UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CARLOS A. P. CAMPANI

Avaliação da Compressão de Dados e da Qualidade de Imagem em Modelos de Animação Gráfica para Web: uma nova abordagem baseada em Complexidade de Kolmogorov

Tese apresentada como requisito parcial
para a obtenção do grau de
Doutor em Ciência da Computação

Prof. Dr. Paulo Blauth Menezes
Orientador

Porto Alegre, abril de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Campani, Carlos A. P.

Avaliação da Compressão de Dados e da Qualidade de Imagem em Modelos de Animação Gráfica para Web: uma nova abordagem baseada em Complexidade de Kolmogorov / Carlos A. P. Campani. – Porto Alegre: PPGC da UFRGS, 2005.

111 f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Paulo Blauth Menezes.

1. Complexidade de Kolmogorov. 2. Teoria da Informação. 3. Compressão de dados. 4. Qualidade de imagem. 5. Animação gráfica. 6. Teoria da Computação. I. Menezes, Paulo Blauth. II. Título.

‘You should learn not to make personal remarks,’ Alice said with some severity; ‘it’s very rude.’

The Hatter opened his eyes very wide on hearing this; but all he said was, ‘Why is a raven like a writing-desk?’

‘Come, we shall have some fun now!’ thought Alice. ‘I’m glad they’ve begun asking riddles.–I believe I can guess that,’ she added aloud.

Alice’s Adventures in Wonderland, Lewis Carroll

‘When I use a word,’ Humpty Dumpty said in rather a scornful tone, ‘it means just what I choose it to mean – neither more nor less.’

‘The question is,’ said Alice, ‘whether you CAN make words mean so many different things.’

‘The question is,’ said Humpty Dumpty, ‘which is to be master - - that’s all.’

Alice Through the Looking Glass, Lewis Carroll

Dedico este trabalho à Marcia, João Vitor, Luana e Ticiano

AGRADECIMENTOS

Agradeço ao CNPq pelo apoio dado ao projeto ao qual se vincula esta pesquisa, e à CAPES pelo apoio financeiro, através de bolsa modalidade PICDT, que custeou minha atividade de pesquisa na UFRGS, ligada à elaboração deste projeto, e a Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) pelo auxílio financeiro para a apresentação de um artigo no EUROCAST 2001 na Espanha, que foi de grande importância para este projeto. Agradeço a todos aqueles que de alguma forma contribuíram para a elaboração deste trabalho, especialmente ao colega Fernando Accorsi. Agradeço ao meu Orientador, Prof. Paulo Blauth Menezes, sem o qual este trabalho não teria acontecido.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho versa sobre a avaliação da compressão de dados e da qualidade de imagens e animações usando-se complexidade de Kolmogorov, simulação de máquinas e distância de informação. Complexidade de Kolmogorov é uma teoria da informação e da aleatoriedade baseada na máquina de Turing. No trabalho é proposto um método para avaliar a compressão de dados de modelos de animação gráfica usando-se simulação de máquinas. Também definimos formalmente compressão de dados com perdas e propomos a aplicação da distância de informação como uma métrica de qualidade de imagem. O desenvolvimento de uma metodologia para avaliar a compressão de dados de modelos de animação gráfica para web é útil, a medida que as páginas na web estão sendo cada vez mais enriquecidas com animações, som e vídeo, e a economia de banda de canal torna-se importante, pois os arquivos envolvidos são geralmente grandes. Boa parte do apelo e das vantagens da web em aplicações como, por exemplo, educação à distância ou publicidade, reside exatamente na existência de elementos multimídia, que apoiam a idéia que está sendo apresentada na página. Como estudo de caso, o método de comparação e avaliação de modelos de animação gráfica foi aplicado na comparação de dois modelos: GIF (Graphics Interchange Format) e AGA (Animação Gráfica baseada em Autômatos finitos), provando formalmente que AGA é melhor que GIF ("melhor" significa que AGA comprime mais as animações que GIF). Foi desenvolvida também uma definição formal de compressão de dados com perdas com o objetivo de estender a metodologia de avalição apresentada. Distância de informação é proposta como uma nova métrica de qualidade de imagem, e tem como grande vantagem ser uma medida universal, ou seja, capaz de incorporar toda e qualquer medida computável concebível. A métrica proposta foi testada em uma série de experimentos e comparada com a distância euclidiana (medida tradicionalmente usada nestes casos). Os resultados dos testes são uma evidência prática que a distância proposta é efetiva neste novo contexto de aplicação, e que apresenta, em alguns casos, resultados superiores ao da distância euclidiana. Isto também é uma evidência que a distância de informação é uma métrica mais fina que a distância euclidiana. Também mostramos que há casos em que podemos aplicar a distância de informação, mas não podemos aplicar a distância euclidiana. A métrica proposta foi aplicada também na avaliação de animações gráficas baseadas em frames, onde apresentou resultados melhores que os obtidos com imagens puras. Este tipo de avaliação de animações é inédita na literatura, segundo revisão bibliográfica feita. Finalmente, neste trabalho é apresentado um refinamento à medida proposta que apresentou resultados melhores que a aplicação simples e direta da distância de informação.

Palavras-chave: Complexidade de Kolmogorov, Teoria da Informação, Compressão de dados, Qualidade de imagem, Animação gráfica, Teoria da Computação.

Evaluating Data Compression and Image Quality in Computer Animation Models for Web: A New Approach Based on Kolmogorov Complexity

ABSTRACT

This work deals with the evaluation of data compression and image quality applyed to images and computer animations, using Kolmogorov complexity, machine simulation, and information distance. Kolmogorov complexity is a theory of information and randomness based on Turing machine. We propose a method to evaluate data compression in computer animation models using machine simulation. Also, we formally define lossy data compression and present an application of information distance as an image quality metric. The development of such methodology to evaluate data compression of computer animation models is very usefull, as the web pages are enriched with animations, sound and video, and the band-width must be optimized, because the files envolved are generally very large. Much of the appeal and advantage of web applications as, for example, education or publicity, rely exactly on the existence of such multimedia elements, which support the ideas presented in the text. As a case study, we present an application of the method to evaluate computer animations, comparing two models: GIF (Graphics Interchange Format) and AGA (Automata-based Graphical Animation), formally proving that AGA is better than GIF ("better" means that AGA compress animations more than GIF). It was developed a formal definition of lossy data compression too, with the intent to extend the methodology presented. Information distance is proposed as a new image quality metric, with the advantage that it is in a formal sense universal, that is, capable of incorporate any computable conceivable metric. The proposed metric was tested in a serie of experiments and compared with the euclidian distance (the metric more used in cases like that). The tests are a practical evidence that information distance is effective in this new application context, and that presents, in some cases, better results than euclidian distance. Moreover, it is an evidence that information distance is finer than euclidian distance. We show cases where we can apply information distance but we can not apply euclidian distance. The proposed metric is applyed to the evaluation of frame-based computer animations too, where we reach better results than with simple images. This kind of evaluation is unpublished, according review we made. Finally, this work introduces a refinement to the metric proposed, which shows better results than the simple and straightforward application of information distance.

Keywords: Kolmogorov Complexity, Information Theory, Data compresion, Image quality, Computer animation, Theory of Computing.

1 INTRODUÇÃO

Segundo Heinz Pagels (PAGELS, 1988), um dos aspectos importantes na ciência moderna é a complexidade[1]. Observamos a complexidade emergir de tudo: nos terrenos e paisagens cuja beleza apreciamos; nos organismos biológicos; no comportamento do clima e seus fenômenos; na organização do homem e suas estruturas sociais; no cérebro humano e nas estruturas cognitivas; etc. Mesmo um dos mais simples organismos vivos, tal como uma bactéria, possui uma complexidade que desafia a ciência em entender seu comportamento.

Um capítulo a parte pode ser consagrado ao encontro das estruturas complexas vistas na Natureza e técnicas da Ciência, que convergem para uma nova Ciência, que poderíamos chamar de Ciências da Complexidade. Vários exemplos podem ser dados desta convergência: fractais; algoritmos genéticos; etc.

Antes de tudo, a própria "complexidade", enquanto conceito, desafia a Ciência a ser definida e entendida. Um passo em direção a este objetivo pode ter sido dado na década de sessenta, quando Kolmogorov, Solomonoff e Chaitin, de forma independente, definiram o que hoje nós conhecemos como "complexidade de Kolmogorov" (LI; VITÁNYI, 1997).

Originalmente proposta como uma teoria da informação e da aleatoriedade, descobriu-se uma grande quantidade de aplicações da complexidade de Kolmogorov em campos bem diversos (GAMMERMAN; VOVK, 1999) que incluem: teoria das probabilidades (VITÁNYI, 2004) (definição formal de "seqüência aleatória"); linguagens formais (LI; VITÁNYI, 1995) (lemas de bombeamento baseados em incompressividade); inteligência artificial (SOLOMONOFF, 1997) (raciocínio indutivo e previsão); complexidade computacional (JIANG; LI; VITÁNYI, 2000) (argumentos de incompressividade para determinar complexidade computacional); teoria de grafos (BUHRMAN et al., 1999) (grafos K-aleatórios); biotecnologia (LI et al., 2003) (reconhecimento de genoma); etc.

Falando a grosso modo, a complexidade de Kolmogorov determina, para cada string binária, um valor numérico associado que é sua "complexidade". Simplificadamente, ela pode ser definida como o tamanho do menor programa que computa a string binária em uma máquina de Turing universal previamente escolhida. Complexidade de Kolmogorov define uma nova teoria da informação chamada *teoria algorítmica da informação*. É importante observar que, na complexidade de Kolmogorov, é irrelevante o tempo necessário para a computação do programa na máquina de Turing, o que diferencia esta complexidade da complexidade dinâmica, que leva em consideração o tempo necessário para

[1] "*Complexidade* é uma das características mais visíveis da realidade que nos cerca. Por ela queremos designar os múltiplos fatores, energias, relações, inter-retro-reações que caracterizam cada ser e o conjunto dos seres do universo. Tudo está em relação com tudo. Nada está isolado, existindo solitário, de si para si. Tudo co-existe e inter-existe com todos os outros seres do universo.", *A águia e a galinha: uma metáfora da condição humana*, Leonardo Boff, Editora Vozes, 1997, página 72.

executar os programas.

Pode-se dizer que a complexidade de Kolmogorov de uma string binária representa a versão comprimida da string. Aliás, o conceito chave neste trabalho é o de "compressão de dados", que será usada como parâmetro para todas as avaliações efetuadas.

Neste trabalho, aplicamos a complexidade de Kolmogorov no contexto de imagens e animações gráficas, com o objetivo de avaliar a taxa de compressão de dados obtida por um modelo de animação gráfica (ou seja, o quanto os modelos comprimem o tamanho dos arquivos das animações) e como uma nova métrica de qualidade de imagem, aplicada tanto a imagens quanto a animações (nossa principal contribuição científica). "Qualidade de imagem" não refere-se a um conceito subjetivo de "qualidade", mas a uma métrica que mede a similaridade entre imagens, permitindo avaliar o quanto uma imagem foi distorcida por uma transformação qualquer aplicada a ela.

O método proposto neste trabalho, para a comparação de modelos de animação gráfica quanto à compressão de dados obtida, é uma idéia ao que se sabe (segundo pesquisa bibliográfica) inédita, e baseia-se em definir um modelo de animação como uma máquina, cuja saída é a seqüência de frames que forma a animação, e usar simulação de máquinas como ferramenta de comparação. Isto permite estabelecer uma hierarquia de modelos de animação gráfica. Aplicamos este método provando que o modelo AGA (Automata-based Graphical Animation) é melhor que o modelo GIF (Graphics Interchange Format), com relação à compressão de dados obtida. O primeiro, AGA (ACCORSI; MENEZES, 2000), desenvolvido pelo grupo de pesquisa ao qual se vincula este trabalho, e daí sua escolha, e o segundo, GIF, escolhido como termo de comparação por dois motivos:

- sua popularidade, advinda de sua simplicidade, em páginas na web, que deverá crescer nos próximos anos, já que o formato tornou-se de livre uso de fato, por terem expirados todos os direitos da Unisys, em todo o mundo, sobre o algoritmo de compressão usado pelo formato GIF (chamado LZW);

- o fato de GIF ser um modelo baseado em mapas de bits, o que o torna adequado para a comparação que desejávamos fazer (isto excluia, por exemplo, animações flash da Macromedia, outro modelo muito popular, por serem animações baseadas em gráficos vetoriais).

Adicionalmente apresentamos uma definição de "compressão de dados com perdas" que estende os resultados apresentados usando-se compressão sem perdas. Baseado nesta definição é proposto o uso de *distância de informação* (BENNETT et al., 1993; LI et al., 2003) como uma métrica de qualidade de imagem. Esta proposta é, ao que se sabe, inédita na literatura.

Baseado nesta idéia, foram feitas medições com o propósito de avaliar a qualidade de imagens e animações submetidas a distorções arbitrariamente escolhidas, com o objetivo de validar a proposta, obtendo evidências de que a métrica é efetiva neste tipo de aplicação.

A motivação, ao início do estudo de complexidade de Kolmogorov, foi investigar de forma extensa e abrangente as possíveis aplicações de complexidade de Kolmogorov nos contextos que o grupo de pesquisa da UFRGS trabalhava. Particularmente procurávamos uma melhor ferramenta para tratar assuntos relacionados com os fundamentos da computação. Inicialmente foi estudada sua relação com os *problemas indecidíveis*: limites dos sistemas formais; número Ω; etc. Como conseqüência deste trabalho inicial, foi publicado um extended abstract e um artigo completo, envolvendo uma aplicação da complexidade

de Kolmogorov no contexto de desenvolvimento de software (CAMPANI; MENEZES, 2001a,b).

A complexidade de Kolmogorov pareceu, desde o começo da pesquisa, uma boa escolha, pois ela se refere a um conceito absoluto de complexidade que pode ser aplicado de forma genérica. Assim, em diferentes contextos das pesquisas que o grupo da UFRGS desenvolvia, se poderia aplica-la. É importante lembrar que existem outras abordagens que poderiam ter sido seguidas pelo projeto, mas o interesse imediato da pesquisa se concentrava em investigar estas aplicações da complexidade de Kolmogorov nestes contextos de atuação do grupo de pesquisa.

Os objetivos gerais deste trabalho são:

Modelos de animação gráfica Formalizar compressão de dados e animação, de forma a comparar, e em certo sentido avaliar, modelos de animação gráfica quanto à compressão de dados obtida pelos modelos. A idéia a ser explorada é o conceito de minoração (simulação de máquinas) para efetuar as avaliações, para isto, definimos modelos de animação como máquinas;

Métrica de qualidade de imagem Propor e validar o uso de *distância de informação* como uma métrica de qualidade de imagem. A distância de informação define, a priori, uma medida de similaridade sobre o conjunto das strings binárias que, segundo a nossa abordagem, permite avaliar a qualidade de imagens submetidas a distorções e transformações. Uma das vantagens da definição de distância proposta é a *universalidade*, que significa que ela é capaz de incorporar toda métrica (computável) concebível. A métrica será aplicada também no contexto de animações gráficas e validada por meio de experimentos envolvendo casos reais.

Os resultados obtidos neste trabalho foram:

Avaliação de animação gráfica A proposta de um método de avaliação da compressão de dados em modelos de animação gráfica baseada em definições formais de compressão de dados sem perdas, e a sua aplicação, como estudo de caso, na comparação dos modelos GIF (Graphics Interchange Format) e AGA (Animação Gráfica baseada em Autômatos finitos). O resultado prova que AGA é melhor que GIF ("melhor" significa que AGA comprime mais as animações que GIF);

Definição de compressão de dados com perdas O desenvolvimento de uma definição formal de compressão com perdas baseada em distância de informação;

Métrica de qualidade de imagem A proposta da distância de informação como uma nova métrica de qualidade de imagem e animações. A aplicação desta métrica com o objetivo de validá-la, assim como a comparação da medida proposta com a distância euclidiana (medida tradicionalmente usada nestes casos). Os resultados obtidos são uma evidência que a métrica é efetiva neste contexto de aplicação e, em alguns casos, é melhor que a distância euclidiana. A aplicação da métrica na avaliação da qualidade de imagem de animações é, segundo pesquisa bibliográfica feita, inédita. O único tipo de avaliação de animações até então desenvolvida, tratava apenas dos aspectos psicológicos da percepção de deformações no movimento (O'SULLIVAN et al., 2003).

Ao longo da pesquisa efetuada, que resultou neste trabalho, foram publicados cinco artigos completos (CAMPANI; MENEZES, 2004, 2003, 2002a,b, 2001a), três deles em

eventos internacionais (EUROCAST'2001/LNCS/Springer, CISST'03 e CISST'04), os outros dois no WMF'2002, um extended abstract em evento internacional (CAMPANI; MENEZES, 2001b) (EUROCAST'2001), um artigo completo selecionado para publicação em periódico nacional (Revista de Informática Teórica e Aplicada) (CAMPANI; MENEZES, 2004), intitulado "Teorias da Aleatoriedade", e está submetido para publicação, ainda sem resposta, o artigo (CAMPANI et al., 2004), intitulado "An Efficient and Flexible Animation Model Based on Automata Theory: Evaluating Data Compression".

Com relação à organização deste texto, os Capítulos 3 e 4 constituem-se em uma revisão bibliográfica abreviada da área de complexidade de Kolmogorov, cobrindo complexidade básica, complexidade de prefixo, incompressividade, probabilidade algorítmica e a definição de seqüência aleatória de Martin-Löf. Esta revisão visa a apresentar as estruturas matemáticas que serão usadas no trabalho. Também procuramos nestes Capítulos defender um ponto importante: a complexidade de Kolmogorov como uma teoria da informação e da aleatoriedade adequada à aplicação que será feita dela ao longo do trabalho. Caso o leitor já conheça estes aspectos, sugerimos uma rápida leitura para identificar a simbologia.

Os teoremas apresentados nesta revisão bibliográfica não são de nossa autoria, mas as provas foram todas refeitas e sistematizadas de forma diferente que a apresentada nos artigos e livros originais.

O Capítulo 2 apresenta a notação usada ao longo de todo o trabalho. Embora tenha sido seguida uma notação relativamente padrão, alguma coisa não segue a notação de outros trabalhos do gênero. Recomenda-se fortemente a leitura deste Capítulo para facilitar a leitura do material subseqüente.

O Capítulo 5 apresenta uma breve revisão da área de compressão de dados que introduz a terminologia da área e apresenta alguns algoritmos de compressão de dados bem conhecidos.

Os Capítulos 6-10 constituem-se na contribuição científica do trabalho, a exceção o Capítulo 8 que é uma revisão bibliográfica, o Teorema 33 e a Seção 6.1.2 que se baseiam em (SUBBARAMU; GATES; KREINOVICH, 1998).

O Capítulo 6 apresenta o método de avaliação da compressão de dados em modelos de animação gráfica, e faz a comparação entre o modelo GIF e o modelo AGA (Teorema 35) como um estudo de caso.

O Capítulo 7 apresenta um resultado que estende o Capítulo 6 para tratar compressão de dados com perdas. Neste Capítulo também é introduzida a proposta da distância de informação como uma métrica de qualidade de imagens.

O Capítulo 8 apresenta uma breve revisão sobre métricas de qualidade de imagem.

O Capítulo 9 apresenta a aplicação da distância de informação como uma métrica de qualidade de imagem. São apresentados alguns experimentos feitos com o objetivo de validar a proposta. A métrica também foi aplicada para avaliar a qualidade de imagem de animações gráficas. Ao final do Capítulo é apresentado um refinamento ao método que permite obter resultados melhores que a aplicação simples e direta do método.

2 NOTAÇÃO

Este Capítulo apresenta a notação usada ao longo de todo o trabalho. Embora tenha sido seguida uma notação relativamente padrão, parte da notação foi adaptada e não segue exatamente a usada em outros trabalhos do gênero.

Neste texto será usada a notação já consagrada no livro de M. Li & P. Vitányi (LI; VITÁNYI, 1997), com pequenas adaptações. Seguimos a convenção de chamar a complexidade básica de C, a complexidade de prefixo de K e a semi-medida enumerável universal discreta de $\mathbf{m}$. Seguindo a convenção usada em (ZVONKIN; LEVIN, 1970), chamamos a maior função monotônica crescente que limita $C(.)$ por baixo de $m(.)$.

$\Pr(A)$ denota a probabilidade de algum evento aleatório A e $\Pr(A|B)$ a probabilidade condicional de A dado que ocorreu B. $\Pr(A \cap B)$ é a probabilidade da ocorrência simultânea dos eventos A e B (LIPSCHUTZ, 1968). $\binom{n}{k}$ denota os coeficientes binomiais.

$\mathbb{R}$ e $\mathbb{N}$ denotam, respectivamente, o conjunto dos números reais e o conjunto dos números naturais. A cardinalidade de um conjunto A é indicada por $\overline{\overline{A}}$. $\sup A$ denota o supremo de um conjunto A. Se $x \in A$ e $y \in B$ então (x, y) é um elemento do produto cartesiano $A \times B$. Denotamos por $[a; b], [a; b), (a; b]$ e $(a; b)$ os intervalos abertos e/ou fechados, com $a, b \in \mathbb{R}$.

Denotamos o tamanho de uma string binária (número de dígitos binários) por $|.|$. Assim, $|1010| = 4$. O valor absoluto de um número é denotado por abs$(.)$. Então, abs$(-2) = 2$. $\min f$ denota o valor mínimo de f. Por exemplo, $\min\{3, 6, 7, 10\} = 3$. $\max$ denota o valor máximo de um conjunto. Representamos a concatenação de duas strings x e y por xy, e chamamos x *prefixo* de xy ou prefixo de y em xy. Todos os $\log$ são sempre logaritmos na base dois. Se $X_1, X_2, X_3, \ldots$ é uma seqüência de variáveis aleatórias, $\overline{X}_n$ denota $\frac{1}{n}\sum X_i$.

α^* designa o conjunto infinito de palavras de tamanho maior ou igual a zero sobre o alfabeto α, e α^+ denota o conjunto infinito de todas as palavras de tamanho maior que zero. Assim, $\{0, 1\}^*$ denota o conjunto de strings binárias com tamanho maior ou igual a zero, e $\{0, 1\}^+$ denota o conjunto de strings binárias com tamanho maior que zero. Denotamos a string $aaaaa \ldots a$, composta por n símbolos a, como a^n. Λ representa a string ou palavra vazia, e $|\Lambda| = 0$. $\{0, 1\}^\infty$ representa o conjunto de todas as strings binárias unidirecionais infinitas. Se $x \in \{0, 1\}^\infty$ e, portanto, $x = x_1x_2x_3 \ldots$, então $x_{1:n} = x_1x_2 \ldots x_n$. O cilindro Γ_x denota o conjunto $\left\{a \in \{0, 1\}^\infty : a_{1:|x|} = x\right\}$.

Se x é uma string binária, $\overline{x}$ denota a versão auto-delimitada de x, $\overline{x} = 1^{|x|}0x$.

$\lfloor . \rfloor$ (chão) representa o maior número inteiro menor ou igual a um número. Assim, $\lfloor 2{,}7 \rfloor = 2$. $\lceil . \rceil$ (teto) denota o menor inteiro maior ou igual a um número. Por exemplo, $\lceil 2{,}7 \rceil = 3$. $\ll$ denota o predicado "muito menor que". $x \approx y$ significa que o valor de x é aproximadamente igual ao valor de y (diferem apenas por uma constante pequena).

Letras caligráficas maiúsculas $\mathcal{M}, \mathcal{U}, \ldots$ são usadas para designar máquinas, de forma que $\mathcal{M}_p = x$ significa que o programa p computa a string x na máquina $\mathcal{M}$. x^* denota o menor programa que computa a string ou o número natural x em uma máquina de referência.

Se f é uma função bijetora então f^{-1} é a função inversa de f. $f(x) \uparrow g(x)$ significa que a função f se aproxima por baixo da função g no ponto x. $f_n \rightarrow f$ significa que a seqüência de funções f_n converge para f.

O somatório será indicado por $\sum$ e o produtório por $\prod$. $\sum_{k=0}^{n} a_k$ denota o somatório $a_0 + a_1 + a_2 + \cdots + a_n$. $\sum_{x \in \alpha}$ denota o somatório sobre todos os elementos de um conjunto α, e $\sum_x$ pode ser usado quando o domínio de x é conhecido e óbvio. Adicionalmente, podemos indicar um predicado que limita o somatório, por exemplo, $\sum_{i:i<j}$ significa o somatório sobre todos os i menores que j.

$O(.)$ designa a notação assintótica. Portanto, se $f(x)$ é $O(g(x))$, então para todo $x \geq x_0$, $f(x) \leq cg(x)$, para algum x_0 e alguma constante c.

Finalmente, usamos o símbolo $\square$ para indicar o fim da prova de um teorema ou lema.

3 COMPLEXIDADE DE KOLMOGOROV: DEFINIÇÃO E PRINCIPAIS PROPRIEDADES

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 a um conceito pontual de aleatoriedade, ao invés de uma aleatoriedade média como o faz a teoria das probabilidades (CAMPANI; MENEZES, 2002a; LI; VITÁNYI, 1997).

Em 1948, em seu trabalho clássico "The Mathematical Theory of Communication", C. Shannon (SHANNON, 1948) fundou uma nova área científica, fundamentada em teoria das probabilidades (JAMES, 1996; LIPSCHUTZ, 1968) e em problemas de engenharia relacionados com comunicação de dados, chamada *teoria da informação*. O propósito original de Shannon era quantificar a informação que transita por um sistema composto de transmissor, receptor e canal. No entanto, Shannon entendia a informação como o tamanho (ou quantidade) em bits das mensagens que informam sobre objetos cuja ocorrência tem uma distribuição de probabilidade bem definida, não expressando nada a respeito das complexidades dos objetos em si. Neste texto nos referimos a esta teoria como *teoria da informação de Shannon* ou *teoria da informação clássica* (COVER; THOMAS, 1991; KHINCHIN, 1957).

Nos anos sessenta, Kolmogorov, Solomonoff e Chaitin, entre outros pesquisadores, (CHAITIN, 1974, 1975; KOLMOGOROV, 1965, 1968; SOLOMONOFF, 1997) desenvolveram, de forma independente, uma teoria da informação baseada no tamanho dos programas para a máquina de Turing (DIVERIO; MENEZES, 2000; HOPCROFT; ULLMAN, 1979). Esta teoria recebeu vários nomes: complexidade algorítmica; teoria algorítmica da informação; complexidade de Kolmogorov; K-complexidade; aleatoriedade de Kolmogorov-Chaitin; complexidade estocástica; complexidade descritiva; complexidade do tamanho de programas; entropia algorítmica; e probabilidade algorítmica; entre outras denominações. Convencionou-se chamar a área genericamente de *complexidade de Kolmogorov* em homenagem ao famoso matemático russo, e fundador da área, Andrei Nikolaevich Kolmogorov (1903-1987).

O objetivo original do trabalho de Kolmogorov era obter uma definição formal de seqüência aleatória (GAMMERMAN; VOVK, 1999). Kolmogorov observou que algumas seqüências binárias (seqüências de zeros e uns) podiam ser comprimidas algoritmicamente. Nos dias de hoje, a maioria das pessoas está acostumada a usar programas de compressão de dados, tais como winzip, gzip, arj, etc. O princípio de funcionamento destes programas é encontrar regularidades no arquivo que deve ser comprimido e substituir estas sub-seqüências regulares por descrições mais curtas.

Kolmogorov postulou que seqüências que não podem ser comprimidas algoritmicamente em uma descrição de tamanho muito menor que a seqüência original são seqüências *aleatórias* (no sentido estocástico do termo). Assim, as seqüências simples são aquelas que apresentam regularidade e as seqüências complexas (ou aleatórias) são aquelas que apresentam irregularidade (são incompressíveis).

Poderíamos comparar a complexidade de Kolmogorov de uma string com o "tamanho comprimido" da string. A complexidade de Kolmogorov associa a cada string binária um valor que é a sua "complexidade", definida como o tamanho de sua descrição. Assim, se diz que um objeto é simples se possui uma descrição curta.

A complexidade de Kolmogorov está relacionada com a máquina de Turing como um dispositivo de "descompressão de dados", não envolvendo de nenhuma maneira dispositivos e algoritmos relacionados com o processo inverso de "compressão de dados".

3.1 Entropia e Informação

Suponha pensar na informação em termos de mensagens enviadas por um canal que conecta um transmissor e um receptor (COVER; THOMAS, 1991; SHANNON, 1948). Podemos transmitir os dados através de códigos alfa-numéricos, binários, código Morse, etc. A única exigência é que transmissor e receptor concordem sobre a interpretação de cada um dos códigos possíveis usados na comunicação. Por exemplo, se estivéssemos informando o receptor sobre os resultados de uma votação sobre qual livro de Érico Veríssimo, entre "Olhai os Lírios do Campo" e "Incidente em Antares", é o melhor, transmissor e receptor deveriam concordar em representar o primeiro livro por um 0 e o segundo por um 1 ou vice-versa. Assim, cada voto seria representado por uma mensagem de tamanho um bit. É importante observar que, neste trabalho, vamos supor que o canal é sem ruído (sem erros).

Pensemos nas possíveis mensagens como seqüências de símbolos do conjunto $\alpha = \{a_1, a_2, \ldots, a_n\}$. Por exemplo, se estivéssemos comunicando resultados de lançamentos sucessivos de um dado teríamos $\alpha = \{1, 2, 3, 4, 5, 6\}$, com cardinalidade $\overline{\overline{\alpha}} = 6$.

Qual a quantidade de informação enviada em uma mensagem deste tipo?

Um indivíduo razoavelmente envolvido profissionalmente com a área de computação naturalmente identificaria o conteúdo de informação de um objeto com o tamanho em bits do objeto, por exemplo, o tamanho de um arquivo de um banco de dados. Então, podemos rotular cada símbolo de α com números binários (strings de bits). Logo, poderíamos representar os n elementos de $\alpha = \{a_1, a_2, \ldots, a_n\}$ usando $\log n$ bits.

Surpreendentemente podemos ter tamanhos de código até menores que $\log n$. Por exemplo, seja $\alpha = \{a_1, a_2, a_3, a_4\}$ com probabilidades de ocorrência dos símbolos $\Pr(a_1) = 3/4$, $\Pr(a_2) = 1/8$, $\Pr(a_3) = 1/16$ e $\Pr(a_4) = 1/16$.

Podemos usar um sistema de codificação mais eficiente que o enumerativo. Este sistema se chama *código prefixado*. A Tabela 3.1 compara os dois tipos de codificação para o exemplo.

O número médio de bits transmitidos em uma mensagem $a_{s_1}, a_{s_2}, \ldots, a_{s_m}$ de tamanho m é $\frac{1}{m}\sum_{k=1}^{m} |E(a_{s_k})|$, onde $E(x)$ denota a palavra de código da mensagem x.

Definimos L, o tamanho do código, como sendo o número médio de bits transmitidos quando $m \to \infty$. Assim,

$$L = \lim_{m\to\infty} \frac{1}{m} \sum_{k=1}^{m} |E(a_{s_k})| = \sum_{i=1}^{\overline{\overline{\alpha}}} \Pr(a_i)\, |E(a_i)| \ .$$

Tabela 3.1: Codificação enumerativa e prefixada

Mensagem	Código Enumerativo	Código Prefixado
a_1	00	0
a_2	01	10
a_3	10	110
a_4	11	111

Calculando para o exemplo,

Código enumerativo $L = \frac{3}{4}2 + \frac{1}{8}2 + \frac{1}{16}2 + \frac{1}{16}2 = 2$ bits $(= \log \overline{\overline{\alpha}})$;
Código prefixado $L = \frac{3}{4}1 + \frac{1}{8}2 + \frac{1}{16}3 + \frac{1}{16}3 = 1,375$ bits $(< \log \overline{\overline{\alpha}})$.

Definição 1 *Uma* codificação binária $E : \alpha \rightarrow \{0,1\}^+$ *é um mapeamento do conjunto* α *de todas as mensagens para o conjunto de todas as strings binárias de tamanho maior que zero. O mapeamento* E *associa uma* palavra de código $E(x)$ *para cada mensagem* $x \in \alpha$.

Definição 2 *Seja* $x, y \in \{0,1\}^*$. *Nós chamamos* x *um* prefixo *de* y *se existe um* z *tal que* $y = xz$. *Um conjunto* $\beta \subseteq \{0,1\}^+$ *é* livre de prefixo *se nenhum elemento de* β *é prefixo de outro elemento de* β.

Definição 3 *Uma codificação binária é* livre de prefixo *se ela gera um conjunto de códigos livres de prefixo (por exemplo, veja o código prefixado da Tabela 3.1).*

Toda codificação livre de prefixo é dita *unicamente decodificável* e *instantaneamente decodificável*, significando que cada mensagem pode ser decodificada sem ambigüidades e sem a necessidade de referências a futuras palavras de código.

O Algoritmo 1 é um exemplo de algoritmo de codificação livre de prefixo (LI; VITÁNYI, 1997).

Algoritmo 1 *Podemos definir uma função simples para codificar strings com códigos livres de prefixo reservando um símbolo, neste caso o* 0*, como símbolo de finalização (LI; VITÁNYI, 1997). Nós podemos prefixar um objeto com o seu tamanho e repetir esta idéia para obter códigos mais curtos usando a função*

$$E_i(x) = \begin{cases} 1^x 0 & \text{para } i = 0 \\ E_{i-1}(|x|)x & \text{para } i > 0 \end{cases} .$$

Então, $E_1(x) = \overbrace{111\cdots1}^{|x| \text{ vezes}}0x = 1^{|x|}0x$, com tamanho $|E_1(x)| = 2|x| + 1$. Esta codificação é tão importante que merece uma notação própria, $\overline{x} = 1^{|x|}0x$.

Para obter códigos mais curtos podemos calcular $E_2(x)$, $E_2(x) = \overline{|x|}x$, com tamanho $|E_2(x)| = |x| + 2\big||x|\big| + 1$, onde $\big||x|\big|$ denota o tamanho em bits do tamanho de x, ou seja, $|l|$, onde $l = |x|$.

Nós chamamos $\overline{x}$ a versão auto-delimitada da string x. Strings binárias codificadas desta forma possuem a notável propriedade de conhecerem seu próprio tamanho. Assim, é possível obter x e y a partir de $\overline{x}y$ sem ambigüidades. Sejam as strings $x = 11001$

e $y = 101$, logo, $\overline{x}y = 11111011001101$, e podemos obter as strings originais lendo o tamanho da primeira string em seu prefixo.

Na teoria das probabilidades um sistema de eventos elementares $A_1, A_2, \ldots, A_n$ expressa um conjunto de eventos em que apenas um pode ocorrer em um resultado de um experimento (por exemplo, o aparecimento de 1,2,3,4,5 ou 6 no lançamento de um dado), e é chamado *espaço amostral*. Dados um sistema de eventos $A_1, A_2, A_3, \ldots, A_n$ junto com suas probabilidades $p_1, p_2, p_3, \ldots, p_n$ com $p_i \geq 0$ e $\sum_{i=1}^{n} p_i = 1$, chamamos

$$A = \begin{pmatrix} A_1 & A_2 & A_3 & \ldots & A_n \\ p_1 & p_2 & p_3 & \ldots & p_n \end{pmatrix} \tag{3.1}$$

de *esquema finito*.

É interessante apresentar uma medida do grau de incerteza associado a um experimento. Esta medida chama-se *entropia* (COVER; THOMAS, 1991; KHINCHIN, 1957).

