**Tiago Vignatti**

# *Sistemas de Prova Interativa com Conhecimento Zero*

Trabalho de graduação

Orientador:
Jair Donadelli Jr.

UNIVERSIDADE FEDERAL DO PARANÁ

Departamento de Informática

setembro de 2006

# *Resumo*

A Teoria da Complexidade é um tema central em Ciência da Computação pois ela estuda os recursos necessários e suficientes para resolver problemas computacionais. Problemas computacionais são mensurados de acordo com a sua dificuldade de resolvê-los e são geralmente classificados em classes. Neste trabalho estudamos a classe de complexidade computacional IP formada pelos problemas que admitem provas interativas. Mais especificamente, estudamos uma subclasse da IP, que é a classe ZK dos problemas que admitem provas interativas com conhecimento zero. Numa prova com conhecimento zero somente a validade da prova é transmitida com o passar do processo interativo através da conversa de duas entidades (máquinas de Turing): o provador e o verificador. A principal característica desse tipo de prova é o fato do verificador não aprender absolutamente nada, ou seja, com conhecimento zero, ele estará convencido da validade da afirmação no final do processo da prova. É necessário definir um modelo que capture toda essa idéia de provas com conhecimento zero. Neste trabalho estudamos o modelo inicialmente proposto em 1985 por Goldwasser, Micali e Rackoff, chamado de sistema de provas interativo e mostramos o concebimento de tal sistema. Provas com conhecimento zero – ou protocolos com conhecimento zero – não ficam somente no plano teórico da ciência da computação, suas aplicações têm grande importância em criptografia e transações eletrônicas.

# *Sumário*

# *Lista de Figuras*

# 1 Introdução

## 1.1 A Carvena do Ali Babá

Como motivação inicial ao estudo, introduziremos o exemplo da *Caverna do Ali Babá* [18]. Peggy quer provar para Vic que ela sabe as palavras secretas que irá abrir a porta de dentro da Caverna de Ali Babá, mas ela não quer revelar o segredo para Vic. Para isso, Peggy usa seus conhecimentos matemáticos de provas com conhecimento zero para demonstrar esse fato. Vejamos.

1. Num primeiro momento, Vic espera fora da caverna enquanto Peggy vai pelo túnel A ou B de dentro da caverna (esquerda da Figura 1.1).

2. Vic, que não sabe por qual lado Peggy foi, entra na caverna e diz para Peggy aparecer por um dos túneis, por exemplo o túnel A (centro da Figura 1.1).

3. Se Peggy não conhece as palavras secretas, então existe somente 50% de chances de ela vir pelo caminho certo do túnel. Caso contrário, se Peggy conhece as palavras secretas, então ela sempre poderá vir pelo túnel que Vic escolheu.

Dessa forma, repete-se todos esses passos quantas vezes forem necessárias até Vic estar satisfeito que, de fato, Peggy conhece as palavras secretas.

Observe que não importando a quantidade de vezes que Peggy repete os três procedimentos acima, Vic nunca irá aprender as palavras secretas (nunca irá aprender qualquer tipo de informação), mas sim somente a validade de que Peggy sabe essas palavras.

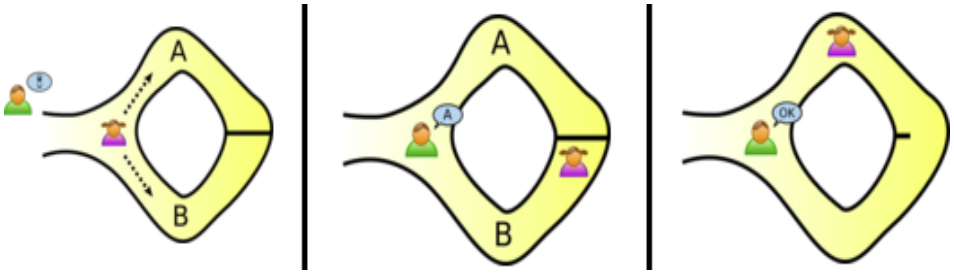


Figura 1.1: Caverna do Ali Baba. (Essa figura tem Creative Commons Atribuição 2.5 e foi legalmente copiada de http://en.wikipedia.org/wiki/Zero-knowledge_proof)

## 1.2 O que é uma prova?

Antes de introduzir as definições de provas com conhecimento zero, devemos saber o que realmente se entende por uma **prova** e também quais são suas características. Nesta seção ao falarmos em *prova*, não estaremos limitados no sentido matemático e lógico dela, mas sim também no sentido de que uma prova pode abrangir outras áreas humanas aonde seja possível de provar fatos de natureza diferente da matemática ao qual estamos acostumados.

**Processo interativo**

O estilo mais familiar de se interpretar uma prova matemática é o concebido no estudo da lógica, aonde a prova é vista como uma sequência, que se deduzem afirmações a partir de axiomas até chegar em teoremas. Numa prova com conhecimento zero abordaremos um estilo de prova menos usual para matemáticos, porém mais parecido com o modo de encarar uma prova em outras atividades humanas. Denominamos de *processo interativo* esse estilo de prova aonde a validade da prova é adquirida com o passar do processo através da interação de dois personagens: o provador e o verificador.

Para fazer uma melhor distinção desses dois tipos de prova, podemos imaginar um ambiente escolar como proposto em [7]. Numa sala de aula, um estudante pode ter grande vantagem ao interagir com o professor que tenta provar uma afirmação. O estudante pode fazer perguntas sobre questões importantes e receber respostas. Isso torna tudo mais fácil. Por outro lado, imagine a mesma afirmação tentando ser entendida pelo aluno num livro. Não há interação e tudo se torna mais difícil.

Nesse simples exemplo acima, já é possível de ter uma noção do poder de uma *prova interativa*.

**Provador e Verificador**

Em ambos tipos de prova vistos acima existe um *provador* e um *verificador*. O objetivo do provador é convencer o verificador de alguma validade, enquanto o objetivo do verificador é de aceitar afirmações corretas. Tipicamente, o processo de verificação, i.e. o verificador, é considerado simples, de modo que o provador fique sobrecarregado de todo sistema de prova. Essa assimetria de complexidade é claramente vista num *sistema de provas da classe NP*, onde sabemos que o processo de verificação é "fácil" (i.e., tempo polinomial), e a prova geralmente é "difícil". Veremos esse exemplo com melhores detalhes na seção 2.4.

**Completude e Validade**

A *completude* e a *validade* são duas propriedades essenciais em todos os sistemas de provas. A *validade* afirma que o processo de verificação não pode ser trapaceado de forma que se aceite afirmações falsas. A *completude* é a habilidade de algum provador convencer o verificador somente de afirmações verdadeiras.

## 1.3 O que é um conhecimento?

Falamos que a principal característica das provas com conhecimento zero é a *quantidade zero de conhecimento* que é transmitida numa prova por um processo iterativo. Mas o que se entende pela palavra *conhecimento*? O que é *ganhar conhecimento*? Vejamos o seguinte exemplo.

