DEMONSTRAÇÕES TRANSPARENTES E A IMPOSSIBILIDADE DE APROXIMAÇÕES

Yoshiharu Kohayakawa

José Augusto R. Soares

Endereço dos autores:

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
Rua do Matão 1010
Cidade Universitária
05508–900 São Paulo, SP

Endereços eletrônicos: `<yoshi@ime.usp.br>`, `<jose@ime.usp.br>`

PREFÁCIO

Uma linha de pesquisa em teoria da computação culminou recentemente com dois teoremas impressionantes. Informalmente falando, podemos enunciar estes resultados da seguinte maneira:

(i) Qualquer teorema dedutível em um sistema formal, como por exemplo a teoria dos conjuntos de Zermelo–Fraenkel ou a aritmética de Peano, admite uma demonstração em um novo sistema em que a sua correção pode ser verificada através do escrutínio de uma quantidade de caracteres não maior que uma *constante universal*.

(ii) A obtenção de uma solução aproximada para uma vasta gama de problemas de otimização combinatória é tão difícil quanto a obtenção de suas soluções exatas. De fato, estes problemas não admitem esquemas eficientes de resolução aproximada, a menos que $\mathrm{P} = \mathrm{NP}$.

O primeiro resultado acima é, sem dúvida, fascinante, e chega a parecer até um tanto paradoxal. Por outro lado, os leitores familiarizados com a teoria da complexidade computacional verão em (ii) um resultado comparável aos resultados clássicos de Cook, Levin, e Karp do começo da década de 70 em termos de impacto à teoria da computação: de fato, com este resultado a teoria P–NP é estendida ao âmbito muito mais genérico dos problemas de aproximação.

Surpreendentemente, os teoremas acima são intimamente ligados. Esta conexão inesperada entre (i) e (ii) e, mais geralmente, os avanços da teoria da complexidade computacional que levaram aos resultados acima são os nosso temas centrais.

Objetivamos com este pequeno texto dar uma introdução elementar aos assuntos aqui discutidos. Mantemos os pré-requisitos a um mínimo, e salientamos os aspectos menos técnicos desta área. Nem sempre discutimos em detalhe os resultados mais refinados, ou damos as demonstrações mais técnicas; entretanto, o conhecimento do material apresentado neste texto é, esperamos, mais que o suficiente para que os leitores interessados possam continuar seus estudos nesta linha de pesquisa. Aqueles que decidirem se aprofundar nesta área encontrarão aqui as referências mais importantes à literatura, que, aliás, cresce vigorosamente nesta direção. Cada capítulo termina com notas bibliográficas, onde fazemos uma breve discussão da literatura relevante.

Um tratado que desse uma visão global fiel de toda a área da teoria moderna da complexidade computacional seria bastante extenso—se o leitor puder usar estas notas como um ponto de partida, ou se pudermos com este texto interessar o leitor nesta área extremamente excitante de pesquisa, teremos atingido o nosso objetivo.

Agradecemos a participação entusiástica dos nossos colegas Paulo Feofiloff, Arnaldo Mandel e Claus Matsushigue numa série de seminários de estudo sobre o assunto abordado neste texto e à Coordenação do *20º Colóquio Brasileiro de Matemática* pela oportunidade de uma ampla divulgação.

Finalmente, é com prazer que agradecemos o apoio da FAPESP (Proc. 93/0603–1) e do CNPq (Projeto ProComb do ProTeM–CC, Fase II; Proc. 300334/93–1 e Proc. 300805/94–2).

Este texto foi preparado com $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX, o sistema LaTeX da *American Mathematical Society*, com o estilo `amsbook`.

Yoshiharu Kohayakawa
José Augusto Soares
Abril de 1995

O segundo autor dedica este livro à Cássia.

CONTEÚDO

Lista de Figuras

CAPÍTULO I

INTRODUÇÃO

O objetivo deste capítulo é o de fazer um apanhado geral dos tópicos de que trataremos neste texto. Seremos aqui informais, com o objetivo de tornar esta discussão inicial numa introdução descontraída aos temas centrais destas notas.

Dois são os temas principais que estudaremos neste texto. Estes temas podem ser classificados dentro de uma área fundamental da teoria da computação, a teoria da *complexidade computacional*. Brevemente falando, estamos nesta área interessados em estudar problemas computacionais do ponto de vista de seu custo de resolução. Resultados clássicos da lógica afirmam que certos problemas *não* podem ser resolvidos algoritmicamente. Temos em mente, por exemplo, o bem-conhecido 'problema da parada'. O que nos interessa aqui são os problemas que *podem* ser resolvidos algoritmicamente, e a pergunta principal concerne à existência de algoritmos *eficientes* para resolvê-los.

Uma série de resultados impressionantes dentro da teoria de complexidade culminaram recentemente com uma nova caracterização de uma classe importante de problemas computacionais. Esta caracterização é o primeiro tema deste texto. O nosso segundo tema versa sobre o impacto surpreendente que este resultado tem sobre a solubilidade de uma vasta gama de problemas de otimização. Como uma conseqüência deste resultado, provou-se que muitos problemas clássicos de otimização combinatória não admitem resoluções algorítmicas eficientes, mesmo que aproximadas, a menos que a conjectura fundamental da teoria da computação conhecida como ‘P $\neq$ NP’ não seja correta. Em outras palavras, provou-se que, para muitos problemas de otimização combinatória, mesmo achar uma solução aproximada é tão difícil quanto achar uma solução exata.

1. Discussão Inicial

Seguindo a tradição nesta área, começamos a nossa discussão com uma fábula onde os personagens principais são o Rei Artur e Merlin.

1.1. Casando seus cavaleiros e donzelas. Um dia, o Rei Artur, em ótimo estado de espírito, tem a seguinte idéia. ‘Tenho na corte 150 cavaleiros e 150 donzelas. Por que não casá-los e ter então 150 casais?’ Sendo o Rei alguém de bom senso, ele pede que as donzelas elaborem, cada uma, uma lista de ‘parceiros aceitáveis’. O Rei compromete-se então a achar 150 casamentos que respeitem a preferência das donzelas. Nenhuma donzela terá como esposo um cavaleiro que não seja de seu agrado.

Vendo as 150 listas entregues a ele, o Rei Artur sente imediatamente uma dor de cabeça—o problema de achar os tais 150 casamentos parece muito complicado. Aqui entra na nossa estória o nosso segundo personagem. Merlin, o mago todo-poderoso, é chamado pelo Rei Artur para resolver este problema: nada é muito complicado para Merlin; ele pode, em um piscar de olhos, descobrir os tais 150 casamentos. Basta, pois, verificar se uma das 150! possibilidades serve.

Merlin pisca um olho, verifica que nenhuma das 150! bijeções serve, e assim informa ao Rei Artur. O Rei, tendo sempre um pouco de desconfiança de Merlin, diz: ‘Como? Impossível de se achar os meus 150 casamentos? Não pode ser; Merlin deve ter se esquecido de analisar alguma das possibilidades’. O Rei Artur é muito paciente, mas também muito ocupado. Ele não tem tempo de verificar todas as $150! = 5.7\ldots\times 10^{262}$ possibilidades ele mesmo.[1] O que deve Merlin fazer para convencer o Rei Artur de que de fato *nenhuma* destas bijeções é uma solução ao problema? Merlin parece estar em apuros.

[1]Verificando uma bijeção por segundo, ele levaria $1.8\ldots\times 10^{255}$ anos. A idade estimada do universo é da ordem de 10^{22} anos, e o número total de prótons no mundo conhecido é da ordem de 10^{79}.

Entretanto, lembrando-se repentinamente de suas aulas de teoria dos grafos da sua academia para magos, Merlin percebe o que fazer: basta usar o teorema de König! De fato, este teorema implica que tais 150 casamentos são possíveis se, e *somente se*, para qualquer subconjunto S de donzelas, o número de elementos na união das listas de parceiros aceitáveis das donzelas em S tem pelo menos $|S|$ cavaleiros. Note que, de fato, se tal condição não for verificada, isto é, se houver um subconjunto de donzelas cuja união das listas correspondente tem *menos* que $|S|$ elementos, então não podemos esperar casar todas estas donzelas respeitando as suas listas de preferência. Tal conjunto pode ser considerado como uma *obstrução* ao nosso objetivo.