O conceito de entropia originou-se na mecânica estatística, onde ela é definida como o logaritmo natural do número de estados de um ensemble, assim, $H = \ln N$. O uso do logaritmo objetivava definir uma escala logarítmica para a entropia. Shannon definiu a entropia usando o logaritmo na base dois (devido à codificação binária) e atribuindo pesos (probabilidades) a cada um dos estados do ensemble, $H = -p_1 \log p_1 - p_2 \log p_2 - \cdots - p_N \log p_N$. É fácil ver que se os estados são equiprováveis ($p_i = 1/N$), esta formulação se reduz a $H = \log N$.

Definição 4 *A entropia do esquema finito (3.1) é definida como*

$$H(p_1, p_2, p_3, \ldots, p_n) = -\sum_{k=1}^{n} p_k \log p_k \, .$$

Um resultado interessante de teoria da informação é a prova de que a entropia de um processo estocástico (fonte ergódica) (GRIMMETT; STIRZAKER, 1992) coincide com o tamanho de código de um código ótimo (que minimiza redundância na comunicação) para as mensagens da fonte (COVER; THOMAS, 1991). Apresentaremos este resultado no Teorema 25.

Uma ferramenta muito importante para prova de propriedades da entropia é a *Desigualdade de Jensen*, que pode ser aplicada sobre funções convexas (COVER; THOMAS, 1991; FIGUEIREDO, 1996; KHINCHIN, 1957).

Definição 5 *Seja I um intervalo não vazio em $\mathbb{R}$. Uma função ϕ é dita* convexa *em I quando $\phi[\lambda_1 x_1 + \lambda_2 x_2] \leq \lambda_1 \phi(x_1) + \lambda_2 \phi(x_2)$, para $\lambda_1, \lambda_2 \geq 0$, $\lambda_1 + \lambda_2 = 1$ e $x_1, x_2 \in I$.*

Para uma função convexa ϕ em I, o valor da função ϕ dentro do intervalo I está sempre abaixo do segmento $\overline{P_1 P_2}$, onde $P_1 = (x_1, \phi(x_1))$ e $P_2 = (x_2, \phi(x_2))$. Portanto, para que isto ocorra, basta que a primeira derivada da função seja sempre crescente no intervalo I.

Exemplo 1 *A função $\log(1/x)$ é convexa no intervalo $(0; 1]$, pois sua segunda derivada é $\log e/x^2$, que é sempre positiva no intervalo.*

Teorema 1 *(Desigualdade de Jensen) Seja I um intervalo não vazio em $\mathbb{R}$ e $\phi : I \to \mathbb{R}$ convexa em I. Então para qualquer conjunto $\{x_1, x_2, \ldots x_n\} \subset I$ e qualquer conjunto de números $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$ satisfazendo $\lambda_k \geq 0$, para $1 \leq k \leq n$ e $\sum_{k=1}^{n} \lambda_k = 1$, vale que*

$$\phi\left(\sum_{k=1}^{n} \lambda_k x_k\right) \leq \sum_{k=1}^{n} \lambda_k \phi(x_k)\,.$$

Prova: O enunciado é trivialmente verdadeiro para $n = 2$, $\phi(\lambda_1 x_1 + \lambda_2 x_2) \leq \lambda_1\phi(x_1) + \lambda_2\phi(x_2)$, pois esta é a própria definição de função convexa.

Supomos que a relação vale para $n-1$. Seja agora um conjunto $\{x_1, x_2 \ldots x_n\} \subset I$ e $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$ nas condições da hipótese. Observamos que se $\lambda_n = 0$ ou $\lambda_n = 1$ nada há para provar. Supomos então $\lambda_n \in (0;1)$, então $\sum_{k=1}^{n-1} \lambda_k \in (0;1)$ e $\sum_{k=1}^{n-1} \lambda_k = 1 - \lambda_n \in (0;1)$.

Fazemos $\lambda'_k = \frac{\lambda_k}{1-\lambda_n}$, para $k = 1, 2, \ldots n-1$. Observamos que $\sum_{k=1}^{n-1} \lambda'_k x_k \in I$.

Sendo assim, pela convexidade de ϕ em I segue que

$$\begin{aligned}\phi\left(\sum_{k=1}^{n} \lambda_k x_k\right) &= \phi\left((1-\lambda_n)\sum_{k=1}^{n-1} \lambda'_k x_k + \lambda_n x_n\right)\\ &\leq (1-\lambda_n)\phi\left(\sum_{k=1}^{n-1} \lambda'_k x_k\right) + \lambda_n\phi(x_n)\end{aligned}$$

e então, pela hipótese indutiva,

$$(1-\lambda_n)\phi\left(\sum_{k=1}^{n-1} \lambda'_k x_k\right) \leq (1-\lambda_n)\sum_{k=1}^{n-1} \lambda'_k \phi(x_k) = \sum_{k=1}^{n-1} \lambda_k \phi(x_k)\,.$$

Assim,

$$\phi\left(\sum_{k=1}^{n} \lambda_k x_k\right) \leq \sum_{k=1}^{n} \lambda_k \phi(x_k)\,.$$

□

Baseado na Desigualdade de Jensen podemos provar a seguinte propriedade da função convexa $\log 1/x$, que será muito útil mais adiante.

Teorema 2 *Seja $a_i, b_i \geq 0$, para $1 \leq i \leq n$, com $\sum_{i=1}^{n} a_i > 0$ e $\sum_{i=1}^{n} b_i > 0$. Então,*

$$\sum_{i=1}^{n} a_i \log\left[\frac{a_i}{b_i}\right] \geq \left[\sum_{i=1}^{n} a_i\right] \log\left[\frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i}\right]\,.$$

Prova: Seja $x_i = \frac{b_i}{a_i}$, com $y_i = \frac{a_i}{\sum_{j=1}^{n} a_j}$ que implica $\sum_{i=1}^{n} y_i = 1$. Segue que,

$$\sum_{i=1}^{n} a_i \log\left[\frac{1}{x_i}\right] = \left[\sum_{i=1}^{n} a_i\right] \sum_{i=1}^{n} y_i \log\left[\frac{1}{x_i}\right]\,.$$

Pela convexidade da função $\log(1/x)$, e aplicando a Desigualdade de Jensen, com $x_k = 1/x_i$ e $\lambda_k = y_i$ (observe que satisfaz o enunciado do Teorema 1, pois $\lambda_k \geq 0$ e $\sum_k \lambda_k = 1$), obtemos

$$\left[\sum_{i=1}^{n} a_i\right] \sum_{i=1}^{n} y_i \log\left[\frac{1}{x_i}\right] \geq \left[\sum_{i=1}^{n} a_i\right] \log\left[\frac{1}{\sum_{i=1}^{n} y_i x_i}\right] = \left[\sum_{i=1}^{n} a_i\right] \log\left[\frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i}\right]\,.$$

E está provado o Teorema. □

3.2 Complexidade de Kolmogorov

3.2.1 Complexidade Básica

Assumimos que existe um *método de especificação* f que associa no máximo um objeto x com uma descrição y (LI; VITÁNYI, 1997). 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}, se existir, 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, $|s_i| = \lfloor \log(i+1) \rfloor$. Lembramos que $|.|$ 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. Então, podemos representar objetos e descrições como strings binárias ou números naturais.

No que se segue $f : \mathbb{N} \to \mathbb{N}$ e $g : \mathbb{N} \to \mathbb{N}$ denotam duas funções parciais quaisquer.

Definição 6 *Seja $x \in A$ um objeto qualquer. A complexidade do objeto x com respeito a um método de especificação f é definida como*

$$C_f(x) = \min_{f(p)=n(x)} |p| .$$

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 da 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 7 *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 $C_f(x) \leq C_g(x) + c_{f,g}$.*

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 (CAMPANI; MENEZES, 2001a).

Definição 8 *Dois métodos f e g são* equivalentes *se* $\text{abs}(C_f(x) - C_g(x)) \leq c_{f,g}$*, onde $c_{f,g}$ não depende de x.* $\text{abs}(.)$ *denota o valor absoluto de um número.*

Dois métodos f e g são equivalentes se cada um minora o outro.

Por exemplo, suponha duas linguagens de programação tais como BASIC e PROLOG (LI; VITÁNYI, 1997). Como podemos escrever um interpretador BASIC em PROLOG e um interpretador PROLOG em BASIC, podemos dizer que

$$\text{abs}(C_{\text{BASIC}}(x) - C_{\text{PROLOG}}(x)) \leq c_{\text{BASIC,PROLOG}} ,$$

onde $c_{\text{BASIC,PROLOG}}$ depende apenas das linguagens BASIC e PROLOG. Seja π_{BASIC} o código necessário para interpretar BASIC em PROLOG e π_{PROLOG} o código necessário para interpretar PROLOG em BASIC, então $c_{\text{BASIC,PROLOG}}$ é $O(|\pi_{\text{BASIC}}|)+O(|\pi_{\text{PROLOG}}|)$, a soma dos tamanhos de π_{BASIC} e π_{PROLOG}.

Teorema 3 *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 sub-seqüência $q_1, q_2, q_3, \ldots$ 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. Originalmente alguns autores chamaram a complexidade de Kolmogorov de *entropia algorítmica* ou *informação algorítmica*.

Sejam s (sucessor), o (zero) e p_k (projeção) funções definidas como: $s(x) = x + 1$; $o(x_1, \ldots, x_n) = 0$; e $p_k(x_1, x_2, \ldots, x_k, \ldots, x_n) = x_k$. Seja $f(x_1, \ldots, x_n)$ definida usando-se as funções g e $h_1, h_2, \ldots h_n$, tal que

$$f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_n(x_1, \ldots, x_n))$$

e chamamos isto de *composição generalizada*. Nós definimos a função f por recursão primitiva (CLARK; COWELL, 1976) se, para funções g e h quaisquer:

$$\begin{aligned} f(x_1, \ldots, x_n, 0) &= g(x_1, \ldots, x_n), \\ f(x_1, \ldots, x_n, s(y)) &= h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)). \end{aligned}$$

Nós denotamos por $\mu_y(f(x_1, \ldots, x_{n-1}, y) = x_n)$ o menor número a para o qual $f(x_1, \ldots, x_{n-1}, a) = x_n$ e chamamos esta operação de minimização.

Definição 9 *Uma função f é chamada* função parcial recursiva *se ela é obtida das funções s, o e p_k pela aplicação de um número finito de operações de composição generalizada, recursão primitiva e minimização.*

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* (DIVERIO; MENEZES, 2000; HOPCROFT; ULLMAN, 1979). 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 4 *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 (DIVERIO; MENEZES, 2000), 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, $\phi_0(i, x) = \phi_i(x)$ (DAVIS, 1956; LI; VITÁNYI, 1997).

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 3.1). 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.

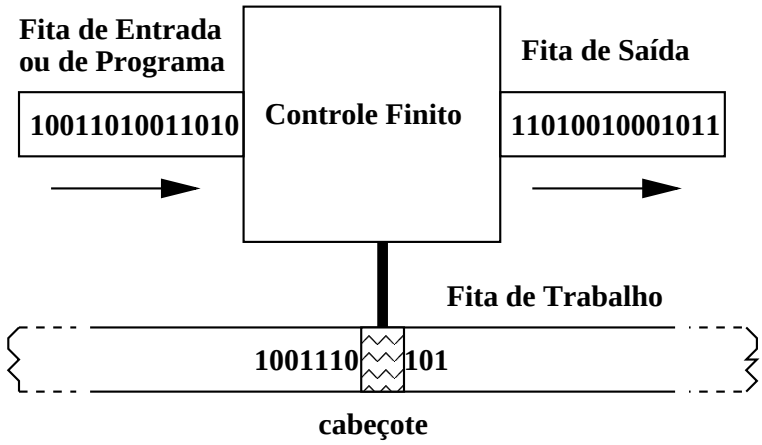

Figura 3.1: Computador concreto

Definição 10 *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 (DAVIS, 1956; GÁCS, 1993; KOLMOGOROV, 1965, 1968).

Nós codificaremos as máquinas da enumeração mencionada no Corolário 1 como

$$\overbrace{111\cdots1}^{i \text{ vezes}}0p = 1^i 0p \,,$$

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 5 *(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* $\mathcal{U}_{1^i 0p,y} = \mathcal{M}_{i_{p,y}}$*, 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

$$C_{\mathcal{U}}(x|y) = \min_{\mathcal{U}_{1^n 0p,y}=x} |1^n 0p| = \min_{\mathcal{U}_{1^n 0p,y}=x} |p| + n + 1 = C_{\mathcal{M}}(x|y) + n + 1 \,.$$

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 $\mathcal{M}$. □

Observe que podemos interpretar a desigualdade $C_{\mathcal{U}}(x|y) \leq C_{\mathcal{M}}(x|y) + c$ de duas formas diferentes:

$$\begin{array}{c} \text{Prova do Teorema} \\ \longrightarrow \\ C_{\mathcal{U}}(x|y) \leq C_{\mathcal{M}}(x|y) + c \ . \\ \longleftarrow \\ \text{Significado do Teorema} \end{array}$$

Interpretando a desigualdade da esquerda para a direita, obtemos a idéia utilizada na prova do Teorema 5, 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 caráter invariante e absoluto a esta medida de complexidade.

Observe que a constante c do Teorema 5, embora possa ser grande, é assintoticamente desprezável, pois não depende de x (KOLMOGOROV, 1965, 1968). 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 (LI; VITÁNYI, 1997). Em (CAMPANI; MENEZES, 2001a) 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* $\text{abs}(C_{\mathcal{U}}(x|y) - C_{\mathcal{V}}(x|y)) \leq c$*, onde c depende apenas de $\mathcal{U}$ e $\mathcal{V}$.*

Isto é verdade pois uma máquina simula a outra.

Teorema 6 *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 5 (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 7 *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 8 *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 $C(f(x)) \leq C_{\mathcal{M}}(f(x)) + c = C(x) + c$. □

Isto significa que a aplicação de uma função computável sobre uma string não pode aumentar a complexidade da string resultante em relação à original.

3.2.2 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 (VITÁNYI, 2004). 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*) (CHAITIN, 1974; KOLMOGOROV, 1965, 1968).

Esta definição desloca a discussão de fundamentos de teoria das probabilidades do conceito de aleatoriedade para o conceito de incompressividade (veja Capítulo 4).

Sejam as seguintes strings binárias:

```
11111111111111111111
01010101010101010101
01001101011000110101
```

Nossa intuição nos diz que as duas primeiras strings não podem ser aleatórias pois tem pequena probabilidade de ocorrer em uma seqüência de lançamentos de uma moeda honesta, enquanto que a última parece ser aleatória. No entanto, todas as três 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:

$$\texttt{FOR i} := \texttt{1 TO 20 PRINT 1}$$

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

$$\texttt{FOR i} := \texttt{1 TO } n \texttt{ PRINT 1}$$

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 última 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:

$$\texttt{PRINT 01001101011000110101}$$

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

Na Definição 11, a constante c desempenha o papel de "taxa de compressão".

Definição 12 *Se uma string x é c-incompressível para um valor $c > 0$, então nós dizemos que x é* incompressível.

É fácil provar a existência de strings binárias incompressíveis no sentido da Definição 12.

Teorema 9 *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 10 *(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.

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 6, $C(x) = C(\mathcal{M}_q) \leq C_{\mathcal{M}}(q) + O(1) \leq |q| + O(1)$, 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 $n - |v| = |u| + |w|$,

$$\begin{aligned} C(x) &\leq C(v) + 2\,|C(v)| + 2\big||u|\big| + n - |v| + O(1) \\ &\leq C(v) + 2\,|C(v)| + 2|n| + n - |v| + O(1)\,. \end{aligned}$$

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.

3.2.3 Algumas Propriedades da Complexidade Básica

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 $|x^*| = C(x)$. Lembre-se que x^* representa a menor descrição de x.

Como não podemos fazer desaparecer este termo logarítmico que depende de x e y, concluímos 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.

Veremos mais adiante que a complexidade de prefixo resolve este problema.

Teorema 11 *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 10, já que sabemos pelo Teorema 7 que $C(x|y) \leq C(x) + c$, sua complexidade é $C(x, y) > C(\langle x, y\rangle |\Lambda) \geq \log \overline{\overline{A}} - 1$. 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 6 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 12 *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 12 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, $C(1^m) > C(1^n)$.

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 (VITÁNYI, 2004).

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 6). Portanto, a complexidade $C(x)$ tem como limite superior a curva $\log x + c$ (veja Figura 3.2).

Teorema 13 $\lim_{x\to\infty} C(x) = \infty$.

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 3.2) (ZVONKIN; LEVIN, 1970).

Teorema 14 *Todo conjunto infinito enumerável contém um subconjunto infinito solúvel.*

Prova: Veja M. Li & P. Vitányi (LI; VITÁNYI, 1997), página 33, Lema 1.7.4. □

Teorema 15 *Para qualquer função parcial recursiva $f(x)$ tendendo monotonicamente a infinito a partir de um x_0, nós temos $m(x) < f(x)$.*

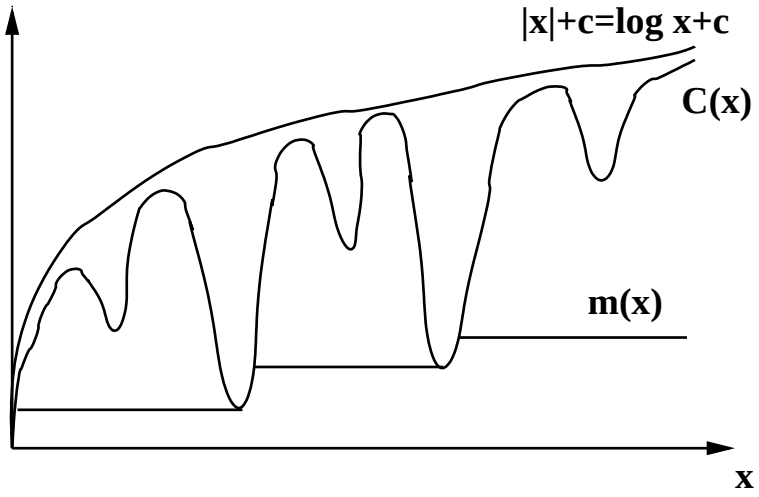

Figura 3.2: Funções $C(.)$ e $m(.)$

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 14, 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 8, $C(h(a)) \leq C(a) + O(1) = \log a + O(1)$. 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 16 *(Propriedade da Continuidade)* 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 17 *A função $C(.)$ não é parcial recursiva, além disto, nenhuma função parcial recursiva $\phi(.)$ definida sobre um conjunto infinito de pontos pode coincidir com $C(.)$ em todo seu domínio.*

Prova: Provaremos por contradição. Selecionamos um conjunto infinito solúvel $V \subset U$ no domínio U de definição de $\phi(.)$ (isto é possível pelo Teorema 14). A função $F(i) = \min_{C(x)\geq i, x\in V} x$ é uma função recursiva (total), já que $C(x) = \phi(x)$ em V. Além disto, $C(F(i)) \geq i$ (por construção). Pelo Teorema 8, $C(F(i)) \leq C(i) + O(1)$. Logo,

$$i \leq C(F(i)) \leq C(i) + O(1) \leq \log i + O(1)\,,$$

o que resulta em $i \leq \log i + O(1)$, que é falso. □

Definição 13 *Uma função f é* semi-computável *se existe uma função (total) recursiva ϕ tal que $\phi(x,t)$ é monotônica em t e $\lim_{t\to\infty}\phi(x,t) = f(x)$.*

Seja p um programa, t um número inteiro, então $\overline{\mathcal{U}}_p^t$ pode ser definido como o predicado que representa a computação na máquina de Turing universal $\mathcal{U}$ que computa o valor Φ_p em t passos e caso contrário resulta em um valor indefinido. Usando-se a Tese de Turing-Church é fácil provar que $\overline{\mathcal{U}}_p^t$ é decidível e que Φ_p é computável (GÁCS, 1993).

Corolário 3 *(Problema da Parada Limitado) $\overline{\mathcal{U}}_p^t$ é decidível e Φ_p é computável.*

Teorema 18 *A função C é semi-computável.*

Prova: Sabemos que $C(x) < \log x + c$, pois um número pode ser representado por sua codificação binária. Seja a máquina de Turing universal $\mathcal{U}^t$ definida como a máquina $\mathcal{U}$ que computa um programa p em t passos, caso contrário resulta em uma computação indefinida. Seja $C^t(x)$ um valor menor que $\log x + c$ e

$$C^t(x) = \min\left\{|p| \leq t : \mathcal{U}_p^t = x\right\}\,.$$

Isto implica que a função $C^t(x)$ é computável, não crescente, monotônica em t e $\lim_{t\to\infty} C^t(x) = C(x)$. □

3.3 Complexidade de Prefixo

A complexidade de prefixo é uma definição mais robusta de complexidade e apresenta uma série de vantagens sobre a complexidade básica C, entre elas uma relacionada com a *Desigualdade de Kraft*, como veremos adiante (CHAITIN, 1975).

Supomos que se $\mathcal{M}_{p,y}$ está definida então $\mathcal{M}_{q,y}$ não está definida para nenhum prefixo q de p. Obtemos isto através do uso de codificação livre de prefixo (veja o Algoritmo 1) e chamamos estes programas de *programas auto-delimitados*. A máquina $\mathcal{M}$ que recebe como entrada programas auto-delimitados é chamada de máquina de Turing de prefixo.

Definição 14 *Seja $\mathcal{M}$ uma máquina de Turing de prefixo. A* complexidade condicional de prefixo $K_{\mathcal{M}}(x|y)$ *de um número x com respeito a um número y é o comprimento da menor descrição p, representada como string livre de prefixo, tal que $\mathcal{M}_{p,y} = x$,*

$$K_{\mathcal{M}}(x|y) = \min_{\mathcal{M}_{p,y}=x} |p|\,.$$

Se não existe uma descrição p de x então dizemos, por definição, que $K_{\mathcal{M}}(x|y) = \infty$.

Da mesma forma como fizemos com a complexidade básica $C(.)$, podemos definir uma complexidade de prefixo incondicional $K(.)$. Também podemos apresentar um Teorema da Invariância para a complexidade K, cuja prova é similar à prova do Teorema da Invariância para a complexidade C.

Agora podemos provar que a complexidade de prefixo possui a propriedade subaditiva.

Teorema 19 *Para alguma constante $c > 0$ independente de x e y, $K(x, y) \leq K(x) + K(y) + c$.*

Prova: Se p é a menor descrição livre de prefixo de x e q é a menor descrição livre de prefixo de y então pq ou qp é a menor descrição de x e y. Existe uma máquina de Turing de prefixo $\mathcal{M}$ que recebendo pq ou qp recupera p e q (pela propriedade de strings livres de prefixo) e computa x e y. Assim, $K_{\mathcal{M}}(x, y) = |p| + |q|$. Aplicando o Teorema da Invariância, $K(x, y) \leq K_{\mathcal{M}}(x, y) + c = K(x) + K(y) + c$. □

Significa que $K(x, y) \leq K(x) + K(y) + \Delta$ e Δ é constante em relação a x e y.

É interessante que se investigue a relação que existe entre as complexidades C e K. Afirma-se no Teorema 20 que as complexidades C e K são assintoticamente iguais.

Teorema 20 *(M. Li & P. Vitányi (LI; VITÁNYI, 1997), exemplo 3.1.3, página 194) Para todo x e um y fixo, $C(x|y) \leq K(x|y) + O(1) \leq C(x|y) + 2\log C(x|y) + O(1)$.*

Prova: A primeira desigualdade, $C(x|y) \leq K(x|y)$, é trivial pois a complexidade de prefixo é definida como uma restrição da definição de programa. A segunda, $K(x|y) \leq C(x|y) + 2\log C(x|y) + O(1)$, pode ser provada da seguinte maneira.

Um programa para a máquina de prefixo possui em seu prefixo uma codificação auto-delimitada do seu tamanho. Seja $\mathcal{V}$ uma máquina universal de prefixo que simula $\mathcal{U}$. Logo, $K(x|y) \leq K_{\mathcal{V}}(x|y) + c = \left|\overline{|x^*|}x^*\right|$, e como $|x^*| = C(x|y)$, temos $K(x|y) \leq C(x|y) + 2\log C(x|y) + O(1)$. □

O Teorema 23 prova que H e K são equivalentes em um sentido formal (K converge a H em probabilidade) (COVER; THOMAS, 1991).

Definição 15 *Se a seqüência de funções mensuráveis f_n e a função mensurável f são finitas (FERNANDEZ, 1996), diz-se que f_n converge a f em probabilidade se e somente se, para todo $\epsilon > 0$,*

$$\lim_{n\to\infty} \Pr\left(\text{abs}(f_n - f) \geq \epsilon\right) = 0\,.$$

Isto significa que $f_n \to f$ em todos os pontos, a exceção de um conjunto de pontos que tem medida zero.

Teorema 21 *(Lei dos Grandes Números) Seja $X_1, X_2, X_3, \ldots$ uma seqüência de variáveis aleatórias independentes e identicamente distribuídas com média μ. Seja $\overline{X}_n = \frac{1}{n}\sum X_i$. Então, para qualquer $\varepsilon > 0$, $\lim_{n\to\infty} \Pr(\text{abs}(\overline{X}_n - \mu) \geq \varepsilon) = 0$.*

Prova: Veja (JAMES, 1996), Capítulo 5, e (LIPSCHUTZ, 1968), Teorema 5.11, página 87. □

Teorema 22 *(Propriedade da Equipartição Assintótica) Se as variáveis aleatórias*

$$X_1, X_2, \ldots, X_n$$

são independentes e identicamente distribuídas, então

$$-\frac{1}{n}\log \Pr(X_1X_2\cdots X_n) \to H(X)$$

em probabilidade.

Prova: Pela Lei dos Grandes Números e pela independência das variáveis X_i,

$$-\frac{1}{n}\log \Pr(X_1X_2\cdots X_n) = -\frac{1}{n}\log \prod_i \Pr(X_i) = -\frac{1}{n}\sum_i \log \Pr(X_i) \to H(X)$$

em probabilidade. □

Isto significa que as seqüências que não satisfazem a Propriedade da Equipartição Assintótica pertencem a um conjunto de medida zero. Assim, esta propriedade divide as seqüências em típicas (que respeitam a propriedade) e atípicas (que não respeitam a propriedade e pertencem a um conjunto de medida zero) (KHINCHIN, 1957).

Uma seqüência *típica* é aquela que converge e pertence a um conjunto de probabilidade um. Obviamente, as seqüências típicas respeitam a seguinte propriedade:

$$2^{-n(H(X)+\epsilon)} \leq \Pr(X_1X_2\cdots X_n) \leq 2^{-n(H(X)-\epsilon)}\,. \tag{3.2}$$

Seja $X_1, X_2, X_3, \ldots$ uma seqüência de variáveis aleatórias independentes e identicamente distribuídas, com cada $X_i = 0$ ou $X_i = 1$, para $i \geq 1$. Nós chamamos $X_1, X_2, X_3, \ldots$ de uma p-seqüência de Bernoulli se os X_i tomam o valor 1 com probabilidade p.

Teorema 23 *Seja a seqüência de variáveis aleatórias independentes e identicamente distribuídas $X_1, X_2, \ldots, X_n$ uma p-seqüência de Bernoulli (processo estocástico). Então, $\frac{1}{n}K(X_1X_2\cdots X_n) \to H(p, 1-p)$ em probabilidade.*

Prova: Primeiro devemos provar o seguinte Lema.

Lema 2 *Seja x uma string binária de tamanho $|x| = n$ que possui $k < n$ uns. Então, $K(x) \leq nH(k/n, 1-k/n) + 2\log k + c$.*

Prova: O menor programa que computa x de uma forma genérica deve gerar as strings de tamanho n que possuem k uns e imprimir a i-ésima. Para isto, este programa deve armazenar dentro dele o valor k, usando codificação livre de prefixo com tamanho $2\log k$ bits, e i usando $\log \binom{n}{k}$ bits ($\binom{n}{k}$ é o número máximo de combinações de strings possíveis com tamanho n e k uns). Então, $|x^*| = 2\log k + \log \binom{n}{k} + c$. Como sabemos que $\binom{n}{k} \leq 2^{nH(k/n, 1-k/n)}$ (resulta da aplicação da fórmula de Stirling (COVER; THOMAS, 1991) que afirma $n! \approx (2\pi n)^{\frac{1}{2}} e^{n\ln n - n}$), então $K(x) = |x^*| \leq 2\log k + nH(k/n, 1 - k/n) + c$. □

De acordo com a Definição 15, devemos primeiro provar que (condição necessária)

$$\Pr\left(\text{abs}\left(\frac{1}{n}K(X_1X_2\cdots X_n) - H(p, 1-p)\right) \geq \epsilon\right) \to 0\,. \tag{3.3}$$

Seja $\overline{X}_n = \frac{1}{n}\sum X_i$. Então, $K(X_1X_2\cdots X_n) \leq nH(\overline{X}_n, 1-\overline{X}_n) + 2\log n + c$, e como sabemos que $\overline{X}_n \to p$, a Equação (3.3) vale, com $\epsilon = \frac{2\log n + c}{n}$.

Seja T o conjunto das seqüências típicas e X o conjunto das seqüências que satisfazem $K(x) < n(H(p, 1-p) - c)$, para algum $c > 0$.

Para provar a condição suficiente, devemos perceber que as seqüências $x = X_1 X_2 \cdots X_n$ típicas satisfazem a Propriedade da Equipartição Assintótica, de forma que podemos afirmar que a probabilidade da complexidade de x ser menor que $n(H(p, 1-p) - c)$ para um $c > 0$ é

$$\begin{aligned} \Pr(K(x) < n(H(p,1-p)-c)) &\leq \Pr(x \notin T) + \Pr(x \in T \cap X) \\ &\leq \epsilon + \sum_{x \in T \cap X} \Pr(x) \\ &\leq \epsilon + \sum_{x \in X} 2^{-n(H(p,1-p)-\epsilon)} \\ &\leq \epsilon + 2^{n(H(p,1-p)-c)} 2^{-n(H(p,1-p)-\epsilon)} \\ &= \epsilon + 2^{-n(c-\epsilon)} \end{aligned} \tag{3.4}$$

que é arbitrariamente pequeno, provando a convergência. Note que (3.4) segue de (3.2). □

A tentativa original de Kolmogorov de definir seqüências aleatórias infinitas como sendo aquelas que possuem todos os seus prefixos incompressíveis falhou, devido à flutuação da complexidade (LI; VITÁNYI, 1997; VITÁNYI, 2004). Tais seqüências não existem. Porém, usando-se a complexidade K é possível obter tal definição de uma forma correta (VITÁNYI, 2004).

Definição 16 *Seja x uma string binária infinita, então x é aleatória se $K(x_{1:n}) \geq n - c$, para todos os n e algum $c > 0$.*

Ou seja, uma string binária é aleatória se todos os seus prefixos são incompressíveis pela complexidade K.

3.4 Probabilidade Algorítmica

3.4.1 Computações Aleatórias

Suponha uma máquina de Turing determinística $\mathcal{M}$ (SOLOMONOFF, 1997). Suponha que p é um programa binário que descreve uma dada linguagem L. O programa p sempre espera encontrar concatenado à sua direita um número n, representado em binário, que denota a string $s_n \in L$ que está na n-ésima posição em uma particular e arbitrária enumeração das palavras da linguagem L. Assim, $\mathcal{M}_{pn} = s_n$. Nós não vamos nos preocupar como p encontra n na fita de entrada da máquina. A Figura 3.3 (a) ilustra a situação da fita da máquina neste caso.

Podemos generalizar o que foi definido acima para uma *máquina probabilística*, abandonando a máquina determinística. Suponha a distribuição uniforme (medida de Lebesgue). Seja p um programa que descreve uma linguagem probabilística e r uma string binária infinita aleatória (veja Figura 3.3 (b)). Então, $\mathcal{M}_{pr} = s$ representa a probabilidade de s ocorrer na linguagem L (Figura 3.3 (c)).

Como r é uma string infinita, devemos adaptar nossa definição de máquina de Turing provendo espaço para armazenar r. A expressão pr é formada por uma parte aleatória precedida por uma descrição determinística da linguagem L.

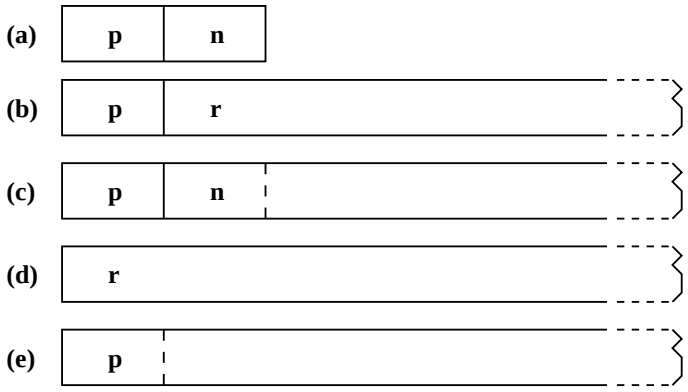

Figura 3.3: Fita de entrada da máquina de Turing

Podemos ir em frente imaginando que a parte p foi retirada da fita da máquina (Figura 3.3 (d)). Agora, $\mathcal{M}_r$ representa uma computação puramente aleatória. Se $|p|$ é o tamanho em bits da descrição p, então r tem probabilidade $2^{-|p|}$ de começar com p e gerar a linguagem L (Figura 3.3 (e)).

Assim, $\mathcal{M}_r$ é uma distribuição de probabilidade a priori sobre todas as strings binárias finitas e a probabilidade de cada string s pode ser medida por $2^{-|p|}$, onde p é o programa que computa s. Mas, se mais de um programa computar s (usualmente infinitos farão isto)? Como a ocorrência de cada um dos programas representa um evento independente, a probabilidade de ocorrência de uma string s é a soma destas probabilidades, $\sum_p 2^{-|p|}$. No entanto, devemos garantir que esta série seja convergente. Veremos mais adiante como isto será tratado.

3.4.2 Definição da Probabilidade Algorítmica

A complexidade de Kolmogorov provê uma definição formal de "simplicidade". Uma string binária é dita simples se ela é regular, ou seja, se $K(x) \ll |x|$. Ao contrário, uma string complexa é aquela em que $K(x) \geq |x| - c$, para algum $c > 0$ (neste caso se diz que x é *c-incompressível*). Um objeto "regular" é aquele com significativamente menos complexidade que a máxima.

Podemos identificar x^* com uma teoria para x. A construção de teorias na ciência é basicamente a compressão de grandes quantidades de informações sobre os fenômenos estudados em uma pequena quantidade de bits que modelam aqueles fenômenos (teoria, programa para a máquina de Turing) (CHAITIN, 1974).

Através de um argumento simples de contagem podemos provar que o número de objetos regulares é pequeno e, portanto, considerando a distribuição uniforme (medida de Lebesgue), estes eventos regulares têm pequena probabilidade de ocorrer (LI; VITÁNYI, 1997).

Suponha que você observasse uma string binária x de tamanho n e desejasse saber se devemos atribuir a ocorrência do evento x como puro acaso ou a uma causa. Devemos formalizar o que entendemos por "acaso" e "causa" (LI; VITÁNYI, 1997).

Acaso Obtenção de x pelo uso de uma moeda honesta (probabilidade 2^{-n});

Causa Significando que uma máquina de Turing (computador) computou x através de um programa aleatório (obtido pelo lançamento de uma moeda honesta).

Ambas formam duas distribuições diversas. Na primeira (distribuição uniforme), objetos irregulares são os típicos; na segunda, os objetos regulares é que têm máxima probabilidade de ocorrer. Veremos que a segunda probabilidade pode ser aproximada para $2^{-K(x)}$. Se x é regular, então possui baixa complexidade. Assim, $K(x) \ll n$ e a razão entre acaso e causa é $2^{n-K(x)}$.