É sabido que um grafo é euleriano se, e somente se, esse grafo é conexo e se todos seus vértices têm grau par[1]. Claramente, essa propriedade pode ser verificada em tempo polinomial no tamanho do grafo. Por outro lado, não é conhecida nenhuma caracterização "boa" de grafos hamiltonianos[2], isto é, assumindo que $P \neq NP$, nenhum procedimento eficiente de verificação

[1] Veja a definição de Euleriano na seção 2.3.
[2] Seção 2.3

existe.

Consideremos uma conversa entre Alice e Bob aonde Bob faz perguntas para Alice sobre um grafo conhecido por ambos [5]. Consideremos primeiro o caso em que Bob pergunta para Alice se o grafo é Euleriano ou não. Claramente, dizemos que Bob não ganha conhecimento da resposta de Alice, pois ele mesmo pôde facilmente determinar essa resposta usando o Teorema de Euler. Por outro lado, se Bob pergunta para Alice se o grafo é Hamiltoniano, e ela de alguma maneira consegue responder essa pergunta, então dizemos Bob *ganhou* conhecimento, pelo fato de nao sabermos um procedimento eficiente que o proprio Bob utilize para determinar essa resposta.

Portanto, dizemos que Bob *ganhou conhecimento* da interação se sua *habilidade computacional aumentou*. Ou seja, depois da interação ele pôde facilmente computar algo que não podia computar eficientemente antes da interação.

## 1.4 Organização do texto

Objetivamos neste trabalho o resgate de todo o concebimento do sistema de provas interativa dotadas da característica do conhecimento zero, inicialmente introduzido por Goldwasser, Micali e Rackoff [7] em 1985. Sendo assim, o trabalho está organizado da seguinte maneira. No capítulo 2, exibimos alguns aspectos introdutórios importantes para o entendimento do tema em estudo tais como o modelo computacional usado, algumas classes de complexidade e outras definições abordando principalmente algoritmos probabilísticos. No capítulo 3, mostramos o concebimento do sistema de provas interativa e também um exemplo desse sistema. No capítulo 4, mostramos o sistema de provas interativas com conhecimento zero e o exemplo clássico do isomorfismo de grafos com sua demonstração. A conclusão se encontra no capítulo 5.

# 2 Preliminares

Essa seção inclui uma breve revisão a algumas classes de complexidade computacional focalizando principalmente o estudo de algoritmos probabilísticos.

## 2.1 Modelo Computacional

É necessário estabelecer modelos computacionais abstratos que exprimam os aspectos práticos de uma computação. O modelo mais usado, e o estudado em questão, é o da máquina de Turing.

**Definição 2.1** (Máquina de Turing determinística)**.** *Uma máquina de Turing determinística é uma quádrupla* $M = (S, \Sigma, \delta, s)$. *Aqui S é o conjunto finito de estados, onde* $s \in S$ *determina o* estado inicial *da máquina. A máquina usa um conjunto finito de símbolos, denotados* $\Sigma$*; esse conjunto inclui os símbolos especiais BLANK e FIRST*[1]. *A função* $\delta$ *é a função de transição da máquina de Turing, mapeando* $S \times \Sigma$ *em* $(S \cup \{HALT, YES, NO\}) \times \Sigma \times \{\leftarrow, \rightarrow, STAY\}$. *A máquina têm três estados: HALT (o* estado de parar*), YES (o* estado de aceitar*), NO (o* estado de rejeitar*) (esses estados formalmente não estão em S).*

A máquina de Turing determinística funciona da seguinte maneira [16]. A entrada da máquina é geralmente vista como sendo escrita numa *fita*; ao menos que outro jeito seja especificado, a máquina pode ler e escrever nessa fita. Assumimos que $HALT$, $YES$ e $NO$, e também os símbolos $\leftarrow$, $\rightarrow$ e $STAY$, como não pertencentes a $S \bigcup \Sigma$. A máquina começa no estado inicial $s$ com o cursor no primeiro símbolo da entrada $x$; este símbolo é sempre $FIRST$.

[1] Acredito que a *não*-tradução de algumas palavras no texto não atrapalhe o leitor. Sendo assim, no decorrer do texto deixarei algumas palavras como originalmente foram concebidas em sua lingua mãe para não fazer nenhuma confusão com outras bibliografias.

O resto da entrada é uma seqüência finita de símbolos ("string") de $\Sigma \setminus \{BLANK, FIRST\}$; o símbolo *BLANK* mais a esquerda da fita identifica o final da "string" de entrada.

A função de transição dita as ações da máquina e pode ser vista como o seu *programa*. Em cada passo, a máquina lê o símbolo $\alpha$ da entrada que está sendo apontada pelo *cursor*. Baseada nesse símbolo e no estado corrente da máquina, é escolhido o próximo estado, um símbolo $\beta$ a ser sobreescrito em $\alpha$ e um movimento do cursor em direção a $\{\leftarrow, \rightarrow, STAY\}$, onde $\leftarrow$ e $\rightarrow$ diz para o cursor mover para o próximo símbolo a esquerda ou a direita, respectivamente, e *STAY* diz para o cursor permanecer na posição presente. A função de transição é projeta para garantir que o cursor nunca saia fora do final da entrada, identificado por $FIRST$. A máquina pode claramente sobreescrever o símbolo *BLANK*.

Se a máquina pára no estado $YES$, dizemos que ela *aceitou* a entrada $x$. Se a máquina pára no estado $NO$, dizemos que ela *rejeitou* a entrada $x$. O terceiro estado, $HALT$, é para a computação de funções que não sejam booleanas; nesses casos, a saída da função computada é escrita na própria fita. Um **algoritmo** corresponde a uma máquina de Turing que sempre pára.

Uma *máquina de Turing probabilística* corresponde a uma variação da máquina de Turing vista acima adicionada da habilidade de gerar bits aleatórios em um passo [16]. Numa entrada $x$, a máquina de Turing probabilística aceita $x$ com alguma probabilidade, como veremos na seção 2.2. Essa máquina corresponde a um **algoritmo probabilístico**.

## 2.2 Classes de Complexidade

Definiremos agora algumas classes de complexidade computacional visando o estudo de algoritmos probabilísticos, porém antes veremos algumas definições básicas [10].

Um *alfabeto* é um conjunto finito e não-vazio, por exemplo, é suficiente assumirmos $\Sigma = \{0,1\}$. Denotamos por $\Sigma^*$ o conjunto de todas as possíveis *palavras* ("strings") sobre esse alfabeto. Denotamos por $|s|$ o comprimento da palavra $s$. Uma *linguagem* $L$ é uma coleção de palavras sobre $\Sigma$, ou seja $L \subseteq \Sigma^*$. O *problema de reconhecimento de linguagem*, isto é, o

problema de decidir se uma palavra $x$ de $\Sigma^*$ pertence a $L$. Um algoritmo resolve um problema de reconhecimento de uma linguagem específica $L$ *aceitando* (a máquina de Turing pára no estado $YES$) qualquer entrada pertencente a $L$.

Uma *classe de complexidade* é uma coleção de linguagens as quais problemas de reconhecimento podem ser resolvidos sob limites nos recursos computacionais como, por exemplo, limitando em função do tamanho da entrada o número de transições da função de transição (limitação de tempo) ou o número de posições percorridas na fita (limitação de espaço). Estamos interessados em algoritmos eficientes, aonde eficiência é definida como sendo limitação de tempo por uma função polinomial no tamanho da entrada. Um algoritmo tem *tempo polinomial* se ele pára com tempo $n^{O(1)}$ em qualquer entrada de tamanho $n$.

