efetivos). Isto permite que exista um elemento universal no conjunto dos métodos de especificação, já que existe uma função parcial recursiva universal. As funções parciais recursivas foram propostas como uma formalização do conceito de "computável" (em um sentido vago e intuitivo). Uma formalização alternativa foi proposta por Turing, baseada em uma máquina de manipulação simbólica chamada *máquina de Turing* [4]. Prova-se que os formalismos máquina de Turing e funções parciais recursivas são equivalentes, o que permitiu a Turing e Church enunciarem a seguinte Tese.

Tese de Turing-Church A classe das funções algoritmicamente computáveis (em um sentido intuitivo) coincide com a classe das funções parciais recursivas (ou das funções computadas pela máquina de Turing).

Teorema 2. *O conjunto dos programas é enumerável.*

Prova: Basta apresentar uma bijeção entre o conjunto dos programas e o conjunto dos naturais (veja discussão em T. Divério & P. Menezes [4], seção 3.1, página 68, a qual deve ser adaptada para prover a bijeção).

Se associarmos à cada máquina de Turing um programa que representa o seu controle finito, chamado "programa de hardware", podemos concluir que o conjunto das máquinas também é enumerável.

Corolário 1. *O conjunto das máquinas de Turing é enumerável.*

Assim, podemos definir uma enumeração padrão das máquinas de Turing, $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots$ Seja $\phi_1, \phi_2, \phi_3, \ldots$ uma enumeração das funções parciais recursivas. Então, existe uma função ϕ_0, chamada *função parcial recursiva universal*, tal que, para todo i, $\phi_0(i, x) = \phi_i(x)$ [3,9].

Consideremos uma máquina de Turing $\mathcal{M}$ que a partir de uma string binária p e um número natural y computa a saída x, $\mathcal{M}_{p,y} = x$. Nós dizemos que $\mathcal{M}$ interpreta p como uma descrição de x na presença da informação lateral y. Em outras palavras, p é um $\mathcal{M}$-programa que transforma y em x.

Devemos mostrar como construir um computador concreto para tratar com esta definição (veja a Figura 1). A máquina trabalha sobre strings binárias e possui três fitas. A primeira é chamada de *fita de entrada* (ou *fita de programa*) e é uma fita limitada, somente de leitura e unidirecional. A segunda é chamada de *fita de saída* e é uma fita unidirecional somente de escrita. A terceira fita é chamada *fita de trabalho* e é uma fita bidirecional, infinita e de leitura e escrita. Inicialmente a fita de entrada armazena a descrição (entrada ou programa) e a fita de trabalho armazena a informação lateral de forma literal. Todos os outros campos da fita de trabalho estão preenchidos com brancos. A fita de saída está vazia. A máquina pode ler ou escrever um símbolo (0 ou 1), mover o cabeçote para a esquerda ou a direita uma posição ou apagar símbolos da fita de trabalho, ou então escrever algum símbolo na fita de saída. Depois de uma quantidade finita de tempo a máquina eventualmente pára, tendo lido toda a string da fita de entrada (o programa), com a saída armazenada na fita de saída. Não importa o tempo de execução do programa (isto só é importante na complexidade de tempo de execução).

Caso a saída desejada seja uma string de tamanho infinito, a máquina fica para sempre imprimindo bit após bit na fita de saída.

Definição 4. *A* complexidade condicional $C_{\mathcal{M}}(x|y)$ *de um número x com respeito a um número y é o tamanho da menor descrição p tal que $\mathcal{M}_{p,y} = x$, $C_{\mathcal{M}}(x|y) = \min_{\mathcal{M}_{p,y}=x} |p|$. Se não existe uma descrição p de x então dizemos, por definição, que $C_{\mathcal{M}}(x|y) = \infty$.*

À primeira vista parece que a complexidade condicional $C_{\mathcal{M}}(.|.)$ depende da máquina $\mathcal{M}$. Kolmogorov e Solomonoff observaram que, ao contrário, depende muito pouco porque existem máquinas de Turing universais, capazes de simular qualquer outra máquina de Turing cuja descrição é fornecida [3,6,7].

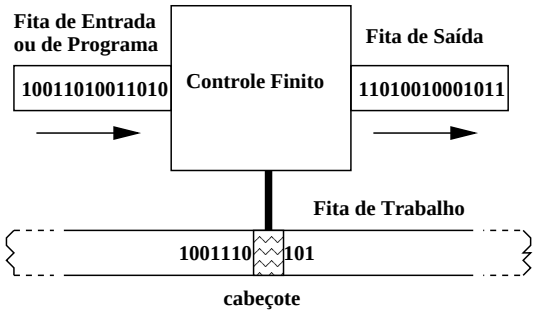

Figura 1. Computador concreto

Nós codificaremos as máquinas da enumeração mencionada no Corolário 1 como $\overbrace{111\cdots1}^{i \text{ vezes}}0p = 1^i0p$, significando que a máquina universal espera encontrar concatenado à esquerda da descrição p uma descrição de qual máquina irá simular. Ou seja, a máquina universal $\mathcal{U}$ irá simular a execução do programa p em $\mathcal{M}_i$, a i-ésima máquina da enumeração padrão.

