Modelo de Computação Probabilístico e Suas Aplicações

Ediana da Silva de Souza, Erica Peters do Carmo

Santa Catarina, Brasil

Abstract

A Teoria da Computação define uma máquina de Turing probabilística como uma máquina de Turing não-determinística que escolhe cada transição de acordo com um processo aleatório. Este artigo apresenta a definição do modelo de computação probabilístico e suas classes de complexidade, bem como a classificação destes algoritmos de acordo com seus objetivos e algumas de suas aplicações. De forma a conceituar os tópicos propostos, a metodologia adotada para a elaboração deste trabalho caracteriza-se como uma pesquisa bibliográfica. Como resultado, entende-se que alguns problemas podem ser solucionados de maneira eficiente por algoritmos probabilísticos, isto demonstra a importância do estudo acerca do tema.

Keywords: Máquina de Turing Probabilística, Teoria da Computação, Computabilidade, Não-Determinismo

1. Introdução

Algoritmos probabilísticos são algoritmos que utilizam em seu processamento o resultado de algum evento aleatório. Antes vistos apenas como uma ferramenta para a Teoria dos Números, hoje, esses algoritmos são utilizados em diversas áreas de aplicação principalmente devido à duas de suas características principais: simplicidade e velocidade [1].

Historicamente, define-se que a utilização desses algoritmos teve início com os métodos Monte Carlo utilizados na análise numérica e estatística. Já em Teoria da Computação, [2], [3], [4] e [5] fazem as primeiras menções às máquinas de Turing probabilísticas.

Com o intuito de apresentar uma pesquisa bibliográfica relacionada à computação probabilística, este trabalho explana acerca da definição das máquinas de Turing probabilísticas, suas classes de complexidade, classificações e aplicações.

Preprint submitted to VII SUCU *21 de Junho de 2019*

Assim, a seção 2 apresenta a definição de máquina de Turing probabilística enquanto a seção 3 explana sobre as classes probabilísticas. A seção 4 apresenta a classificação dos algoritmos probabilísticos e a seção 5 aborda duas de suas aplicações. Finalmente, a seção 6 apresenta as considerações finais.

2. Definição

Em Teoria da Computação, a primeira menção à máquina de Turing probabilística aparece em [2]. Posteriormente, o termo foi explorado e definido em [3], [4] e [5]. Nesse sentido, o presente trabalho apresenta três definições de máquinas de Turing probabilísticas de acordo com os primeiros trabalhos exploratórios acerca do termo.

DEFINIÇÃO I. De acordo com Rabin em [3], um autômato probabilístico (p.a.) sobre o alfabeto Σ é um sistema $\mathfrak{V} = \{S, M, s_0, F\}$ onde $S = \{s_0, ..., s_{\mathrm{n}}\}$ é um conjunto finito de estados, s_0 é o estado inicial, F é o conjunto de estados finais e M é uma função S X Σ em $[0, 1]^{\mathrm{n}+1}$ (tabela de transições possíveis) onde para cada transição $(s, \sigma) \in S$ X Σ:

$$M(s, \sigma) = (p_0(s, \sigma), ..., p_{\mathrm{n}}(s, \sigma)), \text{ com } 0 \leqslant p_{\mathrm{i}}(s, \sigma) \text{ e } \sum_i p_{\mathrm{i}}(s, \sigma) = 1.$$

Assim, cada transição (s, σ) pode chegar a $i = n + 1$ próximos estados com probabilidade p_{i}. Assume-se ainda que essas probabilidades permanecem fixas independente da entrada da máquina e do tempo de computação.

DEFINIÇÃO II. Gill em [5] define uma máquina de Turing probabilística como uma máquina de Turing não determinística [6] com estados adicionais chamados de estados de arremesso-de-moeda. Para cada um desses estados, a função de computação da máquina define dois próximos estados possíveis. Assim, a computação da máquina depende da sua entrada e do resultado dos arremessos de moeda que em cada passo não determinístico decidem entre os dois próximos estados. De maneira geral, para cada entrada x, uma máquina probabilística M produz y como saída com probabilidade $Pr\{M(x) = y\}$.