**Definição 2.2.** *A classe* P *consiste de todas as linguagens L tais que existe um algoritmo A de tempo polinomial tal que para cada entrada* $x \in \Sigma^*$,

- $x \in L \Rightarrow A(x)$ *aceita.*
- $x \notin L \Rightarrow A(x)$ *rejeita.*

**Definição 2.3.** *A classe* NP *consiste de todas as linguagens L tais que existe um algoritmo A de tempo polinomial tal que para cada entrada* $x \in \Sigma^*$,

- $x \in L \Rightarrow \exists y \in \Sigma^*$, $A(x,y)$ *aceita, onde* $|y|$ *é limitado por um polinômio em* $|x|$.
- $x \notin L \Rightarrow \forall y \in \Sigma^*$, $A(x,y)$ *rejeita.*

Obviamente, $P \subseteq NP$, mas não é sabido se $P = NP$ e esse é um dos problemas não resolvidos mais importantes dentro da matemática [1].

Para toda classe de complexidade $C$, definimos a classe co-$C$ como sendo o conjunto de linguagens as quais o complemento está na classe $C$, isto é co-$C = \{\mathrm{L} \subseteq \Sigma^* \mid \bar{\mathrm{L}} \in C\}$.

É fácil ver que $P =$ co-$P$. Por outro lado não é conhecido se $NP =$ co-$NP$. É sabido que se $NP \neq$ co-$NP$ então $P \neq NP$ e a recíproca também não se sabe se vale ou não.

A quantidade de espaço (memória) ocupado é outro fator que consideremos na classificação de problemas de acordo com sua dificuldade de computação. Se modificarmos as definições 2.2 e 2.3 de modo que utilizem espaço polinomial, ao invés de tempo, então temos as classes *PSPACE* e *NPSPACE*. O espaço parece ser algo mais poderoso que o tempo, pois o espaço pode ser reutilizado enquanto o tempo não. É sabido que $PSPACE = NP$ [19].

Agora iremos generalizar essas classes vistas acima de modo que permitam que sejam vistas com algoritmos probabilísticos. Algoritmos probabilísticos diferem dos algoritmos que vimos até agora em duas maneiras:

1. Eles têm a habilidade de lançar moedas e gerar bits aleatórios.

2. Eles podem cometer erros.

**Definição 2.4.** *A classe* RP *("randomized polynomial") consiste de todas as linguagens L para as quais existe um algoritmo probabilístico A de tempo polinomial tal que para cada entrada* $x \in \Sigma^*$,

- $x \in L \Rightarrow Prob[A(x)\ aceitar\ ] \geq \frac{1}{2}$.
- $x \notin L \Rightarrow Prob[A(x)\ aceitar\ ] = 0$.

A escolha da constante $\frac{1}{2}$ é arbitrária; notemos que iterações independentes do algoritmo podem ser usadas para diminuir essa probabilidade de erro. Portanto, a probabilidade de sucesso pode ser trocada por qualquer função polinomial inversa do tamanho da entrada sem significantemente afetar a definição 2.4.

Observe que um algoritmo na classe *RP* pode errar somente quando $x \in L$. Dessa maneira denominamos essa classe de "one-sided error" (erro de um lado).

Consideremos agora a classe de problemas que admite algoritmos probabilísticos da forma "two-sided error" (erro dos dois lados).

**Definição 2.5.** *A classe* PP *("probabilistic polynomial time") consiste de todas as linguagens L para as quais existe um algoritmo probabilístico A de tempo polinomial tal que para cada entrada* $x \in \Sigma^*$,

- $x \in L \Rightarrow Prob[A(x)\ aceitar\ ] > \frac{1}{2}$.
- $x \notin L \Rightarrow Prob[A(x)\ aceitar\ ] < \frac{1}{2}$.

Da mesma maneira que na definição 2.4 podemos reduzir a probabilidade de erro fazendo sucessivas iterações independentes da mesma entrada e produzir uma saída que ocorre na maioria dessas iterações. Entretanto, a probabilidade de aceitar uma entrada que *pertence* e a probabilidade de aceitar uma entrada que *não pertence* a uma linguagem dessa classe é igual. Dessa forma existe uma definição mais útil para o nosso propósito.

**Definição 2.6.** *A classe* BPP *("bounded-error probabilistic polynomial time") consiste de todas as linguagens L para as quais existe um algoritmo probabilístico A de tempo polinomial tal que para cada entrada* $x \in \Sigma^*$,

- $x \in L \Rightarrow Prob[A(x)\ aceitar\ ] \geq \frac{3}{4}$.
- $x \notin L \Rightarrow Prob[A(x)\ aceitar\ ] \leq \frac{1}{4}$.

Claramente temos que $RP \subseteq BPP \subseteq PP$ [17].

Está em aberto se $BPP \subseteq NP$. Se $NP \subseteq BPP$ então $RP = NP$ [12].

Analogamente, poderíamos definir a classe *PPSPACE* trocando tempo por espaço na definição de *PP* mas mostra-se que tal classe é igual a *PSPACE* [2].

**Definição 2.7.** *Uma* redução polinomial *de uma linguagem* $L_1 \subseteq \Sigma^*$ *para uma linguagem* $L_2 \subseteq \Sigma^*$ *é uma função* $f : \Sigma^* \to \Sigma^*$ *tal que:*

- *Existe um algoritmo de tempo polinomial que computa* $f$.
- *Para todo* $x \in \Sigma^*$, $x \in L_1$ *se e somente se* $f(x) \in L_2$.

**Definição 2.8.** *Numa classe C uma linguagem L é C-*difícil *se, para toda* $L' \in C$, *existe uma redução polinomial de* $L'$ *para L.*

**Definição 2.9.** *Numa classe C uma linguagem L é NP-*completa *se ela está em C e for NP-*difícil.

Note que existência de um algoritmo polinomial para decidir uma linguagem $NP$-completa implica em $P = NP$. A primeira linguagem que foi mostrada ser $NP$-completa foi a das fórmulas booleanas verdadeiras. Uma linguagem $PSPACE$-completa é a da *fórmula booleana quantificada* formada pelas fórmulas booleanas com quantificadores universal e existencial verdadeiras. Não se sabe se existe uma linguagem completa em $BPP$-completa.

## 2.3 Grafos

**Definição 2.10.** *Um* grafo *é um par $G = (V,E)$, onde $V$ é um conjunto finito de* vértices *e $E$ é um conjunto de* arestas $E = \{\{u,v\} \mid u,v \in V\}$. *As vezes denotaremos a aresta $\{u,v\}$ por $uv$.*

**Definição 2.11.** *Dizemos que os grafos $G$ e $H$ são isomorfos se existe uma bijeção $f$ dos vértices de $G$ nos vértices de $H$ tal que $\{x,y\}$ é uma aresta de $G$ se, e somente se, $\{f(x),f(y)\}$ é uma aresta de $H$.*

Definimos as linguagens

$$ISO = \{< G,H > \mid G \text{ e } H \text{ são grafos isomorfos}\}$$

e

$$NONISO = \{< G,H > \mid G \text{ e } H \text{ não são grafos isomorfos}\}$$

onde $< X >$ denota uma codificação de $X$ em $\Sigma^*$.