Teorema 3. *(Teorema da Invariância) Existe uma máquina $\mathcal{U}$, chamada* universal, *tal que para qualquer máquina $\mathcal{M}$ e números x e y, e para algum $c > 0$, $C_{\mathcal{U}}(x|y) \leq C_{\mathcal{M}}(x|y) + c$ e c depende apenas de $\mathcal{M}$.*

Prova: Para $C_{\mathcal{M}}(x|y) = \infty$ a desigualdade é trivialmente verdadeira. Podemos mostrar uma máquina universal simulando outra máquina qualquer. Por exemplo, considere a máquina universal $\mathcal{U}$ tal que $\mathcal{U}_{1^i0p,y} = \mathcal{M}_{i_{p,y}}$, onde $\mathcal{M}_i$ é a i-ésima máquina na enumeração das máquinas. Suponha que $\mathcal{M}$ é a n-ésima máquina na enumeração, ou seja, $\mathcal{M}_n = \mathcal{M}$. Logo, $C_{\mathcal{M}}(x|y) = \min_{\mathcal{M}_{p,y}=x} |p|$ e $C_{\mathcal{U}}(x|y) = \min_{\mathcal{U}_{1^n0p,y}=x} |1^n0p| = \min_{\mathcal{U}_{1^n0p,y}=x} |p| + n + 1 = C_{\mathcal{M}}(x|y) + n + 1$. Ou seja, o limite superior da complexidade expressa em $\mathcal{U}$ é $C_{\mathcal{M}}(x|y) + n + 1$ e eventualmente existe um p' tal que $\mathcal{U}_{p',y} = x$ e $|p'| < |p| + n + 1$. Assim, $C_{\mathcal{U}}(x|y) \leq C_{\mathcal{M}}(x|y) + n + 1$. Tomando $c = n + 1$ prova-se o Teorema, com c dependendo apenas de $\mathcal{M}$.

Podemos interpretar a desigualdade $C_{\mathcal{U}}(x|y) \leq C_{\mathcal{M}}(x|y) + c$ de duas formas diferentes. Interpretando a desigualdade da esquerda para a direita, obtemos a idéia utilizada na prova do Teorema 3, significando que o limite superior da complexidade expressa em $\mathcal{U}$ é a complexidade expressa em $\mathcal{M}$. Por sua vez, interpretando a desigualdade da direita para a esquerda, obtemos o significado principal do Teorema, indicando que nenhum método pode ser melhor, a não ser por uma constante, que o método universal (a complexidade expressa em $\mathcal{U}$ é o limite inferior da complexidade expressa usando-se qualquer outro método), dando um carater invariante e absoluto a esta medida de complexidade.

Observe que a constante c do Teorema 3, embora possa ser grande, é assintoticamente desprezável, pois não depende de x [6,7]. Além disto, por causa da

constante c, não necessariamente a complexidade expressa em $\mathcal{U}$ é menor que a expressa em $\mathcal{M}$. Mas podemos afirmar que, *assintoticamente*, a complexidade expressa em $\mathcal{U}$ não é maior que a expressa em $\mathcal{M}$.

Restringir as descrições às efetivas (computáveis) permite que exista um método de especificação universal (invariante ou assintoticamente ótimo) que obtém uma descrição mínima com respeito a todos os outros métodos (a diferença entre as complexidades expressas usando-se diferentes métodos universais é limitada por uma constante, veja Corolário 2). Como conseqüência, a complexidade de um objeto é um atributo intrínseco do objeto independente de um método particular de especificação [9]. Em [1] é feita uma discussão bem completa sobre a boa fundamentação, objetividade e utilidade desta definição de complexidade.

Fixando uma máquina universal $\mathcal{U}$, chamada *máquina de referência*, e não fornecendo nenhuma informação lateral, podemos definir a *complexidade incondicional* como $C_{\mathcal{U}}(x|\Lambda) = C(x|\Lambda) = C(x)$. $C(x|y)$ representa intuitivamente a quantidade de informação que é necessário adicionar à informação contida em y para obter x.

Corolário 2. *Sejam $\mathcal{U}$ e $\mathcal{V}$ duas máquinas de Turing universais. Então, para todo x e y,* $\text{abs}(C_{\mathcal{U}}(x|y) - C_{\mathcal{V}}(x|y)) \leq c$, *onde c depende apenas de $\mathcal{U}$ e $\mathcal{V}$.*

Teorema 4. *Existe uma constante $c > 0$ tal que $C(x) \leq |x| + c$ para toda string binária x.*

Prova: Existe uma máquina $\mathcal{M}$ tal que $\mathcal{M}_p = p$ para todos os programas p. Então, pelo Teorema 3 (Teorema da Invariância), para toda string binária x, $C(x) \leq C_{\mathcal{M}}(x) + c = |x| + c$.

Isto é verdade porque existe uma máquina que executa a cópia do próprio programa na saída.

Teorema 5. *Existe uma constante $c > 0$ tal que, para todo x e y, $C(x|y) \leq C(x) + c$.*

Prova: Construa uma máquina $\mathcal{M}$ que para todo y e z computa x a partir da entrada (z, y) se e somente se a máquina universal $\mathcal{U}$ computa a saída x com a entrada (z, Λ). Então, $C_{\mathcal{M}}(x|y) = C(x)$. Pelo Teorema da Invariância, $C(x|y) \leq C_{\mathcal{M}}(x|y) + c = C(x) + c$.

Teorema 6. *Para qualquer função computável f existe uma constante $c > 0$ tal que $C(f(x)) \leq C(x) + c$, para todo x em que $f(x)$ é definida.*

Prova: Construa uma máquina universal $\mathcal{M}$ que computa x simulando $\mathcal{U}$ e depois usa x e computa $f(x)$, então $C_{\mathcal{M}}(f(x)) = C_{\mathcal{U}}(x)$ e $C(f(x)) \leq C_{\mathcal{M}}(f(x)) + c = C(x) + c$.