Assim como [7], este trabalho utiliza a definição de Gill [5] para apresentar as seguintes definições relacionadas aos erros e probabilidades das máquinas probabilísticas de Turing:

Seja b um ramo da computação não determinística de M sobre uma entrada w e k o número de passos do tipo arremesso-de-moeda que ocorrem no ramo b, a probabilidade de aceitação do ramo é:

$$Pr[b] = 2^{\text{-k}}$$

e a probabilidade da entrada w ser aceita por M é:

$$Pr[M \text{ aceita } w] = \sum Pr[b].$$

Define-se ainda que para uma máquina de Turing probabilística reconhecer uma linguagem, ela deve aceitar todas as palavras pertencentes à linguagem e rejeitar todas as palavras não pertencentes, com uma probabilidade de erro ϵ [7]. Assumindo $0 \leqslant \epsilon < \frac{1}{2}$, M reconhece uma linguagem L com probabilidade de erro ϵ se:

1. $w \in L \implies Pr[M \text{ aceita } w] \geqslant 1 - \epsilon$, e
2. $w \notin L \implies Pr[M \text{ rejeita } w] \geqslant 1 - \epsilon$.

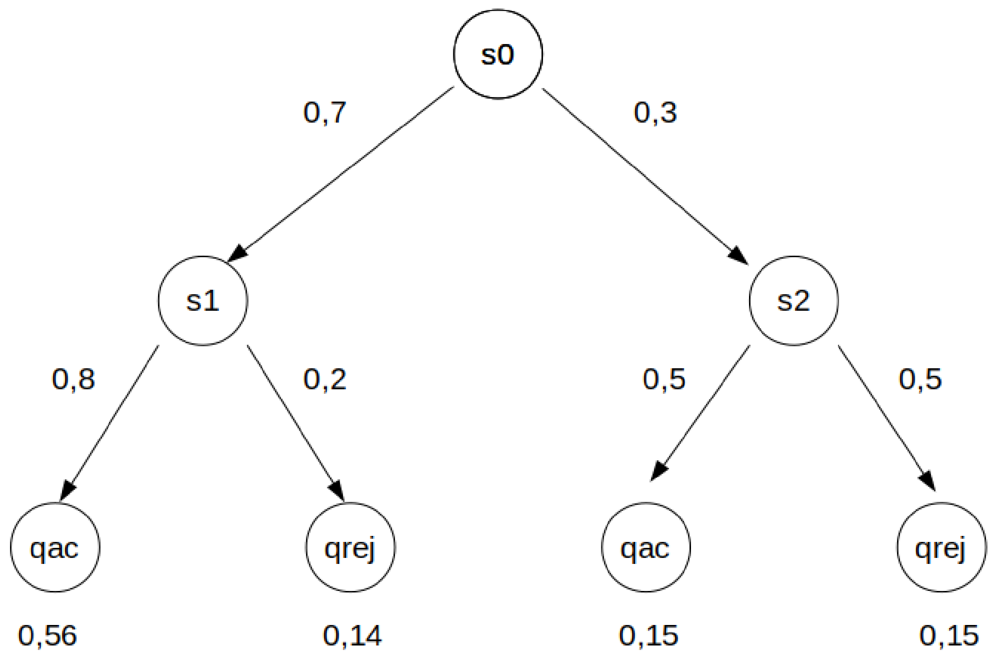
Assim, é possível afirmar que ao simular w em M a probabilidade máxima de obter um resultado errado é de $\frac{1}{2}$.

A Figura 1 apresenta um exemplo de computação de uma máquina de Turing probabilística - sem definir os movimentos do cabeçote e a escrita na fita. Observa-se na figura que s0, s1 e s2 são estados de arremesso-de-moeda e a soma das probabilidades de cada transição originadas nesses estados resulta em 1. Verifica-se que a partir de uma entrada w qualquer, $Pr[M$ aceita $w] = 0,71$. Logo, $w \in L(M)$ e a probabilidade de erro de M é $0,29$. Logo, $L(M) \in BPP$.

Outra questão referente à máquina de Turing probabilística é se seu poder computacional difere-se do poder de um modelo comum de máquina de Turing. [5] e [4] provaram que a resposta para essa questão é negativa e um jeito simples de entender essa prova é a partir de uma outra definição para a máquina de Turing probabilística:

DEFINICAO III. De acordo com Mandrioli et al. [8], é possível entender uma máquina de Turing probabilística como uma máquina de Turing multifita. Além da fita principal de trabalho, a máquina possui uma fita especial preenchida aleatoriamente com bits. Assim, a cada passo da função de transição, a máquina verifica o bit sob o cabeçote da fita especial, move o cabeçote para a direita e faz um movimento de transição de acordo com o valor do bit lido. Em uma mesma transição é possível definir movimentos di-

Figura 1: Computação de uma máquina de Turing probabilística



ferentes para os bits 0 e 1, isso representa um estado de arremesso-de-moeda, onde o valor do bit equivale ao valor obtido aleatoriamente no arremesso de uma máquina de Turing probabilística.