**Definição 2.12.** *Um* caminho *é um grafo não-vazio $P = (V,E)$ da forma*

$$V = \{x_0, x_1, \cdots, x_k\}$$

*e*

$$E = \{x_0x_1, x_1x_2, \cdots, x_{k-1}x_k\}.$$

*Denotamos esse camiho por $P = x_0 \cdots x_k$.*

**Definição 2.13.** *Se $P = x_0 \cdots x_{k-1}$ é um caminho e $k \geq 3$, então o grafo $C := P + xk - 1x_0$ é chamado de* ciclo.

**Definição 2.14.** *Um grafo não vazio G é* conexo *se todos os seus vértices, dois a dois são ligados por um caminho em G.*

**Definição 2.15.** *Um grafo conexo é dito* euleriano *se o conjunto de arestas E pode ser escrito como uma seqüência* $e_1, e_2, \ldots, e_{|E|}$ *tal que* $e_i \cap e_{i+1} \neq \emptyset$ *para todo* $1 \leq i \leq |E| - 1$ *e* $e_1 \cap e_2 \neq e_1 \cap e_{|E|} \neq \emptyset$.

Um famoso teorema caracteriza os grafos eulerianos, para uma prova veja [4] por exemplo.

**Teorema 2.16** (Euler 1736, Hierholzer 1873)**.** *Um grafo conexo é euleriano se e somente se todos os seus vértices têm grau par.*

**Definição 2.17.** *Um ciclo que contém todos os vértices de G é chamado de* ciclo hamiltoniano *de G. Um grafo é* hamiltoniano *se e somente se contém um ciclo hamiltoniano.*

## 2.4 Sistema de provas

Nas seções anteriores, mencionamos algumas vezes o termo *sistema de provas*, porém ainda não foi definido formalmente o que seja isso. Um *sistema de provas* é visto como duas máquinas de Turing – ou dois algoritmos – que se comunicam através de leitura/escrita numa fita comum [13]. Chamaremos essas máquinas de $V$, o *verificador*, e $P$, o *provador*. As duas máquinas têm acesso a uma fita de leitura contendo a entrada $x$, ou seja, o fato a ser provado. Existem mais duas fitas compartilhadas por $P$ e $V$, que são as fitas de leitura/escrita. Numa delas, o provador $P$ pode escrever e $V$ somente fazer leituras. Na outra delas, os papéis são trocados, ou seja, $V$ pode escrever e $P$ somente fazer leituras.

A classe *NP* é um bom exemplo [5] de um sistema de provas (Figura 2.1), onde a única restrição é na complexidade do processo de verificação, que precisa ter tempo polinomial. Note que na Figura 2.1 apenas uma fita compartilhada é usada para o *sistema NP de provas*, diferente do *sistema de provas*. Isso porque no sistema *NP* permitimos somente o envio de uma mensagem $y$ do Provador para o Vericador que, juntamente com o entrada $x$, faz uma computação de tempo polinomial de modo que a classe das linguagens que admitem tal sistema de provas coincida com a classe *NP*.

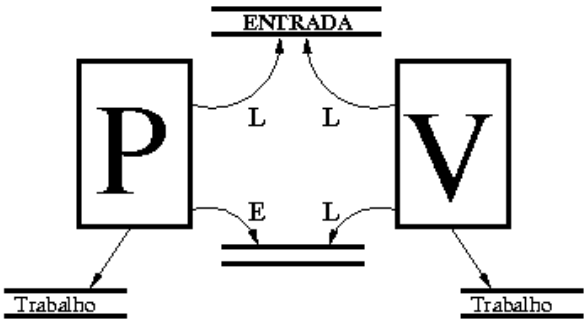


Figura 2.1: Sistema NP de provas

Portanto, num sistema de provas para uma linguagem $L$ em $NP$, a prova para a afirmação "$x \in L$" consiste do provador enviar uma testemunha $y$, e o verificador checar com a ajuda da prova $y$ e em tempo polinomial se $x \in L$. Tal testemunha existe somente se a afirmativa é verdadeira, portanto somente afirmações verdadeiras podem ser provadas por esse sistema. Observe que não existe restrições na complexidade do tempo de encontrar a prova (testemunha).

**Definição 2.18** (Sistema NP de provas)**.** *Uma linguagem $L \subseteq \Sigma^*$ está na classe de linguagens NP se existe um sistema de provas tal que para todo $x \in \Sigma^*$*

1. *a máquina de Turing V tem complexidade de tempo polinomial;*
2. *P computa computa uma palavra y e a escreve na fita $F_P$;*
3. *se $x \in L$ a máquina de Turing V, no término de suas computações, escreve YES na fita $F_V$, após eventuais consultas à fita $F_P$;*
4. *se $x \notin L$ a máquina de Turing V, no término de suas computações, escreve NO na fita $F_V$, após eventuais consultas à fita $F_P$;*

Observe que no exemplo acima, a prova foi interpretada pelo processo dinâmico (citado na seção 1.2), e não pelo estático ao qual estamos mais familiarizados. Num sistema de provas encarado pelo processo dinâmico, podemos ver que existe uma rodada trivial onde uma mensagem é enviada pelo provador ao verificador, *uma única vez*, que então pode verificar sozinho a validade da prova de modo determinístico. Alguns autores chamam esse tipo de interação como sendo *unidirecional*. No entanto, ganharíamos muito mais "expressividade" e poder, em

termos de complexidade computacional, se permitíssemos que o processo de verificação fosse **probabilístico** e trocas de mensagens – uma real **interação** – entre o provador e o verificador acontecesse (interação do tipo bidirecional).

Adicionando aleatoriedade e interação, o sistema de provas é expandido e então surge o **sistema de provas interativa**. Tal modelo exige que o processo de verificação – o verificador – seja limitado na complexidade computacional. Além disso, o fato do verificador "lançar moedas" (usar a aleatoriedade) traz em evidência uma probabilidade de erro, que pode ser reduzida por sucessivas aplicações independentes do sistema de provas. Veremos como que funciona esse sistema de provas no capítulo que segue.

# 3 *Sistema de prova interativa*

Justo como no sistema de provas, o objetivo de um *sistema de provas interativas* é o de permitir o provador a convencer o verificador da validade de uma afirmação. Introduziremos a noção de provas interativas, inicialmente apresentando esse modelo visto como uma máquina de Turing. Para isso precisamos definir uma máquina de Turing especial dita *máquina de Turing interativa*.

## 3.1 Máquina de Turing interativa

Uma máquina de Turing interativa é uma máquina determinística e multi-fita que envia e recebe mensagens enquanto faz suas computações. Chamaremos essas máquinas de verificador e provador, $V$ e $P$ respectivamente. As duas máquinas de Turing interativas têm acesso comum a uma fita de leitura contendo uma palavra $x$ (o fato a ser provado). Cada máquina tem acesso a uma fita particular contendo bits aleatórios. Existem ainda mais duas fitas compartilhadas, chamadas de fitas de comunicação $F_P$ e $F_V$. Numa delas, na fita $F_P$, o provador $P$ pode escrever e $V$ somente fazer leituras. Na outra delas, na fita $F_V$, os papéis são trocados, ou seja, $V$ pode escrever e $P$ somente fazer leituras [13].