A observação importante é que o teorema de König diz também que, por outro lado, se os 150 casamentos procurados não existem, *então existe uma tal obstrução S*. Como o Rei Artur pode facilmente julgar se um dado subconjunto S de donzelas é ou não uma obstrução, basta a Merlin fornecer ao Rei Artur uma tal obstrução S para ele convencer o nosso Rei de que, de fato, tais 150 casamentos não existem.

1.2. Os cavaleiros à mesa de jantar. O Rei Artur é forçado a se esquecer de sua ótima idéia de melhorar seu reino casando suas 150 donzelas com seus 150 cavaleiros. Entretanto, seu reino pode ser, percebe ele, melhorado de outro modo. Ao se sentarem à mesa para os seus importantes banquetes, os 150 cavaleiros comportam-se de forma menos que elegante quando a disposição dos lugares não é boa: dois cavaleiros que não se dão bem que por acaso fiquem em lugares adjacentes têm a forte tendência de acertarem suas diferenças antes do fim do jantar, de formas um tanto violentas (omitimos os detalhes). Como o único tipo de sangue que o nosso Rei gosta de ver nestes banquetes é aquele que vem com o seu bife mal passado, ele tem uma idéia. Por que não achar uma disposição de lugares para os seus 150 cavaleiros à mesa de forma que apenas cavaleiros amigos sentem-se um ao lado do outro?

Naturalmente, esta é uma nova tarefa para Merlin. O mago todo-poderoso é chamado à presença do Rei Artur, e é-lhe dado a nova incumbência. Merlin pisca um olho, verifica as 150! possíveis disposições de lugares e descobre que, novamente, o Rei Artur deu-lhe uma tarefa impossível. Sabendo que o Rei não aceitará sua palavra apenas, ele pesquisa em suas notas de aula o que a teoria dos grafos tem a dizer sobre este problema. Merlin fica contente ao descobrir que este problema nada mais é que decidir se um certo grafo é hamiltoniano.

O grafo G em questão tem 150 vértices, um para cada cavaleiro, e ligamos dois deles com uma aresta se e só se os cavaleiros correspondentes aceitam-se mutuamente como vizinhos à mesa de jantar. A solução do problema é simplesmente um *circuito hamiltoniano* neste grafo, uma seqüência circular de vértices, contendo todos os 150 vértices de G, e tal que vértices adjacentes *na seqüência* são sempre adjacentes *no nosso grafo*.

Entretanto, Merlin fica surpreso ao saber que não se conhece uma boa caracterização de grafos hamiltonianos; nada como o resultado de König foi até hoje provado. Em outras palavras, não se conhece uma obstrução simples que necessariamente está presente em todo grafo que não contém um circuito hamiltoniano. Desta vez Merlin está realmente em apuros.

1.3. Boas caracterizações. Vimos na discussão acima que os dois problemas levantados pelo Rei Artur parecem ter, pelo menos dentro da teoria dos grafos, dois *status* diferentes. Na verdade, a teoria da complexidade computacional indica que estes dois problemas são intrinsecamente diferentes. Passemos a tratar os dois problemas do Rei Artur na linguagem da teoria dos grafos.

O primeiro problema corresponde a achar um emparelhamento perfeito em um certo grafo bipartido. De fato, consideremos B, um grafo bipartido com bipartição $V(B) = X \cup Y$, onde X corresponde ao conjunto das 150 donzelas e Y ao dos 150 cavaleiros. A lista de possíveis parceiros de cada donzela corresponde no grafo B ao conjunto de vizinhos do vértice representando esta donzela. Achar os 150 casamentos do nosso Rei corresponde a achar um *emparelhamento perfeito* em B, *i.e.*, um conjunto de arestas de B que incide sobre cada vértice exatamente uma vez. Por outro lado, como discutido acima, o segundo problema do Rei Artur corresponde a encontrar um circuito hamiltoniano em um certo grafo G. Consideremos por ora apenas os problemas de decidir se os objetos procurados, *i.e.*, o emparelhamento perfeito e o circuito hamiltoniano, existem.[2] Caso estes objetos existam, Merlin

[2]Tais problemas, cujas respostas são ou SIM ou NÃO, são conhecidos como 'problemas de decisão'.

tem uma forma muito fácil de convencer o Rei deste fato. Basta exibir um tal objeto explicitamente, e o Rei simplesmente verificará se de fato o objeto apresentado é uma solução ao problema em questão. Em outras palavras, se a resposta ao problema de decisão for SIM, há um *certificado* 'simples' para esta resposta, que pode ser verificado eficientemente.[3]

Caso um certo grafo bipartido B como acima não contenha um emparelhamento perfeito, Merlin pode exibir ao Rei Artur uma obstrução, cuja existência nos é garantida pelo teorema de König, e a inexistência da solução fica provada. Por outro lado, no caso do problema de decidir se um dado grafo G contém um circuito hamiltoniano, se a resposta for negativa, não se sabe como 'provar' este fato através de um 'certificado' eficientemente verificável. Em resumo, a resposta negativa ao primeiro problema pode ser certificada de forma simples, mas não se sabe, e suspeita-se que não seja possível, certificar a resposta negativa ao segundo problema.

Problemas de decisão que têm a característica de que a resposta SIM pode ser eficientemente certificada são conhecidos como problemas da classe NP. Por outro lado, se a resposta a um problema for NÃO e esta resposta puder ser certificada eficientemente, então dizemos que este problema pertence à classe coNP. Assim, o problema de decidir se um dado grafo bipartido contém um emparelhamento perfeito pertence tanto à classe NP como à classe coNP. O problema de decidir se um circuito hamiltoniano existe em um dado grafo pertence à classe NP, mas suspeita-se que ele não pertença à classe coNP.

Como tanto a existência como a inexistência de emparelhamentos perfeitos em grafos bipartidos admitem certificados eficientemente verificáveis, ou, na linguagem introduzida acima, o problema de decisão correspondente está tanto em NP como em coNP, dizemos que a existência destes objetos admite uma *boa caracterização*. Este conceito foi introduzido na década de 60, nos primórdios da otimização combinatória e da teoria de complexidade.

Um problema de aparência inocente, mas cuja solução teria conseqüências profundas em teoria da computação, é o de decidir se a

[3]Em nossa fábula, o Rei Artur representa um ente que está disposto a executar procedimentos eficientes, enquanto que Merlin representa um ente capaz de executar *qualquer* procedimento computacional.

existência de circuitos hamiltonianos admite uma boa caracterização.

1.4. Efetividade e eficiência. Na discussão até o momento, temos usado o termo *eficiente* sem maiores discussões. Devemos neste ponto fazer alguns comentários a esse respeito. Na década de 30, a lógica estava descobrindo a existência de problemas que não podiam ser resolvidos 'algoritmicamente' ou 'efetivamente'. O mais famoso deles é, talvez, o *Entscheidungsproblem* de Hilbert: dado um sistema formal e uma afirmativa dentro deste sistema, pode esta afirmativa ser provada dentro deste sistema?

Os problemas nos quais a teoria da computação tem interesse são essencialmente sempre efetivamente solúveis. Tipicamente, o conjunto das possíveis soluções para uma dada 'instância' do problema é finito e bem conhecido, e uma busca exaustiva neste espaço das possibilidades constitui um processo efetivo de se resolver o problema em questão com aquela particular entrada. Entretanto, o espaço das possibilidades quase sempre cresce de forma muito rápida com o 'tamanho' da instância a ser resolvida. Por exemplo, se o Rei Artur tivesse apenas 5 cavaleiros, o problema do 'banquete pacífico' poderia ser resolvido pelo próprio Rei com um pouco de paciência, mas, como vimos, com os seus 150 cavaleiros, uma tentativa ingênua de se achar um circuito hamiltoniano é fadada ao fracasso, dado o grande número de possibilidades a serem verificadas. Esta explosão do espaço das possibilidades é típico em problemas computacionais de interesse.

Em teoria da computação, a noção aceita de 'eficiência' é a seguinte: dizemos que um algoritmo que resolve um problema é eficiente se o tempo do qual este algoritmo precisa para resolver uma instância de 'tamanho' n é limitado por algum polinômio em n. Isto é, se existe alguma constante k tal que o tempo de computação para qualquer entrada de tamanho n é limitado por n^k para n suficientemente grandes. Aqui, o tamanho da instância é o número de bits que são necessários para se codificar a entrada em algum sistema 'sensato' de codificação, e medimos o tempo usado pelo algoritmo de alguma forma 'sensata'.[4] Esta noção de eficiência não deve ser interpretada como um critério prático; um algoritmo polinomial não é necessariamente utilizável em aplicações reais. Este critério tem sido, entretanto, muito frutífero do

[4]Definições precisas serão dadas no Capítulo II.

ponto vista teórico. A classe dos problemas para os quais existem algoritmos polinomiais é conhecida como a classe P.