Como explanado em [7], a máquina de Turing multifita tem o mesmo poder computacional de uma máquina de Turing comum. Assim, entende-se que o fator aleatório não adiciona poder computacional, tudo que uma máquina de Turing probabilística computa pode ser computado por uma máquina de Turing normal.

Contudo, o comportamento da máquina de Turing probabilística difere-se dos demais modelos, principalmente no que diz respeito à possibilidade de erro do seu resultado. Esse fator está diretamente relacionado à análise da sua complexidade, explorada na próxima seção.

3. Classes Probabilísticas

Através da análise da complexidade de tempo é possível definir classes de complexidade, sendo este um critério para classificar linguagens. No decorrer desta seção será abordada a definição das classes probabilísticas BPP, RP e ZPP, bem como suas relações com NP e P. A partir deste ponto, o termo algoritmos probabilísticos será também utilizado para referir-se às máquinas de Turing probabilísticas.

3.1. Definição da classe BPP

A classe de complexidade BPP (bounded-error probabilistic polynomial time) compreende os problemas que podem ser resolvidos por máquinas de Turing probabilísticas de tempo polinomial com uma probabilidade de erro de $\frac{1}{3}$ [7]. Isso significa que, dado uma linguagem L, ela está em BPP se e somente se existe uma máquina de Turing probabilística M, tal que [9]:

- Possui um algoritmo de tempo polinomial para o pior caso, ou seja, M é executada em tempo polinomial em toda entrada
- Se $x \in L$, então M aceita x com probabilidade $\geq \frac{2}{3}$
- Se $x \notin L$, então M aceita x com probabilidade $\leq \frac{2}{3}$

A classe BPP contém os algoritmos probabilísticos com erro de dois lados. Isto é, permite que a máquina M produza uma probabilidade de erro tanto se $x \in L$ como se $x \notin L$ [10].

Existem mudanças acerca das restrições do pertencimento de uma linguagem L em BPP, como visto em [11], na qual menciona que a probabilidade de erro nesta classe é de no máximo de $\frac{1}{4}$, sendo que se $x \in L$, então M aceita x com probabilidade $\geq \frac{3}{4}$ e se $x \notin L$, então M aceita x com probabilidade $\leq \frac{1}{4}$. Do mesmo mesmo, [12] refere-se a classe BPP com a probabilidade de erro de $\frac{1}{4}$, sendo que se $x \in L$, pelo menos $\frac{3}{4}$ dos caminhos de computação de M em x irão aceita-lo e se $x \notin L$, pelo menos $\frac{3}{4}$ dos caminhos de computação de M irão rejeita-lo. Por fim, o autor [9] cita que a definição padrão refere-se a máquinas probabilísticas com uma probabilidade de erro de no máximo $\frac{1}{3}$. Essas mudanças sucedem pois a classe pode ser definida com qualquer probabilidade de erro constante estritamente entre 0 e $\frac{1}{2}$, conforme [7] e [13].

3.2. Definição da classe RP

A classe de complexidade RP (*randomized polynomial time*) é definida como sendo a classe de linguagens que são reconhecidas por máquinas de Turing probabilísticas de tempo polinomial em que entradas na linguagens são aceitas com uma probabilidade de, no mínimo, $\frac{1}{2}$ e que entradas que não estão na linguagem são rejeitadas com uma probabilidade de 1 [7]. Diferentemente da classe BPP que possui erros de ambos os lados, apresentando erros tanto se a linguagem aceita ou rejeita tal entrada, a classe RP possui uma característica chamada de erro unilateral. Isto é, quando o algoritmo produz uma resposta *não*, esta é sempre certa, quando produz uma resposta *sim*, esta pode estar errada [7]. Portanto, uma linguagem L pertence à classe RP se e somente se existir uma máquina de Turing probabilística M tal que para todo x $\in \sum^*$ [14]:

- Se $x \in L$, então M aceita x com probabilidade $\geq \frac{1}{2}$
- Se $x \notin L$, então M rejeita x

A fração $\frac{1}{2}$ na definição é arbitrária, visto que o conjunto RP conterá exatamente os mesmos problemas se o valor for substituído por qualquer probabilidade constante diferente de zero e inferior a 1 [15].