Sumarizando, a computação nesse par de máquinas consiste na troca de mensagens entre máquinas, utilizando-as na computação junto com a entrada em comum e também das entradas aleatórias (distintas) 3.1.

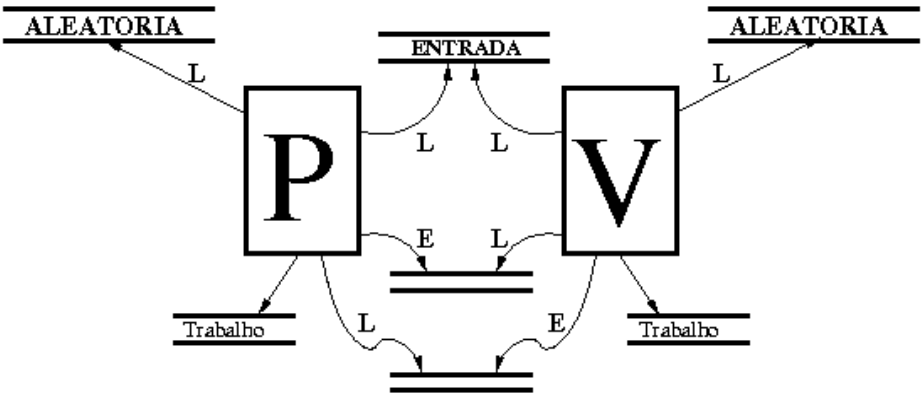


Figura 3.1: Sistema de provas interativo

## 3.2 Definição

Intuitivamente, podemos pensar na interação como sendo perguntas "espertas" feitas pelo verificador ao qual o provador deve responder convincentemente. O verificador não deve aceitar afirmações falsas, ou seja, nenhum provador desonesto deve ser capaz de trapaceá-lo com fatos falsos. Além disso, estamos preocupados com a eficiência desse tipo de prova: não importa qual é o tempo que o provador demore no processo de prova, mas é essencial que a computação feita pelo verificador seja de tempo polinomial no comprimento da entrada comum as máquinas interativas.

Estamos tratando de algoritmos probabilísticos, portanto o veridito do verificador de aceitar ou rejeitar uma afirmação é uma variável aleatória. Mais especificamente, o verificador é uma "máquina *BPP*"; desse modo a aceitação, no lado do verificador, de afirmações válidas deve ser realizada com alta probabilidade e, existe uma probabilidade de erro na verificação de um fato, isto é, o verificador aceita afirmações falsas com uma probabilidade pequena. Na definição 3.2 garantimos tudo isso que foi discutido nesses últimos dois parágrafos.

**Definição 3.1.** *Seja $P$ e $V$ um par de máquinas interativas de Turing e suponha que todas as possíveis interações de $P$ e $V$ numa entrada em comum terminem num número finito de passos. Então denotamos $(P,V)(x)$ sendo a* variável aleatória *que representa a saída de $V$ interagindo com a máquina $P$ numa entrada em comum $x$.*

**Definição 3.2** (Sistema de provas interativo e a classe IP)**.** *Um par de máquinas interativas $(P,V)$ é chamado de um* sistema de provas interativo para uma linguagem *$L$ se a máquina $V$ é*

*de tempo polinomial e valem as seguintes condições:*

- *(**completude**) para todo $x \in L$*

$$Prob[(P,V)(x)\ aceitar\ ] \geq \frac{3}{4}$$

- *(**validade**) para todo $x \notin L$ e qualquer que seja a máquina interativa $P^*$ interando com $V$ no lugar de $P$*

$$Prob[(P^*,V)(x)\ aceitar\ ] \leq \frac{1}{4}$$

*A classe de linguagens que admite provas interativas é denotada IP*

Justo como já dissemos, o valor de negligência ($1/4$, no caso da definição 3.2) pode ser diminuido ao repetir várias vezes a interação entre o verificador e provador.

Apresentamos na seção seguinte, um exemplo que demonstra o poder da classe *IP*.

## 3.3 Exemplo: $NON \in IP$

A linguagem *ISO* está em *NP*. Porém, não se sabe se *NONISO* está em *NP*, pois não conseguimos mostrar testemunhas curtas (de tempo polinomial) que digam se os grafos não são isomorfos. Apesar disso, um *provador* — Máquina de Turing com grande pode computacional — consegue convencer um *verificador* — Máquina de Turing polinomialmente limitada — desse fato utilizando um sistema de provas interativo.

Seja $< G_1, G_2 >$ uma instância do problema, onde os grafos dados são $G_1 = (V, E_1)$ e $G_2 = (V, E_2)$. Definimos o seguinte protocolo [13]:

1. O verificador escolhe aleatoriamente e uniformemente $c \in \{1,2\}$ e uma permutação $\pi$ de $V$.

2. O verificador computa o grafo $G_3 = (V, E_3)$, onde $E_3$ é definido por $\{\pi(u), \pi(v)\} \in E_3$ se e somente se $\{u, v\} \in G_c$.

3. O verificador envia $G_3$ ao provador (escreve na fita $F_V$).

4. O provador, depois de ler $G_3$, envia ao verificador o valor de $c$ (escreve na fita $F_P$).

5. Se o provador acerta no passo 4 o valor de $c$, o verificador ACEITA.

6. Se o provador erra no passo 4 o valor de $c$ e o protocolo foi executado uma única vez, o verificador começa uma nova rodada a partir do passo 1. Se o protocolo foi executado duas vezes o verificador REJEITA.

Se os grafos $G_1$ e $G_2$ não são isomorfos, então o provador pode sempre distinguir o caso em que $G_3$ é isomorfo a $G_1$ do caso em que $G_3$ é isomorfo a $G_2$, e sempre acertar o passo 4 (i.e., temos a completude com probabilidade 1). Entretanto, se os grafos são isomorfos, então devido à escolha aleatória da permutação $\pi$ pelo provador, o grafo $G_3$ pode ter vindo de $G_1$ ou $G_2$, e mesmo com um grande poder computacional, o provador acerta a resposta em cada rodada do protocolo com probabilidade no máximo $1/2$, portanto ao fim de duas rodadas (os sorteios em cada rodada são independentes) temos o limitante desejado de $1/4$ da validade do sistema.

O algoritmo do verificador apresentado acima é facilmente implementado em tempo polinomial probabilístico. Não é conhecido um algoritmo dessa mesma complexidade para o provador, no entanto isso também não é necessário pela definição de prova interativa.

Trivialmente, $NP \subseteq IP$. O exemplo anterior nos mostra que $NONISO \in IP$, mas ainda assim não sabemos se $NONISO \in NP$, ou seja, não sabemos se a inclusão $NP \subseteq IP$ é própria ($NP \subsetneq IP$).