Suponha que um Orangotango Eterno[1] fosse posto na frente de um teclado. A saída deste teclado é uma seqüência de bits. Qual seria a probabilidade deste Orangotango, ao teclar aleatoriamente, obter uma obra completa de Érico Veríssimo com 1000000 de bits (esta probabilidade é positiva, veja (JAMES, 1996), página 202, Corolário e Exemplo 3)?

Dentre todas as possíveis $2^{1000000}$ strings binárias de 1000000 de bits, a probabilidade de ser obtida a string x_0, que é a obra de Érico Veríssimo, é $\Pr(x_0) = 2^{-1000000}$.

Por outro lado, se este Orangotango Eterno estivesse teclando programas aleatórios para serem executados em uma máquina de Turing $\mathcal{U}$, a probabilidade da máquina computar a string x_0 da obra de Érico Veríssimo seria $P_{\mathcal{U}}(x_0) = 2^{-K(x_0)}$. Podemos supor que $P_{\mathcal{U}}(x_0) = 2^{-250000}$ é uma boa aproximação de $P_{\mathcal{U}}$ (empiricamente sabemos que um texto em português pode ser comprimido na proporção de 1 para 4). Esta probabilidade, embora ainda pequena, é 2^{750000} maior que $2^{-1000000}$.

Neste sentido, podemos interpretar a máquina de Turing como um construtor de significados. Ela é como um filtro que transforma desordem em ordem.

Seja $P_{\mathcal{U}}(x)$ a probabilidade de uma máquina de Turing universal $\mathcal{U}$ computar x quando é fornecido um programa obtido aleatoriamente, por lançamentos de uma moeda,

$$P_{\mathcal{U}}(x) = \sum_{\mathcal{U}_p = x} 2^{-|p|} \, .$$

No entanto, esta definição tem um grave problema, pois $\sum_{\mathcal{U}_p=x} 2^{-|p|}$ é uma série que diverge. Sabemos que para todo n existe um x_0, $|x_0| = n$, tal que $C(x_0) = O(\log n)$. Isto significa que para todo n existe um x_0 tal que $P_{\mathcal{U}}(x_0) \approx 2^{-\log n} = 1/n$. Portanto, como a soma infinita $1/2 + 1/3 + 1/4 + \cdots$ diverge, assim também $P_{\mathcal{U}}(x)$ diverge.

Definição 17 *(Desigualdade de Kraft) Seja α um conjunto de códigos, então*

$$\sum_{x \in \alpha} \left(\frac{1}{2} \right)^{|x|} \leq 1 \, .$$

Estamos interessados em provar que as codificações livres de prefixo satisfazem esta desigualdade (COVER; THOMAS, 1991).

Teorema 24 *Toda codificação livre de prefixo E satisfaz a desigualdade de Kraft.*

Prova: Mostramos na Figura 3.4 uma árvore que ilustra todos os possíveis caminhos para obter uma string binária. A árvore segue infinitamente para baixo. Cada caminho representa uma string binária. Podemos ver na Figura 3.5 os pontos marcados indicando três códigos livres de prefixo, desde que nos caminhos abaixo dos pontos não exista nenhum outro código. São eles 00, 010 e 1.

Uma vez chegando a um ponto de bifurcação na árvore, devemos tomar uma de duas ações:

[1] "O Orangotango Eterno de Dee, munido de uma pena resistente, de tinta que bastasse e de uma superfície infinita, acabaria escrevendo todos os livros conhecidos, além de criar algumas obras originais", *Borges e os Orangotangos Eternos*, Luis Fernando Verissimo, Companhia das Letras, 2000, página 70.

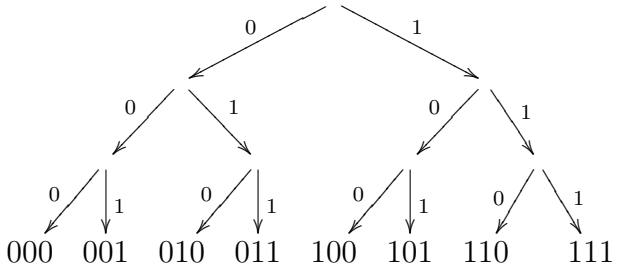

Figura 3.4: Árvore da codificação binária

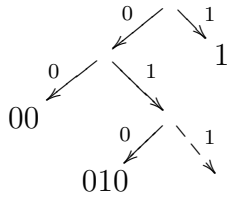

Figura 3.5: Árvore mostrando códigos livres de prefixo

- *Neste ponto já foi encontrado um código, então pare;*
- *Ainda não é um código, então mova-se mais um passo para baixo com* $\Pr(0) = \Pr(1) = \frac{1}{2}$.

Após caminhar n *passos na árvore nós podemos ter terminado em algum nível* $m < n$ *se encontramos um código (pois os códigos são livres de prefixo), ou podemos ter atingido o nível* n *sem encontrar um código. De qualquer forma a probabilidade de qualquer caminho na árvore pode ser calculada como:*

- *Caminho com* n *bits (não encontrou código):* $\Pr(x_1 x_2 \cdots x_n) = \left(\frac{1}{2}\right)^n$*;*
- *Caminho com* $m < n$ *bits (encontrou código):* $\Pr(x_1 x_2 \cdots x_m) = \left(\frac{1}{2}\right)^m$.

Pela propriedade da conservação da probabilidade *a soma das probabilidades de qualquer nível da árvore deve ser igual a 1. Ilustramos esta propriedade na Figura 3.6. Nesta figura apresentamos uma árvore hipotética em que o desenvolvimento de três níveis da árvore resultou em uma codificação encontrada no nível 2 (código 01). Podemos somar as probabilidades obtendo*

$$\frac{1}{8} + \frac{1}{8} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = 1\,.$$

Desta forma podemos dizer que

$$\sum_{\text{códigos com } |x|<n} \left(\frac{1}{2}\right)^{|x|} + \sum_{\text{caminhos ainda não terminados}} \left(\frac{1}{2}\right)^n = 1\,.$$

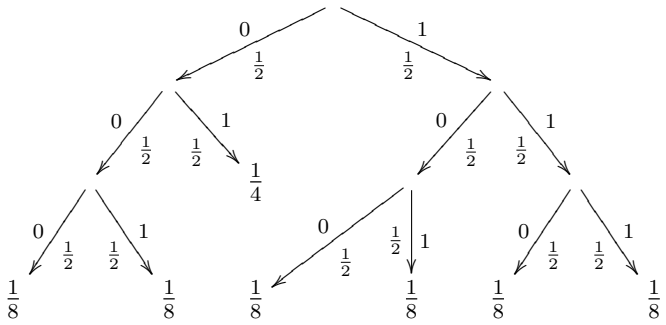

Figura 3.6: Árvore mostrando a conservação da probabilidade

Como

$$\sum_{\text{caminhos ainda não terminados}} \left(\frac{1}{2}\right)^n \geq 0\,,$$

segue que

$$\sum_{\text{códigos com } |x|<n} \left(\frac{1}{2}\right)^{|x|} \leq 1\,.$$

Tomando o $\lim_{n\to\infty}$ *obtemos*

$$\lim_{n\to\infty} \sum_{\text{códigos com } |x|<n} \left(\frac{1}{2}\right)^{|x|} = \sum_{x\in\alpha} \left(\frac{1}{2}\right)^{|x|}\,,$$

e está completa a prova. □

Teorema 25 *Existe um código livre de prefixo* E *que codifica mensagens* $x \in \alpha$ *tal que*

$$H(X) \leq L(E) < H(X) + 1\,.$$

Prova: Primeiro provamos a desigualdade $H(X) \leq L(E)$*. Subtraindo os dois lados da desigualdade*

$$\begin{aligned} L(E) - H(X) &= \sum_{x\in\alpha} p(x)\,[|x| + \log p(x)] = \\ &= \sum_{x\in\alpha} p(x) \left[\log p(x) - \log \left(\frac{1}{2}\right)^{|x|}\right] = \\ &= \sum_{x\in\alpha} p(x) \log \frac{p(x)}{\left(\frac{1}{2}\right)^{|x|}}\,. \end{aligned}$$

Aplicando a desigualdade característica da função convexa $\log(1/x)$ *(veja Teorema 2), com* $a_i = p(x)$ *e* $b_i = (1/2)^{|x|}$*, obtemos*

$$\sum_{x\in\alpha} p(x) \log \frac{p(x)}{\left(\frac{1}{2}\right)^{|x|}} \geq \left[\sum_{x\in\alpha} p(x)\right] \log \left[\frac{\sum_{x\in\alpha} p(x)}{\sum_{x\in\alpha} \left(\frac{1}{2}\right)^{|x|}}\right]\,.$$

Como sabemos que $\sum_{x\in\alpha} p(x) = 1$

$$\left[\sum_{x\in\alpha} p(x)\right] \log \left[\frac{\sum_{x\in\alpha} p(x)}{\sum_{x\in\alpha}\left(\frac{1}{2}\right)^{|x|}}\right] = -\log\left[\sum_{x\in\alpha}\left(\frac{1}{2}\right)^{|x|}\right] \geq 0\,,$$

pois pela desigualdade de Kraft $\sum_{x\in\alpha}\left(\frac{1}{2}\right)^{|x|} \leq 1$*, e está provada a primeira desigualdade.*

Fica evidente que a melhor codificação possível é aquela em que $|x| = \log(1/p(x))$*. Escolhemos* $|x| = \lfloor \log(1/p(x)) \rfloor$*. Logo,*

$$L(E) = \sum_{x\in\alpha} p(x)\left\lfloor \log\frac{1}{p(x)}\right\rfloor < \sum_{x\in\alpha} p(x)\left[\log\frac{1}{p(x)} + 1\right] = H(X) + 1\,,$$

pois $\lfloor x \rfloor < x + 1$. □

Toda codificação que satisfaz a desigualdade de Kraft é livre de prefixo (e unicamente decodificável) e o conjunto de tamanhos de código satisfaz $|x| = \left\lceil \log\frac{1}{\Pr(x)}\right\rceil$.

Uma das preocupações que devemos ter é de como obter códigos unicamente decodificáveis ótimos em relação ao tamanho médio das palavras de código. Um algoritmo bem sucedido é a *codificação de Shannon-Fano* (COVER; THOMAS, 1991). O princípio básico da codificação de Shannon-Fano é definir o código baseado na função de distribuição cumulativa $F(x)$. Seja $a_1, a_2, a_3, \ldots, a_n$ uma ordenação de todos os elementos do conjunto de mensagens α, então $F(a_i) = \sum_{j:j\leq i} \Pr(a_j)$.

Seja

$$\overline{F}(a_i) = \frac{1}{2}\Pr(a_i) + \sum_{j:j<i} \Pr(a_j)\,,$$

representando a soma das probabilidades de todas mensagens menores, na ordenação, que a_i mais a metade da probabilidade de a_i. A ordenação não necessita seguir qualquer ordem preestabelecida (crescente, decrescente, etc.).

Se $F(x)$ é uma função em degrau, $\overline{F}(x)$ representa a metade do degrau correspondente à mensagem x.

Em geral $\overline{F}(x)$ é um número real que só pode ser expresso com um número infinito de bits. Se arredondarmos $\overline{F}(x)$ para $l = \lceil \log\frac{1}{\Pr(x)}\rceil$ bits obteremos um número que ainda permanece no intervalo do degrau. Assim l bits é suficiente para descrever x.

Podemos provar que tal codificação é livre de prefixo (e portanto unicamente decodificável) e que o tamanho médio do código L satisfaz $L < H + 2$. Assim, este código está a um bit do ótimo que é $L < H + 1$.

A Tabela 3.2 apresenta um exemplo de construção do código para cinco mensagens a_1, a_2, a_3, a_4 e a_5 com probabilidades 0,125, 0,5, 0,25, 0,0625 e 0,0625.

O tamanho médio das palavras de código obtido neste exemplo é $2,875$. Observe que, neste caso, o último bit dos códigos correspondentes a a_4 e a_5 pode ser eliminado, obtendo assim um código mais eficiente (ou mais curto).

Podemos agora, baseados em codificação livre de prefixo, redefinir $P_{\mathcal{U}}(x)$ de forma a restringir que todos os programas sejam programas autodelimitados e que $\mathcal{U}$ seja uma máquina de prefixo. Isto não é tão difícil de aceitar pois as linguagens de programação usualmente são delimitadas por um comando de finalização (por exemplo, no PASCAL, o `END.`).

Tabela 3.2: Exemplo de construção do código de Shannon-Fano

x	$\Pr(x)$	$F(x)$	$\overline{F}(x)$	$\overline{F}(x)$ em binário	$l = \lceil \log \frac{1}{\Pr(x)} \rceil$	código
a_1	0,125	0,125	0,0625	0,0001	4	0001
a_2	0,5	0,625	0,375	0,011	2	01
a_3	0,25	0,875	0,75	0,11	3	110
a_4	0,0625	0,9375	0,90625	0,11101	5	11101
a_5	0,0625	1,0	0,96875	0,11111	5	11111

Seja $P_{\mathcal{U}}(x)$ definida como a probabilidade de uma máquina de Turing universal de prefixo computar x quando é fornecido um programa autodelimitado aleatório,

$$P_{\mathcal{U}}(x) = \sum_{\mathcal{U}_p = x} 2^{-|p|} \, .$$

Pela desigualdade de Kraft é evidente que $P_{\mathcal{U}}(x)$ é uma probabilidade, pois a série acima converge e $0 \leq P_{\mathcal{U}}(x) \leq 1$. Baseados nos resultados desta seção, podemos afirmar que $\sum_x 2^{-C(x)}$ é uma série divergente, enquanto que $\sum_x 2^{-K(x)}$ converge.

Teorema 26 *$P_{\mathcal{U}}(.)$ não é computável.*

Prova: Podemos simular todos os programas para a máquina $\mathcal{U}$ na ordem lexicográfica e coletar todos os programas que param e computam uma dada string. É fácil ver que, pelo problema da parada, esta estimativa não é computável, pois não podemos determinar quais programas não irão parar. □

3.4.3 Probabilidade Universal

Seja $\alpha = \{0, 1\}$ (alfabeto binário). Então, $\alpha^* = \{\Lambda, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. Considere o conjunto das seqüências binárias infinitas unidirecionais $\{0, 1\}^\infty$. Se $a \in \{0, 1\}^\infty$ então $a = a_1 a_2 a_3 \ldots$ e $a_{1:n} = a_1 a_2 \ldots a_n$. Cada elemento de $\{0, 1\}^\infty$ corresponde a um elemento $r \in \mathbb{R}$, $r = \sum_i \frac{a_i}{2^i}$ e r converge com $0 \leq r \leq 1$.

O conjunto de todas as seqüências infinitas unidirecionais que tem x como prefixo é chamado de *cilindro* Γ_x, $\Gamma_x = \{a : a_{1:|x|} = x \text{ e } a \in \{0, 1\}^\infty\}$. O cilindro pode ser interpretado como o intervalo $[0.x; 0.x + 2^{-|x|})$, em $[0; 1)$. O tamanho do cilindro Γ_x é $2^{-|x|}$.

Devemos atribuir probabilidades a estes cilindros. Para isto, definimos uma σ-álgebra gerada pelos intervalos (esta σ-álgebra é chamada de σ-álgebra de Borel) e uma medida de probabilidade para esta σ-álgebra (FERNANDEZ, 1996). Usaremos como medida o tamanho dos cilindros. Para simplificar a notação usamos $\mu(x)$, ao invés de $\mu(\Gamma_x)$, para indicar a medida do intervalo Γ_x.

Definição 18 *Uma função $\mu : \alpha^* \to \mathbb{R}$ é uma* semi-medida *se*

$$\begin{aligned} \mu(\Lambda) &\leq 1 \, ; \\ \mu(x) &\geq \sum_{a \in \alpha} \mu(xa) \, . \end{aligned}$$

Podemos compensar as desigualdades, obtendo novamente igualdades, ao supor a existência de um elemento indefinido u, $u \notin \alpha$, tal que $\mu(\Lambda) + \mu(u) = 1$ e $\mu(x) = \mu(xu) + \sum_{a \in \alpha} \mu(xa)$. u concentrará o excesso de ambas as desigualdades transformando-as em igualdades. O conceito de semi-medida não pertence à teoria da medida clássica (LI; VITÁNYI, 1997). Uma semi-medida é dita *discreta* se seu espaço amostral é discreto.

Definição 19 *Seja $\mathcal{C}$ uma classe de semi-medidas discretas. Uma semi-medida μ_0 é* universal *(ou* máxima*) para $\mathcal{C}$ se $\mu_0 \in \mathcal{C}$ e para todo $\mu \in \mathcal{C}$ existe uma constante $c_\mu > 0$ tal que $c_\mu \mu_0(x) \geq \mu(x)$, com c_μ independente de x.*

Se $c_\mu \mu_0(x) \geq \mu(x)$ para todo x então nós dizemos que μ_0 *domina* (multiplicativamente) μ.

Vamos aproximar o resultado da avaliação de funções de valor real usando valores racionais p/q, representados como pares (p, q), para $p, q \in \mathbb{N}$.

Definição 20 *Uma função real f é dita* enumerável *se existe uma função recursiva $\phi(x, t)$, não decrescente em t, com $\lim_{t \to \infty} \phi(x, t) = f(x)$.*

Ou seja, uma função enumerável é aquela que pode ser aproximada por baixo.

Definição 21 *Uma função real f é dita* co-enumerável *se existe uma função recursiva $\phi(x, t)$, não crescente em t, com $\lim_{t \to \infty} \phi(x, t) = f(x)$.*

Uma função co-enumerável é aquela que pode ser aproximada por cima. Se f é enumerável então $-f$ é co-enumerável. Se uma função f é enumerável e co-enumerável então ela é dita *recursiva*. Por exemplo, $K(.)$ e $C(.)$ são funções co-enumeráveis, e $-K(.)$, $-C(.)$ e $P_{\mathcal{U}}(.)$ são funções enumeráveis, mas nenhuma delas é recursiva.

Definição 22 *Uma medida μ é dita* enumerável *se existe uma função recursiva $\phi(x, t)$, não decrescente em t, tal que $\lim_{t \to \infty} \phi(x, t) = \mu(x)$.*

Muito do que será discutido a seguir é a prova da existência de uma semi-medida universal $\mathbf{m}$ na classe das semi-medidas enumeráveis discretas (LI; VITÁNYI, 1997).

Primeiramente, devemos provar que a complexidade $K(x)$ e a probabilidade $P_{\mathcal{U}}(x)$ são medidas equivalentes, ou seja, $2^{-K(x)} \approx P_{\mathcal{U}}(x)$ (COVER; THOMAS, 1991).

Teorema 27 *Existe uma constante c, tal que para todo x, $2^{-K(x)} \leq P_{\mathcal{U}}(x) \leq c 2^{-K(x)}$, onde c não depende de x.*

Prova: A primeira desigualdade é fácil de ser provada. Seja x^* o menor programa que computa x, então $P_{\mathcal{U}}(x) = \sum_{\mathcal{U}_p = x} 2^{-|p|} \geq 2^{-|x^*|} = 2^{-K(x)}$.

Podemos reescrever a segunda desigualdade como $K(x) \leq \log \frac{1}{P_{\mathcal{U}}(x)} + c'$.

Um dos problemas com esta parte da prova é que não podemos computar $P_{\mathcal{U}}(x)$, então a estratégia da prova é encontrar o menor programa que descreve strings x com alta probabilidade $P_{\mathcal{U}}(x)$.

Para isto, usaremos codificação de Shannon-Fano e construiremos a árvore de códigos de Shannon-Fano de forma a satisfazer a desigualdade de Kraft. À medida que esta árvore é construída teremos aproximações cada vez melhores de $P_{\mathcal{U}}(x)$. Nós não conhecemos previamente a profundidade dos nodos na árvore, mas podemos sucessivamente atribuir x aos nodos da árvore mais próximos da raiz, à medida que a nossa estimativa de $P_{\mathcal{U}}(x)$ melhora.

Supomos uma enumeração dos programas $p_1, p_2, p_3, \ldots$. Então, nós construimos uma máquina $\mathcal{M}$ que executa um passo da computação do primeiro programa simulando a computação do programa na máquina $\mathcal{U}$. A seguir, executamos dois passos da computação dos dois primeiros programas e assim sucessivamente, de forma que em algum momento $\mathcal{M}$ estará executando i passos da computação dos primeiros i programas da enumeração, simulando a máquina $\mathcal{U}$. Ou seja, $\mathcal{M}$ executará o Algoritmo 2.

Algoritmo 2 *(Determina os p_k e x_k)*

1. *Faça i:=0 e k:=1;*
2. *Faça i:=i+1. Gere os primeiros i programas da enumeração. Simule a computação de i passos de cada um dos i programas gerados em $\mathcal{U}$. Se um programa pára, armazene o programa p_k e a sua saída x_k e faça k:=k+1. Vá para 2.*

Desta forma, eventualmente a máquina irá executar todos os possíveis programas por um tempo cada vez mais longo, de forma que se um programa eventualmente pára, nós descobriremos.

Assim, podemos obter a lista ordenada $p_1, p_2, p_3, \ldots$ de todos os programas que param junto com as suas saídas.

Para cada programa p_k e a sua saída x_k calculamos uma estimativa de $P_{\mathcal{U}}(x_k)$,

$$\tilde{P}_{\mathcal{U}}(x_k) = \sum_{\mathcal{U}_{p_i}=x_k : i \leq k} 2^{-|p_i|},$$

e então calculamos o número de bits n_k necessários para descrever x_k baseados na estimativa atual de $P_{\mathcal{U}}(x_k)$,

$$n_k = \left\lceil \log \frac{1}{\tilde{P}_{\mathcal{U}}(x_k)} \right\rceil .$$

Podemos então construir a árvore, garantindo que todos os n_k de um determinado x_k são distintos pela remoção dos nodos em que ocorre coincidência de x_k e n_k. Isto garante que existe no máximo um nodo em cada nível da árvore que corresponde a um dado x_k. Sempre que um nodo é usado todos os nodos abaixo dele ficam indisponíveis, garantindo que a atribuição de nodos é livre de prefixo.

Ilustramos este processo com o seguinte Exemplo.

Exemplo 2 *Sejam os programas, suas saídas e os cálculos de $\tilde{P}_{\mathcal{U}}(x_k)$ e n_k ilustrados na Tabela 3.3.*

A partir dos dados da tabela podemos construir a árvore como ilustra a Figura 3.7. Observe que na árvore x_5 foi descartado, pois $x_3 = x_5$ e $n_3 = n_5$, e x_6 foi renomeado como x_5.

Usando a árvore obtida podemos concluir que

$$\begin{aligned}
\sum_{k:x_k=x} 2^{-(n_k+1)} &= 2^{-1} \sum_{k:x_k=x} 2^{-n_k} \\
&\leq 2^{-1} \left(2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor} + 2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor - 1} + 2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor - 2} + \cdots \right) \qquad (3.5) \\
&= 2^{-1} 2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor} \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots \right) \\
&= 2^{-1} 2^{\lfloor \log P_{\mathcal{U}}(x) \rfloor} 2 \\
&\leq P_{\mathcal{U}}(x) ,
\end{aligned}$$

Tabela 3.3: Construção da Árvore

k	p_k	x_k	$\tilde{P}_{\mathcal{U}}(x_k)$	n_k	$Comentário$
1	101	10	$\tilde{P}_{\mathcal{U}}(x_1) \geq 2^{-\lvert p_1\rvert} = 2^{-3} = 0{,}125$	3	
2	100011	1101	$\tilde{P}_{\mathcal{U}}(x_2) \geq 2^{-\lvert p_2\rvert} = 2^{-6} = 0{,}015625$	6	
3	11	10	$\tilde{P}_{\mathcal{U}}(x_3) \geq 2^{-\lvert p_1\rvert} + 2^{-\lvert p_3\rvert} = 2^{-3} + 2^{-2} = 0{,}375$	2	
4	1001	111	$\tilde{P}_{\mathcal{U}}(x_4) \geq 2^{-\lvert p_4\rvert} = 2^{-4} = 0{,}0625$	4	
5	10000	10	$\tilde{P}_{\mathcal{U}}(x_5) \geq 2^{-\lvert p_1\rvert} + 2^{-\lvert p_3\rvert} + 2^{-\lvert p_5\rvert} = 2^{-3} + 2^{-2} + 2^{-5} = 0{,}40625$	2	descartado
6	1000101	11111	$\tilde{P}_{\mathcal{U}}(x_6) \geq 2^{-\lvert p_6\rvert} = 2^{-7} = 0{,}0078125$	7	
⋮	⋮	⋮	⋮	⋮	⋮

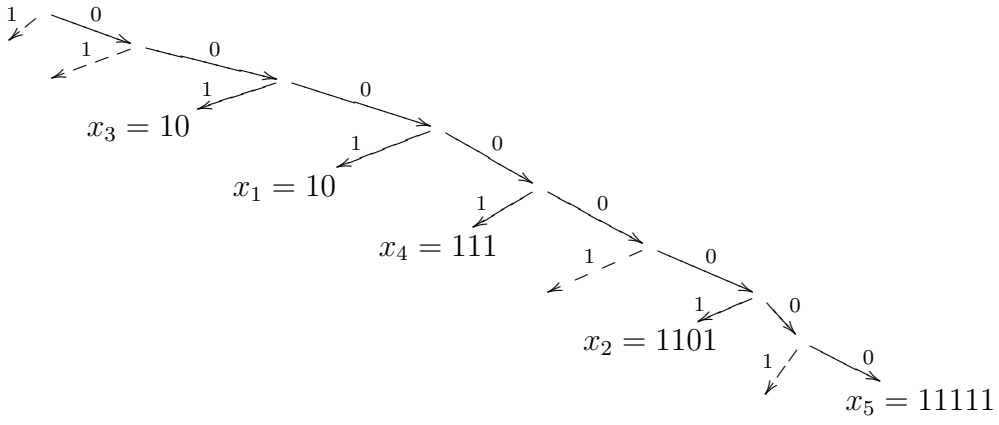

Figura 3.7: Árvore Construída Parcialmente

onde (3.5) é verdadeiro pois existe no máximo um nodo em cada nível da árvore que computa x, e é assumido que o limite superior da soma é aquele em que todos os níveis estão alocados para x.

Portanto,

$$\sum_{k=1}^{\infty} 2^{-(n_k+1)} \leq \sum_{x\in\{0,1\}^*} P_{\mathcal{U}}(x) \leq 1\,.$$

A última desigualdade acima justifica-se pela desigualdade de Kraft.

Ou seja, existe um procedimento efetivo para construir a árvore a partir das aproximações de $P_{\mathcal{U}}(x)$, e $\tilde{P}_{\mathcal{U}}(x) \uparrow P_{\mathcal{U}}(x)$. Isto significa que $P_{\mathcal{U}}$ pode ser aproximado por baixo e, portanto, $P_{\mathcal{U}}$ é enumerável.

Então, é possível identificar um dado x pelo menor caminho que leva a um nodo que computa x. Chamaremos este nodo de $\tilde{l}$. Sabemos por construção (pela codificação de Shannon-Fano) que $|\tilde{l}| \leq \log \frac{1}{P_{\mathcal{U}}(x)} + 2$, pois a codificação obtida satisfaz a desigualdade de Kraft.

Podemos construir então uma máquina $\mathcal{M}$ que computa x usando a árvore. O programa primeiro constrói a árvore e depois, a partir de $\tilde{l}$, imprime x. O tamanho do programa para $\mathcal{M}$ é $|\tilde{l}| + c''$, onde c'' é o tamanho do programa que constrói a árvore. Pelo Teorema da Invariância

$$K(x) \leq K_{\mathcal{M}}(x) + c''' = |\tilde{l}| + c'' + c''' \leq \left\lfloor \log \frac{1}{P_{\mathcal{U}}(x)} \right\rfloor + 1 + c'' + c''' .$$

Fazendo $c'' + c''' + 1 = c'$ esta completa a prova do Teorema. □

Este resultado é surpreendente pois, mesmo sabendo que existem infinitos programas que computam x, significa que a probabilidade $P_{\mathcal{U}}(x)$ concentra-se fortemente em $2^{-K(x)}$. Além disto, obtivemos uma equivalência $K(x) \approx \log \frac{1}{P_{\mathcal{U}}(x)}$ semelhante a existente na teoria da informação clássica, $H(X) \approx \log \frac{1}{p(X)}$ (veja Teorema 22).

Assim, podemos entender a complexidade K como os tamanhos de palavras de código de um código de Shannon-Fano baseado na distribuição $P_{\mathcal{U}}$.

Teorema 28 *(Universalidade de $P_{\mathcal{U}}$) Seja $p_0, p_1, p_2, \ldots$ uma enumeração arbitrária das distribuições de probabilidade enumeráveis. $cP_{\mathcal{U}}(x) \geq p_i(x)$, para $i = 1, 2, 3, \ldots$, onde c não depende de x apenas de p_i.*

Prova: Seja uma distribuição qualquer p_i da enumeração que pode ser aproximada por baixo por uma máquina $\mathcal{M}_i$ (veja a prova do teorema anterior). Pelo Teorema da Invariância $K_{\mathcal{U}}(x) \leq K_{\mathcal{M}_i}(x) + c'$.

Assim, $2^{-K(x)} \geq 2^{-c'} p_i(x)$, e como pelo Teorema 27, $P_{\mathcal{U}}(x) \geq 2^{-K(x)}$, então $cP_{\mathcal{U}}(x) \geq p_i(x)$, onde $c = 2^{c'}$. □

Ou seja, $P_{\mathcal{U}}$ domina todas as distribuições de probabilidade enumeráveis (ZVONKIN; LEVIN, 1970).

Restringir as distribuições de probabilidade apenas às semi-medidas enumeráveis permite que exista um elemento universal na classe das distribuições de probabilidade enumeráveis.

Chamamos $\mathbf{m}$ de *distribuição universal discreta*. Desta forma, os Teoremas 27 e 28 provam $\mathbf{m}(x) \approx P_{\mathcal{U}}(x) \approx 2^{-K(x)}$.

4 TEORIAS DA ALEATORIEDADE

Os matemáticos do Século XX buscavam obter uma base matemática para teoria das probabilidades. Esta base se constituiria em uma definição formal de "seqüência aleatória", que permitiria dar um significado ao cálculo de probabilidades ao estilo do proposto por Kolmogorov (KOLMOGOROV, 1933). Esta axiomatização, proposta por Kolmogorov, embora seja bem sucedida no cálculo de probabilidades, nada afirma sobre o significado da aleatoriedade dos fenômenos físicos. A busca por tal definição é um objetivo fundamental para dar suporte a toda a teoria das probabilidades e às aplicações práticas em estatística. Veremos que este esforço foi plenamente recompensado, e a meta atingida através da complexidade de Kolmogorov.

Este Capítulo apresenta uma revisão bibliográfica abreviada das principais teorias desenvolvidas com este propósito (veja também (USPENSKY; SEMENOV; SHEN, 1990; VITÁNYI, 2004; VOLCHAN, 2002; ZVONKIN; LEVIN, 1970)). Procuramos enfatizar a teoria de Martin-Löf (MARTIN-LÖF, 1966a,b) e sua relação com incompressividade (complexidade de Kolmogorov). O assunto é apresentado na ordem cronológica do desenvolvimento das diversas teorias que foram propostas, assim como procuramos mostrar as relações entre elas. O texto é relativamente auto-contido, embora algum conhecimento de teoria da medida (FERNANDEZ, 1996) seja desejável para o entendimento das provas de alguns teoremas (particularmente o Teorema 31). Porém, o núcleo da argumentação pode ser entendido sem este conhecimento.

Veremos que este Capítulo apresenta uma (surpreendente para muitos) identificação entre *aleatoriedade* e *computabilidade*, ambas apresentadas a partir de definições matemáticas. Ou seja, veremos que uma string aleatória é aquela que não pode ser computada por uma máquina de Turing.

Neste Capítulo estamos interessados em formalizar o conceito de seqüência aleatória infinita. Em primeiro lugar deve-se ter clareza que a definição de aleatoriedade é dependente da distribuição de probabilidade envolvida. Por exemplo, se estivéssemos tratando com uma moeda simétrica (com iguais probabilidades de ocorrência de cara e coroa), uma seqüência em que ocorressem mais caras que coroas seria logo identificada como não aleatória. Já uma moeda em que houvesse um desequilíbrio entre suas faces, poderia resultar em uma seqüência deste tipo, a qual seria tratada como aleatória em relação a esta segunda distribuição.

Não iremos discutir aspectos físicos do lançamento da moeda, tais como aceleração, velocidade, ângulo de lançamento, etc. Nem nos preocuparemos na possibilidade de tais parâmetros serem manipulados para a obtenção de resultados "viciados". Estes fatores são irrelevantes para a nossa discussão.

Vamos considerar, na maior parte da discussão (exceção onde indicado), seqüências obtidas pelo lançamento de uma moeda honesta (distribuição Bernoulli uniforme). "Ho-

nesta" aqui significa que a moeda é simétrica e as probabilidades de ocorrer cara ou coroa são iguais ($p = 1/2$). Representaremos esta seqüência de eventos por uma string binária infinita, em que "1" e "0" representam "cara" e "coroa".

É óbvio que tais seqüências (seqüências infinitas) não podem ser obtidas no mundo real. Kolmogorov já havia advertido para o problema de embasar fenômenos que são essencialmente finitos através de entidades infinitas (KOLMOGOROV, 1963). Veremos que a abordagem de Kolmogorov, baseada em incompressividade, tenta definir primeiro "aleatoriedade pontual" para então generalizar para o caso de seqüências infinitas, usando o conceito de incompressividade dos prefixos (teste seqüencial). Originalmente Kolmogorov propôs resolver o problema definindo uma seqüência aleatória como sendo aquela em que todos os prefixos são incompressíveis (KOLMOGOROV, 1965, 1968). A idéia mostrou-se inadequada, pois tais seqüências não existem devido à *flutuação* da complexidade (VITÁNYI, 2004). O problema foi resolvido pelo uso de uma definição de complexidade que garanta que o conjunto das descrições (programas para uma máquina universal) seja *livre de prefixo*.

O Teorema 32 prova a equivalência entre a definição de seqüência aleatória de Martin-Löf, baseada em conjuntos efetivos nulos, e a definição via incompressividade usando-se complexidade de Kolmogorov. Finalmente, na Seção 4.5 apresentamos a definição de Schnorr, que é uma definição mais "branda" de aleatoriedade.

4.1 Kollektivs de Von Mises

Toda a teoria das probabilidades baseia-se no cálculo de novas distribuições de probabilidade a partir de outras conhecidas. Toda esta teoria deve estar baseada em uma definição matematicamente correta de "probabilidade" e "aleatoriedade".

Muitos matemáticos entenderam que o conceito de aleatoriedade deveria ser definido como um fenômeno físico, como o é "massa", "gravidade", "aceleração", etc. Ou seja, "aleatoriedade" como ela aparece nos processos físicos que ocorrem no mundo, como por exemplo o lançamento de uma moeda. Isto não quer dizer que deveremos nos aventurar em aspectos como "mecânica quântica", fractais, etc. Mas que devemos obter uma formulação matemática que explique inteiramente o conceito. Assim, o conceito fundamental que deve ser compreendido é o de *seqüência aleatória*, já que é ela a resultante dos processos estocásticos, tais como o lançar sucessivo de uma moeda.

Von Mises foi o primeiro a propor uma solução para o problema de definir "seqüências aleatórias". Sua definição se baseia na existência de seqüências, que ele chamou de Kollektivs, as quais possuem *limite de freqüência relativa*.