3.3. Definição da classe ZPP

A classe de complexidade ZPP (*zero-error probabilistic polynomial time*) é estabelecida como a classe de linguagens de erro bilateral, isto acontece porque a classe é definida como sendo RP $\cup$ coRP, onde possui duas máquinas de Turing probabilísticas, uma que não tem falsos positivos e outra que não tem falsos negativos. Ou seja, podemos executar cada uma das máquinas alternadamente até que uma resposta definitiva possa ser obtida [16]. Portanto, uma linguagem L pertence à classe ZPP se e somente se existir uma máquina de Turing probabilística M tal que para todo x $\in \sum^*$ [14]:

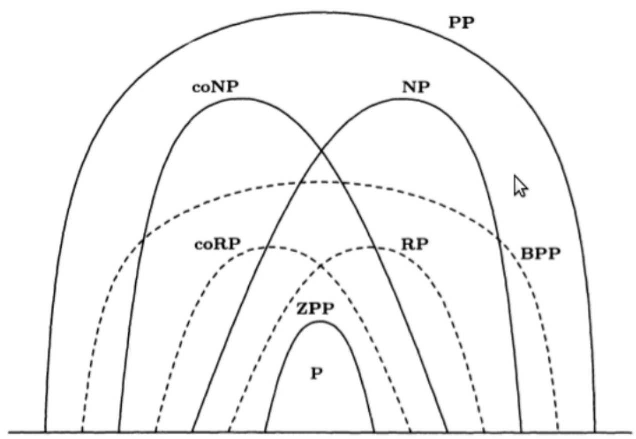
- M retorna que aceita com alta probabilidade de $x \in L$
- M retorna que rejeita com alta probabilidade de $x \notin L$
- M nunca retorna que aceita se $x \notin$ L
- M nunca retorna que rejeita se $x \in L$

Portanto, esta classe não é idêntica a classe de máquina de Turing determinísticas, uma vez que esta ainda possui uma certa indecisão na sua resposta e não é possível saber quando sua execução vai terminar [14].

3.4. Classe probabilísticas e suas relações com NP e P

As classes P e NP são importantes definições em teoria da complexidade. A primeira é definida como sendo a classe de linguagens que são decidíveis em tempo polinomial por uma máquina de Turing determinística [7]. Já a classe NP recebe a definição de ser a classe de linguagens que são decidíveis em tempo polinomial por uma máquina de Turing não determinística ou que são verificadas em tempo polinomial por uma máquina de Turing determinística [7]. Na figura 2 é apresentada a relação destas classes com BPP, RP e ZPP. Desta figura é possível perceber que P $\subset$ ZP $\subset$ RP $\subset$ BPP, bem como RP $\subset$ NP e BP $\subset$ NP. A classe de complexidade PP não será abordada neste trabalho.

Figura 2: Relação Entre as Classes de Complexidade



Esta subseção tem como foco a classe BPP e as relações com P e NP, visto que esta engloba as outras descritas. Sabe-se que P $\subset$ BPP, uma vez que uma máquina de Turing determinista é um caso especial de uma máquina de Turing probabilística [10]. Porém, a questão se BPP = P ainda é uma questão aberta na teoria da complexidade, julgada tão importante quanto descobrir se NP = P [17]. Acredita-se que a igualdade entre BPP e P procede, e se comprovado, revelará que todo algoritmo probabilístico pode ser substituído por um algoritmo determinístico [17]. Caso provado que BPP = P, então teríamos que BPP $\subset$ NP, visto que P $\subset$ NP. Sobre a relação da classe BPP e NP, não sabe-se sobre sua relação de continência ou igualdade. As referências [5], [14] e [18] acreditam que seja improvável que NP $\subset$ BPP. Portanto, acredita-se que nem todos os problemas do NP admitem algoritmos aleatórios eficientes. Sendo assim, a busca por algoritmos aleatórios eficientes para problemas NP-completos é uma questão de pesquisa.

4. Classificação dos Algoritmos Probabilísticos

Os algoritmos probabilísticos podem ser divididos em dois principais grupos: Monte Carlo e Las Vegas. Nesta seção será apresentada suas definições e um exemplo acerca de suas implementações.