No que segue, usamos o termo 'eficiente' para significar 'polinomial'. Em particular, na nossa fábula, admitimos que o Rei Artur é capaz de executar procedimentos polinomiais apenas, enquanto que Merlin é capaz de executar qualquer procedimento efetivo.

2. Verificação Aleatória de Certificados

2.1. Provas interativas. Nesta seção voltamos à nossa fábula. Temos agora Merlin tentanto descobrir como convencer o Rei Artur de que não existe uma disposição pacífica de seus 150 cavaleiros. Vimos na Seção 1.3 que não se sabe se a *inexistência* de circuitos hamiltonianos admite um certificado facilmente verificável, ou, em outros termos, não se sabe se há um meio de Merlin convencer o Rei Artur deste fato.

Uma descoberta recente de múltiplas conseqüências é que se o Rei Artur tiver acesso a um gerador de *números aleatórios*, e se ele estiver disposto a conversar ('interagir') com Merlin por algum tempo, então há uma forma de Merlin convencer o Rei Artur da inexistência de uma solução ao seu problema dos banquetes pacíficos.

Grosseiramente falando, existe um 'jogo' que o Rei Artur e Merlin podem jogar que resulta na vitória certa de Merlin se de fato não existe um circuito hamiltoniano no grafo em questão, como afirmado por ele, e que resulta em vitória *quase* certa do Rei Artur se Merlin está enganado. Esta segunda possibilidade é de natureza probabilística: uma afirmativa mais precisa é que se existe o circuito hamiltoniano em questão, então a probabilidade de o Rei Artur vencer é tão grande quanto se queira; digamos, ela é de pelo menos 99%, sendo o 1% restante devido ao eventual 'azar' do Rei Artur com o seu gerador de números aleatórios. Uma outra característica deste jogo é que ele é um procedimento eficiente: o Rei Artur e Merlin alternam os seus lances, o Rei Artur precisa apenas fazer computações de complexidade polinomial, e o número de lances total no jogo é também polinomial.

Um teorema geral é como segue. Suponha que temos um objeto cuja *existência* admite um certificado verificável eficientemente, *i.e.*, o problema de decisão correspondente pertence a NP. Então a *inexistência* deste objeto admite uma *prova interativa*, ou seja, um jogo com as características acima.

Na verdade, descobriu-se que a classe de afirmativas que admitem provas interativas é extremamente grande. De fato, o resultado final nesta direção é que IP, a classe das afirmativas que admitem provas interativas, é idêntica à classe PSPACE, a classe das afirmativas cujas demonstrações podem ser encontradas por procedimentos que usam espaço apenas polinomial, sem restrição no tempo de computação. Acredita-se fortemente que PSPACE é muito maior que a classe NP, embora esta afirmativa extremamente intuitiva seja ainda hoje uma conjectura.

Concluímos esta seção com a observação seguinte. Introduzimos aqui dois novos elementos em nossa discussão. Em primeiro lugar, estamos agora admitindo que o Rei Artur tem acesso a uma fonte *bits aleatórios*, e, em segundo, que o nosso Rei está disposto a ser convencido de um fato através de uma conversa, ou uma *interação*, com Merlin. O fato surpreendente é que, com esta introdução de um elemento aleatório no processo de verificação de uma afirmativa, a classe de afirmativas verificáveis foi grandemente ampliada.

2.2. Provas verificáveis probabilisticamente. O elemento interativo da seção anterior pode ser, na verdade, eliminado ou pelo menos 'escondido' da seguinte forma. Lembremos que o jogo de Artur–Merlin para convencer o Rei Artur da inexistência de um circuito hamiltoniano no grafo dos seus 150 cavaleiros termina com um número polinomial de lances; em particular, o jogo sempre termina. Assim, Merlin não precisa estar presente durante o jogo: basta que ele deixe escrito em uma 'prova' Π como ele responderia a qualquer lance do Rei Artur. Tal prova Π precisaria ser acessada pelo Rei Artur através de um 'oráculo' (um súdito extremamente eficaz[5]), já que esta prova pode ser extremamente extensa—Merlin precisa levar em conta todos os possíveis lances do Rei. Note também que apenas partes da prova Π serão examinadas pelo Rei Artur, a saber, os trechos correspondentes aos seus lances. Ademais, como o Rei Artur vai basear seus lances em bits aleatórios, ele verificará a prova Π em locais escolhidos *aleatoriamente*.

Temos assim um novo tipo de certificado: para Merlin convencer o Rei de que não existe uma solução para o problema do banquete

[5]e totalmente confiável

pacífico, ele pode escrever uma prova Π, possivelmente bastante longa, que tem a seguinte propriedade. Caso não exista tal solução, esta prova Π convence o Rei Artur deste fato quando ela é examinada pelo Rei. Quando a tal solução *existe*, independentemente do que Merlin apresente como uma 'prova' da *inexistência* desta solução, o Rei Artur tem uma grande chance de descobrir que esta prova é errônea. A verificação do Rei Artur é baseada em bits aleatórios, e ele precisa de um oráculo para acessar a prova apresentada Π, já que esta prova pode ser bastante longa.

Note que neste novo sistema de certificados, não há interação entre as partes. Estes certificados são denominados *certificados* ou *provas verificáveis probabilisticamente*. Um resultado central neste texto é que se temos um problema de decisão e este problema pertence à classe NP, então a resposta SIM a este problema pode ser certificada com uma prova verificável probabilisticamente extremamente eficiente e robusta. Tais provas são conhecidas genericamente como *certificados* ou *provas transparentes* ou *holográficas*.

O resultado principal sobre provas transparentes, devido a Arora, Lund, Motwani, Sudan, e Szegedy [ALM$^+$92b], pode ser informalmente enunciado como segue. Suponha que queiramos certificar a resposta SIM para uma instância de tamanho n de um dado problema. Então existe uma prova verificável probabilisticamente Π que, ao ser examinada pelo Rei Artur, apenas uma quantidade não maior que uma *constante* independente de n de caracteres desta prova Π será lida, sendo a escolha destes caracteres baseada em apenas $O(\log n)$ bits aleatórios.

Consideremos o seguinte exemplo concreto. Suponha que o problema do banquete pacífico *admite* uma solução, *i.e.*, existe um circuito hamiltoniano no grafo dos 150 cavaleiros. Então é possível se apresentar uma prova Π deste fato com a seguinte propriedade: o Rei Artur lê uma quantidade pequena de caracteres desta prova e ele escolhe a localização destes caracteres através de um número também pequeno bits aleatórios. Genericamente, se o Rei Artur tivesse n cavaleiros, um número *constante* de caracteres desta prova seriam examinados e seriam necessários $O(\log n)$ bits aletórios para a escolha destes caracteres. Note que, em particular, muito pouca informação seria adquirida pelo Rei Artur neste processo: a leitura de uma quantidade constante de bits da prova não pode, por exemplo, dar-lhe muita informação sobre como é a solução do seu problema do banquete pacífico. Entretanto, esta leitura será suficiente para convencê-lo de que uma solução *existe*.

O resultado de Arora, Lund, Motwani, Sudan, e Szegedy é um dos temas centrais deste texto.

2.3. Conseqüências para sistemas formais. O leitor não particularmente interessado em teoria de complexidade pode achar que a existência de certificados transparentes para afirmativas NP seja um resultado muito específico desta área. Discutimos brevemente nesta seção que o contexto em que este resultado se encaixa é, na verdade, extremamente amplo.