# 4 *Provas com Conhecimento Zero*

"Extraordinary claims require
extraordinary proof." – Carl Sagan

"Gregory: I'm here for 'la resistance.
Kyle: What's the password?
Gregory: I don't know.
Kyle: Guess.
Gregory: Uhhh . . . bacon.
Kyle: Okay." – South Park (desenho
animado norte-americano)

"Beware of bugs in the above code;
I have only proved it correct, not
tried it" – Donald Knuth

Retomemos o exemplo inicial da caverna de Ali Babá (seção 1.1). O modo trivial de Peggy mostrar que conhece as palavras secretas que abrirá a porta de dentro da caverna é justamente mostrando essas palavras para Vic. No entanto, ao fazer isso, Peggy estará transmitindo muitas informações e fugindo do escopo real problema, que é o de *mostrar que ela sabe as palavras que abrirão a porta de dentro da caverna* e não simplesmente do modo trivial que é *mostrando as palavras secretas*. Ou seja, numa prova interativa informações desnecessárias são transmitidas entre o verificador e o provador. Portanto alguns sistemas podem ser formulados de tal forma que só **transmitam ao verificador somente a validade do que está sendo provado**.

## 4.1 Definições

O fato do verificador *não aprender nada* significa que ele não adquire nenhum conhecimento tal que ele mesmo consiga computar algo que não conseguia antes da interação. Podemos ver isso da seguinte maneira, como descrevemos.

Dizemos que um provador $P$ não transmite nenhuma "informação extra" numa prova se tudo aquilo que um verificador $V^*$ qualquer puder computar ao **interagir** com $P$ e recebendo a entrada $x$, é igual a tudo aquilo que esse mesmo verificador, **não interagindo** com $P$, consegue ele mesmo computar com a entrada $x$. Em outras palavras, existe um algoritmo $M^*$ tal que $(P,V^*)(x)$ e $M^*(x)$ entrada $x$) são duas variáveis aleatórias identicamente distribuídas, denotado por $(P,V^*)(x) \cong M^*(x)$. Dizemos que e $M^*(x)$ é um *simulador* da interação de $V^*$ com $P$.

**Definição 4.1** (conhecimento-zero perfeito)**.** *Seja $(P,V)$ um sistema de prova interativo para uma linguagem L. Dizemos que $(P,V)$, na verdade P, têm* ***conhecimento-zero perfeito*** *se para toda máquina interativa de tempo polinomial $V^*$ existe uma máquina probabilística de tempo polinomial $M^*$, tal que para todo $x \in L$*

$$(P,V^*)(x) \cong M^*(x).$$

*A máquina $M^*$ é chamada de* simulador perfeito *para a interação de $V^*$ com P.*

Para propósitos práticos a formulação de conhecimento zero não precisa ser tão estrita do modo que foi definida acima, ou seja, as duas variáveis aleatórias de interesse $(P,V^*)(x)$ e $M^*(x)$ não precisam ser identicamentes distribuídas, mas sim próximas o suficiente. Sendo assim, temos outra definição[1] de conhecimento zero que é mais relaxada, permitindo a negligência do simulador denominada de *conhecimento zero computacional*. Essa definição requer que a probabilidade das distribuições sejam computacionalmente indistinguíveis.

**Definição 4.2.** *Duas distribuições $\{A_x\}_{x\in L}$ e $\{B_x\}_{x\in L}$ são* computacionalmente indistinguíveis *se para cada algoritmo polinomial probabilístico D e dado um polinômio p e dado $x \in L$ sufi-*

[1] Existe também mais uma definição, mais relaxada que o conhecimento zero perfeito e menos relaxada que o conhecimento zero computacional, denominada *conhecimento zero estatístico*

*cientemente longo,*

$$\Big|Prob[D(x,A_x)=1]-Prob[D(x,B_x)=1]\Big| \leq \frac{1}{p(|x|)}$$

*onde as probabilidades são tomados sobre as distribuições em questão ($A_x$ ou $B_x$) e sobre a distribuição dos bits aleatórios de D.*

**Definição 4.3** (conhecimento-zero computacional)**.** *Seja $(P,V)$ um sistema de prova interativa para uma linguagem L. Dizemos que $(P,V)$, na verdade P, têm* ***conhecimento-zero computacional****, ou somente* ***conhecimento-zero****, se para toda máquina interativa de tempo polinomial $V^*$ existe uma máquina probabilística de tempo polinomial $M^*$, tal que as distribuições $(P,V^*)(x)$ e $M^*(x)$ são computacionalmente indistinguíveis.*

**Definição 4.4.** *Denotamos por ZK (também CZK) a classe de linguagens que admitem sistemas de prova interativa com conhecimento zero* computacional. *Da mesma forma, denotamos por PZK a classe de linguagem que admitem sistemas de prova interativa com conhecimento zero* perfeito.

Claramente temos que $BPP \subseteq PZK \subseteq CZK \subseteq IP$.

## 4.2 Exemplo: $ISO \in PZK$

Apresentaremos nesta seção um sistema de prova interativa com conhecimento zero para uma linguagem que não se sabe estar em *BPP*. Tal linguagem é a *ISO*, previamente definida na seção 2.3. Iremos também enunciar o teorema que mostra a pertinência dessa linguagem na classe *PZK* (conhecimento zero perfeito), além de esboçar uma prova para esse teorema.

Assume-se que tanto o provador $P$ como o verificador $V$ conhecem dois grafos $G_1$ e $G_2$ com $n$ vértices. $P$ afirma conhecer um isomorfismo $\phi : G_1 \to G_2$ e quer provar para $V$ que conhece esse isomorfismo sem de fato mostrá-lo. O sistema é construído a seguir, onde $\pi \circ \phi$ significa a função composta da função $\pi$ com a função $\phi$, definida como $\pi \circ \phi(v) = \pi(\phi(v))$.

1. **P1**: o provador sorteia, com probabilidade uniforme de distribuição, uma permutação $\pi$

do conjunto de permutações do vértice $V_2$, e constrói o grafo H (isomorfo a $G_2$):

$$F = \big\{\{\pi(u), \pi(v)\} \mid \{u,v\} \in E_2\big\};$$

o provador envia $H = (V_2, F)$ ao verificador

2. **V1**: Recebido o grafo, denotado por $G' = (V', E')$, o verificador pede ao provador para mostrar um isomorfismo entre $G'$ e um dos grafos de entrada, escolhido aleatoriamente pelo verificador. Em outras palavras, o verificador escolhe uniformemente $\sigma \in \{1,2\}$ e envia ao provador.

3. **P2**: Se a mensagem, $\sigma$, é igual a 2 então o provador responde $\pi$. Caso contrário, se a mensagem é $\sigma = 1$, o provador responde $\pi \circ \phi$ ao verificador.

4. **V2**: Se a mensagem, denotada por $\psi$, recebida pelo provador é um isomorfismo entre $G_\sigma$ e $G'$, então o verificador ACEITA, caso contrário REJEITA.

Vamos denotar o programa do provador por $P_{GI}$. Eis algumas observações. Se os grafos de entrada são isomorfos, então o grafo $H$ enviado pelo provador é isomorfo a ambos. Se os grafos de entrada não são isomorfos, então não existe nenhum grafo que pode ser isomorfo a ambos. Se a condição do passo V2 não se verificar, então o verificador não fica convencido que o provador conhece o isomorfismo entre $G_1$ e $G_2$. Se essa condição se verificicar, o protocolo é repetido 2 vezes, e o verificador só acredita que o provador conhece esse isomorfismo se em todas essas iterações a condição do passo V2 se verificar.