4.1. Definições

Os algoritmos Monte Carlo podem ser divididos em algoritmo com probabilidade de erro bilateral e probabilidade de erro unilateral, sempre executados em tempo polinomial, sendo eles pertencentes a classe de complexidade BPP e RP, respectivamente [19]. Isto é, o primeiro descreve um algoritmo que pode retornar respostas erradas tanto se a resposta correta for falsa ou verdadeira. Já o segundo, têm-se que se a resposta correta for falsa, o algoritmo indica isso, mas ele pode responder incorretamente em alguns casos em que a resposta correta é verdadeira [11].

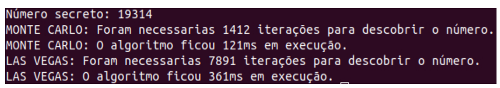
Os algoritmos Las Vegas são aqueles que sempre dão a solução correta, mas podem variar seu tempo de execução para uma determinada entrada, não executando sempre em tempo polinomial. Estes são pertencentes a classe de complexidade ZPP [19].

A escolha de qual algoritmo utilizar depende do objetivo da aplicação. Se é preciso garantia de tempo, em todas as execuções, o algoritmo Monte Carlo é mais indicado. Se há necessidade de se estar sempre diante da resposta correta, o uso do Las Vegas vem a ser o mais apropriado [20]. É possível obter a construção de um algoritmo de Las Vegas para um problema na qual existe algoritmo de Monte Carlo e vice-versa, porém, nem sempre isto apresenta um desempenho suficientemente interessante [20]. Portanto, é preciso escolher entre a certeza ou o desempenho.

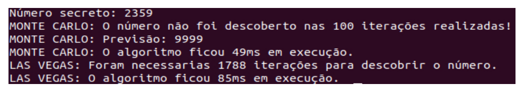
4.2. Implementação

Como forma de retratar a ideia dos algoritmos de Monte Carlo e Las Vegas, [21] criou um simples exemplo na qual busca encontrar um valor secreto gerado aleatoriamente. O código fonte do algoritmo pode ser visto no segundo arquivo apresentado no link: `https://gist.github.com/danielfariati/1535513`. A figura 3 mostra alguns resultados deste algoritmo. Percebe-se que, quanto maior a entrada, mais tempo o algoritmo de Las Vegas ficará executando. Já o algoritmo de Monte Carlo recebe um número limitado de tentativas, mas pode não encontrar a solução correta neste intervalo. Caso não encontre, o algoritmo tenta fazer um cálculo para retornar uma previsão. Este cálculo é dado somente para demonstrar que o Monte Carlo pode responder valores incorretos, visto que o cálculo poderia ser mais aperfeiçoado para responder a previsão com mais precisão, como por exemplo, lidando melhor se o número secreto fosse algum dos extremos, como o número 1.

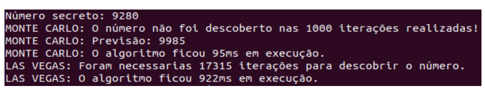
Figura 3: (a) Execução 1, (b) Execução 2, (c) Execução 3 e (d) Execução 4



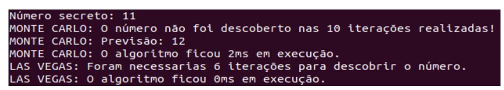
(a)



(b)



(c)



(d)

5. Aplicações

Os algoritmos probabilísticos tem sido utilizados em uma grande variedade de aplicações, principalmente devido à dois dos seus maiores benefícios: simplicidade e rapidez [1].

Com o intuito de apresentar algumas de suas aplicações e com base na explanação em [7], essa seção explora dois algoritmos probabilísticos: o teste de primalidade e o teste de equivalência de programas ramificantes.

5.1. Teste de Primalidade

O problema de determinar se um número é primo ou não sempre foi alvo de pesquisas devido às suas diversas aplicações. Em 2002, foi publicado o artigo [22] que prova que esse problema pertence à classe P ao descobrir um algoritmo determinístico que o resolve em tempo polinomial.

Anteriormente, alguns algoritmos probabilísticos foram propostos para determinar a primalidade de um número. Estes algoritmos tem como base o pequeno teorema de Fermat apresentado a seguir.

TEOREMA. Se p é um número primo e $a \in Z^+$ é não divisível e menor que p, então $a^{\mathrm{p-1}} \equiv 1 \pmod{p}$.