2.2 Seqüências Aleatórias e Incompressividade

O propósito original da complexidade de Kolmogorov era definir seqüências aleatórias formalmente, embasando uma teoria matemática das probabilidades. Naquele tempo, apenas evidências empíricas, advindas de jogos de azar e salões de

cassinos, apoiavam a teoria do limite da freqüência relativa [12]. A teoria do limite da freqüência relativa afirma que, em uma seqüência longa de tentativas, as freqüências relativas das ocorrências de sucesso ou fracasso em um experimento devem convergir a um limite, $\lim_{n\to\infty} \lambda(n)/n = p$, onde λ conta o número de ocorrências de sucesso no experimento e p é o limite da freqüência. Por exemplo, com uma moeda honesta, em uma seqüência *suficientemente longa*, $p = 1/2$.

A dificuldade com esta abordagem é definir o que é "uma seqüência longa", pois qualquer seqüência de tentativas é, obviamente, limitada em tamanho. Kolmogorov propôs uma nova abordagem, definindo primeiro seqüências aleatórias finitas (aleatoriedade pontual), via incompressividade, e depois seqüências aleatórias infinitas.

String binária aleatória Uma string binária é dita *aleatória* se sua complexidade é aproximadamente igual ao tamanho da string.

Assim, definimos as strings simples como sendo aquelas que são *regulares* ou *compressíveis*, e as strings aleatórias ou complexas como sendo aquelas que possuem irregularidade (são *incompressíveis*) [2].

Esta definição de Kolmogorov foi mais tarde aprimorada por Martin-Löf [9,12]. Demonstra-se que a definição de Martin-Löf garante que toda string aleatória que a satisfaça, com certeza possui todas as propriedades de aleatoriedade que podem ser testadas efetivamente (por um programa de computador) produzida por uma fonte aleatória (estocástica) na média. Observe que é necessário supor que exista um algoritmo de computador capaz de testar a string por deficiência de aleatoriedade.

Suponha que você possui uma definição própria de "teste estatístico para evidenciar não-aleatoriedade". Podemos dizer que toda string que respeita a sua definição com certeza respeitará a definição de Martin-Löf. A prova de Martin-Löf baseia-se na existência de um *teste de aleatoriedade universal* [12].

Sejam as seguintes strings binárias:

```
11111111111111111111
01001101011000110101
```

Nossa intuição nos diz que a primeira string não pode ser aleatória pois tem pequena probabilidade de ocorrer em uma seqüência de lançamentos de uma moeda honesta, enquanto que a segunda parece ser aleatória. No entanto, ambas as seqüências tem, segundo a teoria das probabilidades, a mesma probabilidade de ocorrer, considerada a distribuição uniforme (medida de Lebesgue), o que parece ser um paradoxo.

A primeira string pode ser computada pelo seguinte programa:

```
FOR i := 1 TO 20 PRINT 1
```

Strings deste tipo, com tamanho n, podem genericamente ser computadas pelo seguinte programa:

```
FOR i := 1 TO n PRINT 1
```

cujo tamanho (em número de bits, pois supomos programas binários) é o tamanho da representação em bits de n mais c bits para a rotina que imprime. Assim, o tamanho do programa é $O(\log n)$.

Já a segunda string parece ter sido criada pelo lançamento de uma moeda. Por não possuir uma regra simples para a sua formação (ou geração), não poderia ser computada de uma forma compacta. Seria necessária a escrita da própria string na saída da máquina pelo programa:

```
PRINT 01001101011000110101
```

Logo, o tamanho do programa para computar strings deste tipo seria $n + c$, ou $O(n)$. Ou seja, aproximadamente igual ao tamanho da própria string.

Definição 5. *Para uma constante $c > 0$, nós dizemos que a string x é c-incompressível se $C(x) \geq |x| - c$.*

Na Definição 5, a constante c desempenha o papel de "taxa de compressão".

Definição 6. *Se uma string x é c-incompressível para um valor $c > 0$, então nós dizemos que x é* incompressível.

É facil provar a existência de strings binárias incompressíveis no sentido da Definição 6. Na prova do Teorema 7 supomos a distribuição uniforme (medida de Lebesgue).

Teorema 7. *Existe pelo menos uma string incompressível de tamanho menor ou igual a n.*

Prova: Segue do fato que existem $2^n - 1$ programas binários de tamanho menor que n, porque $\sum_{i=0}^{n-1} 2^i = 2^n - 1$, mas existem muitas mais strings binárias de tamanho menor ou igual a n $(2^{n+1} - 1)$.

Isto se deve ao fato de existirem menos descrições curtas que longas. Então, existem strings que não tem uma descrição significativamente menor que seu próprio tamanho.

Podemos determinar quantas strings de tamanho n são c-incompressíveis. Sabemos que do total de 2^n strings de tamanho n temos $2^{n-c} - 1$ que não são c-incompressíveis. Portanto, o número de strings de tamanho n que são c-incompressíveis é $2^n - 2^{n-c} + 1$. A fração de strings de comprimento n que são c-incompressíveis no total de 2^n strings é, para um n grande o suficiente, $1 - 2^{-c}$. Sabemos que para um c, $1 < c < n$, grande o suficiente, $1 - 2^{-c}$ tende a 1. Logo, a maioria das strings são incompressíveis.