Demonstraremos agora que o par de máquinas interativas $P$ e $V$ visto acima constituem um sistema de prova interativa de conhecimento zero perfeito. Utilizaremos nesta demonstração o *paradigma da simulação*: qualquer verificador pode ser simulado por uma máquina que não interage com o provador e, do modo como vimos nas definições de prova com conhecimento zero (4.1 e 4.3), nos garante que nenhuma informação é transmitida além da validade da prova.

**Teorema 4.5.** *ISO está em PZK.*

*Demonstração [5].* A construção dessa prova deve satisfazer as três características de um sis-

tema de prova interativa de conhecimento zero. Elas são: *completude, validade* e *conhecimento zero*.

**Completude**

Se os dois grafos de entrada são isomorfos, então o verificador sempre irá aceitar. De fato, no final do protocolo vale que $H = \psi(G_\sigma)$ pois

1. Caso $\sigma = 2$: o verificador tem $H = \pi(G_2)$ e a permutação $\psi = \pi$, portanto $H = \psi(G_\sigma)$.

2. Caso $\sigma = 1$: o verificador também tem $H = \pi(G_2)$ e a permutação $\psi = \pi \circ \phi$, portanto $H = \psi(G_\sigma)$.

Portanto o verificador aceita com probabilidade 1.

**Validade**

Seja $G_1 \ncong G_2$. Vamos mostrar que a probabilidade de qualquer provador convencer o verificador de que esses grafos são imomorfos é no máximo $1/4$.

1. Considere qualquer $P^*$. Suponha que $H$ não é isomorfo a $G_1$ e nem $G_2$, então $P^*$ falha no P1, onde deveria mostrar um isomorfismo de $G_\sigma$ e $H$, e a probabilidade de convencer o verificador que $< G_1, G_2 > \in ISO$ é zero.

2. Suponha que $H$ é isomorfo a $G_1$ (o caso $H \cong G_2$ é análogo). Se o verificador sorteia $\sigma = 1$, então $P^*$ será capaz de mostrar o isomorfismo entre $H$ e $G_\sigma = G_1$. Caso contrário, se o verificador sortear $\sigma = 2$, então não existe um isomorfismo entre $H \cong G_1$ e $G_\sigma = G_2$, portanto $P^*$ não seria capaz de encontrar um isomorfismo mesmo com seu poder computacional ilimitado. Portanto, nesse caso, $P^*$ teria probabilidade exatamente $\frac{1}{2}$ de convencer o verificador que os grafos são isomorfos.

   Consequentemente, para qualquer provador $P^*$ temos

$$Prob\Big[(P^*, V)(< G_1, G_2 >) \; aceitar \mid G_1 \ncong G_2\Big] \leq \frac{1}{2}$$

O protocolo é executado duas vezes sequencialmente de forma independente, obtendo uma probabilidade de no máximo $\frac{1}{4}$, e assim, satisfazendo a propriedade da validade dos sistemas de provas interativas.

**Conhecimento zero**

Devemos construir um simulador, $M^*(x)$, que gere a mesma distribuição que $(P,V^*)(x)$. O simulador irá incorporar o papel do $V^*$. Para entrada $x =< G_1, G_2 >$, o simulador procede da seguinte maneira:

1. **Simula a fita aleatória de** $V^*$: Seja $q(\cdot)$ o polinômio que limita o tempo percorrido de $V^*$. $M^*$ sorteia uma *string* $r \in \{0,1\}^{q(|x|)}$ como conteúdo da fita aleatória de $V^*$.

2. **Simula o P1**: O simulador sorteia com probabilidade uniforme $\tau \cong \{1,2\}$ e constrói (usando bits da fita aleatória) uma cópia isomorfa $\psi$ de $G_\tau$. Ou seja, o simulador constrói $G'' = \psi(G_\tau)$.

Reparemos nesse passo que o simulador agiu diferente do $P_{GI}$, mas o grafo gerado ($G''$) é identicamente distribuído com a mensagem enviada no P1 da prova interativa.

3. **Simula o V1**: $M^*$ inicia a execução de $V^*$ colocando $x$ na entrada comum de $V^*$, $r$ na fita aleatória de $V^*$ e $G''$ na fita de entrada de mensagens de $V^*$. Após executar um número polinomial de passos de $V^*$, o simulador pode ler a saída de $V^*$, denotada por $\sigma$.

4. **Simular o P2**: Se $\sigma = \tau$ o simulador pára e mostra $(x, r, G'', \psi)$.

5. **Falha na simulação**: Se $\sigma \neq \tau$ o simulador pára e mostra $\perp$.

Digamos que o simulador gerou $G''$ após permutar os vértices de $G_1$. Então, se $V^*$ pedir para ver o isomorfismo entre $G_1$ e $G''$, o simulador responderá corretamente e, fazendo isso, completa a simulação. Entretanto, se $V^*$ pedir para ver o isomorfismo entre $G_2$ e $G''$, então

o simulador não consiguirá responder corretamente e irá mostrar $\perp$. Sendo assim, devemos demonstrar que $M^*$ imprime $\perp$ com probabilidade no máximo $\frac{1}{2}$. Vejamos.

**Proposição 4.6.** *Suponha que $G_1$ e $G_2$ são isomorfos. Seja $\xi$ uma variável aleatória uniformemente distribuída em $\{1,2\}$, e $\Pi(G)$ a variável aleatória (independente de $\xi$) que descreve o grafo obtido do grafo G após renomear os vértices. Então, para todo grafo $G''$, vale que*

$$Prob[\xi = 1 | \Pi(G_\xi) = G''] = Prob[\xi = 2 | \Pi(G_\xi) = G''] = \frac{1}{2}.$$

A proposição 4.6 nos diz que qualquer processo aleatório com saída em $\{1,2\}$, dado $\Pi(G_\xi)$, imprime $\xi$ com probabilidade exatamente $\frac{1}{2}$. Portanto, dado $G''$, o programa do verificador imprime $\sigma$, tal que $\sigma \neq \tau$, com probabilidade exatamente de $\frac{1}{2}$. Concluímos então que o simulador imprime $\perp$ com probabilidade $\frac{1}{2}$.

Por fim, falta demonstrar que a sequência de tuplas $(x, r, G'', \psi)$, condicionada a não imprimir $\perp$, gerada pelo simulador tem exatamente a mesma distribuição que as tuplas produzidas por uma interação real com o provador. A proposição 4.7 nos garante isso.