Para ilustração, sejam as seguintes strings binárias já apresentadas na página 28:

```
11111111111111111111
01010101010101010101
01001101011000110101
```

Determinar qual delas é mais aleatória, a partir de suas probabilidades em particular, nos conduz a um paradoxo, pois a teoria das probabilidades atribui igual probabilidade para todas (a probabilidade de ocorrer uma dada string binária de tamanho n na distribuição Bernoulli uniforme é 2^{-n}, assim, todas as três strings do exemplo tem probabilidade 2^{-20}). No entanto, duvidaríamos que as duas primeiras pudessem ter sido geradas como resultado do lançamento sucessivo de uma moeda honesta (VITÁNYI, 2004).

A tentativa de definir "seqüência aleatória" como sendo uma seqüência *imprevisível* pode nos conduzir a um segundo problema.

A teoria do limite de freqüência relativa interpreta uma probabilidade como uma razão entre resultados favoráveis e o total de tentativas. Tal razão deve convergir para um valor, digamos p, a medida que o número de repetições do experimento n tende a infinito. Assim, por exemplo, se estivéssemos falando de lançamentos sucessivos de uma moeda honesta, $\lim_{n\to\infty} \lambda(n)/n = 1/2$, onde $\lambda(n)$ é o número de resultados favoráveis. Dizemos que uma seqüência é *imprevisível* se um jogador ao apostar nos resultados do experimento, segundo uma regra de pagamento que considera p, não obterá melhores resultados com qualquer possível e imaginária estratégia de apostas do que obteria ao acaso (como exemplo de esquema de apostas poderíamos tomar a regra "apostar em cara após a ocorrência de três coroas seguidas"). Ou seja, o jogador não poderá ter ganhos infinitos com nenhum esquema de apostas que possa usar.

O problema aparece ao nos perguntarmos o quão longa deve ser a seqüência para garantir a convergência. Podemos dizer que a probabilidade estará na faixa de $p \pm \varepsilon$ para um n grande o suficiente, ou seja, para um $n > n_0$ (nossa definição fica dependendo de ε e n_0). Mas assim, fica evidente que nossa definição de probabilidade é uma definição circular (VITÁNYI, 2004).

Von Mises propôs resolver estes problemas dividindo todas as seqüências infinitas em seqüências aleatórias (que ele chamou de Kollektivs, e nós mantivemos no original neste texto), seqüências estas que possuem limite de freqüência e são apropriadas ao cálculo de probabilidades, e outras seqüências que não são de interesse da teoria das probabilidades. Ele usou evidências empíricas para postular a existência dos Kollektivs (VITÁNYI, 2004). A Definição 24 apresenta a formalização da idéia de Von Mises.

Definimos $\{0,1\}^+$ como o conjunto de todas as strings binárias com um ou mais dígitos, e $\{0,1\}^\infty$ como o conjunto de todas as strings binárias infinitas unidirecionais. Se $x \in \{0,1\}^+ \bigcup \{0,1\}^\infty$ e $x = x_1x_2x_3\cdots$ então $x_1, x_2, x_3 \ldots$ são os sucessivos dígitos binários de x. Cada $x \in \{0,1\}^\infty$ determina um número real $0.x_1x_2x_3\cdots$ (pode ser representado também como $0.x$) no intervalo $[0;1) \subset \mathbb{R}$, fechado a esquerda e aberto a direita. Se x é uma string binária de tamanho maior ou igual a n então $x_{1:n} = x_1x_2x_3\cdots x_n$, ou seja, $x_{1:n}$ denota a seqüência formada pelos n primeiros dígitos binários de x.

Definição 23 *Uma* regra de seleção de posição *é uma função parcial ϕ, que mapeia do conjunto das strings binárias finitas para os valores $\{V, F\}$, $\phi : \{0,1\}^+ \to \{V, F\}$, e que seleciona o índice $k \leq n$ de uma string binária $x = x_1x_2\cdots x_n$ se $\phi(x_{1:k}) = V$. A sub-seqüência obtida pela aplicação de ϕ é $x_{k_1}x_{k_2}x_{k_3}\cdots$, onde x_{k_i} é inserido na sub-seqüência se $\phi(x_{1:k_i}) = V$, e $\phi(x_{1:k}) = F$ para todo k tal que $k_{i-1} < k < k_i - 1$.*

Uma regra de seleção de posição é uma função parcial que seleciona uma sub-seqüência de uma seqüência dada pelo valor calculado da função (V ou F), ou seja, a função "extrai" uma nova seqüência a partir de uma seqüência dada.

A Definição 24 baseia-se claramente no problema das mesas de jogos, ao definir uma seqüência aleatória como sendo aquela cujo limite de freqüência relativa converge (ou seja, existe uma probabilidade de sucesso no jogo) e, além disto, impede que um esquema de apostas seja capaz de "quebrar a banca" (item (2) da Definição), exigindo que esta convergência seja satisfeita por qualquer sub-seqüência da seqüência dada obtida a partir dela pela aplicação de uma função (esquema de aposta).

Definição 24 *Uma seqüência binária infinita $x = x_1x_2x_3\cdots$, é uma seqüência aleatória (Kollektiv) se:*

1. $\lim_{n\to\infty} \frac{\sum_{i=1}^{n} x_i}{n} = p$;

2. *Toda seqüência $x_{k_1}x_{k_2}x_{k_3}\cdots$, obtida de x pela aplicação de alguma regra de seleção de posição* admissível ϕ*, deve satisfazer o item 1 desta Definição.*

Isto significa que a seqüência infinita original e todas as possíveis sub-seqüências obtidas por alguma regra de seleção de posição admissível devem possuir um mesmo limite de freqüência. É importante observar que a Definição 24 não define o que é uma regra de seleção de posição *admissível*.

Existe evidência empírica que (1) da Definição 24 é satisfeito. Mas nenhuma evidência empírica pode garantir a convergência. Eventualmente, a seqüência diverge a partir de algum ponto. O problema é que todo experimento no mundo real é finito, e a evidência empírica não pode garantir uma definição que exige uma seqüência infinita. Quanto a (2), ele garante que nenhum esquema de jogo, usando alguma função parcial como regra de seleção de posição admissível, pode selecionar alguma sub-seqüência em que seja possível obter ganhos infinitos.

Um problema que aparece é o uso de funções parciais como regras de seleção de posição. Seja, por exemplo, a função parcial $\phi_0(x_{1:i}) = V$ se $x_i = 1$ e indefinido caso contrário (VITÁNYI, 2004). Então, para a sub-seqüência selecionada, $p = 1$. Se permitirmos funções como ϕ_0 então não existem Kollektivs.

4.2 Funções de Seleção de Wald-Church

Para viabilizar a proposta de Von Mises, Wald propôs tomar apenas um conjunto contável de funções para ser usado como regras de seleção de posição. Fazendo isto, os Kollektivs passam a existir e a Definição 24 faz sentido (VITÁNYI, 2004).

Church propôs que fossem tomadas como regras de seleção de posição apenas as funções parciais recursivas. Isto significa que uma seqüência aleatória seria imune apenas a toda tentativa de ganho infinito baseada em uma estratégia *computável* (sobre computabilidade veja (KLEENE, 1967)).

Assim, podemos reescrever a Definição 24 restringindo ϕ apenas às funções parciais recursivas. Estas seqüências são chamadas *Wald-Church aleatórias*.

Ainda existe um problema com esta definição: Existem seqüências Wald-Church aleatórias x, com limite de freqüência de $1/2$, mas que satisfazem $\sum_{i=1}^{n} x_i/n \geq 1/2$, para todo n (VITÁNYI, 2004). Estas seqüências permitem ganhos infinitos se o jogador apostar no "1", o que obviamente invalida a seqüência de ser aleatória num sentido mais amplo.

4.3 Definição de Martin-Löf

A definição de seqüência aleatória proposta por Martin-Löf (MARTIN-LÖF, 1966a,b), isenta das dificuldades das outras abordagens apresentadas nas duas Seções precedentes, é baseada em *conjuntos efetivos nulos* e na existência de um *conjunto efetivo nulo maximal* (veja também (USPENSKY; SEMENOV; SHEN, 1990; VITÁNYI, 2004; ZVONKIN; LEVIN, 1970)). Ela é uma abordagem *quantitativa* e baseada em teoria da medida (FERNANDEZ, 1996).

Teoria da medida tem seus fundamentos nas áreas de análise, Calculus e espaços métricos, e foi desenvolvida como uma generalização do cálculo de áreas e volumes. Suas principais aplicações estão no cálculo de integrais, para os casos nos quais a integral de Riemann não pode ser aplicada, e na teoria das probabilidades, ao permitir o cálculo de

probabilidades em espaços "exóticos". A idéia de "medida" fica bem compreendida se pensarmos em medir o tamanho de conjuntos, mesmo que eles sejam não contáveis ou "mal comportados", como é o caso, por exemplo, do conjunto de Cantor (LIMA, 1993). Uma medida pode ser entendida como uma probabilidade (neste caso é chamada de *medida de probabilidade*).

Na abordagem de Martin-Löf isto é importante pois ele identifica "aleatoriedade" com "propriedade de ser típico" (ou seja, com alta probabilidade de ocorrência). Assim, a "medição" do tamanho de conjuntos torna-se central, e determina se os elementos de um dado conjunto são aleatórios ou não.

Considere o conjunto de todas as seqüências binárias infinitas geradas pelo lançamento de uma moeda honesta. Intuitivamente nós chamamos uma seqüência "aleatória" se ela é *típica* (USPENSKY; SEMENOV; SHEN, 1990), porque ela é representativa desta *classe* de seqüências. "Típico" aqui significa "desprovido de características peculiares". Se ela não é típica, então ela é "especial", pois possui alguma propriedade que a distingue das demais (por exemplo, uma seqüência em que cada coroa é seguida por uma cara). As seqüências especiais formam um conjunto com medida zero (significando que tem baixa probabilidade de ocorrerem).

Temos que ter cuidado com o que chamamos de "especial". Se a propriedade "ser aleatório" é considerada como "especial", então obtemos um evidente paradoxo.

Paradoxos (ou antinomias) são bastante freqüentes na história da matemática. Podemos citar o famoso "paradoxo do barbeiro", de autoria de Bertrand Russel, que diz que "o barbeiro é aquele que barbeia os homens que não se barbeiam a si próprios". O paradoxo é obtido ao perguntar se o barbeiro se barbeia a si próprio (NAGEL; NEWMAN, 1973). Obteríamos um paradoxo semelhante ao identificar aleatoriedade com "ser especial".

Entendemos como típico um indivíduo que é representativo de sua classe. Por exemplo, se dissermos que João Vitor é um típico menino de três anos de idade, queremos dizer que além de João Vitor possuir três anos de idade, ele também possui todas as características específicas que o distinguem como pertencente à classe dos meninos de três anos de idade. Ou seja, ele possui todas as propriedades que são compartilhadas por todos os meninos de três anos de idade.

Podemos dizer que as seqüências típicas são aquelas que respeitam todas as leis da aleatoriedade (estocasticamente falando), tal como a Lei dos Grandes Números (GRIMMETT; STIRZAKER, 1992; JAMES, 1996). Cada seqüência satisfazendo uma destas propriedades pertence a uma "maioria". Assim, uma seqüência típica pertence a todas as maiorias, ou satisfaz todas as leis de aleatoriedade. Logo, o conjunto das seqüências aleatórias seria a intersecção de todas estas maiorias.

Isto nos conduz a um problema: Cada seqüência x em particular induz um teste de aleatoriedade, $\delta_x = \{0,1\}^\infty - \{x\}$, que a exclui de uma maioria. Assim, a intersecção de todas as maiorias seria vazia, ou seja, não existiriam seqüências aleatórias!

De outra forma podemos dizer que um elemento $x \in \Omega$ é típico se qualquer subconjunto $U \in \Omega$ que contém uma parte pequena de Ω, não contém x. Mas isto nos conduz a dizer que qualquer x não é aleatório pois $\{x\}$ contém x e é uma parte pequena de Ω. Assim, retornando ao nosso exemplo com crianças de três anos de idade, cada uma pode ser considerada um indivíduo atípico, porque seus gostos pessoais específicos forma uma classe pequena do conjunto de crianças (USPENSKY; SEMENOV; SHEN, 1990).

Martin-Löf resolveu este problema restringindo as leis de aleatoriedade (testes de aleatoriedade) apenas aos testes efetivos (computáveis). Isto significa que apenas "regularidades" computáveis serão testadas, nada interessando eventuais "regularidades" não

computáveis, por não serem mecanicamente detectáveis.

A ocorrência de uma seqüência "regular" no lançamento de uma moeda honesta é um acontecimento "especial" ou "notável". Por exemplo, se a seqüência possuísse um "1" após a ocorrência de dois "0" consecutivos ela seria "especial", e a sua propriedade poderia ser facilmente testada. Ela não possuiria as propriedades estocásticas necessárias para ser considerada aleatória.

Aqui entendemos "regularidade" no sentido da seqüência possuir alguma propriedade estocástica que a exclui do conjunto das seqüências aleatórias. Martin-Löf propõe que estas propriedades devem ser testadas usando-se alguma função parcial recursiva (teste de aleatoriedade), que exclui aquelas consideradas "especiais". Assim, a definição de Martin-Löf propõe a existência de um conjunto de seqüências, ditas aleatórias, que possui complemento recursivamente enumerável.

Seja o conjunto $X \subset \{0,1\}^\infty$. Dizemos que X é um *conjunto nulo* se $\Pr(X)$ é definido e $\Pr(X) = 0$. Uma outra forma de definir conjunto nulo é apresentada na Definição 25.

Definimos o *cilindro* Γ_x denotando o conjunto $\{a \in \{0,1\}^\infty, |x| = n : a_{1:n} = x\}$, ou seja, o conjunto de todas as strings binárias infinitas que tem como prefixo $a_{1:n}$. Um cilindro define geometricamente um sub-intervalo no intervalo $[0;1) \subset \mathbb{R}$. Assim, por exemplo, Γ_x pode ser associado com o intervalo $[0.x; 0.x + 2^{-|x|})$ (LI; VITÁNYI, 1997). O cilindro Γ define uma *medida* trivial (dada pelo tamanho dos intervalos) no intervalo $[0;1)$.

Definição 25 $X \subset \{0,1\}^\infty$ *é um* conjunto nulo *se para todo* $\varepsilon > 0$ *existe uma seqüência de strings binárias* $x_0, x_1, x_2 \ldots$ *tal que:*

1. $X \subset \Gamma_{x_0} \bigcup \Gamma_{x_1} \bigcup \Gamma_{x_2} \bigcup \cdots$;
2. $\sum_{i=0}^{\infty} 2^{-|x_i|} < \varepsilon$.

Na Definição, $\bigcup_{i=0}^{\infty} \Gamma_{x_i}$ é chamado de *cobertura* de X. Os intervalos Γ_x formam uma topologia (LIMA, 1993) em $[0;1)$ e representam subconjuntos de Borel em $[0;1)$ (FERNANDEZ, 1996). A Definição 26 estende a anterior adicionando um requisito de efetividade.

Definição 26 $X \subset \{0,1\}^\infty$ *é um* conjunto efetivo nulo *se existe um algoritmo que recebe um número racional* $\varepsilon > 0$ *como entrada e enumera o conjunto de strings* $\{x_0, x_1, x_2 \ldots\}$ *tal que:*

1. $X \subset \Gamma_{x_0} \bigcup \Gamma_{x_1} \bigcup \Gamma_{x_2} \bigcup \cdots$;
2. $\sum_{i=0}^{\infty} 2^{-|x_i|} < \varepsilon$.

Observe que podemos substituir, na Definição 26, ε por 2^{-m}, $m > 0$, e $<$ por $\leq$ em (2), sem perda de generalidade.

A Definição 26 diz que um conjunto efetivo nulo é um conjunto com medida zero (dado pelo ε que pode ser tão pequeno quanto se queira) que pode ser efetivamente (algoritmicamente) gerado. Ou seja, se o conjunto é nulo e recursivamente enumerável.

Qualquer subconjunto de um conjunto efetivo nulo é também um conjunto efetivo nulo. Um conjunto unitário $\{x\}$ é um conjunto nulo se x é computável (ou não aleatório).

Por "algoritmo de cobertura" para um conjunto efetivo nulo nós entendemos o algoritmo mencionado na Definição 26.

Lema 3 *Seja $X_0, X_1, X_2, \ldots$ uma seqüência de conjuntos efetivos nulos tal que existe um algoritmo que, para todo $i \geq 0$ e $\varepsilon > 0$, produz algum algoritmo de cobertura para X_i. Então, $\bigcup_{i=0}^{\infty} X_i$ é um conjunto efetivo nulo.*

Prova: Para obter uma ε-cobertura de $\bigcup X_i$, nós obtemos uma $(\varepsilon/2)$-cobertura de X_0, uma $(\varepsilon/4)$-cobertura de X_1, uma $(\varepsilon/8)$-cobertura de X_2, e assim por diante. Para gerar esta cobertura combinada, nós usamos o algoritmo, dado no enunciado, que produz coberturas para X_i a partir de i. □

O Lema 3 prova que a união *contável* (ou seja, em que o número de operações de união é equipotente ao conjunto dos números naturais) de conjuntos efetivos nulos também é um conjunto efetivo nulo.

O Teorema 29 prova que a união de todos os conjuntos efetivos nulos é um conjunto efetivo nulo. Ou seja, prova a existência de um *conjunto efetivo nulo maximal* (MARTIN-LÖF, 1966a,b; USPENSKY; SEMENOV; SHEN, 1990).

Teorema 29 *Existe um* conjunto efetivo nulo maximal*, isto é, um conjunto efetivo nulo M tal que $X \subset M$ para todo conjunto efetivo nulo X.*

Prova: Não podemos usar o Lema 3 para todos os conjuntos efetivos nulos pois o Lema se aplica a um conjunto contável deles, e o conjunto de todos os conjuntos efetivos nulos não é contável (já que qualquer subconjunto de um conjunto efetivo nulo também é um conjunto efetivo nulo). Devemos, ao contrário, considerar todos os algoritmos de cobertura (já que o conjunto dos algoritmos é contável).

Para um dado algoritmo, que a partir de um número racional positivo gera strings binárias, não podemos determinar efetivamente se ele é um algoritmo de cobertura ou não. No entanto, podemos faze-lo por aproximação. Se um algoritmo, para um dado $\varepsilon > 0$, gera strings $x_0, x_1, x_2, \ldots$, nós podemos verificar se $2^{-|x_0|} + \cdots + 2^{-|x_k|} < \varepsilon$ ou não. Se não, nós apagamos x_k da seqüência gerada. Seja A' o algoritmo modificado obtido a partir de A. Sabemos que:

1. *Se A é um algoritmo de cobertura de algum conjunto efetivo nulo, então A' é idêntico a A (a condição nunca é violada);*

2. *Para qualquer algoritmo A, A' é um algoritmo de cobertura de algum conjunto efetivo nulo.*

Podemos usar o mesmo argumento do Lema para todos os algoritmos $A'_0, A'_1, A'_2, \ldots$, onde $A_0, A_1, A_2, \ldots$ é a seqüência de todos os algoritmos que recebem como entrada um racional positivo e geram strings binárias. □

Assim, o conjunto efetivo nulo maximal contém todos os conjuntos efetivos nulos. Isto equivale a afirmar a existência de um *teste de aleatoriedade universal*, capaz de encontrar toda e qualquer regularidade (computável) em uma seqüência qualquer. Resolve-se assim os problemas que as outras abordagens para definição de aleatoriedade tinham, pois este conjunto maximal contém realmente *todas* as seqüências não aleatórias.

Definição 27 *Uma seqüência x de zeros e uns é chamada* (Martin-Löf) aleatória *com respeito a medida de Bernoulli uniforme se x não pertence ao conjunto efetivo nulo maximal.*

Uma outra forma de dizer isto é afirmar que x não pertence a nenhum conjunto efetivo nulo.

As Definições 28 e 29 apresentam a idéia de *testes de aleatoriedade* e generalizam a definição de aleatoriedade apresentada até aqui (baseada em distribuição Bernoulli uniforme) para distribuições de probabilidade quaisquer. Importante afirmar que, no caso de seqüências finitas, só podemos falar em *graus de aleatoriedade* (e por isto a existência dos *níveis de significância* nas definições que se seguem), pois tal conceito só se torna absoluto quando aplicado a seqüências infinitas.

Definição 28 *Seja P uma distribuição de probabilidade recursiva sobre o espaço amostral S. Uma função $\delta : S \to \mathbb{N}$ é um P-*teste *(*teste de Martin-Löf*) se:*

1. *δ é enumerável (o conjunto $V = \{(m; x) : \delta(x) \geq m\}$ é recursivamente enumerável);*
2. *$\sum\{P(x) : \delta(x) \geq m, |x| = n\} \leq 2^{-m}$, para todo n.*

A função δ testa *deficiência de aleatoriedade*. Assim, a expressão $\delta(x) \geq m$ *rejeita* x com nível de significância 2^{-m}. As *regiões críticas* associadas ao teste são os conjuntos $V_m = \{x : \delta(x) \geq m\}$ (MARTIN-LÖF, 1966a; VITÁNYI, 2004).

Observe que as regiões críticas são aninhadas, ou seja, $V_m \supseteq V_{m+1}$, para $m \geq 1$. O complemento da região crítica V_m é chamado de intervalo de confiança $(1 - 2^{-m})$. Se $x \in V_m$ então "x é aleatório" é rejeitado com nível de significância 2^{-m}.

A definição de Martin-Löf é adequada no sentido que para cada conjunto nulo X, a propriedade "não pertence à X" vale para todas as seqüências aleatórias.

Devemos, a partir da definição de *aleatoriedade pontual* dada na Definição 28, generalizar para o caso de seqüências infinitas, através do conceito de *teste seqüencial*, que significa aplicar a definição anterior aos prefixos da seqüência aleatória e tomar o supremo.

A Definição 29 introduz esta idéia de *teste seqüencial*. Nela, $\mu(.)$ é uma *medida de probabilidade* no espaço amostral $\{0, 1\}^\infty$ com relação à σ-álgebra dos intervalos em $[0; 1)$. $\mu(.)$ é simplesmente uma *medida de probabilidade* que generaliza a medida trivial que usamos até agora, e a σ-álgebra é uma forma matemática de definir um espaço amostral, sobre o qual se aplica a medida, de forma mais genérica. Mais informações sobre teoria da medida podem ser encontradas em (FERNANDEZ, 1996).

Definição 29 *Seja μ uma medida de probabilidade recursiva sobre o espaço amostral $\{0, 1\}^\infty$. Uma função total $\delta : \{0, 1\}^\infty \to \mathbb{N} \cup \{\infty\}$ é um μ-teste seqüencial (μ-teste de aleatoriedade seqüencial de Martin-Löf) se:*

1. *$\delta(x) = \sup_{n\in\mathbb{N}}\{\gamma(x_{1:n})\}$, onde $\gamma : \{0, 1\}^* \to \mathbb{N}$ é uma função enumerável total ($V = \{(m; x) : \gamma(x) \geq m\}$ é um conjunto recursivamente enumerável);*
2. *$\mu\{x : \delta(x) \geq m\} \leq 2^{-m}$, para cada $m \geq 0$.*

O item 1 é aquele que se aplica sobre os prefixos de x ($x_{1:n}$) e toma o supremo (que é o limite do valor de $\gamma(x_{1:n})$). Já o item 2 afirma simplesmente que os conjuntos com menor medida são aqueles que excedem um determinado valor m tomado como nível de significância ($\delta(x) \geq m$ e o nível de significância é 2^{-m}).

Se $\delta(x) = \infty$ então dizemos que x falha para δ, ou que δ rejeita x. O conjunto dos x rejeitados por δ tem, por definição, μ-medida zero (baixa probabilidade de ocorrerem em um processo estocástico). Ou seja, estes rejeitados são considerados "não típicos" ou "não aleatórios".

4.4 Relação da Definição de Martin-Löf com $K(.)$

Nesta Seção vamos mostrar a equivalência entre a definição de Martin-Löf e a definição de seqüência aleatória via incompressividade (complexidade de Kolmogorov).

Kolmogorov pretendia definir uma string binária aleatória como sendo aquela cuja menor descrição não é muito menor que a própria string.

A idéia original de Kolmogorov tornou-se possível quando foram admitidas apenas descrições *livres de prefixo*. Esta exigência está muito relacionada com a desigualdade de Kraft, expressão muito conhecida na teoria da informação clássica. Isto significa simplesmente que basta os programas conhecerem seu próprio tamanho. Isto não é uma exigência absurda, já que a maioria das linguagens de programação possui algum comando que indica o fim do programa (como por exemplo o `END.` do PASCAL).

Esta idéia de "tamanho do menor programa" está associada à idéia de "caótico", no sentido de "desordem" ou "entropia" (veja (COVER; THOMAS, 1991)), embora pareça contraditório estarmos tentando definir formalmente algo que dizemos "desordenado", pois tal definição representaria uma "ordenação". Esta idéia de aleatoriedade associada ao tamanho da descrição das strings pode ser entendida simplesmente ao confrontarmos a possibilidade de definir o número 1.000.000.000, na base 10, simplesmente como 10^9, enquanto teríamos dificuldade de fazer o mesmo com o número 5.359.871.331, formulado ao acaso. Podemos intuir daí uma importante propriedade do conjunto de todas as seqüências: a esmagadora maioria das seqüências são irregulares, simplesmente, por um argumento de contagem, porque faltam descrições curtas suficientes para descreve-las.

É importante observar que esta definição, sendo bem sucedida em definir "seqüências aleatórias", vincula o conceito formal de "computabilidade", como definido por Turing, Church e outros (KLEENE, 1967; DIVERIO; MENEZES, 2000), com o conceito de "aleatoriedade", como aplicado na área de probabilidade e estatística.

O seguinte teorema será usado nas provas subsequentes. Observe que ele só vale para a complexidade $K(.)$, já que a mesma série apresentada não converge para a complexidade $C(.)$.

Teorema 30

$$\sum_{x \in \{0,1\}^+} 2^{-K(x)} \leq 1$$

Prova: Trivial. Conseqüência da desigualdade de Kraft. □

O Teorema 31 apresenta um resultado profundo de complexidade de Kolmogorov que será importante para a argumentação que se seguirá (veja mais sobre este resultado em (CHAITIN, 1975; GÁCS, 1993; LI; VITÁNYI, 1997)) . Nele afirmamos a existência de uma semi-medida enumerável discreta universal.

A teoria das semi-medidas estende a teoria da medida (FERNANDEZ, 1996) (veja uma revisão de teoria das semi-medidas em (LI; VITÁNYI, 1997)). Uma semi-medida enumerável é uma semi-medida cuja função (real ou discreta) não é computável (função parcial recursiva, como definida por Turing, Church e outros), mas pode ser aproximada por cima ou por baixo, com a precisão que se queira. Tal definição é bem menos restritiva que a exigência de ser computável, não tornando assim o conjunto das funções sobre as quais se aplica tão restrito, como poderia parecer a princípio.

Teorema 31 $K(x) \leq -\log \mu(x) + O(1)$, *para qualquer semi-medida enumerável discreta* $\mu(.)$, *onde* $\log(.)$ *denota o logaritmo na base 2, e* $O(1)$ *é a notação assintótica denotando um termo constante.*

Ou seja, o Teorema afirma que $2^{-K(x)}$ é uma semi-medida *universal*, que pode substituir qualquer outra semi-medida $\mu(x)$ (por *dominação* (COVER; THOMAS, 1991; LI; VITÁNYI, 1997)).

O Teorema 32 coroa o presente trabalho fornecendo uma prova da equivalência entre a definição de seqüência aleatória de Martin-Löf e a definição via incompressividade.

Teorema 32 *Uma seqüência $x = x_0x_1x_2x_3\cdots$ é* Martin-Löf aleatória *se e somente se, para alguma constante $c > 0$, $K(x_{1:n}) \geq n - c$, para todo n.*

Prova: (Se) Basta provar que o conjunto N das seqüências x com $K(x_{1:n}) < n - c$, para qualquer n, é um conjunto efetivo nulo. Devemos observar que:

1. *N é enumerável;*
2. *$\sum_{x\in N} 2^{-|x|} < 2^{-c}$.*

A primeira afirmação é trivial pois $K(.)$ é co-enumerável. A segunda afirmação pode ser facilmente provada pois:

$$\begin{aligned} K(x) &< |x| - c \\ -|x| &< -K(x) - c \\ 2^{-|x|} &< 2^{-K(x)}2^{-c} \\ \sum_x 2^{-|x|} &< 2^{-c}\sum_x 2^{-K(x)} \qquad (4.1) \\ \sum_x 2^{-|x|} &< 2^{-c} \qquad (4.2) \end{aligned}$$

A passagem de (4.1) para (4.2) é devido ao fato que $\sum_x 2^{-K(x)}$, pela desigualdade de Kraft, é uma série que converge (veja Teorema 30).

(Somente se) Observe que podemos definir uma função computável f que recebe um valor $c > 0$ qualquer e um inteiro $i = 0, 1, 2, \ldots$ e gera a cobertura

$$\Gamma_{f(c,0)}, \Gamma_{f(c,1)}, \Gamma_{f(c,2)}, \ldots ,$$

que cobre N e tem medida no máximo 2^{-2c}. Observe que esta seqüência existe e pode ser efetivamente gerada.

Considere a função $g(c, i) = |f(c, i)| - c$. Assim, para um dado c,

$$\sum_i 2^{-g(c,i)} = \sum_i 2^{-|f(c,i)|+c} = \sum_i 2^c 2^{-|f(c,i)|} = 2^c \sum_i 2^{-|f(c,i)|} \leq 2^c 2^{-2c} = 2^{-c} .$$

Portanto, a soma $\sum_{c,i} 2^{-g(c,i)}$, sobre todos os c e i, não excede 1, pois esta série é obviamente convergente. Isto garante que $\sum 2^{-g(c,i)}$ é uma semi-medida enumerável, já que $g(c, i)$ é computável, desde que evitemos que ocorram coincidências do valor de $f(c, i)$ para diferentes pares (c, i). Então, considere a semi-medida

$$\mu(x) = \sum \{2^{-g(c,i)} | f(c, i) = x\} .$$

$\mu(.)$ é enumerável por baixo, porque f e g são computáveis.

Sabemos que

$$K(x) \leq -\log \mu(x) + O(1)$$

e, fixando um par (c, i) obtemos

$$K(f(c, i)) \leq g(c, i) + O(1) = |f(c, i)| - c + O(1) .$$

Observe que $f(c, i)$ são os prefixos das seqüências que pertencem a N. Isto completa a prova do Teorema. □

4.5 Definição de Schnorr

Schnorr propôs uma modificação na definição de Martin-Löf, pela adição de um requisito a mais (USPENSKY; SEMENOV; SHEN, 1990). Esta nova definição estabelece um conjunto de strings típicas que é maior que o da definição de Martin-Löf.

Nesta nova abordagem, um conjunto $X \subset \{0,1\}^\infty$ é chamado *conjunto efetivo nulo de Schnorr* se existe uma função $f(\varepsilon, i)$, definida para todos os números racionais $\varepsilon > 0$ e naturais $i \geq 0$, e uma função *computável* $g(\varepsilon, \eta)$, definida sobre todos os racionais $\varepsilon, \eta > 0$, tal que:

1. $X \subset \bigcup_{i=0}^{\infty} \Gamma_{f(\varepsilon,i)}$ para todo $\varepsilon > 0$;
2. $\sum_{i=0}^{\infty} \mu(\Gamma_{f(\varepsilon,i)}) < \varepsilon$ para todo $\varepsilon > 0$;
3. $\sum_{i>g(\varepsilon,\eta)} \mu(\Gamma_{f(\varepsilon,i)}) < \eta$ para todo $\varepsilon, \eta > 0$.

A condição adicional (3) implica que a série $\sum_i \mu(\Gamma_{f(\varepsilon,i)})$ convergirá para um valor menor que ε (pois existem menos termos a serem somados que satisfazem (3)) e que, além disto, *convergirá de forma efetiva*.

Convergir de forma efetiva significa que para cada $\eta > 0$ podemos encontrar de forma algorítmica um valor que não difere da soma da série mais que η. Isto significa que esta série pode ser aproximada de forma computável. Para isto é necessário que as funções f e g sejam totais (requisito que inexistia na definição de Martin-Löf).

A versão de Schnorr da definição de seqüência típica implica que:

1. Para a distribuição Bernoulli uniforme (e para todas as distribuições usuais) não existe um conjunto nulo maximal. Basta observar que para todo conjunto efetivo nulo de Schnorr existe uma seqüência computável que não pertence ao conjunto. Assim, o conjunto $\{x\}$ é um conjunto efetivo nulo de Schnorr para cada seqüência computável x, o que impede a existência de um conjunto maximal;

2. As seqüências típicas de Schnorr são um conjunto mais amplo que as seqüências típicas de Martin-Löf. Isto é conseqüência de uma definição mais "rígida" de conjunto efetivo nulo. Assim, a intersecção dos complementos dos conjuntos efetivos nulos é um conjunto maior que o definido na abordagem de Martin-Löf.

4.6 Conseqüências da Definição de Martin-Löf

Uma importante conseqüência do Teorema 32 é que ele desloca a discussão do conceito de aleatoriedade de uma visão estocástica para uma visão de incompressividade de strings. Isto tem uma grande importância, em primeiro lugar pela consistência e robustez da definição dada, e em segundo lugar porque esta abordagem, embora baseada em uma função não computável (função parcial recursiva universal), pode ser aproximada pelo uso de um algoritmo compressor universal (assintoticamente ótimo), como por exemplo codificação Lempel-Ziv (programa gzip) ou o algoritmo Burrows-Wheeler (programa bzip2), o que permite algumas medições empíricas, como por exemplo, as propostas em (CAMPANI; MENEZES, 2003; CILIBRASI; VITÁNYI; WOLF, 2004; LI et al., 2003).

Duas evidências determinam o mérito da definição de Martin-Löf: a existência de um conjunto efetivo nulo maximal; e a equivalência com a definição via incompressividade. Estas duas propriedades não são satisfeitas pela definição de Schnorr, que representa um conjunto mais amplo de seqüências típicas.

A definição de Martin-Löf representa um avanço crucial na solução do problema de definir o conceito de seqüência aleatória. Ela resolve todos os problemas que apareceram nas Seções 4.1 e 4.2, apresentando-se como uma definição “rígida” de aleatoriedade (ela é uma definição mais “estreita” que a de Wald-Church e a de Schnorr), fornecendo uma definição consistente com a nossa noção intuitiva de “seqüência aleatória” e, ao mesmo tempo, evitando as dificuldades das outras abordagens, sendo assim matematicamente correta.

5 COMPRESSÃO DE DADOS

Este Capítulo tem como objetivo apresentar uma breve revisão sobre a área de compressão de dados, introduzindo assim a terminologia, alguns métodos de compressão, e um pouco da discussão que será feita no restante do trabalho.

Compressão de dados é um ramo da *teoria da informação* e tem como principal objetivo reduzir o tamanho da representação da informação, tanto para armazenamento, como para envio por um canal de comunicação (reduzindo assim os custos de transmissão ao economizar a banda do canal). Intuitivamente poderíamos defini-la como o ato de transformar alguma string binária, que é a representação de um objeto (ou conjunto de dados), em uma nova string de bits que contém a mesma informação mas cujo tamanho é o menor possível. É usual referir-se à compressão de dados como *codificação* (COVER; THOMAS, 1991; LELEWER; HIRSCHBERG, 1987; SHANNON, 1948).