Podemos agora provar o seguinte Teorema, cuja aplicação é muito importante e se constitui no chamado *método da incompressividade.*

Teorema 8. *(Teorema da Incompressividade) Para um número c, dado um y fixo e para todo conjunto A de cardinalidade $\overline{\overline{A}}$ temos ao menos $\overline{\overline{A}}(1 - 2^{-c}) + 1$ elementos $x \in A$ com $C(x|y) \geq \log \overline{\overline{A}} - c$.*

Prova: O número de programas de tamanho menor que $\log \overline{\overline{A}} - c$ *é*

$$\sum_{i=0}^{\log \overline{\overline{A}}-c-1} 2^i = 2^{\log \overline{\overline{A}}-c} - 1 = \overline{\overline{A}}2^{-c} - 1 .$$

Precisamos de $\overline{\overline{A}}$ *programas para descrever cada um dos* $\overline{\overline{A}}$ *elementos de* A*. Logo, existem* $\overline{\overline{A}} - \overline{\overline{A}}2^{-c} + 1$ *elementos em* A *que não tem um programa de comprimento menor que* $\log \overline{\overline{A}} - c$*.*

Os objetos $x \in A$ que satisfazem $C(x|y) \geq \log \overline{\overline{A}} - c$ são chamados de (A, c)-*aleatórios*.

Podemos provar que não é possível determinar se uma string é incompressível, pois este problema é indecidível. Suponha que o conjunto das strings incompressíveis é enumerável. Seja $\mathcal{M}$ uma máquina de Turing que computa a primeira string x_0 com complexidade maior que n. Então, $C(x_0) > n$ e $C(x_0) \leq C_{\mathcal{M}}(x_0) + c \leq \log n + c$. Assim, $C(x_0) > n$ e $C(x_0) \leq \log n + c$, para qualquer n arbitrário, que é uma contradição. É interessante o fato da maioria das strings serem incompressíveis, e no entanto não ser possível determiná-las.

Não é muito difícil provar que a menor descrição de um objeto é uma string incompressível. Se p é a menor descrição de x então $C(x) = |p|$. Se p é incompressível, então $C(p) \geq |p| - c$. Suponha que p não é incompressível. Portanto, existe uma descrição de p que é significativamente menor que p. Logo, existe um q que é a menor descrição de p tal que $|q| < |p| - c$. Defina uma máquina universal $\mathcal{V}$ tal que $\mathcal{V}$ trabalhe exatamente como a máquina de referência $\mathcal{U}$, exceto que $\mathcal{V}$ primeiro simule $\mathcal{U}$ sobre a sua entrada obtendo uma saída e depois simule novamente $\mathcal{U}$ usando a saída como nova entrada. $\mathcal{V}$ pertence à enumeração de máquinas. Assim, podemos supor que $\mathcal{V} = \mathcal{M}_i$. Disto resulta que $\mathcal{U}_{1^i 0 q} = \mathcal{U}_p = x$ e $C(x) \leq |q| + i + 1$. Sabendo que $|q| < |p| - c$ então $C(x) < |p| + i + 1 - c$, o que contradiz que $C(x) = |p|$ para $c \geq i + 1$. Logo, não existe uma descrição menor que p e p é incompressível.

Definimos um *código livre de prefixo*, uma codificação de strings binárias que possui a notável propriedade de que strings codificadas desta forma, chamadas *strings autodelimitadas*, conhecem seu próprio tamanho. Assim, se codificamos a string x como uma string autodelimitada $\overline{x}$, podemos facilmente recuperar da concatenação $\overline{x}y$ ambas as strings x e y [9].

Nós podemos codificar uma string através do seguinte esquema: No início da string x colocamos o seu tamanho codificado como uma seqüência de uns, $1^{|x|}$, seguido do símbolo de terminação 0, $E_1(x) = \overline{x} = \overbrace{111\cdots1}^{|x| \text{ vezes}}0x = 1^{|x|}0x$, com tamanho $|E_1(x)| = 2|x| + 1$. Decodificamos x contando os uns antes do primeiro zero e recuperando x usando seu tamanho.

Podemos melhorar o código repetindo este procedimento para o tamanho de x, obtendo um código mais compacto, $E_2(x) = \overline{|x|}x = 1^{||x||}0|x|x$ com tamanho $|E_2(x)| = |x| + 2\log|x| + 1$.

Seja a string $x = uvw$ de tamanho n, onde $n = |u| + |v| + |w|$. Podemos descrever a string x através de um programa p que reconstrói v e pela string uw tomada literalmente. Precisamos ainda da informação de como separar estas três partes p, u e w. Para isto usamos codificação livre de prefixo, antecedendo cada parte com o seu tamanho. Assim, $q = \overline{|p|}p\overline{|u|}uw$ é uma descrição de x. Existe uma máquina $\mathcal{M}$ que, começando pelo extremo esquerdo de q, primeiro determina $|p|$ e computa v de p. A seguir determina $|u|$ e usa esta informação para delimitar u e w. Finalmente, $\mathcal{M}$ monta x a partir das três partes u, v e w. Assim, $\mathcal{M}_q = x$. Logo, pelo Teorema da Invariância e pelo Teorema 4, $C(x) = C(\mathcal{M}_q) \leq C_{\mathcal{M}}(q) + O(1) \leq |q| + O(1)$, portanto, $C(x) \leq \left|\overline{|p|}p\overline{|u|}uw\right| + O(1) = \left|\overline{|p|}p\right| + \left|\overline{|u|}u\right| + |w| + O(1)$.

Então, como sabemos que $\left|\overline{|p|}p\right| = |p| + 2|l| + 1$ com $l = |p|$ e $|p| = C(v)$, $C(x) \leq C(v) + 2\,|C(v)| + 1 + |u| + 2\big||u|\big| + 1 + |w| + O(1)$ e $C(x) \leq C(v) + 2\,|C(v)| + 2\big||u|\big| + |u| + |w| + O(1)$.