**Proposição 4.7.** *Seja $x =< G_1, G_2 > \in ISO$. Então, para toda string $r$, grafo $H$ e permutação $\psi$*

$$Prob\Big(P_{GI}(x) = (x, r, H, \psi)\Big) = Prob\Big(M^*(x) = (x, r, H, \psi) | M^*(x) \neq \perp\Big)$$

*Demonstração da Proposição 4.7.* Seja $m^*(x)$ a variável aleatória que descreve $M^*(x)$ condicionada a não imprimir $\perp$. Primeiramente observemos que ambos $m^*(x)$ e $P_{GI}(x)$ são distribuídos sobre quádruplas da forma $(x, r, \cdot, \cdot)$, com $r \in \{0,1\}^{q(|x|)}$ uniformemente distribuído, dado um polinômio $q(\cdot)$ qualquer. Seja $\upsilon(x, r)$ a variável aleatória que descreve os últimos dois elementos de $P_{GI}(x)$ condicionado ao segundo elemento ser igual a $r$. Similarmente, seja $\mu(x, r)$ a variável aleatória que descreve os últimos dois elementos de $m^*(x)$ condicionado ao segundo elemento ser igual a $r$. Claramente, é suficiente mostrar que $\upsilon(x, r)$ e $\mu(x, r)$ são identicamente distribuídas, para todo $x$ e $r$. Uma vez que $r$ é fixo, a mensagem $\sigma$ enviada por $V^*$ é unicamente definida na entrada comum $x$, fita aleatória $r$ e mensagem na fita de entrada $H$. Denotemos essa mensagem por $v^*(x, r, H)$. Então, o que precisamos demonstrar é que ambos $\upsilon(x, r)$ e $\mu(x, r)$ são distribuições uniformes sobre o conjunto

$$C_{x,r} = \Big\{(H, \psi) \mid H = \psi(G_{v^*(x,r,H)})\Big\}$$

onde $\psi(G)$ é o isomorfismo de $G$.

A prova foge do escopo do trabalho e não é trivial devido ao fato de estar relacionada ao grupo de automorfismos do grafo $G_2$ (i.e., o conjunto de permutações $\pi$ ao qual $\pi(G_2)$ é idêntico, não somente isomorfo, a $G_2$). Por simplicidade provaremos somente um caso especial e o caso geral omitiremos.

Considere o caso especial no qual o grupo de automorfismos do grafo $G_2$ consiste de ser meramente a permutação identidade (i.e., $G_2 = \pi(G_2)$ se e somente se $\pi$ é a permutação iden-

tidade). Nesse caso, $(H, \psi) \in C_{x,r}$ se e somente se $H$ é isomorfo a ambos $G_1$ e $G_2$, e $\psi$ é o isomorfismo entre $H$ e $G_{v^*(x,r,H)}$. Assim, $C_{x,r}$ contém exatamente $|V_2|!$ pares, cada um contendo um diferente grafo $H$ como primeiro elemento, provando a proposição 4.7 para o caso especial. ■

Lembremos que para provar o conhecimento zero perfeito, precisamos mostrar que $P_{GI}(x)$ e $M^*(x)$ são identicamente distribuídas. Porém isso não foi mostrado. No entanto, $P_{GI}(x)$ e $M^*(x)|M^*(x) \neq \perp$ são identicamente distribuídas, e quando $M^*(x) = \perp$ as distribuições são diferentes. Um modo simples de contornar isso é mudando um pouco a definição de conhecimento zero perfeito. Suponha que permitimos o simulador imprimir um símbolo especial que chamaremos de "falha", mas que esse símbolo seja imprimido com probabilidade de no máximo $\frac{1}{2}$. Nesse caso a construção satisfaria a definição.

Falta dizer que o tempo percorrido do simulador é polinomial em $|x|$. Com a definição modificada do modo que descrevemos acima, conseguimos isso com apenas uma iteração.

Assim, podemos concluir que o teorema 4.5 é uma prova com conhecimento zero perfeito (sobre a definição modificada). ■

# 5 *Conclusão*

A classe de linguagens *NP* trivialmente está contida na classe *IP* ($NP \subseteq IP$). No exemplo da seção 3.3, vimos que o problema que (provavelmente) não está em *NP*, o *NONISO*, está em *IP*. Sendo assim a classe de linguagem que admite provas interativas (provavelmente) é maior que *NP*.

Um fato muito mais forte provado ($PSPACE \subseteq IP$) [20] é a igualdade das classes *IP* e *PSPACE*. Acredita-se que *PSPACE* seja uma classe de linguagens muito maior que *NP*, portanto **ao permitirmos que uma prova seja realizada probabilisticamente, estamos aumentando a quantidade de afirmações que um verificador possa ser convencido**.

Um sistema de prova interativo não tem nenhuma aplicação prática a não ser quando é dotado da característica de *conhecimento zero*. Foi provado em [6] que existe um sistema de prova com conhecimento zero para o problema *NP*-completo do grafo 3-colorido. É possível concluir que *todos os problemas em NP tem um sistema de provas com conhecimento zero*, fato extremamente útil para criptografia.

Sistemas de provas interativas de conhecimento zero de linguagens *NP* permitem construir aplicações de segurança extremamente úteis. Alguns exemplos de aplicações, discutidas em [8], são esquemas de identificação, protocolos de eleição eletrônica e protocolos de pagamento por intermédio de moeda eletrônica. Ironicamente essas aplicações de segurança só são úteis enquanto *NP* for diferente de *P*. Também foi discutido em [15], que problemas *NP* são mais seguros do que os *NP*-Completos.

Por fim, a contribuição principal desse trabalho é que ele constituiu na primeira referência em português sobre o assunto.

# *Referências Bibliográficas*


[1] http://www.claymath.org/millennium/, em Setembro de 2006.

[2] Richard Chang, Benny Chor, Oded Goldreich, and et al. The random oracle hypothesis is false. *J. Comput. System Sci.*, 49(1):24–39, 1994.

[3] Thomas H. Cormen, E. Leiserson, Charles, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[4] Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 2005.

[5] Oded Goldreich. *Foundations of cryptography*. Cambridge University Press, Cambridge, 2001. Basic tools.

[6] Oded Goldreich, Silvio Micali, and Avi Wigderson. *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, volume 38. ACM Press, New York, NY, USA, 1991.

[7] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304, New York, NY, USA, 1985. ACM Press.

[8] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. Summer Course "Cryptography and Computer Security" at MIT, 1996–1999, 1999.

[9] Peter Gács and László Lovász. *Complexity of Algorithms*. 1999.

[10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 2001.

[11] Gaurav Jain. *Zero Knowledge Proofs: A Survey*. University of Pennsylvania.

[12] Ker-I Ko. Some observations on the probabilistic algorithms and NP-hard problems. *Inform. Process. Lett.*, 14(1):39–43, 1982.

[13] Yoshiharu Kohayakawa and José Augusto Soares. *Demonstração transparentes e a impossibilidade de aproximação*. Editora do IMPA – Instituto de Matemática Pura e Aplicada, Rio de Janeiro – RJ, julho de 1995.

[14] László Lovász. *Interactive proofs: a new look at passwords, proofs, and refereeing*. Dept. of Computer Science.

[15] P. Mateus. *Análise de sistemas de prova de conhecimento nulo*. Instituto Superior Técnico, 2005.

[16] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995.

[17] Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[18] Jean-Jacques Quisquater, Louis Guillou, Marie Annick, and Tom Berson. *How to explain zero-knowledge protocols to your children*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[19] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System. Sci.*, 4:177–192, 1970.

[20] Adi Shamir. IP=PSPACE. volume 39, pages 869–877, 1992.

[21] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.