Considere, por exemplo, um sistema formal como a teoria dos conjuntos de Zermelo–Fraenkel ou a aritmética de Peano. Seja A uma afirmativa e Π uma demonstração de A dentro deste sistema formal. Temos assim um *teorema* e pode-se dizer que a demonstração é um certificado de pertinência de A à linguagem constituída pelos teoremas. A discussão acima pode ser aplicada neste contexto para fornecer um novo sistema formal que define o *mesmo* conjunto de teoremas, mas em que o processo de verificação da correção de uma demonstração é outra. Um pouco mais especificamente, a afirmativa A e a prova Π acima podem ser transformadas em uma seqüência de símbolos Π' que, neste novo sistema formal, é uma prova de que A é um teorema. O agente que está verificando a correção da prova Π' (o 'Rei Artur') deve ter acesso a $O(\log n)$ bits aleatórios e, com base nestes bits, ele precisa escolher criteriosamente os pontos da prova Π' a serem verificados. A quantidade de pontos de Π' efetivamente escrutinados ou, mais precisamente, o números de bits de Π' a serem lidos pelo verificador é limitado por uma constante absoluta. Aqui n indica o comprimento total $|A| + |\Pi|$ do par teorema-prova (A, Π) do qual partimos. Ademais, a obtenção da prova Π' é algoritmicamente simples: Π' pode ser obtida de (A, Π) através de um algoritmo polinomial.

3. Conseqüências para Problemas de Otimização

3.1. Problemas NP-difíceis. No início dos anos 70 uma descoberta atingiu fundo a teoria da computação. Constatou-se que um certo grupo bastante amplo de problemas importantes e aparentemente difíceis do ponto vista computacional eram equivalentes no seguinte sentido: caso existisse um algoritmo *eficiente* para resolver qualquer um desses problemas, existiriam algoritmos eficientes para resolver *todos* os outros problemas. Estes problemas são ditos serem NP-*difíceis*, e incluem problemas das mais variadas áreas, como projetar redes telefônicas ou rotas de aviões, seqüenciar tarefas numa fábrica, encontrar soluções inteiras para equações quadráticas do tipo $ax^2 + by = c$, determinar o permanente de matrizes de 0s e 1s, encontrar soluções inteiras para problemas de programação linear, dentre muitos outros. Como um grande número de pesquisadores já haviam tentado resolver separadamente vários desses problemas,[6] a comunidade passou a acreditar na impossibilidade da obtenção de algoritmos eficientes para os problemas nesse grupo. Esta crença tornou-se a bem conhecida conjectura $\mathrm{P} \neq \mathrm{NP}$, que é central em teoria da computação.[7]

Surpreendentemente, o fato de asserções em NP terem certificados probabilisticamente verificáveis extremamente eficientes e robustos implica que encontrar soluções *aproximadas* para uma ampla gama de problemas de otimização é tão difícil quanto encontrar uma solução exata. Esse fato, que veio a público em 1992, teve impacto semelhante ao descrito no parágrafo anterior. Pesquisadores que tentavam 'ingenuamente' encontrar soluções aproximadas para problemas considerados difíceis, tiveram que mudar o rumo de suas pesquisas da noite para o dia em face desses resultados.

3.2. O problema do clique máximo. O primeiro problema para o qual se provou a impossiblidade de aproximações em vista desta nova teoria foi o problema do *clique máximo*. Suponha que temos um grafo G e que queremos encontrar o maior inteiro k tal que G contém um subgrafo completo de ordem k, isto é, um *clique* de ordem k. Em outras palavras, queremos saber o maior k para o qual temos em G um conjunto de k vértices dois-a-dois adjacentes. Escreva $\omega(G)$ para

[6]Um problema comprovadamente não mais difícil é o de fatorar eficientemente números inteiros. Babai [Bab94] observa que Fermat e Gauss provavelmente teriam adorado conhecer uma solução deste problema.

[7]Em correspondência descoberta recentemente, Gödel escreve a von Neumann levantando a questão sobre a relação entre P e NP, não descartando a possibilidade de que $\mathrm{P} = \mathrm{NP}$, e observa que este resultado teria conseqüências profundas; veja Sipser [Sip92].

este k. Os resultados obtidos nos anos 70 mostram que a determinação exata de $\omega(G)$ para um dado grafo G é um problema NP-*difícil*. Uma pergunta natural é então a seguinte: é mais fácil obter uma estimativa de $\omega(G)$?

Infelizmente, o resultado aqui é negativo: existe uma relação um tanto inesperada entre este problema e certificados transparentes, e esta relação fornece, como um corolário do teorema de Arora, Lund, Motwani, Sudan, e Szegedy, o seguinte resultado. Existe uma constante absoluta $\varepsilon > 0$ tal que se existe um algoritmo polinomial para estimar $\omega(G)$ a menos de um fator multiplicativo de n^{ε}, então existe um algoritmo polinomial para determinar o valor exato de $\omega(G)$.

Salientamos que o resultado acima não é de forma alguma um resultado isolado. Como mencionado acima, conhece-se hoje uma ampla *classe de problemas* que não admitem algoritmos de aproximação eficientes, a menos que P = NP e que desta forma todos eles possam ser resolvidos *exatamente*.

4. A Organização deste Texto

Este texto tem basicamente duas partes: a primeira delas tem o objetivo de introduzir o leitor aos resultados recentes da teoria da complexidade que culminaram com o resultado de Arora, Lund, Motwani, Sudan, e Szegedy. Esta parte tem dois capítulos. No Capítulo III discutimos provas interativas, inclusive provando o resultado IP = PSPACE discutido acima. Dedicamos todo o Capítulo IV às provas verificáveis probabilisticamente. Discutimos neste capítulo várias das idéias envolvidas na demonstração do resultado de Arora *et al.*, em particular, provando inteiramente um resultado intermediário que afirma que qualquer afirmativa NP admite um certificado cuja correção pode ser verificada pelo Rei Artur através do escrutínio de uma quantidade *constante* de seus caracteres, desde que ele tenha acesso a uma quantidade polinomial, e não apenas logarítmica, de bits aleatórios.

A segunda parte deste texto, constituída pelo Capítulo V, é dedicada às conseqüências do resultado de Arora *et al.* à área de otimização combinatória. Em particular, veremos como deduzir o resultado negativo relativo ao problema do clique máximo discutido na Seção 3.2.

O Capítulo II contém uma discussão sucinta das bases formais em que a teoria da complexidade computacional se fundamenta. O Apêndice A contém as definições dos problemas que são mencionados no decorrer do texto. O Apêndice B contém a terminologia e os conceitos básicos.

5. Notas Bibliográficas

Parte deste capítulo, incluindo a fábula do Rei Artur e Merlin, segue de perto Babai [Bab90]. Outro trabalho que fornece uma ótima introdução aos tópicos discutidos acima é Babai [Bab94]; em particular, o enfoque brevemente discutido na Seção 2.3 é elaborado detalhadamente neste trabalho. O fato de que provas interativas podem certificar a negativa de asserções NP foi provado por Lund, Fortnow, Karloff, e Nisan [LFKN92]; em realidade, o resultado destes autores é muito mais forte: eles provaram que $\mathrm{P}^{\#\mathrm{P}} \subset \mathrm{IP}$ (veja o Capítulo III para as definições). O resultado de que IP $=$ PSPACE é devido a Shamir [Sha92], que segue de perto [LFKN92].

O resultado principal sobre provas transparentes é devido a Arora, Lund, Motwani, Sudan, e Szegedy [ALM$^+$92b], embora muitas das idéias contidas neste trabalho tenham origem em Arora e Safra [AS92b]. Além dos trabalhos originais de Arora *et al.* [ALM$^+$92a, ALM$^+$92b], Sudan [Sud92], Arora [Aro94], e Hougardy, Prömel, e Steger [HPS94] apresentam demonstrações do resultado central sobre provas transparentes. Estes trabalhos também discutem as conseqüências deste resultado no estudo de problemas de aproximação.

A teoria de complexidade clássica nasceu oficialmente com os trabalhos de Cook [Coo71], Karp [Kar72], e Levin [Lev72] (uma tradução deste último trabalho aparece em Trakhtenbrot [Tra84]). A referência básica é Garey e Johnson [GJ79]. A noção de boas caracterizações de Edmonds aparece em [Edm65a]; veja também Edmonds [Edm65b], que apresenta uma das primeiras discussões específicas sobre a noção de eficiência de um algoritmo como entendemos hoje. Para uma discussão muito interessante sobre o nascimento e a situação atual da conjectura P $\neq$ NP, veja Sipser [Sip92]. Este trabalho contém também uma tradução da carta de Gödel a von Neumann.