Como $n - |v| = |u| + |w|$, $C(x) \leq C(v) + 2\,|C(v)| + 2\big||u|\big| + n - |v| + O(1) \leq C(v) + 2\,|C(v)| + 2|n| + n - |v| + O(1)$. Sabendo que $|C(v)| \leq \log n$ e $|n| = \log n$, então $C(x) \leq C(v) + n - |v| + 4\log n + O(1)$.

Para strings c-incompressíveis x com $C(x) \geq n - c$, obtemos $C(v) + n - |v| + 4\log n + O(1) \geq n - c$, e simplificando, $C(v) \geq |v| - O(\log n)$.

Isto significa que v é incompressível a menos de um termo logarítmico.

2.3 Algumas Propriedades da Complexidade Básica

Nesta seção são apresentadas algumas propriedades da complexidade selecionadas pelos autores, embora nem todas elas sejam usadas na aplicação que é ilustrada ao final deste trabalho.

Seja $\langle ., . \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ uma função de emparelhamento efetiva. Assim, podemos definir $C(\langle x, y \rangle) = C(x, y)$ como a complexidade de x e y simultaneamente. Ou seja, $C(x, y)$ é o tamanho da menor descrição que computa em $\mathcal{U}$ ambas as strings x e y. Seria a complexidade C subaditiva? Ou de outra forma, será que $C(x, y) \leq C(x) + C(y) + O(1)$?

Sendo p a menor descrição de x e q a menor descrição de y, precisamos de uma descrição pq ou qp que permita à máquina $\mathcal{U}$ computar x e y e depois compor ambas as strings. O problema se refere a encontrar a posição da descrição q de y na concatenação pq ou a posição da descrição p de x na concatenação qp. Para isto devemos anteceder a primeira descrição com o seu tamanho, para que a máquina possa encontrar a segunda descrição, usando uma codificação livre de prefixo. Sabemos que $\overline{x} = 1^{|x|}0x$ e $\left|\overline{|x|}x\right| = |x| + 2\big||x|\big| + 1$.

Seja p a menor descrição de x e q a menor descrição de y, então $\overline{|p|}pq$ ou $\overline{|q|}qp$ é a menor descrição de (x, y). Assim, $\left|\overline{|p|}pq\right| = |p| + 2\big||p|\big| + 1 + |q| = |p| + |q| + 2\log|p| + 1$ e $\left|\overline{|q|}qp\right| = |q| + 2\big||q|\big| + 1 + |p| = |p| + |q| + 2\log|q| + 1$.

Logo, $C(x,y) \leq C(x) + C(y) + 2\log(\min(C(x), C(y))) + O(1)$. Não podemos eliminar este termo logarítmico, a não ser que entremos com o tamanho de um dos programas no condicional, $C(x,y|C(x)) \leq C(x) + C(y) + O(1)$. Podemos fazer isto pois, se conhecemos previamente $C(x)$, então podemos encontrar q na descrição x^*q, já que $|x^*| = C(x)$, onde x^* denota a menor descrição de x.

Como não podemos fazer desaparecer este termo logarítmico que depende de x e y, concluimos que C não goza da propriedade subaditiva. Assim, $C(x,y) \leq C(x) + C(y) + \Delta$ e Δ cresce ilimitadamente com x e y.

Teorema 9. *Existe uma constante c tal que, para qualquer x e y, $C(x,y) \geq C(x) + C(y) + \log n - c$, onde $n = |x| + |y|$.*

Prova: É necessário primeiro provar o seguinte Lema.

Lema 1. *Existem $(n+1)2^n$ pares ordenados (x,y) de strings cuja soma dos tamanhos é n.*

Prova: Por meio de um argumento simples de contagem sabemos que o número total de pares (x,y) com $|x|+|y| = n$ é $2^{0+n}+2^{1+(n-1)}+2^{2+(n-2)}+\cdots+2^{(n-1)+1}+2^{n+0}$, onde cada um dos termos da soma resulta em 2^n e temos $n+1$ termos, portanto a soma de todos os termos resulta em $(n+1)2^n$.

Sabemos que existe pelo menos um par (x,y) que é 1-incompressível. Logo, tomando o conjunto de pares (x,y) como A e usando a versão incondicional do Teorema 8, já que sabemos pelo Teorema 5 que $C(x|y) \leq C(x) + c$, sua complexidade é $C(x,y) > C(\langle x,y\rangle | A) \geq \log \overline{\overline{A}} - 1$. Como o conjunto de pares (x,y) é o conjunto A e sabemos que $\overline{\overline{A}} = (n+1)2^n$, obtemos $C(x,y) \geq n + \log(n+1) - 1 \geq n + \log n - 1$.

Pelo Teorema 4 sabemos que, para alguma constante c, $C(x) + C(y) \leq |x| + |y| + c$, e como sabemos que $|x|+|y| = n$, então $C(x,y) \geq C(x) + C(y) + \log n - c$.

Teorema 10. *Seja $x = 1^n$ ou $x = 0^n$. Então, $C(x|n) \leq c$, para algum $c > 0$ independente de x.*

Prova: Trivial, pois podemos computar 1^n ou 0^n através de um programa de tamanho fixo se conhecemos previamente n (fornecido como informação lateral).

$C(x|n)$ é chamada *complexidade condicional de tamanho*. O Teorema 10 permite concluir que uma string x de tamanho n carrega uma quantidade de informação que tem dois sentidos. O primeiro, associado com as irregularidades de x e o segundo, associado ao tamanho da string. O segundo é especialmente evidente para strings de baixa complexidade, como 1^n e 0^n.