O requisito que a nova string contenha exatamente a informação original de forma que a mesma possa ser recuperada intacta, normalmente designa um tipo de compressão de dados dita *sem perdas*. Pode-se obter taxas maiores de compressão se este requisito for relaxado, e este tipo de compressão é dito *com perdas*. Exemplos de esquemas de compressão com perdas são os usados em imagens JPEG e em vídeos MPEG.

A área tem suas raízes na teoria da informação de Shannon (SHANNON, 1948), embora questões relacionados com ruído e ocorrência de erros (típicas da teoria de Shannon) não sejam tão importantes para a compressão de dados como o são em teoria da informação. Portanto, aqui vamos supor que o canal é isento de ruído, isto é isento de erro.

Nós consideraremos um canal de comunicação que liga dois dispositivos de comunicação, um *transmissor* e um *receptor* (COVER; THOMAS, 1991; SHANNON, 1948). Suponha que você deseja comunicar para o receptor uma informação sobre uma seqüência de resultados de um experimento. O conjunto de resultados é chamado *conjunto de mensagens*.

Nós precisamos de duas funções, chamadas *codificadora* e *decodificadora*, para transmitir a mensagem através do canal. A função de codificação mapeia uma mensagem para a correspondente *palavra de código*, e a função de decodificação executa a operação contrária obtendo a mensagem original a partir da palavra de código.

Um código "bom" é um *assintoticamente ótimo*, isto é, um que tenha redundância mínima (LELEWER; HIRSCHBERG, 1987). O Teorema 25 mostra que a entropia coincide com o tamanho médio do melhor código, isto é, um código sem redundância (COVER; THOMAS, 1991). Um código é dito *universal* se ele, independente da fonte das mensagens, produz um tamanho de código que é assintoticamente ótimo, ou seja, um código genérico que produz código com tamanho que é limitado superiormente por $c_1 H + c_2$, para $c_1, c_2 \in \mathbb{R}$, e H é a entropia da fonte.

A quantia ou qualidade de compressão produzida por uma codificação pode ser me-

dida pela *taxa de compressão* (LELEWER; HIRSCHBERG, 1987). Nós definimos a taxa de compressão como

$$R = \frac{\text{tamanho médio da mensagem}}{\text{tamanho médio da palavra de código}}\,.$$

Esta definição faz referência ao problema como apresentado na Seção 3.1. Para o caso específico de imagens e animações comprimidas, a taxa de compressão é

$$R = \frac{\text{tamanho descomprimido}}{\text{tamanho comprimido}}\,.$$

Podemos expressar estes valores como porcentagens, usando:

$$R = \frac{\text{tamanho descomprimido} - \text{tamanho comprimido}}{\text{tamanho comprimido}} \times 100\,.$$

Os valores típicos de compressão obtidos usando-se o programa *compact* do UNIX são 38% para textos, 43% para fontes PASCAL, 36% para fontes C, e 19% para programas binários (LELEWER; HIRSCHBERG, 1987). O programa *compress* pode fornecer taxas de compressão entre 50%-60%. Taxas de compressão típicas obtidas com os programas disponíveis na plataforma UNIX podem chegar a 80%, o que justifica a utilidade destas técnicas (LELEWER; HIRSCHBERG, 1987).

Com relação ao mapeamento entre mensagens e palavras de código, uma codificação pode ser *adaptativa* se os códigos mudam a medida que são calculadas as probabilidades aproximadas das mensagens da fonte.

Diversos métodos de compressão foram desenvolvidos, alguns específicos para uma determinada aplicação outros ditos universais, porque genéricos. O mais simples deles, usado na compressão de arquivos de imagem BMP é o RLE (Run-Length Encoding), que se baseia na idéia de substituir uma longa ocorrência de um determinado símbolo pelo tamanho deste bloco em que o símbolo se repete. Este método não é muito eficiente, apresentando pequena taxa de compressão, embora seja bastante simples de implementar.

Um método desenvolvido com o objetivo de obter um código ótimo é o *código de Huffman* (COVER; THOMAS, 1991). O algoritmo para obter o código, constrói uma árvore binária de baixo para cima, alocando as mensagens de maior probabilidade mais próximas à raiz da árvore. Ilustramos um exemplo de uso deste algoritmo na figura 5.1. Na última coluna aparecem os códigos obtidos.

A codificação obtida respeita a desigualdade de Kraft e, por construção, é ótima no sentido do Teorema 25. Um problema com códigos de Huffman é que precisamos saber de ante-mão todas as probabilidades de ocorrências de símbolos, o que nem sempre é possível ou desejável (podendo reduzir a eficiência do compressor em termos de tempo de compressão). Uma forma de contornar este problema, mantendo uma taxa de compressão ótima, é a *codificação de Shannon-Fano* que foi apresentada na Seção 3.4.2.

Outra codificação assintoticamente ótima é a codificação Lempel-Ziv.

Codificação Lempel-Ziv, assim como seus derivados, é um *código adaptativo*. Isto significa que o comportamento do algoritmo de compressão é dinâmico, mudando, ao longo do processo, o mapeamento do conjunto das mensagens para o conjunto das palavras de código, de acordo com as probabilidades aproximadas computadas durante o processo (LELEWER; HIRSCHBERG, 1987).

O algoritmo Lempel-Ziv executa um processo de parsing da fonte binária produzindo um conjunto de frases. A cada passo, ele procura uma frase que ainda não pertence ao

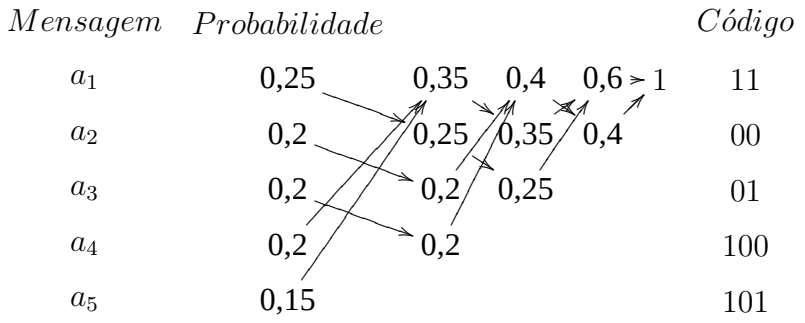

Figura 5.1: Exemplo de código de Huffman

conjunto previamente obtido. Isto significa que cada nova frase tem um prefixo pertencendo ao conjunto anterior mais um novo dígito binário, em uma seqüência de tamanho crescente. Desta forma, é suficiente armazenar a posição do prefixo anterior e o dígito binário finalizador (COVER; THOMAS, 1991; LELEWER; HIRSCHBERG, 1987).

Palavras de código são representadas aqui por pares (i, x), onde i é a posição do prefixo na seqüência de palavras de código e x é o dígito binário finalizador da frase. $(0, x)$ significa que a palavra de código não tem prefixo, tem somente o dígito binário x (isto ocorre no início do processo de parsing).

Exemplo 3 *Seja* 1001101010011 *uma string binária de entrada para o algoritmo. O algoritmo de compressão Lempel-Ziv irá efetuar o parsing obtendo o seguinte conjunto de frases:* $\{1, 0, 01, 10, 101, 00, 11\}$. *Os dados de entrada serão codificados como:* $(0, 1)$, $(0, 0)$, $(2, 1)$, $(1, 0)$, $(4, 1)$, $(2, 0)$ *e* $(1, 1)$.

A compressão é produzida porque, a medida que o tamanho das frases aumenta, é mais eficiente armazenar suas posições (COVER; THOMAS, 1991).

O algoritmo de compressão Lempel-Ziv fornece uma taxa de compressão muito pobre quando o tamanho dos dados de entrada é pequeno, mas podemos provar que, aplicado a uma string produzida por um processo estocástico (fonte ergódica), o tamanho médio das palavras de código aproxima-se assintoticamente da entropia da fonte (COVER; THOMAS, 1991). Isto significa que a codificação Lempel-Ziv é ótima (ou universal). A compressão típica, obtida com o algoritmo, está na faixa de 50-60% (LELEWER; HIRSCHBERG, 1987).

Uma implementação direta e simples do algoritmo leva $O(n^2)$ passos para comprimir uma string de tamanho n (LELEWER; HIRSCHBERG, 1987).

Um outro exemplo de método de compressão assintoticamente ótimo é o algoritmo Burrows-Wheeler. A idéia do algoritmo Burrows-Wheeler é re-ordenar a string de entrada (de uma forma reversível) para torna-la mais adaptada à compressão. Neste método é usada codificação de Huffman para comprimir os dados.

6 COMPARAÇÃO E AVALIAÇÃO DA COMPRESSÃO DE DADOS SEM PERDAS EM MODELOS DE ANIMAÇÃO PARA WEB

A principal meta de compressão de dados é minimizar o tamanho dos dados a serem transmitidos através de um canal de comunicação ou a serem armazenados em arquivos de dados.

Compressão de dados está se tornando um importante assunto a medida que a Internet cresce e as páginas web começam a ser enriquecidas por objetos multimídia. Imagem, vídeo, som e animações demandam transferências de arquivos grandes e bom uso da largura de banda para reduzir custos de transmissão. Com este propósito, formatos multimídia possuem compressão de dados interna.

Este Capítulo introduz um método para comparar e, em certo sentido, avaliar modelos de animação gráfica baseados em uma definição formal de compressão de dados usando complexidade de Kolmogorov (CAMPANI et al., 2004).

Nós apresentaremos um estudo de caso, usando o método. Este estudo de caso é a avaliação do modelo AGA, por comparação com o modelo GIF. AGA (Automata-based Graphical Animation) é um modelo de animação gráfica baseado em teoria dos autômatos e objetiva prover reuso na composição de animações, reduzir o espaço de armazenamento, e o uso de uma linguagem de consulta que permita recuperação de informação (ACCORSI; MENEZES, 2000) .

A razão para a escolha de AGA e GIF para o estudo de caso foi, por parte do AGA, interesses do projeto de pesquisa ao qual esta tese está vinculada, e, por parte do GIF, sua popularidade em páginas web e seu largo uso como mecanismo para produzir animações.

É importante lembrar que durante algum tempo houve restrições ao uso de imagens e animações GIF, pois GIF usa um algoritmo de compressão, chamado LZW, que era propriedade da Unisys. Porém, as patentes da Unisys sobre o LZW expiraram em 7 de julho de 2004, tornando o formato GIF um formato livre de fato, o que deverá aumentar sua popularidade e uso na web.

O método apresentado neste Capítulo aplica-se à modelos de animação gráfica baseados em bitmap com compressão de dados sem perdas, embora o método possa ser adaptado para um caso mais geral (e o faremos na seqüência a este Capítulo).

6.1 Formalizando Compressão de Dados

Permanecem duas questões fundamentais sobre compressão de dados: Podemos obter melhores algoritmos para comprimir dados? E, se sim, existe o melhor algoritmo?

Se a resposta para a primeira pergunta é não, qualquer esforço para comparar e avaliar

algoritmos para compressão de dados é sem sentido, nós apenas precisamos escolher um bom algoritmo para usar. Se a resposta para a segunda questão é sim, provavelmente é mais importante concentrar esforços em desenvolver este "algoritmo perfeito".

Para responder estas questões devemos primeiro definir *algoritmo de compressão*, baseado na definição formal de incompressividade. É importante lembrar que a complexidade de Kolmogorov está relacionada com a máquina de Turing como um "algoritmo de descompressão" e não faz nenhuma menção ao "algoritmo de compressão" usado, sendo este um desafio a ser superado para a obtenção da definição desejada. Além disto, nós devemos definir formalmente o significado de "mais compressão" e "melhor compressão" (SUBBARAMU; GATES; KREINOVICH, 1998).

Nós entenderemos algoritmo de compressão como uma função computável sobre strings binárias que reduz o tamanho da string.

As duas Seções seguintes apresentam resultados bem conhecidos, apresentados originalmente em (SUBBARAMU; GATES; KREINOVICH, 1998). Este artigo inspirou algumas idéias interessantes para o desenvolvimento posterior desta tese, particularmente alguma notação usada e a idéia original para definir esquemas de compressão de dados. Já a Seção 6.1.3 apresenta um resultado nosso (que, no entanto, não é central no desenvolvimento desta tese).

6.1.1 Existem Melhores Algoritmos de Compressão

Definição 30 *Sejam $s, s' \in \{0,1\}^*$ duas strings binárias. Nós chamamos δ um* algoritmo de descompressão *e, para todo s' existe algum s, de tal forma que $s' = \delta(s)$, onde s é o código de s', com $|s| \leq |s'| + c$. Nós chamamos γ um* algoritmo de compressão *se, para todo s, $s = \delta(\gamma(s))$, e γ deve ser uma função injetiva. Nós chamamos o par $\Gamma = (\gamma, \delta)$ um* esquema de compressão.

Definição 31 *Sejam $\Gamma_1 = (\gamma_1, \delta_1)$ e $\Gamma_2 = (\gamma_2, \delta_2)$ dois esquemas de compressão. Nós dizemos que Γ_1 é* melhor que Γ_2 *se, para todo s e algum $c > 0$,*

$$|\gamma_1(s)| \leq |\gamma_2(s)| + c\,. \tag{6.1}$$

Aqui podemos encontrar a ligação entre complexidade de Kolmogorov e o nosso problema. $|\gamma(s)|$ é o tamanho do código de s no esquema de compressão (γ, δ). Um esquema de compressão Γ_1 é melhor que Γ_2 se e somente se Γ_1 *minora* Γ_2.

Agora, nós devemos provar o resultado sobre a existência de melhores algoritmos de compressão (veja (SUBBARAMU; GATES; KREINOVICH, 1998)).

Teorema 33 *Seja $\Sigma = \{(\gamma_1, \delta_1), (\gamma_2, \delta_2), \ldots, (\gamma_n, \delta_n)\}$ um conjunto de esquemas de compressão. Existe um esquema de compressão (γ, δ) que é melhor que todos eles.*

Prova: Suponha Σ acima mencionado. Nós podemos construir um esquema de compressão (γ, δ) melhor pelo seguinte procedimento. Primeiro, para qualquer string s, aplique todos os algoritmos de compressão γ_i, $1 \leq i \leq n$, na string s e escolha a melhor compressão de Σ, identificada por γ_b,

$$|\gamma_b(s)| = \min\{|\gamma_1(s)|, |\gamma_2(s)|, \ldots, |\gamma_n(s)|\}\,.$$

Então, coloque b nos primeiros $\lfloor \log(n+1) \rfloor$ bits e coloque $\gamma_b(s)$ no final do código. Este é $\gamma(s)$. Lembre-se que $\lfloor . \rfloor$ denota o maior número inteiro menor ou igual ao número dado, enquanto que $\log$ representa o logaritmo na base dois.

Nós podemos provar que γ é melhor por construção, de acordo com a Definição 31, porque

$$|\gamma(s)| = \lfloor \log(n+1) \rfloor + |\gamma_b(s)| \,.$$

Isto é, o limite superior do tamanho de $\gamma(s)$ é $|\gamma_b(s)| + c$, onde $c = \lfloor \log(n+1) \rfloor$. Então,

$$|\gamma(s)| \leq |\gamma_i(s)| + c$$

para todo i, $1 \leq i \leq n$, como exigido pela Equação (6.1), mostrando que γ é melhor que todos os outros algoritmos de compressão em Σ.

Finalmente, nós devemos construir o algoritmo de descompressão δ. δ deve olhar os primeiros $\lfloor \log(n+1) \rfloor$ bits do código s', onde está armazenado b e reconstruir s aplicando δ_b à parte final da palavra de código s'. □

6.1.2 Nós Não Podemos Atingir a Melhor Taxa de Compressão

Definição 32 *Nós dizemos que um esquema de compressão Γ é* o melhor *se é melhor que qualquer outro esquema de compressão, no sentido da Definição 31.*

Nós percebemos que o melhor esquema de compressão concorda com o conceito de descrição de tamanho mínimo da complexidade de Kolmogorov, isto é, se Γ_b é o melhor esquema de compressão então, para todo s,

$$|\gamma_b(s)| \approx C(s) \,, \tag{6.2}$$

desde que γ é, obviamente, computável. Assim, nós identificamos δ, o algoritmo de descompressão, com a máquina de Turing universal $\mathcal{U}$. Isto significa que um esquema de compressão é o melhor se *minora* todos os outros esquemas.

Segundo os Teoremas 17 e 18, a complexidade de Kolmogorov não é computável, mas semi-computável. Assim, não podemos decidir se um determinado programa é o menor programa que computa uma determinada string. Logo, por (6.2), nós não podemos decidir se um dado esquema de compressão Γ é o melhor.

Além disto, pelos Teoremas 33 e 17, nós concluímos que é impossível encontrar, entre todos os possíveis algoritmos de compressão, um que seja o melhor, porque nós sempre podemos construir outro esquema de compressão melhor que todos os dados e não podemos provar formalmente qual é o melhor.

Teorema 34 *Não existe o melhor algoritmo de compressão.*
Prova: Segue da discussão dos parágrafos anteriores. □

6.1.3 Mais Tempo Significa Mais Taxa de Compressão

Desejamos desenvolver um ponto de vista mais prático. Se não podemos obter a melhor compressão, quem sabe uma solução não ótima, mas suficientemente boa? Isto significa que desejamos a melhor compressão possível.

Pelo Teorema 18, existe uma função recursiva (total) $\phi : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, de tal forma que $\lim_{t \rightarrow \infty} \phi(t, x) = C(x)$ (GÁCS, 1993).

A função C^t do Teorema 18 é uma função monotônica não-crescente em t, implicando que C é uma *função co-enumerável* (LI; VITÁNYI, 1997).

Concluímos que, lembrando a ligação existente entre a complexidade de Kolmogorov e compressão de dados, mais tempo temos para computar a descrição mínima, mais nos aproximamos assintoticamente da descrição mínima, que representa a compressão "ideal". Isto é, a solução para o problema de obter a máxima compressão possível pode ser aproximada por cima por uma função recursiva (total).

6.2 Modelos de Animação Gráfica para Web

Formatos multimídia fornecem recursos para tratar uma grande quantidade de elementos como imagens, animação e vídeo, dando vida às páginas da web.

Modelos de animação por computador usualmente fornecem ferramentas para mover e deformar objetos, câmeras e fonte de luz colocadas na cena. Nos últimos 20 anos, muitos métodos de animação diferentes foram desenvolvidos (PARENT, 2001) para proporcionar movimento aos objetos na cena. Alguns procuraram obter um movimento com aparência natural. Outros combinaram diversas técnicas para melhorar a taxa de compressão e assim por diante.

Do ponto de vista de aplicações multimídia na web, sensação de movimento é produzida, em geral, pela concatenação de seqüências de imagens que são mostradas na tela uma após a outra, da mesma forma que se usa para simular movimento no cinema ou TV. Enquanto nestes casos nós necessitamos de cerca de 24 ou 30 imagens por segundo para ter a sensação de movimento, para aplicações multimídia, mostrar 15 imagens por segundo é considerada uma taxa aceitável.

Neste contexto, animação pode ser definida como uma seqüência de imagens mostradas em intervalos de tempo. Assim, podemos definir uma animação s como a seqüência $s = s_1 s_2 s_3 \cdots s_n$, onde s_i, para $1 \leq i \leq n$, são chamados *frames*, cada frame correspondendo a uma imagem.

Uma imagem é composta por uma seqüência de dígitos binários, onde cada bit ou agrupamento de bits descreve a intensidade ou a cor de um único pixel na tela. Finalmente, uma imagem pode também ser definida como uma string binária.

Um problema relacionado com o armazenamento e transmissão de imagens é que uma imagem pode conter uma quantidade muito grande de dados. Fotos de alta resolução podem conter centenas de Megabytes de dados (SUBBARAMU; GATES; KREINOVICH, 1998). O problema é especialmente difícil em comunicação em longas distâncias com canal de baixa qualidade, como por exemplo a comunicação com satélites de exploração que operam no espaço profundo.

Usualmente animação e imagens são armazenadas comprimidas para economizar espaço de armazenamento e tempo de transmissão.

Por exemplo, fotos de Uranus transmitidas pela Voyager 2 usaram uma tecnologia de compressão de dados chamada *difference mapping*, na qual a imagem é representada como um array de diferenças em brilho e cor entre pixels adjacentes. Neste caso, foi relatado que cores de 8 bits foram reduzidas para uma média de 3 bits por pixel (LELEWER; HIRSCHBERG, 1987). Isto ilustra a importância das tecnologias de compressão de dados.

O problema de compressão de dados torna-se mais complexo quando aplicado a seqüências de animação, porque animações não são compostas de uma única imagem, mas de um conjunto delas. Como avaliar a compressão obtida? Como comparar diferentes modelos? Apenas avaliações empíricas são suficientes? O desenvolvimento de uma metodologia para comparar e avaliar compressão de dados em animações é um grande e útil avanço.

Nas próximas Seções nós iremos comparar dois modelos de animação para web, considerando a taxa de compressão. O primeiro formato analisado é o GIF, enquanto o segundo corresponde ao modelo AGA, um formato recentemente desenvolvido (ACCORSI; MENEZES, 2000) para fornecer animações baseadas em seqüenciamento de imagens usando teoria dos autômatos.

6.2.1 Modelo GIF

GIF (Graphics Interchange Format) é um formato de imagem muito popular adotado no desenvolvimento de páginas web. Ele usa um algoritmo de compressão interna LZW e proporciona algumas características interessantes como animação por múltiplas imagens, referenciadas como frames e codificadas em um único arquivo.

GIF é definido em termos de blocos e sub-blocos contendo, cada um, dados e informação de controle. As principais divisões definidas no protocolo GIF são: *Header*, identificando o início da stream de dados; *Logical Screen Descriptor*, definindo alguns parâmetros para a área do dispositivo display onde as imagens serão renderizadas, tais como o número de bits disponível para representar cor; *Global Color Table*, para armazenar a tabela de cores representada como seqüências de tripas RGB e mantida para tratar as cores da imagem; *Image Descriptor*; *Image Data*; e *GIF Trailer* para finalizar a stream de dados (GRAPHICS INTERCHANGE FORMAT PROGRAMMING REFERENCE. VERSION 89A., 1990). Cada imagem armazenada em um arquivo de animação GIF é composta de um Image Descriptor e um Image Data. Finalmente, cada bloco tem um campo de tamanho de bloco que conta o número de bytes no bloco.

Com relação ao algoritmo usado na compressão das streams de dados, o formato GIF usa o algoritmo LZW, que é uma variação do algoritmo de compressão Lempel-Ziv (GRAPHICS INTERCHANGE FORMAT PROGRAMMING REFERENCE. VERSION 89A., 1990; LELEWER; HIRSCHBERG, 1987).

O algoritmo LZW difere da implementação direta e simples do algoritmo Lempel-Ziv de várias maneiras, tais como:

- LZW usa um dicionário de frases;
- LZW gerencia frases e palavras de código de uma forma sutilmente diferente de Lempel-Ziv, obtendo uma compressão mais rápida.

6.2.2 Modelo AGA

Accorsi e Menezes descrevem em (ACCORSI; MENEZES, 2000) um modelo para representar animação gráfica baseado em teoria dos autômatos (HOPCROFT; ULLMAN, 1979). O modelo, chamado AGA (Automata-based Graphical Animation), organiza o conteúdo de uma seqüência animada como autômatos. Estes autômatos descrevem o comportamento dos atores em tempo de animação.

As características de AGA favorecem o uso de uma linguagem de consulta para recuperação de informação, incrementam o reuso de animações e contribuem para reduzir o espaço de armazenamento.

O modelo AGA organiza a animação em um conjunto de atores. Estes atores são elementos recortados de imagens que têm um comportamento dinâmico e, juntos, compõem o movimento da cena. Por exemplo, um ator pode ser a imagem de um pássaro em vôo ou o olho piscando em uma face.

Cada ator é modelado como um autômato finito com saída, cujos estados estão ligados com imagens de saída. O comportamento dinâmico dos atores durante a animação é produzido pela adição de símbolos a uma fita de entrada. Quando um símbolo é lido, o estado atual é mudado e a imagem correspondente é mostrada em uma camada da animação. A animação consiste de um conjunto de camadas. A união (ou sobreposição) destas camadas compõe o frame em um instante da animação. Cada camada é controlada por um autômato com sua fita correspondente.

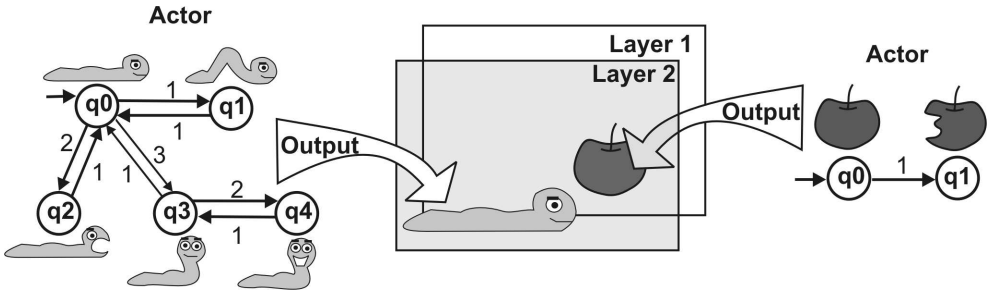

Figura 6.1: Um exemplo de animação AGA

Um exemplo de animação AGA é mostrado na Figura 6.1. No exemplo, nós podemos ver o comportamento individual dos atores, cada ator modelado por uma autômato, e as camadas sobrepostas.

O modelo de ator usado no AGA é baseado em autômatos com saída tradicionais (máquinas de Moore e de Mealy) (HOPCROFT; ULLMAN, 1979), mas algumas extensões foram propostas para inserir informações de temporização, uso de imagens e suporte a recursos de consulta.

A primeira extensão está relacionada com o alfabeto de saída e função de saída. A função de saída é um mapeamento do conjunto de estados para um conjunto de imagens, ao invés de um conjunto de símbolos. Assim, quando um estado é encontrado, uma imagem de saída é mostrada na camada controlada pelo autômato. O armazenamento de imagens distintas é uma importante característica deste modelo, estas imagens são selecionadas pela função de transição quando um símbolo é lido. Como conseqüência, o espaço de armazenamento não cresce na mesma proporção do número de frames, mas depende do número de imagens distintas e símbolos na fita. A função de saída é também estendida para produzir transformações nas imagens de saída. Desta forma, é possível dimensionar, rotar, transladar e aplicar outras transformações sem a necessidade de armazenar novas imagens. Esta função é chamada função de saída contextual. Assim, cada célula da fita possui, além do símbolo de entrada, uma string de símbolos associada representando quais transformações devem ser aplicadas.

Outra extensão proposta para o autômato é a função de transição de tempo. Esta função proporciona a inserção de informação de temporização na fita para controlar o tempo nas mudanças de estado. Então, quando a célula da fita é lida, três tipos de informação são obtidas: o símbolo de entrada, o tempo gasto no estado corrente e a string de transformações. As informações de tempo estão na fita de entrada ao invés de na transição que liga os estados de uma forma estática.

A Figura 6.2 mostra a fita de entrada usada em uma animação, junto com a animação em progresso. Nós podemos ver como o símbolo de entrada lido controla o comportamento do ator.

Uma nova função, chamada função de descrição, foi introduzida para inserir uma documentação semântica dos estados. Esta função é um mapeamento do conjunto de estados para descrições semânticas. O propósito da função é permitir recuperação de informação. É possível recuperar o exato momento quando um ator atinge um estado por sua descrição. Esta recuperação é executada por análise de fita e de diagrama de transição. Esta operação pode ser posteriormente explorada para produzir consultas complexas envolvendo vários atores.

Nas duas definições seguintes serão formalizados ator AGA e animação AGA.

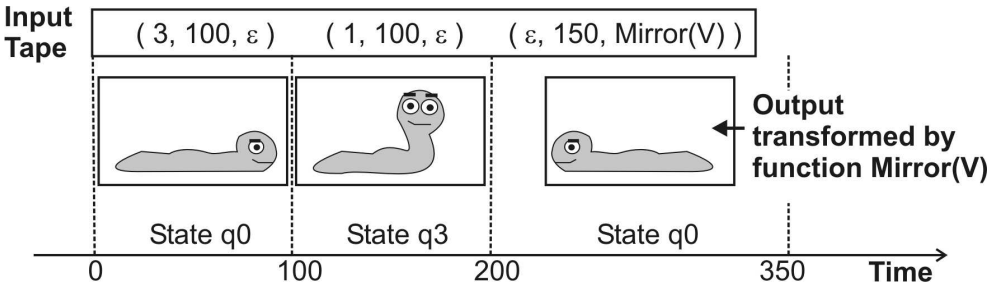

Figura 6.2: Fita de entrada

Definição 33 *Um* ator AGA *é definido como*

$$\mathrm{ACT} = (Q, \Sigma, \Delta, \delta', \lambda^c, \sigma, q_0, F, D)\,,$$

onde

Q *– conjunto de estados do ator;*
Σ *– alfabeto de entrada;*
Δ *– conjunto de imagens do ator;*
$\delta' : Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{N} \rightarrow Q$ *– função de transição de tempo (função parcial);*
$\lambda^c : Q \times F^* \rightarrow \Delta^{*F}$ *– função de saída contextual (função total);*
$\sigma : Q \rightarrow D$ *– função de descrição (função parcial);*
q_0 *– estado inicial;*
F *– conjunto de funções de transformação;*
D *– conjunto de descrições.*

A produção $\delta'(q_1, x, 200) = q_2$ significa que se o estado corrente é q_1, com x na fita de entrada e após 200ms de tempo de espera, o autômato vai para o estado q_2. ε permite definir transições vazias.

Os elementos de Δ^{*F} são seqüências de imagens transformadas por aplicações de elementos de F. ε permite definir transformação vazia (função identidade).

Definição 34 *Uma* animação AGA *é definida como uma ordem total* $\leq^C$ *sobre um conjunto de pares* $(\mathrm{ACT}, \mathrm{fita})$*, onde* ACT *é um ator e* fita *é uma fita de entrada, e*

$$\mathrm{AGA} = (\{(a, \text{fita de } a) | a \text{ é um ator}\}, \leq^C)\,.$$

$\leq^C$ *é definida de tal forma que* $ACT_1 \leq^C ACT_2$ *se e somente se* ACT_1 *está atrás de* ACT_2 *na animação (refere-se à sobreposição de camadas).*

Seqüências animadas geradas com AGA usam compressão LZW e podem enviar as imagens, do servidor web para o cliente, sob demanda para melhorar o tempo de carga da animação.

A estrutura do conteúdo no modelo AGA fornece recuperação de informação orientado a estado e visualização parcial de camadas. Esta nova abordagem contribui para o projeto de linguagens de consulta para recuperação de informação em dados semi-estruturados. O encapsulamento do aspecto e comportamento do ator favorece o reuso de imagens durante a composição de animações.

6.3 Formalizando Animação

Esta Seção do trabalho apresenta uma contribuição do autor, ao formalizar animação, incorporando na definição proposta a definição anterior de "esquema de animação".

Como foi observado, uma animação é formada por uma seqüência de frames mostrados sobre a tela em intervalos de tempo. Para tratar com o conceito de "melhor animação" (no sentido de taxa de compressão obtida) em uma base matemática, nós devemos definir "animação" também formalmente.

Definição 35 *Seja F um conjunto de frames. Uma* animação alvo *é um par (s, t), onde s é uma seqüência $s = s_1 s_2 s_3 \cdots s_n$, cada $s_i \in F$ para $1 \leq i \leq n$, e t é uma seqüência de intervalos de tempo $t = t_1 t_2 \cdots t_n$, cada $t_i \in \mathbb{N}$ para $1 \leq i \leq n$.*

Na Definição 35, nós definimos uma animação como uma string binária "computada" de alguma forma. Esta definição preocupa-se com o "comportamento dinâmico" da animação, isto é, o efeito da execução da animação, mas não faz referência nenhuma a como ela será armazenada e produzida. Nós devemos definir um mecanismo que "computa" a animação.

Definição 36 *Um* descritor de animação *é um par $\phi = (\alpha, \Delta)$, onde α é um conjunto indexado de imagens (ou atores) e Δ é uma função, $\Delta : \alpha \mapsto a$, isto é, Δ mapeia α para a animação alvo a. Δ é uma estratégia para computar uma animação alvo a partir de α.*

Por exemplo, no formato GIF a estratégia Δ é uma função muito simples. Todos os frames estão literalmente armazenados em um arquivo e a seqüência animada segue seqüencialmente, do primeiro frame até o último.

Mas, devemos recordar que usualmente os formatos de animação usam compressão de dados. A Definição 37 combina a estratégia para computar uma animação com um algoritmo de compressão.

Definição 37 *Um* esquema de animação *é um par $G = (\Gamma, \phi)$, onde Γ é um esquema de compressão e ϕ é um descritor de animação.*

O algoritmo de compressão γ é aplicado sobre α, o conjunto de imagens (ou atores) da Definição 36.

Seja G um esquema de animação e seja a uma animação alvo. Definimos $G(a)$ como sendo a animação a executada pelo esquema G, e definimos $|G(a)|$ como o número de bits necessários para armazenar a animação a usando o esquema G. É simples mostrar que, para algum $c > 0$,

$$|G(a)| = |\gamma(\alpha)| + |\Delta| + c\,, \tag{6.3}$$

onde $|\gamma(\alpha)|$ é o número de bits necessários para armazenar todas as imagens (ou atores) em α usando a compressão γ, e $|\Delta|$ é o "tamanho" da função Δ.

O significado de $|\Delta|$ é que nós devemos *descrever* a função Δ de alguma maneira efetiva. Assim, nós podemos contar o número de bits da descrição.

Definição 38 *Sejam G_1 e G_2 dois esquemas de animação. G_1 é* melhor que G_2 *se, para toda seqüência animada a e algum $c > 0$,*

$$|G_1(a)| \leq |G_2(a)| + c \tag{6.4}$$

e

$$\neg\exists c'\forall a|G_2(a)| \leq |G_1(a)| + c' \,. \tag{6.5}$$

(6.4) significa que G_1 não é pior que G_2, e (6.5), fixando c', é equivalente a

$$\exists a|G_2(a)| > |G_1(a)| + c' \,,$$

isto é, basta mostrar a_0 que satisfaça

$$|G_2(a_0)| > |G_1(a_0)| + c' \,.$$

Isto é, G_1 é melhor que G_2 se G_1 *minora* G_2 no sentido induzido pela complexidade de Kolmogorov. Nós chamaremos G_1 e G_2 respectivamente máquina-G_1 e máquina-G_2 para enfatizar este relacionamento. Isto é, a máquina-G_1 é melhor que a máquina-G_2 se a máquina-G_1 *simula* a máquina-G_2 (veja a prova do Teorema 5 – Teorema da Invariância – para mais detalhes sobre este relacionamento entre simulação e minoração).

6.4 AGA É Melhor Que GIF

Pelo fato de termos formalizado o conceito de "melhor animação" como simulação de máquinas, induzido pelo conceito da complexidade de Kolmogorov, nós devemos definir ambos, GIF e AGA, formalmente como máquinas.

Nós percebemos que a taxa de compressão de uma animação é obtida de dois diferentes aspectos ao mesmo tempo:

- O algoritmo de compressão usado para comprimir as imagens que formam a animação;
- A forma com que a animação é computada a partir do conjunto de imagens.

Estas duas partes são representadas na Equação (6.3) por $|\gamma(\alpha)|$ e $|\Delta|$, respectivamente.

Felizmente, ambos, GIF e AGA, usam o mesmo algoritmo de compressão, LZW, reduzindo assim a complexidade da prova, e evitando a necessidade de dados empíricos sobre taxa de compressão.