CAPÍTULO II

PRELIMINARES

O objetivo deste capítulo é o de introduzir alguns conceitos básicos e resultados clássicos que serão usados nestas notas. Damos uma noção do modelo computacional a ser usado e definições das principais classes de complexidade computacional. Terminamos o capítulo tecendo alguns comentários sobre linguagens recursivas com o intuito de facilitar a leitura para o leitor familiarizado com esses conceitos.

1. Modelos Computacionais

A análise de recursos computacionais envolvidos para se resolver problemas pelo computador é o objeto de estudo da Teoria da Complexidade. Para tanto faz-se necessário estabelecer modelos computacionais abstratos que exprimam os aspectos práticos de uma computação.

O principal modelo usado nestas notas é o da máquina de Turing. Não iremos definir o que é uma máquina de Turing. Essa definição requer uma carga de formalismo grande que não ajuda muito no entendimento do que se pode ou não computar com uma máquina de Turing. Para o leitor não familiarizado com esse modelo computacional, tentaremos dar uma idéia geral do que ele vem a ser.

Comentamos inicialmente que, embora a máquina de Turing venha a ser um modelo 'primitivo' de computação, é amplamente aceita a Tese de Church, segundo a qual qualquer *procedimento* pode ser programado numa máquina de Turing. Não existe uma definição precisa de *procedimento,* o que impede, é claro, que a Tese de Church jamais venha a ser provada. Como noção informal, um *procedimento* é uma seqüência finita de instruções que pode ser executada por um agente computacional.

1.1. Tempo e espaço. Quanto aos recursos computacionais envolvidos, estamos interessados principalmente em duas medidas: o tempo e o espaço. O *tempo* de processamento de uma computação numa máquina de Turing é o número de passos executados até a término da computação. O *espaço* usado é a quantidade de bits ou células de memórias usadas na computação.

Fixando-se um problema $\mathcal{P}$, é natural que o tempo e espaço de processamento variem de acordo com a instância do problema considerado. Por exemplo, considere o *problema da primalidade*, ou seja, decidir se um dado número é primo. É de se esperar que o tempo e espaço usados por um algoritmo que resolva este problema cresçam com o tamanho do número a ser analisado. Dessa forma, o tempo e espaço são expressos em função do tamanho da *entrada* ou *instância* do problema. O *tamanho* de uma instância x, ou o *tamanho* de uma entrada x, de um problema $\mathcal{P}$ é o número de símbolos usados na descrição de x obtido por um *esquema de codificação.* Esse tamanho será denotado por $|x|$.

Um *esquema de codificação* é uma forma 'conveniente' de mapear instâncias de um problema em seqüências de caracteres. Para estabelecer um esquema de codificação precisamos inicialmente escolher um conjunto finito de caracteres Σ que será usado na descrição das instâncias do problema. Por exemplo, no caso da primalidade, um esquema de codificação para uma instância do problema poderia ser simplesmente a representação decimal de n. Portanto, o nosso alfabeto seria $\{0, 1, \dots, 9\}$.

A *complexidade em tempo (resp., espaço)* de um algoritmo que resolve um problema $\mathcal{P}$ é uma função do tamanho da entrada que expressa, para cada possível tamanho de entrada, a maior quantidade de tempo (resp., espaço) requerido pelo algoritmo para resolver uma instância daquele tamanho. É claro que essas complexidades dependem do modelo computacional adotado. Essas medidas, porém, não são alteradas substancialmente se o modelo adotado for realista.

1.2. Máquinas de Turing e RAM. Para dar uma idéia concreta do que é uma máquina de Turing, faremos um paralelo entre este modelo e programas de computadores reais. Uma boa aproximação, em termos computacionais de um computador são as Máquinas de Acesso Aleatório (RAM — *Random Access Machines*). Uma máquina RAM, para todos os efeitos, é uma máquina capaz de executar programas exatamente como os computadores existentes. A principal diferença é que qualquer computador real tem capacidade de armazenamento limitada, enquanto que no modelo RAM essa limitação não existe.

O seguinte fato mostra uma eqüivalência entre os modelos RAM e máquinas de Turing.

TEOREMA 1. *O modelo* RAM, *sob o critério de custos logarítmico, e o modelo de máquina de Turing são modelos relacionados polinomialmente.*

Explicamos a seguir o significado dos termos usados. O *critério de custos logarítmico* considera o tempo de execução de instruções proporcional ao comprimento em bits dos operandos. Dizemos que duas funções $f(n)$ e $g(n)$ são *relacionadas polinomialmente* se existirem polinômios $p_1(x)$ e $p_2(x)$ tais que, para todo n, temos $f_1(n) \leq p_1(f_2(n))$ e $f_2(n) \leq p_2(f_1(n))$. Dois modelos computacionais são *relacionadas polinomialmente* se qualquer algoritmo implementado em um modelo pode ser simulado por um outro algoritmo no segundo modelo de forma que as complexidades dos algoritmos são relacionadas polinomialmente.

Assim, o leitor não familiarizado com máquinas de Turing pode imaginar que uma máquina de Turing é um programa de um computador com capacidade de armazenamento ilimitada e onde os custos das operações levam em conta os tamanhos em bits dos operandos. Quando usamos a expressão 'seja M uma máquina de Turing,' O paralelo que deve ser feito é que um *programa* de um computador é dado. Ou seja, uma máquina de Turing corresponde a um programa escrito de acordo com o modelo geral das máquinas de Turing.

Os dados de entrada para um máquina de Turing M são fornecidos numa *fita de entrada* na qual M faz as leituras. O resultado da computação de M é escrito numa *fita de saída.*

2. As Classes P e NP

Os problemas classificados nas classes P e NP são *problemas de decisão,* ou seja, problemas cuja resposta é SIM ou NÃO. O intuito dessa restrição aos problemas de decisão é o de permitir o uso de conceitos e ferramentas envolvidos no formalismo das *linguagens formais.*

Fixe um conjunto finito Σ, chamado de *alfabeto.* O conjunto de todas as seqüências finitas de elementos de Σ, chamadas de *palavras,* é denotado por Σ^*. O tamanho de uma palavra $x \in \Sigma^*$, isto é, o número de símbolos usados para se escrever x, é denotado por $|x|$. Uma *linguagem* L sobre o alfabeto Σ é qualquer subconjunto $L \subseteq \Sigma^*$. O complemento de L, denotado por $\overline{L}$, é definido por $\overline{L} = \Sigma^* \setminus L$. Fixando uma linguagem $L \subseteq \Sigma^*$, o problema da *pertinência* é: dado um $x \in \Sigma^*$, decidir se $x \in L$.

Com esse formalismo, os problemas de decisão nada mais são do que problemas de pertinência em linguagens ou conjuntos. Considere um problema de decisão $\mathcal{P}$ com um esquema de codificação sobre um alfabeto Σ. Considere a linguagem L sobre Σ definida por

$$L = \{x \colon x \text{ é uma codificação de uma instância de } \mathcal{P} \text{ cuja resposta é SIM}\}.$$

Decidir pertinência em L é o mesmo que decidir sobre a resposta ao problema de decisão $\mathcal{P}$. Por exemplo, considere o problema da primalidade. Seja L a linguagem consistindo das palavras sobre o alfabeto $\Sigma = \{0, 1, \ldots, 9\}$ que representam, em notação decimal, números primos. O problema de decisão correspondente, PRIMALIDADE, é: dado uma palavra $x \in \Sigma^*$, decidir se $x \in L$.

Dizemos que uma máquina de Turing M *reconhece* uma linguagem L se M resolve o problema de pertinência para L. Ou seja, para todo $x \in \Sigma^*$, se x é escrito na fita de entrada de M, depois de um número finito de passos M pára escrevendo SIM na fita de saída se $x \in L$, e escrevendo NÃO caso contrário.

Em geral trabalhamos com alfabetos com pelo menos dois símbolos. Não perdemos muito se considerarmos que o nosso alfabeto é sempre o alfabeto binário $\{0, 1\}$. Qualquer linguagem sobre um alfabeto finito Σ pode ser convenientemente codificada em binário, de forma que qualquer palavra $x \in \Sigma^*$ tem tamanho no alfabeto binário no máximo

$|x|(1+\log_2|\Sigma|)$, ou seja, o tamanho das palavras diferem somente de um fator constante, uma vez que $|\Sigma|$ é constante.