Um efeito da quantidade de informação associada com o tamanho da string é o fato de $C(x)$ não ser monotônica sobre prefixos, ou seja, se $m < n$ podemos obter $C(m) > C(n)$. Isto significa que $C(1^m) > C(1^n)$, embora a string 1^m seja prefixo da string 1^n. Assim, poderemos ter $C(x) > C(xy)$, que é um efeito colateral indesejado sobre a complexidade $C(x)$.

Por exemplo, supomos $m < n$, m incompressível e $n = 2^k$. Então, $C(1^n) \leq \log\log n + O(1)$, já que k tem magnitude $\log n$, e $C(1^m) \geq \log m - O(1)$. Mas para m e n quaisquer podemos obter $\log m > \log\log n$ e, neste caso, $C(1^m) > C(1^n)$.

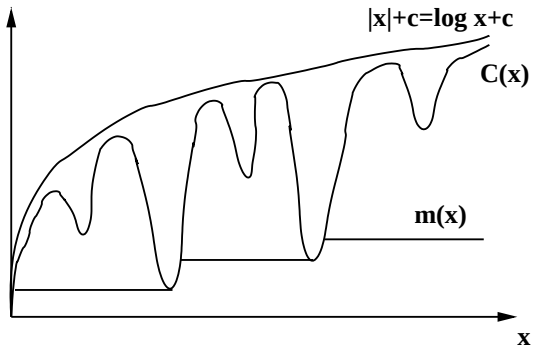

Figura 2. Funções $C(.)$ e $m(.)$

Seja $x = x_1x_2x_3\cdots$ uma string binária infinita, e seja $x_{1:n} = x_1x_2\cdots x_n$. Como conseqüência da não monotonicidade da complexidade C, a complexidade apresenta a propriedade da *flutuação*. Isto significa que, dada uma string binária infinita x, para valores crescentes de n, $\frac{n-C(x_{1:n})}{\log n}$ oscila entre 0 e 1 aproximadamente.

Podemos imaginar $C(.)$ como uma função de naturais em naturais, $C : \mathbb{N} \to \mathbb{N}$.

Sabemos que $C(x) \leq |x| + c = \log x + c$, pois podemos descrever qualquer x por sua codificação binária (veja Teorema 4). Portanto, a complexidade $C(x)$ tem como limite superior a curva $\log x + c$ (veja Figura 2).

Teorema 11. $\lim_{x\to\infty} C(x) = \infty$.

Prova: O número de strings x tal que $C(x) \leq a$ não excede $2^{a+1} - 1$. Assim, para qualquer a arbitrário, existe um x_0 tal que $C(x) > a$ para todo $x > x_0$.

Portanto, a complexidade cresce sempre, porém veremos que cresce muito lentamente.

Definimos $m(x) = \min_{y\geq x} C(y)$ como sendo a maior função monotônica crescente limitando $C(.)$ por baixo (veja Figura 2) [12].

Teorema 12. *Todo conjunto infinito enumerável contém um subconjunto infinito solúvel.*

Prova: Veja M. Li & P. Vitányi [9], página 33, Lema 1.7.4.

Teorema 13. *Para qualquer função parcial recursiva $f(x)$ tendendo monotonicamente a infinito a partir de um x_0, nós temos $m(x) < f(x)$.*

Prova: Suponha que existe uma função parcial recursiva $f(x) \leq m(x)$, para um conjunto infinito de pontos x. Então, f é definida em um conjunto infinito enumerável α. Pelo Teorema 12, α contém um conjunto infinito solúvel (decidível) β. Seja a função

$$\phi(x) = \begin{cases} f(x) & x \in \beta \\ f(y)\,, y = \max\{z : z \in \beta, z < x\} & x \notin \beta \end{cases}.$$

Assim, por construção, ϕ é uma função (total) recursiva, tende monotonicamente a infinito, e $\phi(x) \leq m(x)$ para infinitos x. Definimos $g(a) = \max\{x : C(x) \leq a\}$. Portanto, $g(a) + 1 = \min\{x : m(x) > a\}$ e $\max\{x : \phi(x) \leq a + 1\} \geq \min\{x : m(x) > a\} > g(a)$, para infinitos a. A função $h(a) = \max\{x : \phi(x) \leq a + 1\}$ é (total) recursiva. Segue que $h(a) > g(a)$ para infinitos a. Portanto, $C(h(a)) > a$ para infinitos a. Pelo Teorema 6, $C(h(a)) \leq C(a) + O(1) = \log a + O(1)$. Assim, $C(h(a)) > a$ e $C(h(a)) \leq \log a$ para infinitos a, que é uma contradição.

Assim, $m(.)$ tende a infinito mais lentamente que qualquer outra função parcial recursiva que tende a infinito monotonicamente. Isto prova que $C(.)$ tende a infinito muito lentamente.

Teorema 14. *(Propriedade da Continuidade)* $\text{abs}(C(x + h) - C(x)) \leq 2|h| + c$.

Prova: A string $x + h$ pode ser obtida a partir de $\overline{h}x^*$. Construímos uma máquina $\mathcal{M}$ que recebendo $\overline{h}x^*$ computa $x+h$. Segue do Teorema da Invariância que $C(x+h) \leq C_{\mathcal{M}}(x+h)+c = \left|\overline{h}x^*\right|+c = 2|h|+C(x)+c$. Logo, $C(x+h)-C(x) \leq 2|h| + c$.