Formatos de imagem e animação usualmente possuem um cabeçalho com informações de controle seguidas pelo corpo com a stream de dados. O tamanho do cabeçalho não é afetado pelo tamanho do corpo mais que por um termo logarítmico (sobre o tamanho da codificação de prefixo veja (LI; VITÁNYI, 1997) e o Algoritmo 1), e assim, ele é assintoticamente desprezável.

Assim, seja $G_{\text{AGA}} = (\Gamma_{\text{LZW}}, \phi_{\text{AGA}})$ a máquina-AGA e seja $G_{\text{GIF}} = (\Gamma_{\text{LZW}}, \phi_{\text{GIF}})$ a máquina-GIF (esquemas de animação). Além disto, $\phi_{\text{AGA}} = (A, \Delta_{\text{AGA}})$ e $\phi_{\text{GIF}} = (F, \Delta_{\text{GIF}})$, onde A é um conjunto de atores e F é um conjunto de frames.

Formalizando Δ_{AGA} e Δ_{GIF}, nós devemos levar em conta a forma com que ambos, GIF e AGA, "computam" a animação alvo.

A função Δ_{GIF} é muito simples. O arquivo GIF armazena todos os frames literalmente e executa a animação mostrando os frames seqüencialmente sobre a tela, isto é, $\Delta_{\text{GIF}} : \{s_1, s_2, \ldots, s_n\} \mapsto (s_1 s_2 \cdots s_n, t_1 t_2 \cdots t_n)$, onde $t_1 = t_2 = \cdots = t_n = t \in \mathbb{N}$. Esta abordagem está relacionada com o conceito de armazenar a string dentro do programa como mostrado no Teorema 6 que fala sobre o limite superior da complexidade de Kolmogorov.

Por outro lado, AGA usa uma abordagem mais sofisticada. Δ_{AGA} é definida como a função computada por um autômato finito com saída (HOPCROFT; ULLMAN, 1979). Embora autômatos sejam uma definição fraca, com sérias limitações se comparados com máquinas de Turing, são bons o suficiente para expressar strings de uma forma mais compacta que GIF.

Δ_{AGA} mapeia uma entrada $x_1x_2\cdots x_n$, cada $x_i \in \Sigma$ com $1 \leq i \leq n$, para

$$(\lambda^c(q_0, f_1)\lambda^c(q_1, f_2)\cdots\lambda^c(q_n, f_{n+1}), t_1\cdots t_{n+1})\,, \tag{6.6}$$

onde $q_0, q_1, \ldots, q_n$ é a seqüência de estados tal que $\delta'(q_{i-1}, x_i, t_i) = q_i$, para $1 \leq i \leq n$, e t_{n+1} é obtido da transição $\delta'(q_n, \varepsilon, t_{n+1}) = q_n$ (transição vazia) (ACCORSI; MENEZES, 2000; HOPCROFT; ULLMAN, 1979). $f_1, f_2, \ldots, f_{n+1}$ são funções de transformação.

Teorema 35 *AGA é melhor que GIF.*

Prova: (primeira parte) Nós devemos mostrar que AGA minora GIF, isto é, para qualquer animação alvo a e algum $c > 0$,

$$|G_{\mathrm{AGA}}(a)| \leq |G_{\mathrm{GIF}}(a)| + c\,. \tag{6.7}$$

É suficiente mostrar que, no caso geral, a Equação (6.7) vale.

Seja $a = (s, t)$ uma animação alvo qualquer, com $s = s_1s_2s_3\cdots s_n$ sendo uma seqüência de frames e $t = t_1t_2t_3\cdots t_n$, com $t_1 = t_2 = \cdots = t_n = t'$. t' é o intervalo de tempo entre cada par de frames contíguos na animação.

Então, para o formato GIF, $F = \{s_1, s_2, \ldots, s_n\}$, e $|\gamma_{\mathrm{LZW}}(F)|$ é o espaço de armazenamento necessário para os frames.

Nós usaremos a codificação prefixada E_2 (strings auto-delimitadas) para delimitar as partes que formam as strings (veja o Algoritmo 1). Lembre-se que $E_2(x) = \overline{|x|}x = 1^{||x||}0|x|x$, e o tamanho do código é $|E_2(x)| = |x| + 2\log|x| + 1$.

Nós precisamos delimitar os frames para implementar Δ_{GIF}. Usando a codificação E_2, nós sabemos que são necessários $O(\log|F|)$ bits mais $O(\log n)$ bits para delimitar o frame final. Então, $|\Delta_{\mathrm{GIF}}| = O(\log|F| + \log n)$ e da Equação (6.3),

$$|G_{\mathrm{GIF}}(a)| = |\gamma_{\mathrm{LZW}}(F)| + O(\log|F| + \log n) + c_{\mathrm{GIF}}\,,$$

para algum $c_{\mathrm{GIF}} > 0$.

Para o formato AGA, nós definimos a pior codificação, armazenando literalmente todos os frames e definindo o autômato mostrado na Figura 6.3. Então, $A = \{s_1, s_2, \ldots, s_n\}$. O autômato tem n estados, cada estado provoca a saída de um frame (ator). Observe que, na Equação (6.6), nós atribuímos $t_1 = t_2 = \cdots = t_n = t'$, porque no protocolo GIF todos os frames são mostrados com o mesmo intervalo de tempo, e $f_1 = f_2 = \cdots = f_n = \varepsilon$ é a transformação vazia.

Precisamos armazenar a fita de entrada $111\cdots1$, que são n 1's consecutivos. É sabido que precisamos $\log n$ bits para fazê-lo mais $O(\log|A| + \log n)$ para a codificação de prefixo dos frames. Assim,

$$|G_{\mathrm{AGA}}(a)| = |\gamma_{\mathrm{LZW}}(A)| + O(\log|A| + \log n) + c_{\mathrm{AGA}}\,,$$

para algum $c_{\mathrm{AGA}} > 0$.

É fácil concluir que (6.7) vale, porque o limite superior de $|G_{\mathrm{AGA}}(a)|$ é $|G_{\mathrm{GIF}}(a)| + c$, para algum $c > 0$.

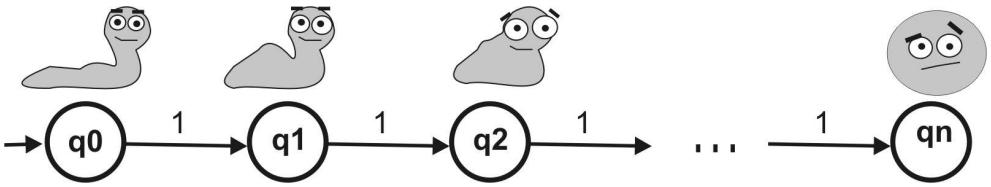

Figura 6.3: Autômato simples

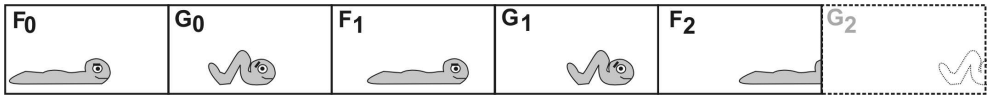

Figura 6.4: Animação a_0

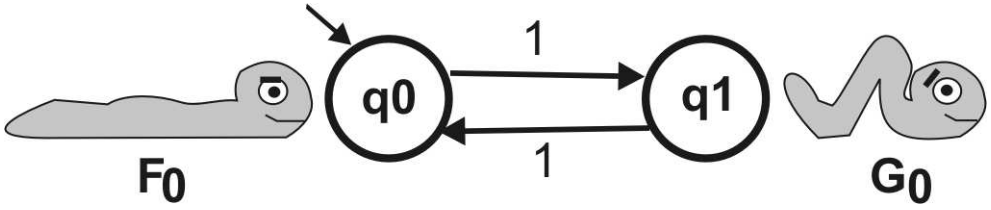

Figura 6.5: Animação a_0 representada usando-se AGA

(segunda parte) Esta encontrar a_0 de forma que $|G_{GIF}(a_0)| > |G_{AGA}(a_0)| + c'$ é satisfeito.

Nós escolhemos a_0 mostrado na Figura 6.4. Os frames F_0 e G_0 são ortogonalmente não relacionados (significa que não podemos encontrar regularidades entre eles, e, neste caso, GIF não pode comprimir a animação usando similaridades intra-frames). Todo F_i, $i > 0$, é obtido de F_{i-1} usando-se alguma transformação (como translação). Para AGA, nós construímos o autômato mostrado na Figura 6.5, F_0 e G_0 são os frames iniciais e a seqüência de transformações está armazenada na fita de entrada. Para GIF nós devemos armazenar todos os frames separadamente. □

O Teorema 35 é conseqüência do fato que a máquina-GIF armazena a animação literalmente e a máquina-AGA computa a animação a partir de um conjunto de atores e da fita de entrada (a compressão GIF é o limite superior da compressão AGA, em um sentido bastante próximo ao do Teorema 6). Isto é, a máquina-AGA simula a máquina-GIF.

6.5 Um Experimento de Comparação de Taxas de Compressão

Esta Seção do trabalho apresenta alguns dados empíricos obtidos de testes executados para comparar animações AGA e GIF com relação ao tamanho dos arquivos. Nosso propósito não é explorar profundamente os dados, mas apenas ilustrar os resultados teóricos das Seções precedentes.

As seqüências animadas usadas nestes testes foram desenvolvidas para ilustrar um curso de linguagens formais, desenvolvido com propósitos didáticos. Todas as três seqüências animadas representam algum autômato em ação (veja Tabela 6.1). A animação 1 mostra um autômato finito reconhecendo uma entrada. Durante a animação, todas

Tabela 6.1: Comparação de tamanhos AGA e GIF (bytes)

	Animação 1	Animação 2	Animação 3
AGA			
Imagens	25,750	26,190	44,648
Autômato	6,745	7,353	9,820
Total	32,495	33,543	54,468
GIF			
Frames completos	111,305	176,020	655,151
Frames parciais	64,558	93,508	278,513
Melhora GIF/AGA			
Frames completos	242.5%	424.8%	1,102.8%
Frames parciais	98.7%	178.8%	411.3%

as transições são mostradas (420×250 pixels, 10 frames, 256 cores). A animação 2 mostra um autômato de pilha durante o processo de reconhecimento. Todas as transições e o estado da pilha são mostrados (550×250 pixels, 14 frames, 256 cores). Finalmente, a terceira seqüência (animação 3) apresenta uma máquina de Turing lendo e escrevendo na fita de trabalho. A animação ilustra todas as transições de estados (650×350 pixels, 30 frames, 256 cores). Ambas as versões das seqüências de animação, GIF e AGA, tem o mesmo aspecto estético (tamanho de tela, contorno, número de frames, número de cores e comportamento dinâmico), para permitir a comparação.

Os resultados apresentados na Tabela 6.1 mostram que o tamanho dos arquivos AGA são menores que os dos arquivos GIF, para as três seqüências de animação descritas acima. Na Tabela, a linha rotulada como "Imagens" significa o tamanho, em bytes, das imagens usadas na animação; a linha "Autômato" significa o tamanho, também em bytes, do autômato associado com a animação. A linha "Total" representa a soma das duas precedentes. "Frames completos" e "Frames parciais" são duas maneiras diferentes de implementar seqüências animadas, disponíveis no protocolo GIF. GIF com frames parciais é um melhoramento de frames completos, permitindo atingir maior taxa de compressão.

Na Tabela 6.1, "Melhora GIF/AGA" significa a percentagem de melhora na compressão obtida no AGA comparado com a obtida no GIF. A fórmula usada foi

$$(\text{tamanhoGIF} - \text{tamanhoAGA})/\text{tamanhoAGA} \times 100\,.$$

Por exemplo, o primeiro valor é calculado como $(111,305 - 32,495)/32,495 \times 100 = 242.5\%$.

Dos testes apresentados, concluímos que bons resultados são obtidos com seqüências de animação muito curtas e simples. Como podemos ver na Tabela, melhores resultados podem ser obtidos com seqüências animadas mais longas.

6.6 Um Modelo Melhor Que AGA

Nós podemos propor uma definição mais sofisticada de $\phi = (\alpha, \Delta)$, o descritor de animação, permitindo que Δ seja qualquer função parcial recursiva. Denotamos isto por $\phi_{\mathcal{U}}$, e o chamamos *descritor universal*.

Assim, $\phi_{\mathcal{U}}$ é baseado na máquina de Turing universal $\mathcal{U}$ ao invés de em autômatos finitos como AGA.

Um argumento simples e direto mostra que o esquema de animação $G_{\mathcal{U}}$, baseado em $\phi_{\mathcal{U}}$, seria melhor que AGA.

Teorema 36 *$G_{\mathcal{U}}$ é melhor que AGA.*

Prova: Trivialmente, pela simulação do autômato finito na máquina de Turing (HOPCROFT; ULLMAN, 1979). □

Nós devemos observar que o Teorema 36 não trata do algoritmo de compressão de dados $\gamma_{\mathcal{U}}$. Supomos que ele é, por simplicidade, LZW, o mesmo de AGA.

Mudar nossa definição de Δ de autômatos finitos para máquina de Turing traz um problema. $\Delta_{\mathcal{U}}$ pode "congelar" durante a animação (porque é uma função parcial). Fora a nossa prova que $G_{\mathcal{U}}$ é melhor que AGA, este é um bom argumento para usarmos AGA ao invés de $G_{\mathcal{U}}$.

6.7 Principais Conclusões e Resultados Apresentados neste Capítulo

Este Capítulo apresenta uma aplicação da metodologia proposta de aplicar complexidade de Kolmogorov para avaliar e caracterizar modelos computacionais e sistemas complexos (veja (CAMPANI; MENEZES, 2001a)).

Os principais resultados apresentados são:

- Definições formais de "algoritmo de compressão", "melhor algoritmo de compressão" e "o melhor algoritmo de compressão", inspiradas em complexidade de Kolmogorov;
- Existem melhores algoritmos de compressão (melhores que um dado conjunto de algoritmos de compressão). Além disto, nós não podemos obter a melhor taxa de compressão (uma idéia originalmente apresentada em (SUBBARAMU; GATES; KREINOVICH, 1998));
- Mais tempo gasto na compressão significa maior taxa de compressão (compressão pode ser aproximada por cima por uma função recursiva);
- Definições formais de "animação" e "melhor animação";
- Aplicando o método proposto para comparar modelos de animação, mostramos que as animações AGA são melhores que as animações GIF ("melhor" significa que AGA é mais eficiente para comprimir dados que GIF);
- Um resultado adicional (apresentado na Seção 6.6), estende a nossa definição de descritor de animação e compara-o (o novo modelo) com o modelo AGA.

O método proposto pode ser generalizado para o caso em que ocorre perda de informação na compressão. Este é o tema do próximo Capítulo deste trabalho.

7 DEFINIÇÃO DE COMPRESSÃO DE DADOS COM PERDAS

O formalismo apresentado no Capítulo anterior está limitado a tratar com esquemas de compressão de dados sem perdas. No entanto, a maioria dos modelos de animação e imagem usa compressão de dados com perdas para obter maiores taxas de compressão.

Neste Capítulo estenderemos as considerações apresentados no Capítulo anterior, propondo uma definição de compressão de dados com perdas, inspirada em complexidade de Kolmogorov, usando o conceito de *distância de informação* (BENNETT et al., 1993; LI et al., 2003). O desafio em obter tal definição é que a complexidade de Kolmogorov refere-se a uma compressão "perfeita", ou seja, diferentemente da compressão com perdas, o descompressor não admite perdas, reconstruindo a string original sem mudanças. Assim, alguma adaptação deverá ser feita para permitir este tipo de definição formal.

7.1 Considerações a Respeito da Definição

No Capítulo anterior formalizamos compressão de dados sem perdas (como podemos perceber pela Equação $s = \delta(\gamma(s))$ da Definição 30). Mas, muitos codecs (especialmente os desenvolvidos para compressão de áudio e vídeo) comprimem sem preservar totalmente os dados (alguns bits podem "mudar"), com o propósito de obter mais compressão.

No presente Capítulo estenderemos a metodologia desenvolvida para tratar também com compressão com perdas. O desafio a enfrentar é que a complexidade de Kolmogorov é definida usando-se a máquina de Turing como um descompressor "perfeito", no sentido em que computa a string original, sem perdas (CAMPANI; MENEZES, 2003).

Um exemplo de uso de compressão com perdas é a compressão MPEG. MPEG é a designação para um grupo de padrões de áudio e vídeo. Ele é uma codificação genérica para imagens em movimento e o áudio associado. MPEG usa transformada cosseno discreto e codifica os dados com código Huffman (uma codificação assintoticamente ótima).

No entanto, não estamos interessados em como um padrão específico para compressão de dados com perdas escolhe o que será descartado quando os dados são comprimidos. Nós enfocamos uma definição matemática genérica para descrever e avaliar compressão de dados com perdas. A vantagem desta abordagem é que poderemos aplicar a definição a qualquer padrão indistintamente.

A idéia principal vem de (KREINOVICH; LONGPRE, 1998). Os autores sugerem que o armazenamento de informação parcial pode ser definido como uma distribuição de probabilidade. Escolhendo uma amostra desta distribuição e aplicando uma função sobre a amostra, computamos uma string similar à original, se e somente se a distribuição de probabilidade é representativa da string que está sendo comprimida. Mas, desta forma, o

método não é muito prático e não define a qualidade da compressão.

Para obter a definição, nós necessitamos usar um conceito de distância para medir a diferença entre a informação antes e depois do processo de compressão e descompressão. Estamos propondo, neste trabalho, o uso de *distância de informação* como esta métrica.

7.2 Distância de Informação

Muitas métricas podem ser definidas para medir a distância entre strings. Optamos por usar uma métrica universal chamada *distância de informação* (BENNETT et al., 1993; LI et al., 2003). Ela representa uma medida absoluta e a priori de distância no conjunto das strings binárias. Logo, sua aplicação independe do objeto ao qual desejamos aplica-la, já que concordamos que todos os objetos podem ser representados como strings binárias (por exemplo, imagens, animações, grafos, categorias, etc.).

Primeiro devemos definir o que é uma métrica, para em um segundo momento definirmos distância de informação e provarmos algumas propriedades que justificam o uso dela no contexto em que será aplicada neste trabalho.

Definição 39 *Seja S um espaço. Uma* métrica *para S é uma função definida sobre o produto cartesiano $S \times S$, $d : S \times S \rightarrow \mathbb{R}$, se e somente se para todo $x, y, z \in S$:*

1. $d(x, y) \geq 0$ *e* $d(x, y) = 0$ *se e somente se* $x = y$*;*
2. $d(x, y) = d(y, x)$ *(simetria);*
3. $d(x, y) \leq d(x, z) + d(z, y)$ *(desigualdade triangular).*

Um espaço provido de uma métrica é chamado um *espaço métrico* (LIMA, 1993).

Podemos observar que $K(x|y)$ já define uma distância entre as strings x e y, pois $K(x|y)$ representa a informação que é necessário adicionar à informação contida em y para computar x. No entanto, uma definição deste tipo poderia levar a se considerar grandes strings mais diferentes que as pequenas. Devemos definir distância de uma forma relativa, interpretando uma diferença de 100 bits entre duas strings de 1.000.000 de bits como menos diferença que os mesmos 100 bits entre duas de 10.000 bits. Assim, uma formulação como $K(x|y)/K(x)$ parece ser mais representativa para esta métrica. Chamamos este tipo de formulação de *distância normalizada* (LI et al., 2003).

No entanto, tal formulação não seria capaz de respeitar as propriedades formuladas para as métricas (Definição 39). Para resolver este problema a distância de informação normalizada será definida como na Definição 40.

Definição 40 Distância de informação *é definida como*

$$d(x, y) = \frac{\max(K(x|y), K(y|x))}{\max(K(x), K(y))}.$$

É importante observar que $d(.,.) \in [0; 1]$. Assim, $d(x, y)$ representará similaridade entre x e y quanto mais $d(x, y)$ se aproximar de 0 e representará diferença quando $d(x, y)$ se aproximar de 1.

É trivial demonstrar que $d(.,.)$ respeita a propriedade da simetria. Podemos mostrar que $d(.,.)$ respeita o *axioma da identidade*, ou seja, que $d(x, x) = O(1/K(x))$ para todo

x (LI et al., 2003), já que $K(x|x) \approx O(1)$. Também podemos mostrar que esta definição de $d(.,.)$ satisfaz o *teorema da desigualdade triangular fraca* (LI et al., 2003), ou seja,

$$d(x,y) \leq d(x,z) + d(z,y) + O\left(\frac{1}{\max(K(x), K(y), K(z))}\right) .$$

Assim, a não ser por termos pequenos, assintoticamente desprezáveis, $d(.,.)$ respeita as propriedades exigidas de uma métrica, ou seja, a distância de informação respeita uma definição mais fraca de métrica sobre o espaço das strings binárias.

Devemos esperar, das medidas que usarmos, por serem normalizadas, que elas respeitem a *condição de densidade* (LI et al., 2003). Seja $f(x,y)$ uma distância normalizada semi-computável por cima qualquer, então:

$$\sum_{y:y\neq x} 2^{-f(x,y)} \leq 1 ,$$

chamada de *condição de densidade*.

Conclui-se que

$$\overline{\overline{\{y : d(x,y) \leq d, K(y) \leq k\}}} \leq 2^{dk} .$$

Pois, supondo o contrário, obtemos a seguinte contradição:

$$\begin{aligned} 1 &\geq \sum_{y:y\neq x} 2^{-f(x,y)K(x)} \geq \sum_{y:y\neq x \wedge d(x,y)\leq d \wedge K(y)\leq k} 2^{-dk} \\ &> 2^{dk}2^{-dk} = 1 \end{aligned}$$

O Teorema 37 prova uma importante propriedade da distância de informação, chamada *universalidade* de $d(.,.)$. Ela significa que, comparada a qualquer distância normalizada semi-computável por cima (ou semi-enumerável por cima), $d(.,.)$ sempre fornece uma distância menor, a não ser por um termo logarítmico assintoticamente desprezável.

Teorema 37 $d(x,y) \leq f(x,y) + O((\log\max(K(x),K(y)))/\max(K(x),K(y)))$*, onde* $f(.,.)$ *é uma distância normalizada semi-computável por cima qualquer.*

Prova: Fixando uma f *e supondo* $f(x,y) = d$*. Pela condição de normalização, dado* x*,*

$$\overline{\overline{\{y : d(x,y) \leq d, K(y) \leq k\}}} \leq 2^{dk} .$$

Assim, fixando x^* *e* k*, podemos enumerar recursivamente os* y *do conjunto, e cada* y *pode ser descrito pelo seu índice na enumeração, de tamanho menor ou igual à* dk*.*

Portanto, já que a complexidade de Kolmogorov é a menor descrição possível, então ela é menor que esta descrição efetiva, e obtêm-se $K(y|x^*,k) = K(y|x,K(x),k) \leq dk$*. Isto é, como nós podemos fornecer a descrição de* k *e* $K(x)$ *em* $O(\log k)$ *bits,* $K(y|x) \leq dk + O(\log k)$*. Dados* x *e* y*, assumimos que* $K(y) > K(x)$ *(logo,* $K(y) = k$*), e*

$$d(x,y) = \frac{K(y|x)}{K(y)} \leq \frac{dk + O(\log k)}{k} = f(x,y) + O\left(\frac{\log k}{k}\right) .$$

O caso $K(x) > K(y)$ *é simétrico e similar.* □

O Teorema 37 prova que $d(.,.)$ incorpora todo tipo de definição de similaridade normalizada semi-computável por cima, significando que se dois objetos são similares em um sentido computável e normalizado, então serão no mínimo tão similares quanto segundo $d(.,.)$ (embora ela própria, $d(.,.)$, não seja semi-computável por cima). Ou seja, se dois objetos são similares segundo alguma métrica normalizada semi-computável por cima, o serão também segundo $d(.,.)$ (LI et al., 2003).

7.3 Definição Formal

Agora já estamos preparados para definir formalmente compressão de dados com perdas, usando distância de informação.

Nesta definição, apresentamos a versão comprimida de uma string s qualquer como sendo um programa p para a máquina de Turing, que uma vez executado, produz uma aproximação s' da string s. Devemos definir a relação necessária entre s e s' (a perda de qualidade máxima permitida na compressão). Observe que a definição não faz menção a forma com que a string s será comprimida (e daí sua generalidade). Apenas define a representação comprimida, o descompressor e o requisito de qualidade.

Definição 41 *Suponha $s' = \delta(\gamma(s))$ e $s \neq s'$ (compressão de dados com perdas). Nós dizemos que s' é uma ϵ-aproximação de s se $d(s, s') \leq \epsilon$, $\epsilon \in [0; 1]$.*

A compressão com perdas, r-comprimida com ϵ-aproximação, aplicada a string s, pode ser definida como o programa p que computa s', $d(s, s') \leq \epsilon$ e $r = |s|/|p|$, de forma que, para algum c, $|p| \leq |s| + c$ e

$$K(s') \leq K(s) + O(1)\,. \tag{7.1}$$

Equação (7.1) segue do teorema que afirma $K(f(x)) \leq K(x) + O(1)$, onde f é qualquer função computável sobre strings binárias (veja o Teorema 8, que apresenta a versão deste Teorema para a complexidade C, e (LI; VITÁNYI, 1997)). No nosso caso, $f = \delta \circ \gamma$. Isto significa que a versão comprimida da string tem menos informação que a original.

Observe que na Definição a ϵ-aproximação representa a qualidade da compressão, usando-se distância de informação.

Esta definição tem dois principais méritos:

- Adapta a definição de complexidade de Kolmogorov, tradicionalmente associada à máquina de Turing como um descompressor sem perdas, ao caso em que ocorre perdas no processo de compressão/descompressão;
- Propõe o uso de distância de informação como uma métrica de qualidade de imagem, idéia que é inédita, segundo pesquisa feita na literatura da área.

Discutimos a escolha da distância de informação como métrica nos Capítulos 8 e 9. Também apresentamos no Capítulo 8 uma revisão bibliográfica resumida relativa a métricas de qualidade de imagens.

8 MÉTRICAS DE QUALIDADE DE IMAGEM

Muitas vezes não é possível (ou desejável) armazenar imagens em sua forma original. Isto ocorre normalmente em casos em que é necessário comprimir os arquivos de imagens e compressão sem perdas não é suficiente (KOSHELEVA; KREINOVICH; NGUYEN, 2004), ou quando temos de aplicar algoritmos de tratamento de imagem que distorcem de alguma forma a imagem original.

Sabe-se que algoritmos de compressão com perdas oferecem, de um modo geral, mais compressão que os sem perdas. Em muitos casos, compressão com perdas (como a compressão JPEG) é a única alternativa para se obter uma taxa de compressão adequada para a aplicação desejada, como por exemplo quando o canal a ser usado para o envio das imagens é muito deficiente em termos de largura de banda, ou a memória disponível para armazenamento é pequena.

Compressão de imagens envolve dois algoritmos, um de compressão e outro de descompressão. O algoritmo de compressão transforma a imagem original I_0 em uma representação comprimida $\gamma(I_0)$, enquanto que o algoritmo de descompressão transforma esta representação comprimida em uma imagem reconstituída I_1. Na compressão com perdas a imagem reconstituída I_1 é, em geral, diferente da original I_0.

Existem muitos métodos de compressão com perdas, e a maioria deles possui muitos parâmetros selecionáveis. Diferentes métodos, usando diferentes parâmetros, podem produzir reconstituições de imagem com diferentes qualidades.

Um outro caso em que ocorre a necessidade de armazenar a imagem transformada é quando precisamos aplicar algum tipo de distorção na imagem para adequa-la a aplicação a que se destina. Por exemplo, para eliminar algum ruído aplicando um efeito de desfocamento. Também neste caso existem muitos parâmetros que podem ser selecionados para ajustar o efeito desejado.

Tendo-se uma forma para medir a qualidade da imagem obtida, é possível ajustar os parâmetros aplicados de modo a obter uma imagem de qualidade satisfatória ao nosso interesse.

Assim, é importante ter uma medida de qualidade, que determine a diferença (ou grau de similaridade) entre a imagem original e a distorcida (KOSHELEVA; KREINOVICH; NGUYEN, 2004). Medidas deste tipo são chamadas de *métricas de qualidade de imagem* (AHUMADA, 1993).

Existem muitos modelos e métricas para qualidade de imagem que determinam a discriminabilidade, ou seja, a diferença visual entre duas imagens dadas, outras métricas enfatizam a detecção de objetos pela observação de um conjunto (normalmente grande) de objetos de imagem e comparação com um conjunto de imagens de fundo (ROHALY; AHUMADA; WATSON, 1995).

Um importante problema envolvido em avaliações deste tipo é o princípio da *loca-*

lidade (KOSHELEVA; KREINOVICH; NGUYEN, 2004). Isto significa que diferentes partes da imagem são independentes umas das outras. Por exemplo, em astronomia, a reconstrução de parte de uma imagem (como uma galáxia), não depende do resto da imagem.

Intuitivamente, duas imagens são semelhantes ou próximas se para cada pixel de ambas, os valores dos pixels (cor ou intensidade) em ambas as imagens são próximos ou iguais.

Devemos formalizar o que chamamos de "distância" entre imagens. Dadas duas imagens I_0 e I_1,

$$Q = (P(I_0) - P(I_1))^2 \tag{8.1}$$

é o quadrado da *distância euclidiana* ou *distância RMS* (AHUMADA, 1993). Se p_{0ij} e p_{1ij} representam os valores dos pixels na posição (i, j) de I_0 e I_1, então

$$Q = \sum (p_{0ij} - p_{1ij})^2 . \tag{8.2}$$

Na equação 8.1, P é a função que representa o *modelo visual*, ou seja, a função que torna a distância probabilisticamente compatível com o que um observador perceberia. Algumas vezes é relatado na literatura que o uso de algum filtro, como função de modelo visual, acabou produzindo resultados piores que a simples aplicação da distância RMS (AHUMADA, 1993).

É importante observar que, neste trabalho, não trataremos com modelos visuais, mas sim apenas com medidas de distância entre imagens.

Existe uma generalização da medida apresentada na Equação 8.2. Chama-se *métrica de Minkowski* (AHUMADA, 1993):

$$Q = \left[\sum (\text{abs}(p_{0ij} - p_{1ij}))^e \right]^{1/e} .$$

Observe-se que se $e = 2$ obtêm-se a distância euclidiana, se $e = 4$ resulta no *somatório de probabilidade*, e aproxima-se a *diferença absoluta máxima* quando e torna-se muito grande. Assim, existe um número muito grande (contável) de medidas, no entanto, a distância euclidiana é a de uso mais amplo, segundo (AHUMADA, 1993).

Outra medida de distância usada como métrica é a *distância de informação de Kullback-Leibler* (também chamada de *entropia relativa*) (AVCIBAS; MEMON; SANKUR, 2003):

$$d = \sum_k p_k \log \frac{p_k}{q_k} .$$

Em (AHUMADA, 1993) encontra-se um resumo das principais métricas propostas na literatura desde 1972 até a data da publicação, quase todas baseadas na distância euclidiana.

Nosso objetivo neste trabalho é propor e validar uma métrica de qualidade de imagem que incorpore em uma única medida de similaridade de imagens toda e qualquer métrica efetiva. Esta medida é a *distância de informação* formalmente definida na Seção 7.2, e a propriedade que buscamos é provada no Teorema 37.

No Capítulo 9 apresentamos a metodologia usada na aplicação da distância de informação, assim como uma série de experimentos, usando esta metodologia, com o objetivo de validar a proposta de uso da distância de informação como métrica de qualidade de imagem. Também propomos um refinamento no método ao final do mesmo Capítulo.

9 AVALIAÇÃO DA QUALIDADE DE IMAGENS E ANIMAÇÕES USANDO DISTÂNCIA DE INFORMAÇÃO

Este Capítulo explora a definição dada no Capítulo 7 de distância de informação (BENNETT et al., 1993; LI et al., 2003) como uma nova métrica de qualidade de imagem. Esta proposta é, ao que se sabe, inédita na literatura e tem como vantagens a universalidade da distância de informação e a possibilidade de aplicar a medida em casos que não é possível aplicar a distância euclidiana (a medida tradicionalmente usada nestes casos).

Os objetivos do trabalho desenvolvido neste Capítulo são:

- Validar a medida proposta neste novo contexto através de um conjunto de experimentos;
- Comparar os resultados obtidos com a medida proposta com os resultados obtidos com a medida tradicionalmente usada (distância euclidiana);
- Estender a medida proposta para avaliar também animações baseadas em frames. Neste aspecto, este trabalho introduz um avanço científico pois, segundo pesquisa bibliográfica feita, o único tipo de avaliação de animações até então desenvolvida, tratava apenas dos aspectos psicológicos da percepção de deformações no movimento (O'SULLIVAN et al., 2003).

A propriedade de *universalidade* da distância de informação (veja Teorema 37) significa que a distância de informação, $d(.,.)$, incorpora toda e qualquer distância computável concebível, ou seja, se existe uma distância normalizada $f(.,.)$ que é uma boa medida em determinado caso, $d(.,.)$ também será uma boa medida.

Esta é uma propriedade distintiva da distância de informação, que a torna, em certo sentido formal, a medida "perfeita" para todos os casos.

Além disto, muitas vezes não é possível aplicar a distância euclidiana, seja porque desejamos comparar distâncias calculadas sobre imagens de tamanho diferente, e a distância euclidiana não é normalizada, seja porque desejamos simplesmente medir a distância entre duas imagens de tamanhos diferentes (o que não pode ser feito com a distância euclidiana). No entanto, veremos que tais avaliações podem ser feitas usando-se a distância de informação, e as avaliações, neste caso, produzem resultados coerentes.

A aplicação proposta aqui tem alguns precedentes na literatura. Em primeiro lugar, citamos a caracterização da literatura russa por meio de entropia clássica (KOLMOGOROV, 1965), para cuja estimativa foi usado o dicionário de russo de Ozhegov. Kolmogorov relata um resultado de $H = 1,9 \pm 0,1$. Embora não tenha sido usada a distância de informação no trabalho, ele tem algum parentesco com a aplicação da complexidade

de Kolmogorov que usamos aqui, haja visto o relacionamento entre H e K (veja Teorema 23).

Já por sua vez em (LI et al., 2003) são apresentadas duas aplicações da distância de informação bastante relacionadas com o que propomos aqui. A primeira é a aplicação da distância de informação em seqüências de DNA para reconhecimento de genoma mitocondrial (LI et al., 2003). Os resultados demonstram que foi possível reconhecer as relações entre três grupos de mamíferos placentários. A segunda aplicação apresentada neste artigo preocupa-se em aplicar a distância de informação para a determinação do parentesco de línguas, formando uma hierarquia de linguagens humanas. O experimento baseou-se na tradução da "Declaração Universal dos Direitos do Homem" para 52 línguas diferentes.

Finalmente, o último trabalho relacionado com o nosso, e o mais recente, foi o uso da distância de informação para classificação automática de música em formato MIDI (CILIBRASI; VITÁNYI; WOLF, 2004). Os resultados foram bons o suficiente para que o programa discernisse entre gêneros musicais (clássico, rock, jazz, etc.) e entre autores (Mozart, Bach, Beethoven, etc.) dentro de um gênero musical.

9.1 Aproximação Usada

Para aplicar a distância de informação como uma métrica de qualidade de imagens, do ponto de vista prático, teremos que enfrentar a não computabilidade de K. Ou seja, teremos que usar uma aproximação computável de K, como por exemplo codificação Lempel-Ziv (programa gzip) ou o algoritmo Burrows-Wheeler (programa bzip2 e biblioteca libbzip2).