As medidas de complexidade de tempo e espaço são dadas em função do tamanho de x, a entrada do problema. Para uma função $t(n)$, uma máquina de Turing M é dita ter complexidade de tempo $t(n)$ se para qualquer n e para qualquer entrada $x \in \Sigma^*$ de comprimento n, a máquina M pára depois de $O(t(n))$ passos. O conjunto das máquinas de Turing *limitadas polinomialmente* consiste da união, para todo $k > 0$, das máquinas de Turing com complexidade de tempo n^k. A máquina de Turing M é dita ter complexidade de espaço $t(n)$ se a maior quantidade de memória usada por M em suas computações é no máximo $O(t(n))$. Aqui, cada unidade de memória armazena um símbolo de Σ.

Uma máquina de Turing que, para qualquer entrada, pára após um número finito de passos é chamada de *algoritmo*. Um *algoritmo polinomial* é uma máquina de Turing limitada polinomialmente. Dizemos que uma função $f\colon X \to Y$ é *computável em tempo polinomial*, ou simplesmente, que f é uma função *polinomial*, se, existe um algoritmo polinomial que, para todo $x \in X$, calcula o valor $f(x)$.

2.1. Sistemas de provas. Um *sistema de provas* consiste de duas máquinas de Turing: V, o *verificador*, e P, o *provador*, que se comunicam. As duas máquinas de Turing têm acesso comum a uma fita de leitura contendo uma palavra x. Iremos impor restrições ao poder computacional da máquina V, mas P sempre terá poder computacional ilimitado. Existem ainda mais duas fitas, F_P e F_V, compartilhadas por V e P. 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. Veja Figura II.1.

2.2. A classe P**.** Definimos aqui a classe de linguagens que, em teoria, correspondem a problemas de decisão algoritmicamente tratáveis, isto é, problemas para os quais soluções computacionais são viáveis.

Definição 2. *Uma linguagem $L \subseteq \Sigma^*$ está na classe de linguagens* DTIME$(t(n))$ (Deterministic TIME) *se existe um sistema de provas tal que para todo $x \in \Sigma^*$, com $n = |x|$,*

(1) *a máquina de Turing V tem complexidade de tempo $t(n)$;*

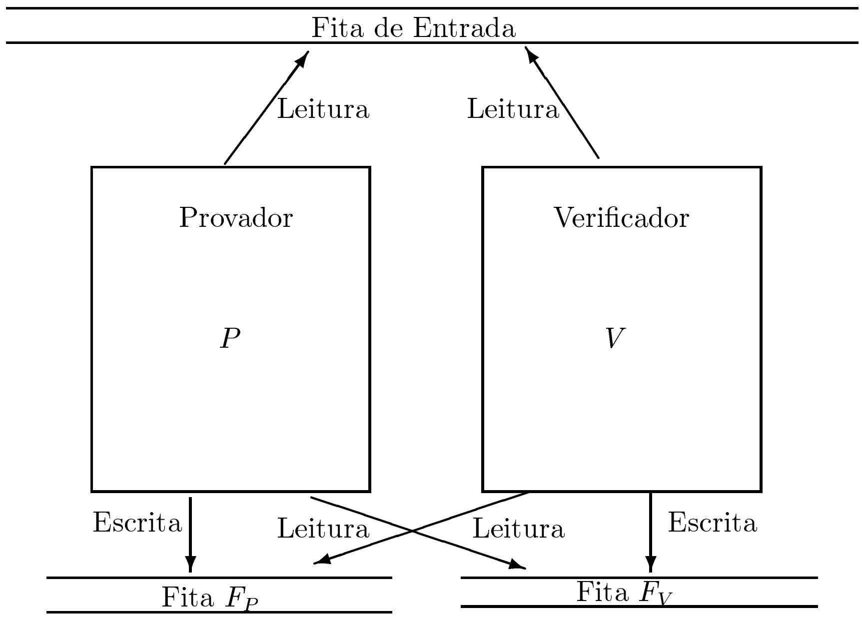

FIGURA II.1. Sistema de Provas

(2) *se $x \in L$ a máquina de Turing V, no término de suas computações, escreve* SIM *na fita F_V sem nunca consultar a fita F_P; neste caso dizemos que V* ACEITA *x.*
(3) *se $x \notin L$ a máquina de Turing V, no término de suas computações, escreve* NÃO *na fita F_V sem nunca consultar a fita F_P; neste caso dizemos que V* REJEITA *x.*

Note que, nessa definição, o provador P e a fita F_P não têm nenhuma importância. Tudo depende somente da máquina de Turing V cuja complexidade de tempo é limitada por $t(n)$.

DEFINIÇÃO 3. *A classe* P (tempo polinomial) *é definida por*

$$\mathrm{P} = \bigcup_{k>0} \mathrm{DTIME}(n^k),$$

e a classe EXP (tempo exponencial) *é definida por*

$$\mathrm{EXP} = \bigcup_{k>0} \mathrm{DTIME}(2^{n^k}).$$

2.3. A classe NP. A classe de linguagens definida a seguir contém a classe P e, intuitivamente, corresponde às linguagens que admitem certificados de pertinência simples.

DEFINIÇÃO 4. *Uma linguagem* $L \subseteq \Sigma^*$ *está na classe de linguagens* NTIME($t(n)$) (Non-Deterministic TIME) *se existe um sistema de provas tal que para todo* $x \in \Sigma^*$, *com* $n = |x|$,

(1) *a máquina de Turing* V *tem complexidade de tempo* $t(n)$*;*
(2) P *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* SIM *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* NÃO *na fita* F_V *após eventuais consultas à fita* F_P.

DEFINIÇÃO 5. *A classe* NP (tempo polinomial não-determinístico) *é definida por*

$$\mathrm{NP} = \bigcup_{k>0} \mathrm{NTIME}(n^k),$$

e a classe NEXP (tempo exponencial não-determinístico) *é definida por*

$$\mathrm{NEXP} = \bigcup_{k>0} \mathrm{NTIME}(2^{n^k}).$$

O sistema de provas para uma linguagem em NP *é chamado de sistema* NP *de provas.*

A definição da classe NP nos diz que se uma linguagem $L \in \mathrm{NP}$ então para todo $x \in L$ existe uma prova y, fornecida pelo provador, de que $x \in L$. Ademais, esta prova y pode ser verificada em tempo polinomial em $|x|$. Por exemplo, para provar que um número inteiro n é composto, basta exibir um de seus divisores d. A verificação de que de fato $d \mid n$ pode ser feita em tempo polinomial no número de dígitos de n. De maneira formal, uma linguagem L está na classe NP se existe um $c > 0$ e existe $L_1 \in \mathrm{P}$ tais que

$$(\forall x \in \Sigma^*)(x \in L \Leftrightarrow (\exists y)(|y| \leq |x|^c \text{ e } (x, y) \in L_1)).$$

Nestes termos, x é a entrada e y é a prova de que $x \in L$, ou um *certificado da pertinência* de x a L.

Abusando da linguagem, vamos dizer que 'um problema de decisão $\mathcal{P}$ está na classe de complexidade X' quando existir um esquema de codifição para $\mathcal{P}$ tal que a linguagem L definida pela codificação das instâncias de $\mathcal{P}$ que têm resposta SIM está em X.

Na definição da classe NP não podemos trocar o papel da resposta SIM pela resposta NÃO. Por exemplo, enquanto trivialmente existe uma prova curta de que um número é composto (basta exibir um divisor próprio do número) a situação é menos clara para se provar que um número é primo.[1] Portanto, é imediato concluir que o problema de decidir se um dado número é composto está em NP, enquanto que não é imediato que o problema de decisão PRIMALIDADE $\in$ NP. Um outro exemplo é o do Problema CIRCUITO HAMILTONIANO. Para provar que um grafo tem um circuito hamiltoniano, basta exibir um tal circuito. Por outro lado não se conhecem provas 'curtas' de que um dado grafo *não* é hamiltoniano. Em outras palavras, não se sabe se o problema de decidir se um grafo *não* é hamiltoniano está em NP.

Pelos comentários acima, faz sentido definirmos a classe de problemas complementares aos problemas em NP, a classe coNP. A classe de linguagens coNP é definida por

$$\mathrm{coNP} = \{\Sigma^* \setminus L : L \subseteq \Sigma^* \text{ e } L \in \mathrm{NP}\}.$$

O maior problema em aberto em Ciência da Computação é se P é diferente de NP. Em outros termos, verificar uma prova é computacionalmente mais fácil do que encontrá-la [LSS+77]? Muitas evidências indicam que P $\neq$ NP. Outra conjectura é que NP $\neq$ coNP. Das definições acima é fácil concluir que P $\subseteq$ NP $\cap$ coNP, e a conjectura é que a inclusão é própria. Também é fácil de ver que P é fechada sobre complementação, isto é, P = coP. Portanto, NP $\neq$ coNP implica que P $\neq$ NP, embora NP = coNP não implique que P = NP.