Analogamente, a string x pode ser obtida a partir de $\overline{h}p$, onde p é uma descrição de $x+h$. Segue que $C(x) \leq C_{\mathcal{M}}(x+h)+c = \left|\overline{h}p\right|+c = 2|h|+C(x+h)+c$. Logo, $C(x) - C(x + h) \leq 2|h| + c$.

Isto significa que, embora $C(x)$ varie muito entre $m(x)$ e $\log x + c$, o faz de uma forma muito suave.

Teorema 15. *(Problema da Parada) Não existe procedimento para a máquina de Turing que, recebendo como entrada um programa p e um valor n, determine se p pára com n como entrada.*

Prova: Veja T. Divério & P. Menezes [4], seção 5.6, página 176.

Teorema 16. *A função C não é computável.*

Prova: Para algum x, nós computamos $C(x)$ executando todos os programas p com $|p| \leq |x| + c$, para algum c, de um modo concorrente, porque sabemos que $C(x) \leq |x| + c$ pelo Teorema 4. Nós fazemos isto gerando todos os programas em uma ordem lexicográfica crescente e provendo tempo de execução crescente para estes programas. Enquanto os programas vão parando, nós armazenamos o tamanho do menor programa que parou computando x. Quando todos programas tiverem parado nós teremos o valor de $C(x)$. Mas, pelo problema da parada, nós não podemos decidir se um programa pára ou não. Assim, por redução ao problema da parada, este processo não é efetivo.

3 Caracterização de Linguagens usando Complexidade de Kolmogorov

3.1 Definição de Regularidade-KC

O estudo de linguagens formais baseia-se em estabelecer uma hierarquia de linguagens. A principal é devida a Chomsky: linguagens regulares; livres de contexto; sensíveis ao contexto e enumeráveis recursivamente [10].

Sabemos que algumas linguagens são regulares e outras não. As provas por "pumping" (bombeamento) [10] são limitadas e não se baseiam em uma definição formal de "regularidade" [9].

Usando-se complexidade de Kolmogorov é possível caracterizar de uma forma muito melhor as linguagens regulares fornecendo lemas de bombeamento muito mais fáceis de serem usados [8,9].

Teorema 17. *(Regularidade-KC) Seja $L \subseteq \alpha^*$ uma linguagem regular, $L_x = \{y : xy \in L\}$. Existe uma constante c, tal que para cada x, se y é a n-ésima string em L_x, então $C(y) \leq C(n) + c$.*

Prova: Seja L uma linguagem regular. Então a string y tal que $xy \in L$, para algum x, pode ser descrita pelo autômato finito que aceita L e pelo estado do autômato após processar x e o número n. O primeiro item requer $O(1)$ bits. Logo, $C(y) \leq C(n) + O(1)$.

Exemplo 1. (M. Li & P. Vitányi [9], exemplo 6.8.2, página 422)

É facil provar que $L = \{1^p : p \text{ é um número primo}\}$ não é regular. Considere a string $xy = 1^p$ com p sendo o $(k+1)$-ésimo primo. Seja $x = 1^{p'}$, com p' sendo o k-ésimo primo. Então $y = 1^{p-p'}$ e y é o primeiro elemento do conjunto L_x. Assim, supondo que L é regular, pelo Teorema 17, $C(p - p') = O(1)$. Mas a diferença entre dois primos sucessivos cresce sem limite. Como só existem $O(1)$ descrições de tamanho $O(1)$ temos uma contradição.

3.2 Caracterização de Linguagens Regulares

Definição 7. *Seja α um alfabeto, e seja y_i o i-ésimo elemento de α^* em ordem lexicográfica, para $i \geq 1$. Para $L \subseteq \alpha^*$ e $x \in \alpha^*$, seja $\chi = \chi_1\chi_2 \ldots$ a* seqüência característica *de $L_x = \{y : xy \in L\}$, definida como $\chi_i = 1$ se $xy_i \in L$ e $\chi_i = 0$ caso contrário. Denotamos $\chi_1\chi_2 \ldots \chi_n$ como $\chi_{1:n}$.*

Teorema 18. *(Caracterização de Regularidade-KC) Seja $L \subseteq \alpha^*$. As seguintes afirmações são equivalentes:*

1. *L é regular;*
2. *Existe uma constante c, que só depende de L, tal que para todo $x \in \alpha^*$ e para todo n, $C(\chi_{1:n}|n) \leq c$;*
3. *Existe uma constante c, que só depende de L, tal que para todo $x \in \alpha^*$ e para todo n, $C(\chi_{1:n}) \leq C(n) + c$;*
4. *Existe uma constante c, que só depende de L, tal que para todo $x \in \alpha^*$ e para todo n, $C(\chi_{1:n}) \leq \log n + c$.*

Exemplo 2. (M. Li & P. Vitányi [8], exemplo 7)

Seja $\alpha^* = \{0, 1\}$. Devemos provar que $L = \alpha^*$ é regular. Existe uma constante c, tal que a sequência característica associada é $\chi = 1, 1, 1, \ldots$, com $C(\chi_{1:n}|n) \leq c$ (veja Teorema 10). Então, L é regular pelo Teorema da Caracterização de Regularidade-KC.

3.3 Caracterização das Linguagens Livres de Contexto

Podemos usar complexidade de Kolmogorov para caracterizar também as linguagens livres de contexto.

Teorema 19. *Seja $L \subseteq \alpha^*$ uma linguagem livre de contexto determinística, sejam c, c' constantes. Seja w uma sequência recursiva sobre α. Seja u um prefixo de vw, seja v um prefixo de w, tal que:*

1. *v pode ser descrito em c bits dado L_u na ordem lexicográfica;*
2. *w pode ser descrito em c' bits dado L_{uv} na ordem lexicográfica;*
3. *$C(v) \geq 2 \log\log |u|$.*

Então existe uma constante c'' dependendo somente de L, c, c', w tal que $C(w) \leq c''$.