Ainda assim, para que possamos usar o programa compressor, devemos eliminar todas as complexidades condicionais. Pela propriedade da simetria da informação algorítmica (GÁCS, 1974), nós sabemos que $K(x|y) \approx K(yx) - K(y)$, que resulta em:

$$d(x,y) = \frac{\max(K(yx) - K(y), K(xy) - K(x))}{\max(K(x), K(y))}\,, \tag{9.1}$$

onde xy e yx são as concatenações das strings x e y. Desta forma, ficam eliminados todos os condicionais. Sabendo que $K(xy) \approx K(yx)$, podemos reescrever a Equação (9.1) como

$$d(x,y) = \frac{K(xy) - \min(K(x), K(y))}{\max(K(x), K(y))}\,, \tag{9.2}$$

que é uma forma de calcular a distância que aumenta a velocidade do cálculo pois elimina uma chamada ao compressor. Então, é apenas necessário usar o compressor para comprimir as strings x, y e xy, e verificar o tamanho dos arquivos comprimidos.

Devemos observar que, se x e y são strings binárias, $0 \leq d(x,y) \leq 1$, e resultados próximos de 0 representam similaridade e próximos de 1 diferença. Mas, pela aproximação usada, devido às imperfeições do algoritmo de compressão de dados, os resultados práticos ficam no intervalo $0 < d(x,y) < 1,1$. Além disto, a medida é mais eficiente (fornece melhores resultados) com objetos mais regulares (mais simples) que com objetos mais complexos (mais aleatórios).

Para os experimentos apresentados neste Capítulo, foi usado o pacote *complearn* de Rudi Cilibrasi, e foi escolhido o compressor bzip2 como aproximação de K, embora o programa possa trabalhar também com o compressor gzip.

9.2 Aplicação da Distância de Informação em Imagens

Nesta Seção deste Capítulo e nas seguintes, apresentamos os experimentos efetuados usando distância de informação, junto com a descrição da metodologia usada para efetuar as medições.

Os experimentos fizeram uso de dois conjuntos distintos de imagens, um primeiro menor (nos quais foram feitos mais testes) e um segundo com maior número de imagens.

Em um primeiro conjunto de experimentos, aplicamos diversas distorções (usando o programa The Gimp), escolhidas arbitrariamente, sobre um conjunto de imagens e determinamos as distâncias das imagens distorcidas em relação à imagem original. Então, procuramos correlações entre as distorções aplicadas nas imagens e as distâncias calculadas.

Usamos como aproximação computável de K, como já dissemos, o algoritmo Burrows-Wheeler, que é usado no programa bzip2 e na biblioteca libbzip2. Para facilitar os cálculos não usamos diretamente o bzip2, mas o pacote complearn, que é um laboratório de experimentos com compressão de dados desenvolvido por Rudi Cilibrasi (url: `http://complearn.sourceforge.net`).

Em um segundo conjunto de experimentos, efetuamos comparações das distâncias calculadas usando distância de informação com as calculadas usando distância euclidiana, procurando verificar qual das duas apresentava as correlações mais corretas para o conjunto de imagens escolhido.

Com relação às imagens, as mesmas foram obtidas arbitrariamente na Internet, em sites de fotografia e em grupos de discussão, ou digitalizadas com uma câmera digital. No entanto, algumas foram desenhadas para testar alguns casos limites, particularmente imagens simples, monocromáticas com fundo branco.

Finalmente, aplicamos a mesma avaliação de qualidade de imagem a um conjunto de animações baseadas em frames, e comparamos os resultados com os obtidos com imagens.

Tivemos alguns cuidados nos experimentos para não prejudicar os resultados. As imagens tiveram seus tamanhos padronizados para 250 por 200 pixels para eliminar outros fatores nos testes. Este tamanho foi obtido tanto por redimensionamento, quanto por corte de partes da imagem para não prejudicar o aspecto.

Já que qualquer esquema de compressão embutido no formato iria interferir com a compressão do programa bzip2, usada para aproximar K, todas as imagens foram salvas em formato BMP não comprimido e não indexado (RGB). Além disto, os cabeçalhos dos arquivos BMP foram retirados usando-se um programa Python desenvolvido especialmente para estes experimentos.

Assim, ao final, os arquivos das imagens estavam reduzidos a uma seqüência de triplas RGB (três bytes para cada pixel, fornecendo uma cor RGB), mais apropriada, segundo nossa experiência pessoal, para ser aplicada ao cálculo das distâncias.

Uma preocupação importante foi não usar imagens muito pequenas (pois diminui a eficiência do compressor), nem maiores que 450KB, por limitação da libbzip2 que foi implementada com tamanho de bloco máximo de 900KB (imagens maiores que 450KB não seriam comprimidas muito bem, pois há a necessidade de comprimir a concatenação xy de x e y, para aplicar a Equação 9.2).

Os softwares usados nos experimentos foram os seguintes:

The Gimp usado para tratar as imagens, converter de formato, e produzir as distorções usadas nos testes (url: `http://gimp.org`);

strip.py programa Python que retira o cabeçalho de arquivos BMP e adicionalmente fornece uma série de informações importantes sobre a imagem (veja listagem na Figura 9.1);

difeuclidrgb programa Python que calcula a distância RMS entre duas imagens;

complearn pacote de autoria de Rudi Cilibrasi que contém um conjunto de programas para experimentos com compressão de dados usando a biblioteca libbzip2 (url: `http://complearn.sourceforge.net`);

refina programa Python usado para efetuar os refinamentos das medições (veja listagem na Figura 9.8).

Trabalhamos com dois conjuntos distintos de imagens. O primeiro conjunto é menor, possuindo apenas 12 imagens, mas sobre ele foram aplicadas mais distorções para os testes. Este primeiro conjunto é exibido na figura 9.2 (as 12 imagens estão ordenadas da esquerda para a direita e de cima para baixo) e é composto, na ordem, por três imagens de desenhos animados, três imagens simples de fundo branco, desenhadas especialmente para os testes, três paisagens e três rostos fotografados com câmera fotográfica digital.

O segundo conjunto é maior, composto de 32 imagens, porém sobre ele foram aplicados menos testes. Este segundo conjunto foi todo obtido arbitrariamente em sites e grupos de discussão na Internet, tendo sido descartados apenas as imagens que não se adaptavam a serem redimensionadas para o tamanho padrão que foi usado em todos os testes. Este conjunto de imagens é apresentado na Figura 9.3.

As Distorções usadas são apresentadas a seguir e ilustradas na Figura 9.4 (em ordem, da esquerda para a direita, e de cima para baixo, iniciando pela imagem original no topo à esquerda):

Brilho -25% Significa redução de brilho de 25%;

Desfocar Aplicou-se o desfocamento gaussiano IIR, com raio de desfocamento de 5 pixels na horizontal e 5 pixels na vertical;

Deslocar 5 Significa deslocar a imagem 5 pixels na horizontal da esquerda para a direita com “embrulhamento” (ou seja, os pixels retirados da direita são inseridos na esquerda ao deslocar);

Deslocar 15 O mesmo que o anterior, porém deslocando 15 pixels na horizontal, da esquerda para a direita, com “embrulhamento”;

Espalha 2 pixels Significa introduzir um ruído na imagem com raio de 2 pixels, que faz com que a imagem pareça “granulada”.

Os resultados do cálculo das distâncias entre a imagem original e a distorcida são apresentados nas Tabelas 9.1 e 9.2. A primeira apresenta as distâncias calculadas para as distorções deslocar 5 pixels, deslocar 15 pixels e desfocar. A segunda tabela apresenta os resultados para redução de brilho de 25% e espalhamento.

Em primeiro lugar devemos observar na análise dos resultados que, na primeira tabela, a diferença média entre as distâncias para desfocar e deslocar 5 pixels foi 0,028113561 e entre deslocar 5 pixels e deslocar 15 pixels foi 0,0003963498, cerca de 70 vezes menor, o que torna as distorções deslocar 5 pixels e deslocar 15 pixels mais próximas uma da outra que a distorção desfocar.

```
#!/usr/bin/python
# programa que retira o cabecalho de arquivo BMP
# autor: Carlos Campani
from struct import *
from sys import *
f = open(argv[1],"r")
x = f.read()
f.close()
magic = x[0:2]
offset = unpack("I",x[10:14])[0]
headersize = unpack("I",x[14:18])[0]
width = unpack("I",x[18:22])[0]
height = unpack("I",x[22:26])[0]
nbits = unpack("H",x[28:30])[0]
compression = unpack("I",x[30:34])[0]
sizeimage = unpack("I",x[34:38])[0]
ncolors = unpack("I",x[46:50])[0]
tamanhototal = 14+40+4*ncolors
print "magic=%s\noffset=%u\nheadersize=%u\nwidth=\
%u\nheight=%u\nnbits=%u\ncompression=%u\nsizeimage=\
%u\nncolors=%u\ntamanhototal=%u"%(magic,offset,\
headersize,width,height,nbits,compression,sizeimage,\
ncolors,tamanhototal)
if compression == 0 :
   print "sem compressão!"
else:
   print "BMP comprimido!"
print "stripping..."
f = open(argv[1]+".strip","w")
f.write(x[offset:])
f.flush()
f.close()
print "stripped."
```

Figura 9.1: Programa strip.py

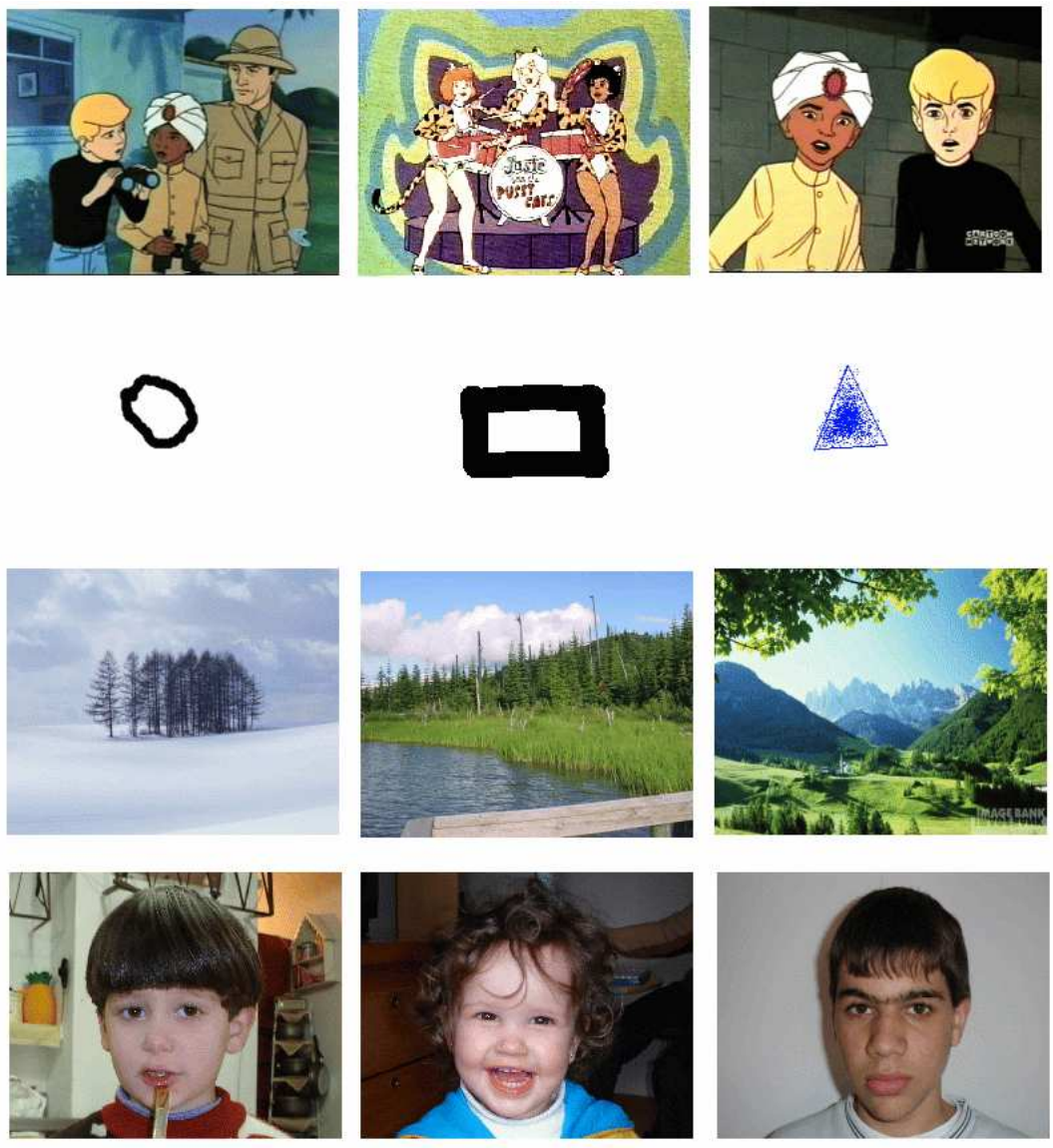

Figura 9.2: Primeiro Conjunto de Imagens Usadas nos Experimentos

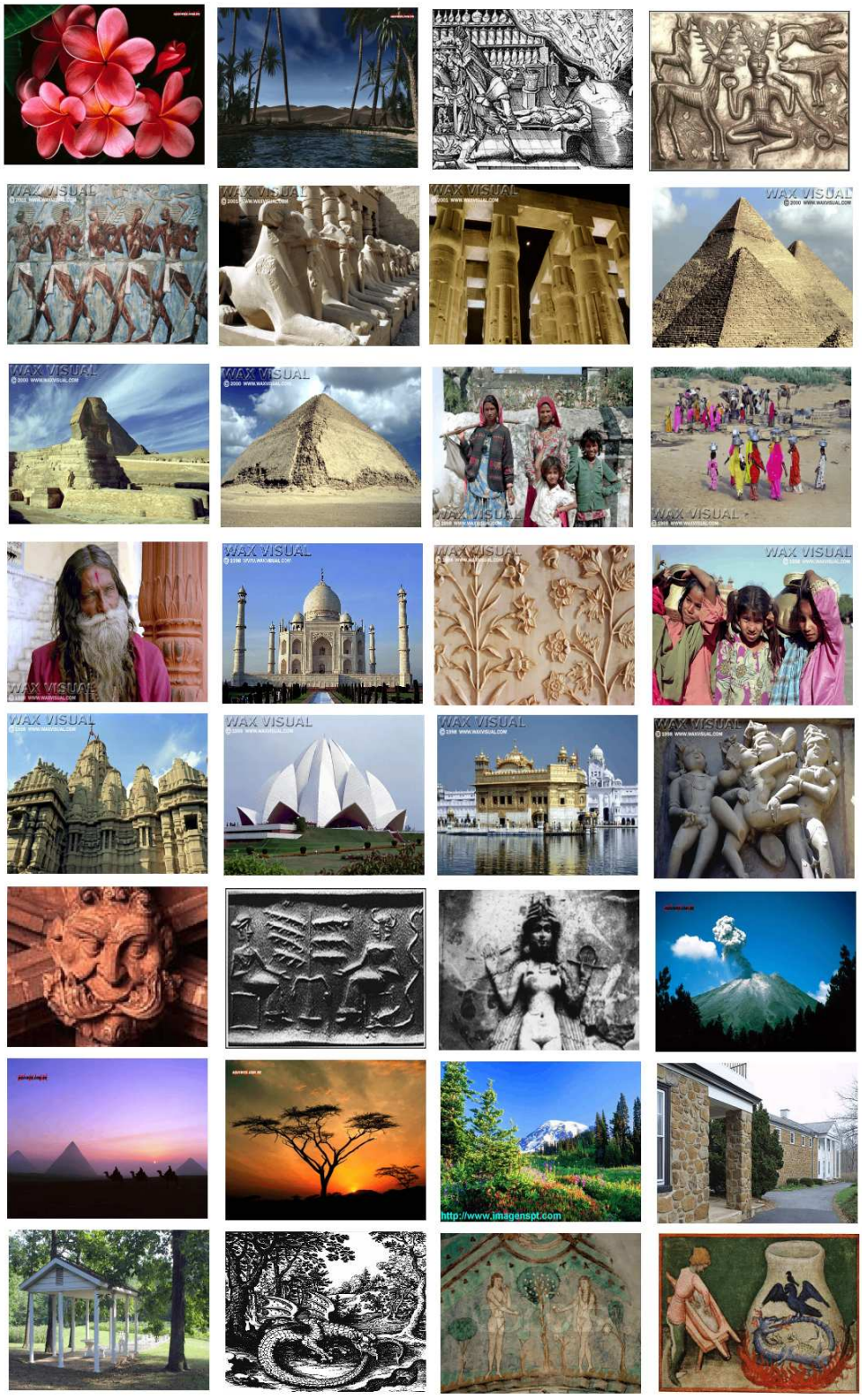

Figura 9.3: Segundo Conjunto de Imagens Usadas nos Experimentos

Figura 9.4: Distorções Usadas nos Experimentos

Tabela 9.1: Resultados de Experimento (Deslocar e Desfocar)

	Deslocar 5		Deslocar 15		Desfocar
1	0,988454481298518	<	0,988460344778941	<	0,99809956034605
2	0,993519626862591	<	0,99360455520713	<	1,01384577176208
3	0,995465847593974	<	0,995494177541108	<	1.00105667909243
4	0,922101449275362	>	0,921266968325792	<	1,02131496966716
5	0,938983050847458	<	0,940677966101695	<	1,021071919377
6	0,944967773921666	<	0,945409429280397	<	1,0124593716143
7	0,99595295307955	<	0,996553250695674	<	1,00098647573588
8	1,00451352851729	<	1,00470400926328	>	1,00147782057805
9	0,986316683808076	>	0,98628082896916	<	1,00715054245494
10	0,986084786833329	<	0,986344688802645	<	1,00334448160535
11	0,996896059953581	<	0,996958705357143	<	0,999831616759745
12	0,996125196245449	<	0,996602038776734	<	1,00003354128933

Tabela 9.2: Resultados de Experimento (Brilho e Espalhamento)

	brilho -25%		Espalha 2 pixels
1	0,999121763272707	>	0,992283178382365
2	1,00951947353172	>	0,994888033445093
3	1,00231041441231	>	1,00048344898191
4	0,961050724637681	>	0,945525291828794
5	0,944162436548223	<	0,966292134831461
6	0,96575682382134	>	0,957264957264957
7	1,00155926809865	>	0,997179246089409
8	1,0052084589231	<	1,00589304015653
9	0,998834968857821	>	0,986876816454281
10	0,999111580666074	>	0,987112721687585
11	0,999607348198009	>	0,997829491441602
12	0,999463339370765	>	0,997267031062525

Tabela 9.3: Resultados de Experimento com Composição de Distorções

	Deslocar 15		Desloc.+Desfocar		Desfocar
1	0,988460344778941	$<$	1,00446140662408	$>$	0,99809956034605
2	0,99360455520713	$<$	1,01064329545788	$<$	1,01384577176208
3	0,995494177541108	$<$	1,00501810182851	$>$	1.00105667909243
4	0,921266968325792	$<$	1,02113714566607	$<$	1,02131496966716
5	0,940677966101695	$<$	1,01993127147766	$<$	1,021071919377
6	0,945409429280397	$<$	1,01715237302248	$>$	1,0124593716143
7	0,996553250695674	$<$	1,00337559972584	$>$	1,00098647573588
8	1,00470400926328	$<$	1,00635397771443	$>$	1,00147782057805
9	0,98628082896916	$<$	1,00956256713446	$>$	1,00715054245494
10	0,986344688802645	$<$	1,00934009967623	$>$	1,00334448160535
11	0,996958705357143	$<$	1,00466293479872	$>$	0,999831616759745
12	0,996602038776734	$<$	1,00461566062662	$>$	1,00003354128933

Além disto, observa-se, ao contar o número de vezes em que cada distorção ocupou maior ou menor valor na tabela, que:

- Desfocar: $1^{\underline{o}}$ em $8/12$ casos;
- Brilho -25%: $2^{\underline{o}}$ em $8/12$ casos;
- Espalha 2 pixels: $3^{\underline{o}}$ em $10/12$ casos;
- Deslocar 15 pixels: $4^{\underline{o}}$ em $9/12$ casos;
- Deslocar 5 pixels: $5^{\underline{o}}$ em $9/12$ casos.

Combinados estes resultados demonstram que ocorreu uma caracterização da distorção, dada pelos valores das distâncias calculadas em relação às imagens originais. Houve, na média, *correlação* entre a distorção aplicada e a distância calculada.

Uma questão importante é se a métrica proposta é transitiva, ou seja, se ao compor distorções a distância aumentará proporcionalmente. A Tabela 9.3 apresenta o teste feito para verificar isto. Ela apresenta as distâncias para “deslocar 15”, “desfocar” e a composição de ambas (obtido ao desfocar no *The Gimp* a imagem que já fora deslocada).

Confirma-se a hipótese, como se pode ver pelos resultados, que estão resumidos na Figura 9.5, onde $A \longrightarrow B$ significa que A apresenta distâncias menores que B.

Apenas as linhas 2, 4 e 5 da Tabela 9.3 não apresentaram a correlação esperada para “Deslocar+Desfocar” e “Desfocar”. Relativo a isto, a linha 4 da Tabela apresenta uma diferença (0,000177824) cerca de vinte vezes menor que a diferença média destas colunas (0,0037186316). A aplicação de um limiar, abaixo do qual poderíamos considerar a diferença muito pequena, tornaria este um resultado considerado “igual” ou “não conclusivo”, melhorando ainda mais os resultados apresentados na Tabela.

Esta idéia poderia ser aplicada em diversos resultados em que as diferenças são pequenas (atribuído ao erro na aproximação de K usada).

Resumindo os resultados dos experimentos desta Seção, podemos afirmar que, na média, ocorreu correlação entre as distorções aplicadas e as distâncias calculadas, indicando

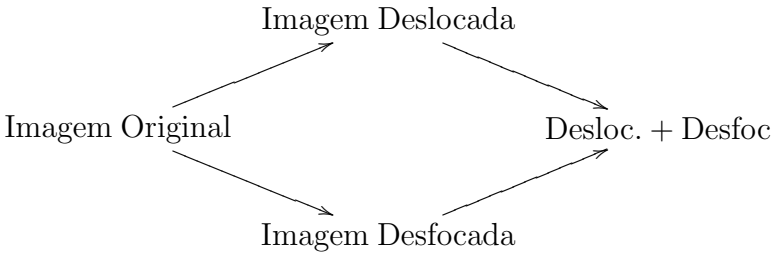

Figura 9.5: Relações entre as Distorções e Distâncias Calculadas

que a distância de informação foi capaz de caracterizar estas distorções, servindo assim como uma medida de qualidade de imagem.

9.3 Comparação da Distância de Informação com a Euclidiana

Um aspecto de interesse em nossa pesquisa era avaliar a medida proposta comparando-a com a distância euclidiana, que é a medida mais usada como métrica de qualidade de imagem (AHUMADA, 1993).

Para isto, foram aplicadas as distorções "desfocar" e "deslocar 5 pixels" sobre as 32 imagens do segundo conjunto (Figura 9.3). Depois, foram calculadas ambas as distâncias (informação e euclidiana) em relação à imagem original e foram comparadas as correlações obtidas com ambas.

Os resultados destes testes estão tabulados nas Tabelas 9.4 e 9.5. Na verdade, na Tabela 9.5 aparece o valor do quadrado da distância euclidiana (distância RMS) dividido arbitrariamente por 10^{15}.

Ao analisar os resultados, constatamos que, para a distância de informação, em $29/32$ dos casos, a distância resultou na relação *desfocado*$>$*deslocado*, que é coerente com os experimentos anteriores. Assim, em apenas três casos não se verificou a correlação esperada.

Já no caso da aplicação da distância euclidiana, nos 32 casos verificou-se a relação *desfocado*$<$*deslocado*, confirmando-se a correlação.

O fato da relação entre as distâncias ser a inversa da observada com a distância de informação não é problema, pois cada medida interpreta as distâncias de uma forma diferente, e estamos tratando com *diferenças* de distâncias, e o mais importante é a ocorrência de correlação.

É importante observar que o ponto flutuante, que aparece na distância euclidiana, foi arbitrário para tornar a leitura dos números mais fácil (obtido por divisão por 10^{15}), já que os valores do quadrado da distância euclidiana são valores inteiros. Além disto, a distância euclidiana não é normalizada. Assim, seus valores tendem ao infinito a medida que as imagens são maiores, o que inviabiliza comparações entre distâncias tomadas de imagens de tamanhos diferentes.

O outro teste efetuado para comparar as duas distâncias foi feito usando-se as distorções "brilho -25%" e "espalhamento". Os resultados deste outro experimento estão na Tabela 9.6.

Podemos concluir deste experimento que houve correlação com a distância de informação (na proporção de $10/12$ dos casos), no entanto, não ocorreu correlação com a

Tabela 9.4: Testes Usando Distância de Informação

Distância de Informação			
	Desfocado		Deslocado 5 pixels
1	0,998577524893314	>	0,989126871380177
2	1,00082785161319	>	0,99039492494639
3	0,987564693371145	>	0,984365698086463
4	1,00089126559715	>	0,983538166647875
5	0,99974890932488	>	0,983766844249452
6	1,00101277218252	>	0,985787485506975
7	1,00296256396445	>	0,99225302648432
8	1,00286051862446	<	1,00491999834622
9	0,999480107939494	>	0,998447702353086
10	1,00257546578807	>	0,999907435262537
11	1,00020133613984	>	0,990038607226107
12	1,00572148552072	>	0,993318898855552
13	1,00175954385496	>	0,990289099737182
14	1,00182232346241	>	1,00121022641319
15	0,999180286732355	>	0,989188556914645
16	1,00218742804513	>	0,994811277435945
17	1,00095592140202	<	1,00249978815355
18	1,0	>	0,996955903271693
19	1,00130502668313	>	1,00048853114968
20	1,00162037037037	>	0,986771011594593
21	0,999960167297351	>	0,9810061193674
22	0,991984094423581	>	0,986808550829582
23	0,97549777770542	<	0,977868839130364
24	1,00198746369057	>	0,997413760316434
25	1,002884277172	>	0,99057127010538
26	0,999834404186262	>	0,998124012638231
27	1,00621924985623	>	1,00057394298833
28	1,00545495560158	>	0,991517629024016
29	1,0045293957786	>	0,99753979739508
30	0,999590863811015	>	0,976549659544494
31	1,0008176045925	>	0,98556564937208
32	1,00419567437111	>	0,992512278750811

Tabela 9.5: Testes Usando Distância Euclidiana

Distância Euclidiana			
	Desfocado		Deslocado 5 pixels
1	60,889247306184944	$<$	422,651063948054430
2	52,434439492538969	$<$	274,689349448507780
3	457,654081092433476	$<$	1590,792227036750128
4	146,078796526846173	$<$	677,668621050946394
5	90,491557134852246	$<$	508,958855861193911
6	80,380166034464767	$<$	545,333238387641730
7	47,638703110125102	$<$	355,559827390075407
8	100,714276738025203	$<$	324,923032571297968
9	49,636009548550893	$<$	251,940803497331359
10	65,166095694895952	$<$	248,864522575054413
11	125,006217346043796	$<$	781,990969051966516
12	82,909049112792451	$<$	383,640566315482242
13	61,032714812902550	$<$	428,830050027680398
14	106,455515423004061	$<$	437,221212507833729
15	102,725878214688514	$<$	498,755274154354945
16	107,881657082465497	$<$	638,206295758015347
17	96,641574578047504	$<$	462,982141829967429
18	52,3517554862291553	$<$	218,361323393560266
19	103,629762395951943	$<$	420,247675508658150
20	67,227145047030475	$<$	444,585456446267841
21	44,600220506826830	$<$	302,475573197309854
22	221,991039359985038	$<$	600,683773456466577
23	62,350204318364430	$<$	544,538280028824058
24	30,209767143089667	$<$	201,248773899581198
25	7,329297761774975	$<$	38,469806203701764
26	51,984090942932659	$<$	249,320289977271454
27	156,147448681272435	$<$	545,045805747515670
28	99,095826659225789	$<$	457,988536316804804
29	112,395500402179591	$<$	515,962690263432175
30	930,026877270928146	$<$	2688,636043170360004
31	37,428007894941422	$<$	173,252804210066286
32	69,637825359944329	$<$	329,672542789132097

Tabela 9.6: Comparando Distância de Informação e Euclidiana

	brilho -25%		Espalha 2 pixels
Distância de Informação			
1	0,999121763272707	>	0,992283178382365
2	1,00951947353172	>	0,994888033445093
3	1,00231041441231	>	1,00048344898191
4	0,961050724637681	>	0,945525291828794
5	0,944162436548223	<	0,966292134831461
6	0,96575682382134	>	0,957264957264957
7	1,00155926809865	>	0,997179246089409
8	1,0052084589231	<	1,00589304015653
9	0,998834968857821	>	0,986876816454281
10	0,999111580666074	>	0,987112721687585
11	0,999607348198009	>	0,997829491441602
12	0,999463339370765	>	0,997267031062525
Distância Euclidiana			
1	29,209826647574905	<	49,711746112608689
2	56,773376915496188	<	247,584546558677197
3	31,228968595811195	<	65,842767885316200
4	131,565438346958104	>	16,522246461357718
5	120,549979668490000	>	58,973514012925650
6	132,922619649217182	>	43,722446318319283
7	78,083089782525070	>	13,515141193753829
8	47,894718498863623	>	43,618261178713632
9	38,305774539887879	<	77,094344912742851
10	31,069250807093691	<	35,690400043311677
11	18,775438219975196	<	18,973430565437990
12	40,240626482533506	>	9,111493328389656

distância euclidiana (onde o resultado foi 50%-50%).

Este segundo caso, envolvendo "redução de brilho" e "espalhamento", tem como característica ocorrer diferenças menores entre as distâncias, e o resultado superior da distância de informação indica que ela é mais *fina* (em um sentido formal), em casos extremos, que a distância euclidiana.

Assim, embora a distância euclidiana tenha sido um pouco melhor no experimento anterior, neste último experimento ela falhou totalmente, e a distância de informação mostrou resultados coerentes, na média, com o que esperávamos.

É importante enfatizar que os resultados desta Seção estão restritos a este conjunto de 44 imagens com que trabalhamos (12 do primeiro conjunto e 32 do segundo conjunto). No entanto, ela serve como evidência do que a propriedade de universalidade afirma, ou seja, que a distância de informação, comparada a outras medidas (computáveis), é uma boa medida em todos os casos.

A propriedade de universalidade pode ser interpretada, tomando a definição de equivalência dada em espaços métricos (LIMA, 1993), como uma indicação de que a distância de informação "cobre" a distância euclidiana, mas que o contrário não acontece.

Dadas duas métricas d_1 e d_2, num espaço M, diz-se que d_1 é *mais fina* que d_2 se existir constante c tal que $d_1(x, y) \leq cd_2(x, y)$, para quaisquer $x, y \in M$. Se cada uma é mais fina que a outra, ou seja, se existirem constantes c_1 e c_2 tais que $d_1(x, y) \leq c_1 d_2(x, y) \leq c_2 d_1(x, y)$ então as métricas são *equivalentes* (ou seja, se uma *cobre* a outra) (LIMA, 1993). Importante observar que a distância euclidiana define o tamanho de uma *bola* no espaço das strings binárias (LIMA, 1993).

Os experimentos desta Seção são uma evidência que a distância de informação é mais fina que a euclidiana, mas que o contrário não é verdadeiro.

9.4 Usando Redimensionamento e Escala de Imagens

Já sabemos que a distância euclidiana não pode ser usada para medir a distância de imagens de tamanho diferente ou de razão de aspecto diferente, pois exige o casamento um para um de todos os termos da série apresentada na Equação 8.2. Isto a inviabiliza para ser aplicada em transformações do tipo redimensionar, escalar, rotação, etc.

No entanto, faz sentido medir a distância de informação de strings de tamanhos diferentes, já que não há nenhuma restrição com relação aos tamanhos das strings x e y no condicional $K(x|y)$. Neste caso, a pergunta fundamental é se as correlações entre distorções e distâncias calculadas se manterão.

Para isto, tomamos as dez primeiras imagens do segundo conjunto de imagens (formado por 32 imagens) e escalamos $\times 2$ (ou seja, obtivemos imagens de 500x400). Para cada uma destas dez imagens tomamos outras dez arbitrárias (as dez seguintes) e escalamos também. Estas serão usadas como *contraste*. Assim, a primeira será contrastada com a décima-primeira, a segunda com a décima-segunda e assim por diante. Então, medimos as distâncias entre a original e ela própria escalada, e a original e o contraste escalado (veja Figura 9.6), e comparamos as distâncias calculadas. Os resultados são apresentados na Tabela 9.7.

A Tabela demonstra que a distância de informação considerou mais semelhantes as imagens em relação a elas próprias escaladas, que em relação ao contraste tomado arbitrariamente (em $8/10$ casos). Isto confirma que a distância de informação pode ser usada nos casos em que os tamanhos das imagens são diferentes.

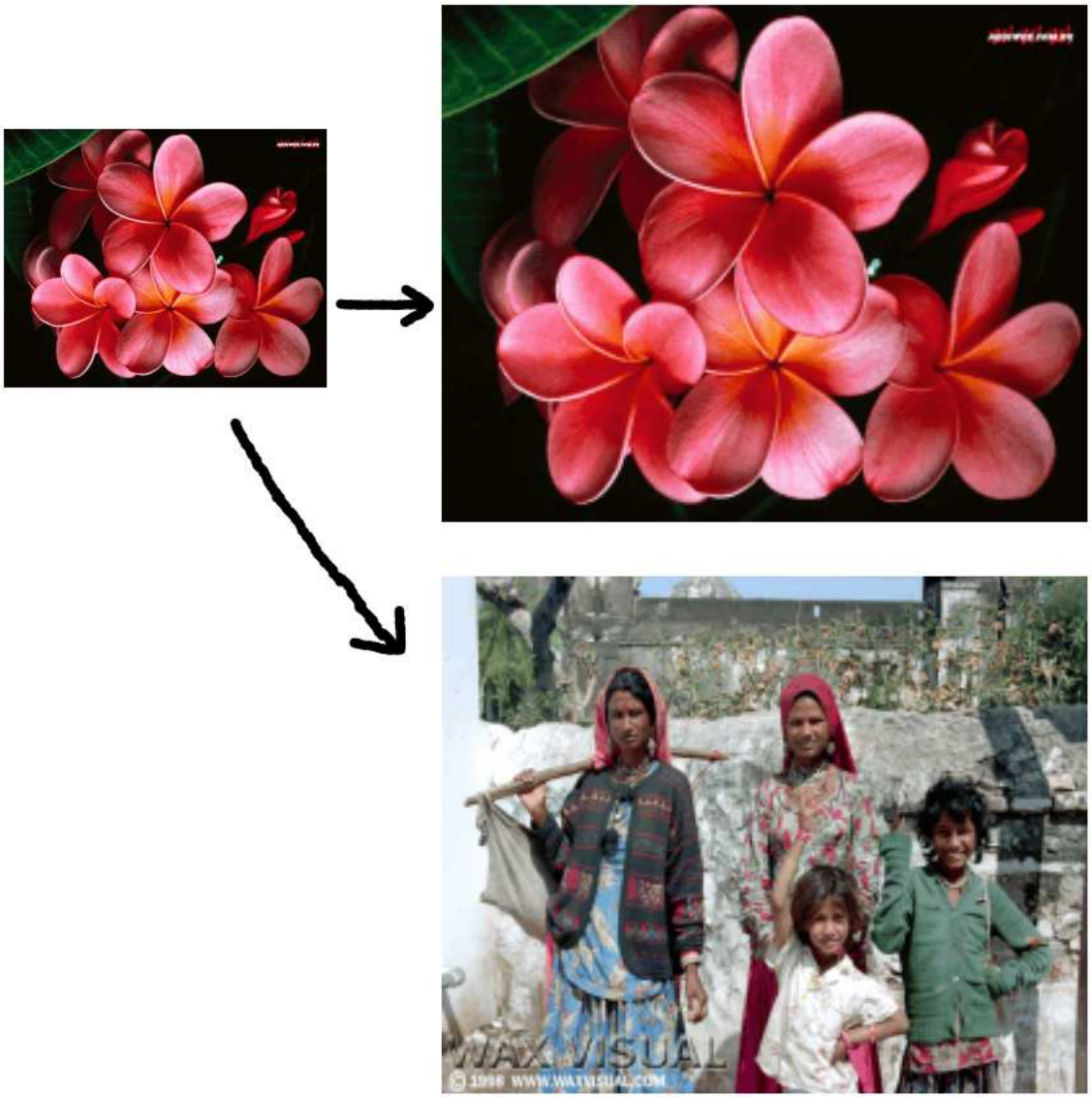

Figura 9.6: Experimento com Uso de Escala

Tabela 9.7: Resultados de Experimento com Escala

	Mesma figura escalada		Outra figura escalada
1	1,00115400069147	$<$	1,00143672855066
2	1,00088128848293	$<$	1,00297795900525
3	0,995824408558157	$<$	1,00464532564107
4	0,999931429858868	$<$	1,00056736582685
5	1,00036533266397	$<$	1,00075204954161
6	1,00070406855341	$<$	1,00266466630093
7	1,00296411687708	$>$	1,00294468415834
8	1,00265047186185	$<$	1,00600243865424
9	1,0015353716762	$<$	1,00277278011002
10	1,00292431819295	$>$	1,00261186313964

9.5 Avaliação da Qualidade de Imagem de Animações

Será que poderíamos estender estas idéias para avaliar também animações baseadas em frames? Neste caso, se confirmariam as mesmas correlações?