2.4. A classe PSPACE. Mencionamos por último que as classes de complexidade relativas ao *espaço* usado pelas computações são definidas de maneira análoga, trocando-se, nas definições, complexidade

---

[1] Embora exista também, para todo número primo, uma prova curta de sua primalidade [Pra75].

de tempo por complexidade de espaço. Uma classe que iremos usar é a classe PSPACE, definida por

$$\mathrm{PSPACE} = \bigcup_{k>0} \mathrm{DSPACE}(n^k).$$

Veja que neste caso não há limitação em relação ao tempo de processamento. Não é muito difícil de ver que NP $\subseteq$ PSPACE. Para tanto, suponha que exista um sistema NP de provas com provador P e verificador V para uma linguagem L. Podemos, baseado nesse sistema, contruir um sistema de provas que caracterize a pertinência de L em PSPACE. Para cada entrada x, um novo verificador V' simula, para cada possível conteúdo y da fita F_P, a computação de V. (Lembre que y tem comprimento polinomial em $|x|$.) O verificador V' pára com resposta SIM se e somente se para algum dos y o verificador V pára com resposta SIM. Como o tempo usado por V é polinomial em $|x|$, é claro que V' usa espaço polinomial para cada uma das simulações.

Embora PSPACE contenha também coNP e várias outras classes, não se sabe nem mesmo se P $\neq$ PSPACE.

3. Problemas de Enumeração e a Classe #P

Nos problemas de decisão estamos interessados em saber se uma determinada instância de um problema tem ou não uma certa propriedade. Por exemplo, dado um grafo, queremos saber se ele tem um circuito hamiltoniano. Um *problema de enumeração* consiste em contarmos quantas soluções um problema tem. No caso do CIRCUITO HAMILTONIANO, o problema de enumeração é saber quantos circuitos hamiltonianos distintos um grafo tem (eventualmente esse número é zero).

Mais formalmente, um *problema de enumeração* $\mathcal{P}$ consiste de um conjunto I de instâncias para $\mathcal{P}$ e, para cada $x \in I$, um conjunto finito $S(x)$ de soluções para x. A solução do problema de enumeração é a cardinalidade de $S(x)$. Se $\mathcal{P}$ é o Problema CIRCUITO HAMILTONIANO, o conjunto I consiste de todos os grafos finitos e para cada grafo $x \in I$, o conjunto $S(x)$ é constituído por todos os circuitos hamiltonianos do grafo x.

Um problema de enumeração $\mathcal{P}$ está na classe #P se existe um sistema NP (veja Definição 5) de provas tal que para cada instância x

de $\mathcal{P}$, o número dos y fornecidos pelo provador que levam o verificador a aceitar x é igual à cardinalidade de $S(x)$. Abusando da notação, dizemos que uma função $f\colon \Sigma^* \to \mathbb{N}$ está em #P se o valor de f, para cada x, é o número dos y que levam o verificador a aceitar x.

4. Linguagens Recursivas

A leitura desta seção não é essencial para o entendimento do restante dos capítulos. Nosso objetivo aqui é esclarecer, para o leitor familiarizado com o conceito de funções recursivas, alguns pontos que possam ter ficados obscuros neste capítulo.

Seja Σ um alfabeto finito. Uma função $f\colon \Sigma^* \to \Sigma^*$ é *recursiva* ou *computável* se existe uma máquina de Turing que para qualquer entrada $x \in \Sigma^*$ calcula $f(x)$ após um número finito de passos.

Uma linguagem L é *recursiva* se sua função característica f_L, definida sobre Σ^* e dada por

$$f_L(x) = \begin{cases} 1 & \text{se } x \in L \\ 0 & \text{caso contrário,} \end{cases}$$

é recursiva. Ou seja, se existe uma máquina de Turing T que resolve o problema de pertinência para L. Neste caso dizemos que T *decide* L, ou que L é *decidível*. Uma linguagem L que não é recursiva, ou seja, L não é decidível, é dita ser *indecidível*.

Pelas definições acima, dizer que uma linguagem L é indecidível é equivalente a dizer que não existe uma máquina de Turing capaz de decidir o problema de pertinência para L. Pela Tese de Church isso quer dizer que não existe um procedimento efetivo para decidir pertinência em L.

A 'maior' classe definida na seção anterior é a classe NEXP. Destacamos aqui que, embora NEXP contenha linguagens que são consideradas computacionalmente intratáveis (acredita-se que este já é o caso para algumas linguagens em NP), é óbvio que a classe NEXP está contida na classe de linguagens recursivas. Ou seja, embora possamos considerar as linguagens em NEXP fáceis de serem computadas por serem linguagens recursivas, a estória é outra quando consideramos os recursos computacionais envolvidos nessa computação.

Veremos a seguir classes de linguagens que contém propriamente a classe de linguagens recursivas. Uma linguagem L é *recursivamente* *enumerável* se $L = \emptyset$ ou existe uma função recursiva f tal que a imagem de f é L. Ou seja, podemos enumerar os elementos de L: $L = \{f(w_0), f(w_1), \dots\}$, se $\Sigma^* = \{w_0, w_1, \dots\}$.

Uma definição alternativa de linguagens recursivamente enumeráveis é dada pelo seguinte lema.

Lema 6. *Uma linguagem L é recursivamente enumerável se e somente se existe uma máquina de Turing que com entrada x pára se e somente se $x \in L$.*

É um fato bem conhecido que se L é recursiva, o seu complemento $\overline{L} = \Sigma^* \setminus L$ também é. E que se L e $\overline{L}$ são recursivamente enumeráveis, então L é recursiva.

4.1. O problema da parada. Vamos mostrar a seguir uma linguagem que é recursivamente enumerável mas não recursiva.

Seja T uma máquina de Turing. O *problema da parada* para T é decidir, para todas as possíveis entradas x, se T com entrada x pára. Seja $\mathcal{L} = \{\langle T, x\rangle : T \text{ com entrada } x \text{ pára}\}$. Aqui, T deve ser entendido como uma codificação conveniente da máquina de Turing T. O seguinte teorema é bastante conhecido.

Teorema 7. *A linguagem $\mathcal{L}$ é recursivamente enumerável mas não recursiva.*

Em outras palavras, o problema da parada é computacionalmente indecidível. Ou seja, como comentado na Introdução, o problema da parada não pode ser resolvido por processos algorítmicos.

Segue também que, pelo exposto acima, o complemento de $\mathcal{L}$ não é recursivamente enumerável. Embora esse exemplo de linguagem não recursiva possa parecer não muito natural, existem muitos outros problemas que são também indecidíveis. De fato, considere o seguinte problema.

Problema 8. *Dado um polinômio $p(x_1, x_2, \dots, x_n)$ com coeficientes inteiros e n variáveis, decidir se a equação $p = 0$ tem uma solução inteira.*

A indecidibilidade do problema acima foi provada por Matiyasevich em 1970, resolvendo assim o *Décimo Problema de Hilbert*. (Veja, por exemplo, Matiyasevich [Mat93].)

5. Notas Bibliográficas

As máquinas de Turing foram formalizadas por Alan Turing [Tur36]. O Teorema 1 é devido a Cook e Reckhow [CR73]. Para uma introdução mais detalhada às classes de complexidade descritas neste capítulo, veja Garey e Jonhson [GJ79].

A classe #P foi definida de maneira similar por Valiant [Val79] e Janos Simon [Sim75].

CAPÍTULO III

PROVAS INTERATIVAS

Num sistema de provas interativo temos um ser todo-poderoso, o Mago Merlin, e um humano comum, o Rei Artur. Suponha que Merlin queira convencer o desconfiado Rei Artur de que uma palavra x pertence a uma linguagem L. Embora o Rei seja bastante inteligente, dispõe somente de tempo polinomial para ser convencido de que $x \in L$. Quais são as classes de linguagens para as quais Merlin está apto a convencer o Rei Artur da resposta afirmativa ao problema de pertinência? Suponha que o Rei não se incomode em ser 100% convencido que $x \in L$, mas se tiver, digamos, 99% de certeza se dará por convencido. Será que com essas condições Merlin pode convencer o Rei da resposta SIM ao problema de pertinência para uma classe grande de linguagens?