Exemplo 3. (M. Li & P. Vitányi [8], exemplo 9)

Desejamos provar que $L = \{x : x = x^R, x \in \{0,1\}^*\}$ não é uma linguagem livre de contexto determinística. Supomos o contrário, que L é uma LLC determinística. Atribuimos $u = 0^n 1$ e $v = 0^n$, $C(n) \geq \log n$ (supomos que n é incompressível). $C(v) \geq \log\log(n+1)$ pois $C(n) \geq \log n$. Como v é lexicograficamente a primeira palavra em L_u, o primeiro item do Teorema é satisfeito. O primeiro elemento lexicograficamente não vazio em L_{uv} é 10^n e $C(w|L_{uv}) \leq c$, satisfazendo o segundo item do Teorema. Mas, temos $C(w) = O(\log n)$, o que é uma contradição. Logo, L não é LLC determinística.

4 Exemplos de Outras Aplicações

Existem muitas aplicações de complexidade de Kolmogorov em Ciência da Computação. Adicionalmente à aplicação apresentada neste trabalho desejamos introduzir algumas outras aplicações.

Em [11] é proposta uma aplicação em Inteligência Artificial, que se constitui em um método geral de inferência indutiva. Tal inferência é feita substituindo-se, na regra de Bayes, a probabilidade prévia por uma *prévia universal* baseada no tamanho do menor programa para a máquina de Turing que computa uma saída. Este método está presente no núcleo dos métodos de aprendizado MDL (minimum description length) e MML (minimum message length). Em [5] é apresentado um método para obter a Complexidade Computacional de um algoritmo baseado no método da incompressividade. Finalmente, em [1] é apresentada uma aplicação em Engenharia de Software, que se constitui em uma caracterização do desenvolvimento de software baseada em complexidade de Kolmogorov. O trabalho tem três conclusões principais: o processo de desenvolvimento de software é formalmente imprevisível; os métodos formais são incompletos (devido à incompletude dos sistemas formais); e o desenvolvimento de software é, embora os esforços para sua formalização, também subjetivo e empírico.

Estas aplicações demonstram quão flexivel e variadas são as aplicações da complexidade de Kolmogorov em Ciência da Computação, justificando nosso desejo que as pesquisas nesta área, no nosso país, atinjam a projeção que já tem no exterior.

5 Conclusão

Neste trabalho apresentamos uma revisão bibliográfica selecionada da área de complexidade de Kolmogorov. Ao final incluimos uma aplicação em Ciência da Computação que mostra a potencialidade da área.

A aplicação apresentada constitui-se em lemas de bombeamento para linguagens regulares e livres de contexto que são mais expressivos que a técnica tradicional. O lema de bombeamento para linguagens enumeráveis recursivamente usa o mesmo tipo de raciocínio e, por isto, foi omitido [8].

Finalmente, os autores planejam publicar, em um trabalho futuro, os assuntos probabilidade algorítmica, complexidade de prefixo e seqüências aleatórias de Martin-Löf, que são obrigatórios para a completude de uma revisão bibliográfica sobre o assunto, mas que não puderam ser apresentados devido ao espaço disponível.

Referências

1. Campani, C. ; Menezes, P. Characterizing the Software Development Process: A New Approach Based on Kolmogorov Complexity. In: *Computer Aided Systems Theory - EUROCAST'2001*, 8th International Workshop on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, feb. 19-23, 2001, Lecture Notes in Computer Science, Springer, v. 2178, p. 242-256, 2001.
2. Chaitin, G. Information-Theoretic Computational Complexity. *IEEE Transactions on Information Theory*, v. 20, p. 10-15, 1974.
3. Davis, M. A Note On Universal Turing Machines. In: Shannon, C. ; McCarthy, J. *Automata Studies*, p. 129-153, Princeton University Press, 1956.
4. Diverio, T. ; Menezes, P. *Teoria da Computação:* Máquinas Universais e Computabilidade. segunda edição, Porto Alegre: Sagra-Luzzatto, 2000. 224 p.
5. Jiang, T. ; Li, M. ; Vitányi, P. A lower bound on the average-case complexity of Shellsort. *J. Assoc. Comp. Mach.*, v. 47, n. 5, p. 905-911, 2000.
6. Kolmogorov, A. Three Approaches to the Quantitative Definition of Information. *Problems of Information Transmission*, v. 1, p. 4-7, 1965.
7. Kolmogorov, A. Logical Basis for Information Theory and Probability Theory. *IEEE Transactions on Information Theory*, v. 14, n. 5, p. 662-664, 1968.
8. Li, M. ; Vitányi, P. A New Approach to Formal Language Theory by Kolmogorov Complexity. *SIAM J. Comput.*, v. 24, n. 2, p. 398-410, 1995.
9. Li, M. ; Vitányi, P. *An Introduction to Kolmogorov Complexity and its Applications.* 2nd edition, New York: Springer, 1997. 657 p.
10. Menezes, P. *Linguagens Formais e Autômatos.* Porto Alegre: Sagra-Luzzatto, 1997. 168 p.
11. Solomonoff, R. The Discovery of Algorithmic Probability. *Journal of Computer and System Sciences*, v. 55, n. 1, p. 73-88, aug. 1997.
12. Zvonkin, A. ; Levin, L. The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms. *Russian Math. Surveys*, v. 25, n. 6, p. 83-124, 1970.