O problema em utilizar o teorema como um teste de primalidade é que existem certos números compostos, chamados números de Carmichael, que comportam-se como primos e passam nos testes para todos os valores de a. Assim, o teste de Fermat é considerado um algoritmo probabilístico: se p não passa no teste para algum valor de a, p certamente é um número composto (não primo); se p passa no teste para todos os valores de a analisados, p quase certamente é primo (mas pode ser um número de Charmichael), por isso, dizemos que p possui um caráter pseudoprimo.

Conforme descrito em [7], uma máquina de Turing probabilística para testar o caráter pseudoprimo é apresentada a seguir:

MTP = Sobre a entrada p

1. Selecione $a_1, a_2, ..., a_{\mathrm{k}}$ tal que $a \in Z+$ é menor e não divisível por p.
2. Calcule $a_{\mathrm{i}}{}^{\mathrm{p - 1}}$ para cada i.
3. Se todos os resultados forem 1, aceite. Caso contrário, rejeite.

É possível observar que se p for pseudoprimo (um número primo ou um número de Charmichael), ele passa em todos os testes e é aceito. Se p não for pseudoprimo, ele passa em cada teste com probabilidade de $\frac{1}{2}$, logo, a probabilidade de que ele passe em todos k testes é $2^{\mathrm{-k}}$ [7].

Para resolver o problema dos números de Charmichael, um outro teste é adicionado ao algoritmo. Sabe-se que o número 1 possui apenas duas raízes quadradas (1 e -1) módulo de um primo p qualquer. Para os números compostos, incluindo os números de Charmichael, 1 possui quatro ou mais raízes quadradas módulo p. Por isso, é possível criar um algoritmo de primalidade que encontra a raiz quadrada de 1 e verifica se ela é 1 ou -1, se não for, determina que o número não é primo. Para encontrar cada raiz o cálculo feito é $a^{\mathrm{(p - 1)/2}}$ mod p. Se o valor encontrado for 1, é possível dividir o expoente por 2 até encontrar um valor diferente. Se o valor encontrado for -1, o número e primo, caso contrário, o número é composto [7].

MTP = Sobre a entrada p

1. Se p for par, aceite caso $p = 2$ e rejeite caso contrário.
2. Selecione $a_1, a_2, ..., a_{\mathrm{k}}$ aleatoriamente tal que $a \in Z+$ é menor e não divisível por p.
3. Para cada i faça:
4. Calcule $a_{\mathrm{i}}{}^{\mathrm{p - 1}}$ mod p e rejeite se for diferente de 1.
5. Faça $s = p - 1$ com s ímpar e t uma potência de 2.

6. Calcule a sequência $a_i^{s \cdot 2^0}$, $a_i^{s \cdot 2^1}$, ..., $a_i^{s \cdot 2^h}$ módulo p.

7. Se algum resultado da sequência não for 1, encontre o último elemento diferente de 1 e verifique se ele é -1. Se não for, rejeite.

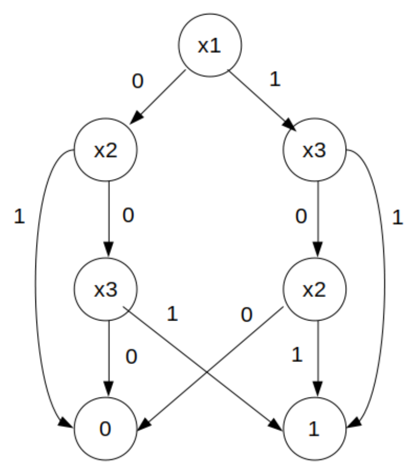
8. Se os testes passaram até aqui, aceite.

Sipser em [7] demonstra ainda que se p é primo, o algoritmo o identifica corretamento com probabilidade 1. E se p é composto, a probabilidade de erro é menor que 2^{-k}. Assim, esse algoritmo pertence à classe BPP.

Sobre o problema da primalidade de um número, pode-se definir sua maior aplicação dentro da Criptografia, principalmente nos sistemas baseados em chaves públicas e privadas. A Criptografia RSA, por exemplo, gera suas chaves a partir da multiplicação de dois números primos grandes (p e q), assim, garante-se que o processo de fatoração desse número será o mais difícil possível. Os algoritmos para testar o caráter primo de p e q são probabilísticos, sendo os mais conhecidos o teste de Miller Rabin e de Solovay–Strassen [23].