Avaliação da qualidade de animações é um assunto que pouco aparece na literatura. Em nossa pesquisa encontramos apenas um artigo (O'SULLIVAN et al., 2003), onde os autores consideram-se os proponentes de avaliações de animações. No entanto, as avaliações que os autores descrevem no artigo, não se preocupam com a qualidade das imagens, mas sim em avaliar como as pessoas percebem, do ponto de vista psicológico, variações no movimento de objetos em uma animação. Assim, nosso objetivo é um pouco diferente, pois desejamos mais medir a qualidade da imagem dos frames, que o movimento propriamente dito.

Na verdade, nos parece que avaliações do tipo das feitas pelos autores do artigo, centradas no movimento mais que na imagem, poderiam também ser feitas usando-se distância de informação, mas seria um enfoque diferente do que apresentamos aqui neste trabalho.

Nossa abordagem toma as idéias já desenvolvidas nas seções precedentes e adapta a idéia às animações, pela decomposição dos frames que compõem a animação.

Os experimentos foram feitos usando-se quinze animações GIF, doze obtidas arbitrariamente na Internet e três feitas para testar casos limites (animações simples, de fundo branco, com um objeto em movimento). Sobre cada animação, aplicamos as distorções "desfocar" e "deslocar 5 pixels", frame a frame, e medimos as distâncias entre a animação original e a distorcida.

Assim, cada animação foi decomposta em seus frames originais, salvos em BMP (não comprimido), não indexado (RGB). Então, foram aplicadas as distorções e os arquivos resultantes tiveram seu cabeçalho retirado (usando-se o programa strip.py). Finalmente, foram concatenados os frames obtidos usando-se o comando *cat* do UNIX.

O resultado deste experimento é apresentado na Tabela 9.8.

A conclusão é que houve correlação entre as distorções aplicadas e as distâncias calculadas em todos os quinze casos, e a diferença média entre as duas colunas foi maior que no caso das mesmas distorções, quando aplicadas a imagens (0,091876621 contra 0,028113561).

9.6 Refinamento das Avaliações com Imagens

Um das dificuldades envolvidos na aplicação da distância de informação em imagens é que uma imagem é um objeto bidimensional, que é linearizado pela concatenação das linhas da imagem. Isto faz com que o *princípio da localidade* fique prejudicado, ou seja, um objeto formado por um conjunto de pontos localizados em uma área pequena da imagem, que são contíguos na imagem, acabam distantes na representação linearizada da imagem. Isto prejudica o compressor, resultando em medidas de distância menos acuradas.

Uma forma de superar este problema é linearizar os pixels da imagem, antes do processo de compressão, usando-se uma regra que aproxime os pontos de uma área de interesse (princípio da localidade) no início da string binária que será aplicada ao compressor. Isto permitiria maior acuidade na medida de distância, no que se refere aquele objeto.

A abordagem que foi seguida nos experimentos seguintes foi recortar a imagem, na forma de um quadrado de 200x200, a partir do centro da imagem (normalmente a zona

Tabela 9.8: Resultados com Animações

	Desfocado		Deslocado 5 pixels
1	1,02270483711747	>	0,974719101123595
2	1,02325029655991	>	0,954985754985755
3	1,00558168063791	>	0,993989243910155
4	1,02572763015542	>	0,917188198480107
5	1,01393156606427	>	0,983268356075374
6	1,01128300198172	>	0,985203973789896
7	1,01479975657443	>	0,990379359352793
8	1,01820241293926	>	0,8996138996139
9	1,02561631735325	>	0,882938703252932
10	1,01366496002556	>	0,969231961559327
11	1,01290541131218	>	0,980030795881051
12	1,01360544217687	>	0,834285714285714
13	1,01445398405745	>	0,981175566653861
14	1,01538180592235	>	0,977839335180055
15	1,02990978178748	>	0,691217413313085

de maior interesse visual), tomando os pixels em espiral, como mostra a Figura 9.7. Este refinamento permite que a string resultante da linearização contenha no seu prefixo todo o objeto (ou objetos) que aparecem no centro da imagem, com todos os pixels próximos, de forma a facilitar a compressão.

O programa usado para o refinamento é apresentado na Figura 9.8. O programa recebe como argumentos o nome do arquivo a ser refinado (resultante da aplicação do programa strip.py), o ponto central do recorte (no nosso caso, 125 e 100) e o tamanho do quadrado a ser recortado (neste caso, 200).

Usando-se este refinamento foram refeitos os testes anteriores, cujos resultados são apresentados nas Tabelas 9.9, 9.10 e 9.11.

Observa-se, da análise dos resultados, que todos os resultados negativos dos experimentos anteriores, relacionados com as distorções “desfocar”, “deslocar 5 pixels”, “redução de brilho” e “espalhamento”, foram revertidos, a exceção da linha 23 da Tabela 9.11. Uma das razões para que a imagem número 23, deste segundo conjunto de imagens, tenha dado resultado falso neste caso, é que observa-se que ela é de muito baixa qualidade, esmaecida e desfocada (veja na Figura 9.3), e a deformação que está sendo usada neste caso é exatamente “desfocamento”.

Para testar esta hipótese de que o problema ocorreu porque a distorção aplicada foi “desfocamento”, aplicamos nesta imagem as distorções “brilho -25%” e “espalhamento”. A aplicação simples da distância de informação deu um resultado inconsistente com o resultado esperado (distância para “brilho” foi 0,980609869267213 e para “espalhamento” 0,980968041287466), mas ao aplicar o refinamento, o resultado foi corrigido (para “brilho”, 0,995343145989997, e para “espalhamento”, 0,994452166268696).

É importante observar que o refinamento proposto pode ser modificado para considerar também as laterais, seja pela redefinição da maneira como se faz a espiral, formando um retângulo que cubra toda a imagem, seja pela concatenação das laterais ao final da string obtida no recorte. Estas duas possibilidades não foram testadas neste trabalho.

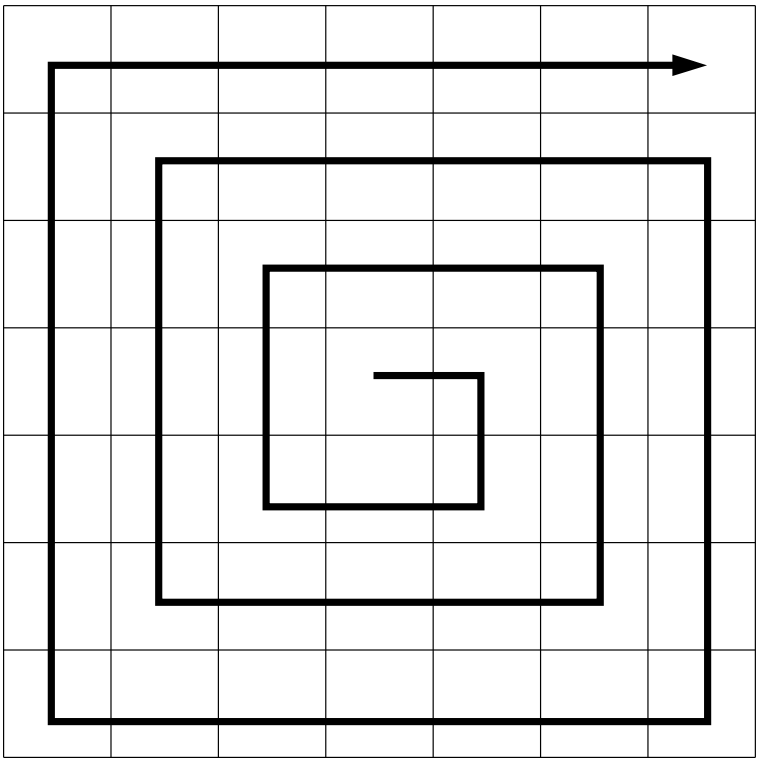

Figura 9.7: Refinamento da Medida de Distância

Tabela 9.9: Primeiro Teste com Refinamento – Primeiro Conjunto

	Desfocado		Deslocado 5 pixels
1	1,00065463769895	>	0,998000484730974
2	1,00778036295952	>	0,995458352722196
3	1,00668555742495	>	0,997443014183622
4	1,02240040728013	>	0,940422805893658
5	1,02329144958627	>	0,9420454545455
6	1,04882459312839	>	0,967085171360706
7	1,00038136885996	>	1,00030120481928
8	1,00350503644255	>	1,00153959544673
9	1,00395237307756	>	0,995005787994047
10	1,0024795649646	>	0,995218323520403
11	1,00021577089055	>	0,993744868837718
12	1,00193833048535	>	0,996168582375479

```
#!/usr/bin/python
from sys import *
from string import *
# Refinamento em espiral
# autor: Carlos Campani
def posi(x,y):
   return 3*(250*(y-1)+(x-1))+400
f = open(argv[1],"r")
v = f.read()
f.close()
dx=dy=1
sx=sy=1
x = atoi(argv[2]) # posicao inicial
y = atoi(argv[3])
nv = v[posi(x,y)]
while dx < atoi(argv[4]): # tamanho quadrado
  for c in range(dx):
    x = x+sx
    nv = nv+v[posi(x,y):posi(x,y)+3]
  for c in range(dy):
    y = y+sy
    nv = nv+v[posi(x,y):posi(x,y)+3]
  dx+=1
  dy+=1
  sx=-sx
  sy=-sy
f = open(argv[1]+".refin","w")
f.write(nv)
f.flush()
f.close()
```

Figura 9.8: Programa refina

Tabela 9.10: Segundo Teste com Refinamento – Primeiro Conjunto

	Brilho -25%		Espalha 2 pixels
1	1,00194345566875	>	0,996095033733496
2	1,00807742415783	>	0,99557003257329
3	1,0071947223944	>	0,997281670693821
4	0,998684210526316	>	0,962432915921288
5	0,952830188679245	>	0,951774340309372
6	1,02516838000709	>	0,966678482807515
7	1,01507678226381	>	0,996869928124275
8	1,00863156170666	>	1,00188670194906
9	1,00464693236316	>	0,995506616630456
10	1,00376952073236	>	0,995191806977955
11	1,00129462534327	>	0,993072986593505
12	1,00882450457803	>	0,995896344707045

9.7 Conclusões deste Capítulo

Fica comprovada a hipótese inicial, ou seja, a distância de informação foi capaz de caracterizar as distorções aplicadas nos testes feitos, pois ocorreu correlação entre as distorções aplicadas em imagens e as distâncias calculadas.

A propriedade universal da distância de informação indica que ela é uma boa medida em todos os casos, o que a diferencia das outras medidas que não possuem esta propriedade. Além disto, a distância de informação mostrou-se interessante para casos em que a distância euclidiana não se aplica (redimensionar, escalar, rotação, etc.). Foi possível também aplicar a métrica em animações baseadas em frames, o que representa uma proposta inédita, segundo pesquisa bibliográfica feita, e um avanço na avaliação de animações.

Os resultados obtidos neste trabalho com imagens não foram tão bons quanto os obtidos em outros contextos (literatura, genoma e música), e isto deve ser conseqüência do fato que textos literários, DNA e partituras musicais tem uma característica linear e imagem é um objeto bidimensional, assim como do fato que imagens são objetos mais complexos que literatura, DNA e música, e a distância de informação trabalha melhor com objetos mais regulares. Este último aspecto explica porque os resultados com animações foram melhores que com imagens, já que animações são objetos com mais regularidade.

O refinamento proposto neste trabalho foi capaz de reverter a maior parte dos resultados negativos dos testes, e representa um resultado importante da pesquisa desenvolvida. É interessante observar que, embora tenhamos feito o recorte da imagem em forma quadrada a partir do centro da imagem, este refinamento poderia ser adaptado para efetuar o recorte em um formato de retângulo, para cobrir toda a imagem. Além disto, o ponto central da espiral poderia ser mudado (é um parâmetro de linha de comando do programa apresentado na Figura 9.8), o que permitiria valorizar mais outras áreas da imagem que não o seu centro.

Muitos dos valores que desviaram da média esperada nas distâncias podem ser efeito da aproximação computável. Em alguns casos a aplicação de um limiar pode eliminar alguns resultados negativos, pois os erros foram muito pequenos.

Ainda resta a possibilidade de melhorar ainda mais os resultados pelo uso de melhores

Tabela 9.11: Testes com Refinamento – Segundo Conjunto

	Desfocado		Deslocado 5 pixels
1	1,00169243349117	>	0,994055983896084
2	1,00287876991855	>	0,993869913467563
3	0,995737577558241	>	0,994834699898164
4	1,00140180809485	>	0,995442893762461
5	1,00154587504982	>	0,994603839946038
6	1,00133380515557	>	0,993462459901734
7	1,00127860887355	>	0,994882471670825
8	1,00345646679051	>	0,999741990317048
9	1,00206821668017	>	0,992597310930156
10	1,00496876315023	>	0,99525196761681
11	1,00180969999196	>	0,993696749552629
12	1,00191746069206	>	0,994384747681136
13	1,00333825511973	>	0,99815693313674
14	0,999666287150092	>	0,994323152594436
15	1,00302251385319	>	0,995610014332012
16	1,00415828760353	>	0,996165905527912
17	1,00357570339843	>	0,995121951219512
18	1,00296697626419	>	0,998427271954433
19	1,00268334814765	>	1,00058340139683
20	0,999644972094807	>	0,993241619183622
21	1,00077485524295	>	0,990166382553078
22	0,994909166986297	>	0,99320240815711
23	0,991648550079	<	0,993620585195481
24	1,00449148022345	>	0,997957642055787
25	0,996811568169571	>	0,989605813697424
26	1,00725239688349	>	1,00174976189949
27	1,00484799691968	>	0,995228974403622
28	1,00192786946913	>	0,995771472988729
29	1,00193436153087	>	0,995448120242248
30	1,01443642420877	>	0,985204182284474
31	1,00049697116224	>	0,99333789091299
32	0,999776489488021	>	0,997401690298247

algoritmos de compressão (mais próximos da compressão “ideal”). Isto pode ser resultado da pesquisa com algoritmos de compressão.

Uma outra possibilidade de refinamento do método seria usar um esquema de compressão adaptado à compressão de imagens, como compressão JPEG com perdas, já que então seria possível obter mais compressão, e mais acuidade no cálculo da distância. Neste sentido, seria necessário redefinir os conceitos de complexidade e distância, que foram definidos sobre um esquema de compressão sem perdas. Esta é uma possibilidade indicada como trabalho futuro.

10 CONCLUSÃO

Este trabalho versa sobre a avaliação da compressão de dados e da qualidade de imagens e animações usando-se complexidade de Kolmogorov, simulação de máquinas e distância de informação. Nele, é proposto um método para avaliar a compressão de dados de modelos de animação gráfica usando-se simulação de máquinas. Também definimos formalmente compressão de dados com perdas e propomos a aplicação da distância de informação como uma métrica de qualidade de imagem, aplicada em imagens e animações.

O desenvolvimento de uma metodologia para avaliar a compressão de dados de modelos de animação gráfica para web é útil, a medida que as páginas na web estão sendo cada vez mais enriquecidas com animações, som e vídeo, e a economia de banda de canal torna-se importante, pois os arquivos envolvidos são geralmente grandes. Boa parte do apelo e das vantagens da web em aplicações como, por exemplo, educação à distância ou publicidade, reside exatamente na existência de elementos multimídia, que apoiam a idéia que está sendo apresentada na página.

O método proposto neste trabalho permite estabelecer uma hierarquia de modelos de animação, baseada na taxa de compressão de dados obtida pelo modelo.

O método de comparação e avaliação de compressão de dados, em modelos de animação gráfica, foi aplicado, como estudo de caso, em dois modelos: AGA e GIF. O primeiro, AGA (ACCORSI; MENEZES, 2000), desenvolvido pelo grupo de pesquisa ao qual se vincula este trabalho, e daí sua escolha, e o segundo, GIF, escolhido como termo de comparação por sua popularidade, advinda principalmente de sua simplicidade, e por ser um modelo de animação baseado em mapas de bits, o que o adaptava bem ao método proposto.

Esta metodologia baseia-se em uma definição formal de compressão de dados sem perdas, desenvolvida para viabiliza-la, inspirada em complexidade de Kolmogorov e no artigo (SUBBARAMU; GATES; KREINOVICH, 1998).

Apresentamos também uma definição de compressão de dados com perdas, com o objetivo de estender o método desenvolvido, e que baseia-se na idéia de medir a qualidade da compressão usando uma nova medida de qualidade de imagem. Esta medida é a distância de informação (BENNETT et al., 1993; LI et al., 2003), cuja aplicação neste contexto é, segundo o que se sabe, inédita na literatura.

A distância de informação goza de uma propriedade muito importante, chamada *universalidade*, que a faz ser, no mínimo, tão boa quanto qualquer outra medida (computável).

Foi feita uma série de experimentos de medição da qualidade de imagens, submetidas a algumas distorções, escolhidas arbitrariamente, com o propósito de validar a proposta de aplicação da distância de informação neste novo contexto. Também foi feita a comparação dos resultados obtidos com distância de informação, aos obtidos usando-se a distância

euclidiana, que é a medida tradicionalmente usada nestes casos. Os resultados obtidos indicam que a medida proposta é uma boa medida de qualidade de imagem.

Além disto, pode-se deduzir dos resultados que a distância de informação é uma métrica mais *fina* (no sentido do termo usado na área de espaços métricos, veja (LIMA, 1993)) que a distância euclidiana, mas o contrário não é verdadeiro. Isto mostrou-se particularmente verdadeiro quando foram consideradas, nos testes, deformações em que as diferenças eram pequenas. Neste caso, a distância euclidiana não mostrou bons resultados, ao contrário da distância de informação, que continuou mostrando correlações coerentes.

A mesma idéia foi aplicada na avaliação da qualidade de imagem de animações baseadas em frames. Os resultados obtidos foram melhores que os obtidos com imagens puras, pois animações são objetos mais "regulares", o que facilita o compressor, tornando a medida de distância mais acurada. Neste aspecto, este trabalho introduz um avanço, pois, segundo pesquisa bibliográfica, o único tipo de avaliação de animações até então desenvolvida, tratava apenas dos aspectos psicológicos da percepção de deformações no movimento (O'SULLIVAN et al., 2003).

Finalmente, foi proposto um refinamento ao método de cálculo da distância de imagens, que resultou em melhoria dos resultados obtidos. O refinamento baseia-se em um recorte em espiral da imagem, podendo este recorte ser feito com centro em qualquer posição da imagem, permitindo valorizar diferentes partes da imagem.

Como conclusão geral do trabalho podemos dizer que a complexidade de Kolmogorov mostrou-se efetiva nas aplicações propostas. Particularmente, a aplicação da distância de informação neste novo contexto de imagens apresenta as maiores possibilidades de outros desenvolvimentos.

Já está em andamento pesquisa para aplicar a distância de informação na classificação dos tipos de impressões digitais, segundo padrão aceito na dactiloscopia. Tecnologia que pode representar um avanço na velocidade do reconhecimento de impressões digitais.

De um modo geral, poderiamos aplicar a distância de informação em contextos mais próximos de reconhecimento de padrões, como por exemplo, na detecção de plágio em programas de computador em trabalhos de alunos de cursos de programação, ou para auditoria relacionada com direitos autorais de programas de computador.

As investigações, relacionadas com este trabalho, que pretendemos desenvolver como trabalhos futuros são:

- Estender a metodologia de comparação de compressão de dados de modelos de animações apresentada neste trabalho para tratar com animações vetoriais;

- Modificar o programa apresentado na Figura 9.8 para tomar o corte em forma retangular, de forma a considerar toda a imagem no refinamento. Fica a dúvida se este tipo de corte teria um efeito tão bom quanto o corte em quadrado;

- Uso da compressão JPEG com perdas para aproximar K no cálculo da distância de informação, aplicado à avaliação de qualidade de imagens. Este seria um refinamento da medida de distância apresentada neste trabalho. Isto representa tomar uma definição de complexidade de Kolmogorov um pouco diferente, já que originalmente ela está definida em um esquema de compressão sem perdas. A vantagem é que, aplicando um algoritmo de compressão desenvolvido especialmente para imagens, e não um algoritmo de compressão genérico, a compressão deverá ser maior, as medidas de distância podem ser mais acuradas, e os resultados melhorados;

- Uso da distância de informação para discernir objetos em imagens e para medir a diferença de movimentos de objetos em animações. Esta aplicação está mais próxima de reconhecimento de padrões, e depende, para seu sucesso, do refinamento da medida de distância. Neste caso, seria possível determinar a presença ou ausência de um determinado objeto em uma imagem, e/ou a detecção de diferenças no movimento de um objeto numa animação, tais como o ângulo da trajetória ou a velocidade do movimento.

Ainda haveria um trabalho correlacionado com este desenvolvido aqui que seria a aplicação da complexidade de Kolmogorov no contexto de teoria de jogos. A idéia seria perceber um jogo como uma máquina ou um programa, e foi inspirada no artigo (VANY, 1998). Neste artigo, o autor aplica idéias semelhantes no contexto de economia. Nossa contribuição seria generalizar estas idéias em um contexto mais amplo de teoria de jogos. Assim como em (VANY, 1998), o autor conseguiu um significado para uma "instituição universal", como sendo uma instituição pública capaz de simular o contrato de qualquer outra instituição, seria interessante definir a idéia de jogo universal. Que conseqüências estas idéias teriam? Na minha opinião, seria uma opção de pesquisa bem interessante e com muitas aplicações práticas (por exemplo, mercado financeiro).

REFERÊNCIAS

ACCORSI, F.; MENEZES, P. B. Animação Gráfica Baseada na Teoria de Autômatos. In: WMF'2000 - 3RD WORKSHOP ON FORMAL METHODS, 2000, João Pessoa, Brazil. **Proceedings...** [S.l.: s.n.], 2000. p.122–127.

AHUMADA, A. J. Computational Image-Quality Metrics: a review. In: SOCIETY FOR INFORMATION DISPLAY INTERNATIONAL SYMPOSIUM, DIGEST OF TECHNICAL PAPERS, 1993. **Proceedings...** [S.l.: s.n.], 1993. v.24, p.305–308.

AVCIBAS, I.; MEMON, N.; SANKUR, B. Steganalysis Using Image Quality Metrics. **IEEE Transactions on Image Processing**, [S.l.], v.12, n.2, p.221–229, Feb. 2003.

BENNETT, C. H.; GÁCS, P.; LI, M.; VITÁNYI, P.; ZUREK, W. H. Information Distance. In: ACM SYMP. THEORY OF COMPUT., 25., 1993. **Proceedings...** [S.l.: s.n.], 1993. p.21–30.

BUHRMAN, H. et al. Kolmogorov Random Graphs and the Incompressibility Method. **SIAM J. Comput.**, [S.l.], v.29, n.2, p.590–599, 1999.

CAMPANI, C. A. P.; ACCORSI, F.; MENEZES, P. B.; NEDEL, L. P. **An Efficient and Flexible Animation Model Based on Automata Theory**: evaluating data compression. Submetido para publicação em 2004.

CAMPANI, C. A. P.; MENEZES, P. B. Teorias da Aleatoriedade. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v.11, n.2, p.75–98, dez. 2004.

CAMPANI, C. A. P.; MENEZES, P. B. Characterizing the Software Development Process: a new approach based on Kolmogorov complexity. In: INTERNATIONAL WORKSHOP ON COMPUTER AIDED SYSTEMS THEORY, EUROCAST, 8., 2001. **Computer Aided Systems Theory**. Berlin: Springer, 2001. p.242–256. (Lecture Notes in Computer Science, v.2178).

CAMPANI, C. A. P.; MENEZES, P. B. Characterizing the Software Development Process: a new approach based on Kolmogorov complexity. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED SYSTEMS THEORY AND TECHNOLOGY, 8., 2001, Las Palmas de Gran Canaria. **Formal Methods and Tools for Computer Science**: extended abstracts. Las Palmas de Gran Canaria: IUCTC, 2001. p.78–81.

CAMPANI, C. A. P.; MENEZES, P. B. Complexidade de Kolmogorov e Caracterização da Hierarquia de Classes de Linguagens Formais - Uma Introdução. In: WORKSHOP ON FORMAL METHODS, 5., 2002, Brazil. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 2002. p.68–83.

CAMPANI, C. A. P.; MENEZES, P. B. Aplicação da Complexidade de Kolmogorov na Caracterização e Avaliação de Modelos Computacionais e Sistemas Complexos. In: WORKSHOP ON FORMAL METHODS, 5., 2002, Brazil. **Proceedings...** Porto Alegre: Instituto de Informática da UFRGS, 2002. p.100–112.

CAMPANI, C. A. P.; MENEZES, P. B. Evaluating Computer Animation Models with Lossy Data Compression Using Kolmogorov Complexity. In: INTERNATIONAL CONFERENCE IMAGING SCIENCE, SYSTEMS, AND TECHNOLOGY, CISST, 2003. **Proceedings...** [S.l.]: CSREA Press, 2003. p.721–725.

CAMPANI, C. A. P.; MENEZES, P. B. On the Application of Kolmogorov Complexity to the Characterization and Evaluation of Computational Models and Complex Systems. In: INTERNATIONAL CONFERENCE ON IMAGING SCIENCE, SYSTEMS AND TECHNOLOGY, CISST, 2004, Las Vegas. **Proceedings...** [S.l.]: CSREA Press, 2004. v.1, p.63–68.

CHAITIN, G. J. Information-Theoretic Computational Complexity. **IEEE Transactions on Information Theory**, [S.l.], v.20, p.10–15, 1974.

CHAITIN, G. J. A Theory of Program Size Formally Identical to Information Theory. **Journal of the ACM**, [S.l.], v.22, p.329–340, 1975.

CILIBRASI, R.; VITÁNYI, P.; WOLF, R. de. **Algorithmic Clustering of Music**. Disponível em: <http://homepages.cwi.nl/~paulv/papers/music.ps>. Acesso em: 17 dez. 2004.

CLARK, K.; COWELL, D. **Programs, machines, and computation**: an introduction to the theory of computing. London: McGraw-Hill, 1976.

COVER, T. M.; THOMAS, J. A. **Elements of Information Theory**. New York: Wiley, 1991.

DAVIS, M. A Note On Universal Turing Machines. In: SHANNON, C.; MCCARTHY, J. (Ed.). **Automata Studies**. [S.l.]: Princeton University Press, 1956. p.167–175.

DIVERIO, T.; MENEZES, P. B. **Teoria da Computação**: máquinas universais e computabilidade. 2.ed. Porto Alegre: Sagra-Luzzatto, 2000. 224 p.

FERNANDEZ, P. **Medida e Integração**. Rio de Janeiro: Instituto de Matemática Pura e Aplicada (Impa), 1996. 202 p.

FIGUEIREDO, D. G. **Análise I**. Rio de Janeiro: LTC, 1996. 172 p.

GÁCS, P. On the symmetry of algorithmic information. **Soviet Math. Dokl.**, [S.l.], v.15, p.1477–1480, 1974.

GÁCS, P. **Lecture Notes on Descriptional Complexity and Randomness**. Boston: Boston University, Computer Science Dept., 1993.

GAMMERMAN, A.; VOVK, V. Kolmogorov Complexity: sources, theory and applications. **The Computer Journal**, [S.l.], v.42, n.4, p.252–255, 1999.

GRAPHICS Interchange Format Programming Reference. Version 89a. [S.l.]: CompuServe Inc., 1990.

GRIMMETT, G. R.; STIRZAKER, D. R. **Probability and Random Processes**. 2nd ed. Oxford: Claredon Press - Oxford University Press, 1992. 541 p.

HOPCROFT, J. E.; ULLMAN, J. D. **Introduction to Automata Theory, Languages and Computation**. Reading: Addison-Wesley, 1979.

JAMES, B. **Probabilidades**: um curso em nível intermediário. Rio de Janeiro: Instituto de Matemática Pura e Aplicada (Impa), 1996. 299 p.

JIANG, T.; LI, M.; VITÁNYI, P. A lower bound on the average-case complexity of Shellsort. **J. Assoc. Comp. Mach.**, [S.l.], v.47, n.5, p.905–911, 2000.

KHINCHIN, A. **Mathematical Foundations of Information Theory**. New York: Dover, 1957. 126 p.

KLEENE, S. **Mathematical Logic**. New York: Wiley, 1967.

KOLMOGOROV, A. N. **Grundbegriffe der Wahrscheinlichkeitsrechnung**. [S.l.]: Springer-Verlag, 1933. Tradução para o inglês de N. Morrison: Foundations of the Theory of Probability, Chealsea, 1956.

KOLMOGOROV, A. N. On tables of random numbers. **Sankhya, Series A**, [S.l.], v.25, p.369–376, 1963.

KOLMOGOROV, A. N. Three Approaches to the Quantitative Definition of Information. **Problems of Information Transmission**, [S.l.], v.1, p.4–7, 1965.

KOLMOGOROV, A. N. Logical Basis for Information Theory and Probability Theory. **IEEE Transactions on Information Theory**, [S.l.], v.14, p.662–664, 1968.

KOSHELEVA, O.; KREINOVICH, V.; NGUYEN, H. T. **On the Optimal Choice of Quality Metric in Image Compression**. [S.l.]: University of Texas at El Paso, Computer Science Department. (UTEP-CS-00-25c). Disponível em: <http://www.cs.utep.edu/vladik/2000/tr00-25c.ps.gz>. Acesso em: 15 maio 2004.

KREINOVICH, V.; LONGPRE, L. **Human visual perception and Kolmogorov complexity**: revisited. [S.l.]: University of Texas at El Paso, Computer Science Department, 1998.

LELEWER, D.; HIRSCHBERG, D. Data Compression. **ACM Computing Surveys**, [S.l.], v.19, n.3, p.261–296, Sept. 1987.

LI, M.; CHEN, X.; LI, X.; MA, B.; VITÁNYI, P. The similarity metric. In: ACM-SIAM SYMP. DISCRETE ALGORITHMS, SODA, 14., 2003. **Proceedings...** [S.l.: s.n.], 2003.

LI, M.; VITÁNYI, P. A New Approach to Formal Language Theory by Kolmogorov Complexity. **SIAM J. Comput.**, [S.l.], v.24, n.2, p.398–410, 1995.

LI, M.; VITÁNYI, P. **An Introduction to Kolmogorov Complexity and its Applications**. New York: Springer, 1997.

LIMA, E. L. **Espaços Métricos**. Rio de Janeiro: Instituto de Matemática Pura e Aplicada (Impa), 1993. 299 p.

LIPSCHUTZ, S. **Theory and Problems of Probability**. New York: McGraw-Hill, 1968. 153 p.

MARTIN-LÖF, P. The Definition of Random Sequences. **Information and Control**, [S.l.], v.9, p.602–619, 1966.

MARTIN-LÖF, P. On the concept of a random sequence. **Theory Probability Applyed**, [S.l.], v.11, p.177–179, 1966.

NAGEL, E.; NEWMAN, J. R. **Prova de Gödel**. São Paulo: Ed. Perspectiva, 1973. (Debates).

O'SULLIVAN, C.; DINGLIANA, J.; GIANG, T.; KAISER, M. K. Evaluating the Visual Fidelity of Physically Based Animations. **ACM Transactions on Graphics**, New York, v.22, n.3, p.527–536, 2003. Trabalho apresentado na ACM SIGGRAPH, 2003.

PAGELS, H. R. **Os Sonhos da Razão**: o computador e a emergência das ciências da complexidade. Lisboa: Gradiva, 1988.

PARENT, R. **Computer Animation**: algorithms and techniques. [S.l.]: Morgan Kaufmann Publishers, 2001.

ROHALY, A. M.; AHUMADA, A. J.; WATSON, A. B. A Comparison of Image Quality Models and Metrics Predicting Object Detection. In: SID INTERNATIONAL SYMPOSIUM DIGEST OF TECHNICAL PAPERS, 1995, Santa Ana. **Proceedings...** [S.l.]: Society for Information Display, 1995. p.45–48.

SHANNON, C. A Mathematical Theory of Communication. **Bell Sys. Tech. Journal**, [S.l.], v.27, p.379–423, 623–656, 1948.

SOLOMONOFF, R. The Discovery of Algorithmic Probability. **Journal of Computer and System Sciences**, [S.l.], v.55, n.1, p.73–88, Aug. 1997.

SUBBARAMU, S.; GATES, A.; KREINOVICH, V. Application of Kolmogorov Complexity to Image Compression: it is possible to have a better compression, but it is not possible to have the best one. **Bulletin of the European Association for Theoretical Computer Science**, [S.l.], v.69, p.150–154, Oct. 1998.

USPENSKY, V.; SEMENOV, A.; SHEN, A. Can an Individual Sequence of Zeros and Ones Be Random? **Russian Math. Surveys**, [S.l.], v.45, n.1, p.121–189, 1990.

VANY, A. D. How much information is there in an economic organization and why can't large ones be optimal? **Brazilian Electronic Journal of Economics**, [S.l.], v.1, n.1, 1998.

VITÁNYI, P. **Randomness**. In: 'Matematica, Logica, Informatica', Volume 12 do 'Storia del XX Secolo', a ser publicado pelo 'Instituto della Enciclopedia Italiana'. Disponível em: <http://homepages.cwi.nl/~paulv/papers/ency.ps>. Acesso em: 17 dez. 2004.

VOLCHAN, S. B. What Is a Random Sequence. **The American Mathematical Monthly**, [S.l.], v.109, p.46–63, Jan. 2002.

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**, [S.l.], v.25, n.6, p.83–124, 1970.