5.2. Equivalência de programas ramificantes

Figura 4: Programa ramificante

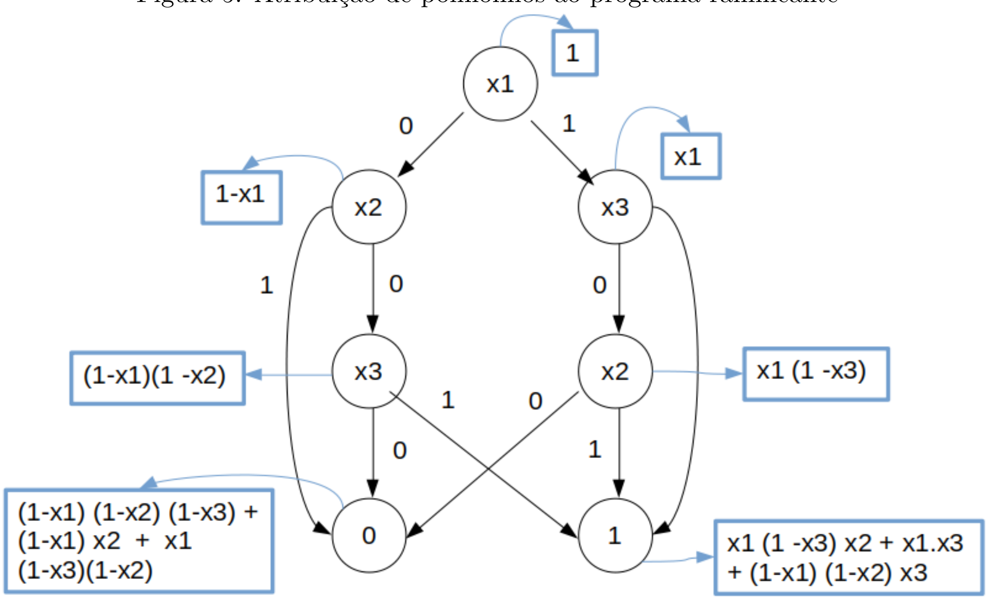


Um programa rafimifante (*branching program*) é um grafo direcionado acíclico cujos vértices representam variáveis, exceto por dois nós de saída rotulados 0 e 1 [7]. Cada nó que representa uma variável possui grau de saída 2, com uma aresta rotulada como 1 e outra rotulada como 0. Esse programa determina uma função booleana ao sair do nó inicial e seguir o caminho indicado por cada aresta de acordo com a valoração das variáveis [24]. A Figura 4 apresenta um exemplo desse tipo de programa.

O problema de determinar a equivalência de dois programas ramificantes tem como objetivo determinar se dados dois programas B1 e B2, eles determinam as mesmas funções.

A prova para esse problema baseia-se em associar um polinômio para cada um dos m nós e arestas dos programas da seguinte forma: a constante 1 é atribuída ao nó inicial; se um x é atribuído a um nó, deve ser atribuído xp à sua aresta de saída com rótulo 1 e $(1 - x)p$ à sua aresta de saída com rótulo 0. Atribui-se também a cada nó a soma das suas arestas de entrada. E o polinômio atribuído ao nó de saída 1 é também atribuído ao próprio programa ramificante redSipser. A figura 5 apresenta a atribuição de polinômios ao programa ramificante apresentado anteriormente.

Figura 5: Atribuição de polinômios ao programa ramificante



A partir dessa atribuição, é possível apresentar o seguinte algoritmo probabilístico:

MTP = Sobre a entrada <B1, B2 >

1. Selecione elementos $a_1, ...a_{\mathrm{m}}$ de um conjunto F com no mínimo $3.m$ elementos.

2. Calcule o valor dos polinômios p1 e p2 em cada um dos m elementos.

3. Se $\mathrm{p1}(a_1, ...a_{\mathrm{m}}) = \mathrm{p2}(a_1, ...a_{\mathrm{m}})$, aceite. Caso contrário, rejeite.

Sipser em [7] demonstra que caso os programas B1 e B2 sejam equivalentes, MTP sempre aceita a entrada. E caso contrário, MTP rejeita com uma probabilidade de erro de $\frac{1}{3}$. Assim, este algoritmo pertence à BPP.

Os programas ramificantes possuem sua maior aplicação dentro de sistemas computacionais relacionados ao Desenho Assistido por Computador (CAD). Principalmente no que diz respeito à verificação de circuitos combinacionais e teste de padrões [24].

6. Considerações Finais

Este artigo apresentou uma pesquisa bibliográfica com o objetivo de conceituar a máquina de Turing probabilística e suas classes de complexidade, bem como relatar a classificação dos algoritmos probabilísticos e suas aplicações. Foi possível constatar que muitas classes de complexidade são definidas em torno da máquina de Turing probabilística e que ainda há questões abertas acerca destas classes, sendo uma considerada igualmente importante a questão de N = NP, dada por descobrir se BPP = P. A utilização dos algoritmos probabilísticos é vista como um grande potencial, dado que com algoritmos probabilísticos é possível resolver problemas que ainda não descobriu-se uma maneira possível de resolver com algoritmos determinísticos. Portanto, há uma forte tendência deste tipo de computação continuar sendo gradativamente discutida e aprimorada.

Referências

[1] R. Motwani, Randomized algorithms, ACM Computing Surveys (CSUR) 28 (1996) 33 – 37.

[2] K. D. Leeuw, E. F. Moore, C. E. Shannon, N. Shapiro, Computability by probabilistic machines, Automata Studies, Annals of Mathematics Studies (1956) 183 – 212.

[3] M. O. Rabin, Probabilistic automata, Information and Control 6 (1963) 230 – 245.

[4] E. S. Santos, Probabilistic turing machines and computability, Proceedings of the American Mathematical Society 22 (1969) 704 – 710.

[5] J. T. Gill, Computational complexity of probabilistic turing machines, SIAM J. Comput. 6 (1977) 675 – 695.

[6] A. M. Turing, On computable numbers, with an application to the entscheidungs problem, Proceedings of the London Mathematical Society (1937) 230 – 265.

[7] M. Sipser, Introdução à Teoria da Computação, Cengage Learning, 2016.

[8] D. Mandrioli, C. A. Furia, M. Rossi, A. Morzenti, Modeling Time in Computing, Springer, 2012.

[9] O. Goldreich, Computational Complexity: A Conceptual Perspective, Cambridge University Press, 1ª edition, 2008.

[10] Computational Complexity: A Modern Approach, http://theory.cs.princeton.edu/complexity/bppchap.pdf, 2007. Acessado em: 19 jun. 2019.

[11] Randomized Algorithms, https://www.cs.cmu.edu/afs/cs/academic/class/15859-f04/www/scribes/lec2.pdf, 2019. Acessado em: 19 jun. 2019.

[12] Bounded Probabilistic Polynomial, https://www.csie.ntu.edu.tw/ lyuu/complexity/2015/2015 2015. Acessado em: 19 jun. 2019.

[13] D.-Z. Du, K.-I. Ko, Theory of Computational Complexity, John Wiley Sons, 1ª edition, 2008.

[14] H. Hempel, Randomized algorithms and complexity theory, Journal of Universal Computer Science 12 (2006) 746–761.

[15] I. Kononenko, M. Kukar, Machine Learning and Data Mining, Woodhead Publishing, 2007.

[16] T. A. Rocha, Complexidade descritiva de classes probabilísticas de tempo polinomial e das classes p e npconp através de lógicas com quantificadores generalizados de segunda ordem, Dissertação(Ciência da Computação) - Universidade Federal do Ceará, Departamento de Computação (2014).

[17] Complexity Classes, https://brilliant.org/wiki/complexity-classes/, 2019. Acessado em: 19 jun. 2019.

[18] Computational Complexity, https://people.eecs.berkeley.edu/ luca/cs278-04/notes/lecture08.pdf, 2002. Acessado em: 19 jun. 2019.

[19] D. Antonova, D. Kunkle, Theory of randomized computation: A survey for csg714, Northeastern University, Boston (2005).

[20] C. de Figueiredo, G. da Fonseca, M. Lemos, V. de Sá, Computational Complexity: A Conceptual Perspective, Associação Instituto Nacional de Matemática Pura e Aplicada - IMPA, 2007.

[21] Algoritmos Las Vegas - Monte Carlo, https://gist.github.com/danielfariati/1535513, 2011. Acessado em: 19 jun. 2019.

[22] M. Agrawal, N. Kayal, N. Saxena, Primes is in p, Annals of Mathematics (2004) 781 – 793.

[23] Notes on primality testing and public key cryptography part 1: Randomized algorithms miller–rabin and solovay–strassen tests, http://www.cis.upenn.edu/ jean/RSA-primality-testing.pdf, 2019. Accessed: 2019-05-19.

[24] I. Wegener, Branching Programs and Binary Decision Diagrams: Theory and Applications, 1987.