Se exigimos que o Rei Artur seja 100% convencido, a resposta é que, para todo $x \in L$, o Rei pode ser convencido de que $x \in L$ se e somente se $L \in$ NP. A razão disso é que, primeiro, no sistema de provas que define NP, Merlin faria o papel do Provador P e o Rei Artur do Verificador V. E em segundo lugar, uma prova interativa totalmente determinística pode ser facilmente simulada por um sistema NP de provas. Por outro lado, se 99% de certeza for o suficiente, a resposta é surpreendente: para todo $x \in L$, o Rei pode ser convencido de que $x \in L$ se e somente se $L \in$ PSPACE. Como já mencionamos na Introdução, acredita-se que PSPACE seja uma classe de linguagens muito maior que NP.

1. Definições e Modelos

Informalmente, um sistema de provas interativo consiste de um jogo entre um *Provador* P ('Merlin') e um *Verificador* V ('Rei Artur'). Dado uma linguagem $L \subseteq \Sigma^*$ e um $x \in \Sigma^*$, a tarefa de Merlin é convencer o Rei Artur de que $x \in L$. Merlin nem sempre é honesto (nem sempre $x \in L$) e o honesto Rei Artur precisa ter uma estratégia para não se deixar enganar. O Rei tem acesso a bits aleatórios. O que ele precisa fazer é, baseado na entrada x e eventualmente consultando bits aleatórios, interagir com Merlin para se certificar que $x \in L$. De acordo com as respostas de Merlin, o Rei pode querer fazer mais perguntas, usando ou não mais bits aleatórios. Como Merlin é todo-poderoso, ele não gasta tempo nenhum para dar suas repostas. Assim, o processo todo fica limitado somente pelo Rei Artur, cujo tempo total para fazer suas computações é limitado polinomialmente. Dizemos que a linguagem $L \in \mathrm{IP}$ se para qualquer $x \in L$ Merlin consegue convencer o Rei disso, e para qualquer $x \notin L$, qualquer que seja a estratégia de Merlin, a probabilidade do Rei ser convencido de que $x \in L$ é muito pequena. Convencionamos dizer que Merlin *ganha* o jogo se o Rei Artur se convence de que $x \in L$. Caso contrário, Merlin *perde*.

Um *sistema de provas interativo* consiste de duas máquinas de Turing: o verificador V e o provador P que se comunicam. As duas máquinas de Turing têm acesso comum a uma fita de leitura contendo uma palavra x. A máquina de Turing V é limitada polinomialmente e tem acesso a uma fita contendo bits aleatórios. A máquina de Turing P tem poder computacional ilimitado. Existem ainda mais duas fitas, F_P e F_V, compartilhadas por V e P. 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. Usando essas fitas, P e V interagem trocando mensagens. Veja Figura III.2.

Um 'detalhe' não comentado até agora é se Merlin pode 'ver' os bits aleatórios do Rei Artur na medida em que este acessa esses bits. Esse detalhe, que conforme veremos não é tão importante, deu origem à definição de duas classes de complexidade em provas interativas, as classes IP e AM, que definiremos a seguir.

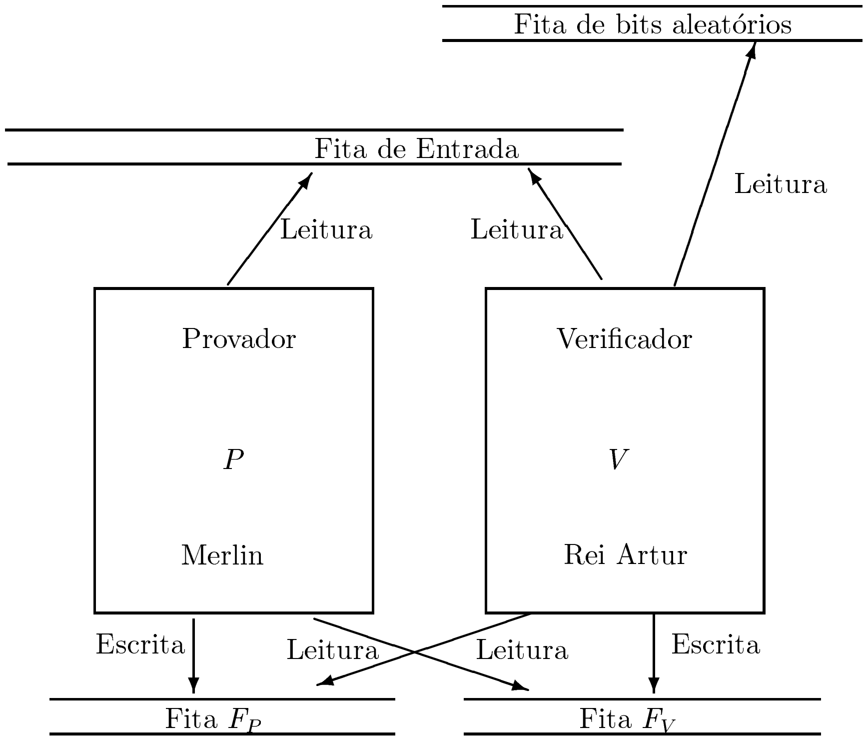

FIGURA III.2. Sistema de Provas Interativo

1.1. As classes IP **e** AM. Para facilidade de leitura (e por já termos uma certa familiaridade com sua majestade o Rei Artur), chamaremos sempre de Artur a máquina de Turing V limitada polinomialmente e de Merlin a máquina de Turing P com poder computacional ilimitado. Uma linguagem L está na classe $\mathrm{IP}(t(n))$ se existe um sistema de provas interativo e uma função $t(n)$ que, para uma entrada comum x, de comprimento $|x| = n$, as seguintes condições são satisfeitas:

(1) Artur e Merlin trocam no máximo $t(n)$ mensagens usando as fitas F_P e F_V;
(2) para todo $x \in L$, Merlin sempre tem uma estratégia ganhadora, isto é, existe um P tal que Artur ACEITA a prova de P

terminando suas computações escrevendo SIM na fita F_V (esse uso da fita não faz parte das $t(n)$ mensagens trocadas). Ou seja, Merlin sempre pode convencer o Verificador de que $x \in L$ se isso for verdade.

(3) para todo $x \notin L$, para qualquer Provador P, Artur REJEITA a prova de P terminando suas computações escrevendo NÃO na fita F_V com probabilidade $\geq 3/4$. Ou seja, por mais que Merlin tente enganar Artur, a probabilidade deste ser enganado é no máximo $1/4$. Essa probabilidade é medida em relação aos bits aleatórios consultados por Artur.

(4) Artur é o primeiro a fazer uso de sua fita de mensagens.

(5) Merlin não tem acesso aos bits aleatórios consultados por Artur.

Quando Artur termina o protocolo escrevendo SIM em sua fita, dizemos que Artur ACEITA [a prova de Merlin]. Caso contrário, dizemos que Artur REJEITA. A classe de linguagens $\mathrm{AM}(t(n))$ é definida de maneira análoga, exceto pelo item 5, onde agora Merlin e Artur têm acesso simultâneo aos bits aleatórios. O valor $3/4$ colocado na definição é arbitrário e poderia ser qualquer constante estritamente entre 0 e 1. Se repetirmos de forma independente e em paralelo o sistema acima um número conveniente (constante) de vezes, podemos amplificar arbitrariamente o grau de certeza da resposta, sem aumentar o número de lances no jogo.

As classes IP e AM são definidas por

$$\mathrm{IP} = \bigcup_{k \geq 0} \mathrm{IP}(n^k) \qquad \text{e} \qquad \mathrm{AM} = \bigcup_{k \geq 0} \mathrm{AM}(n^k).$$

As definições acima correspondem simplesmente a não limitar o número de mensagens trocadas entre P e V, porém mantendo-se a restrição de polinomialidade de V.

Uma variante da classe AM é a classe MA: agora quem usa a fita de mensagens primeiro é Merlin.

1.2. Um exemplo de um sistema de prova interativo. É fácil ver que $\mathrm{NP} \subseteq \mathrm{IP}(2)$. Se uma linguagem L está em NP, então existe um sistema de provas para L, conforme Definição 5. Esse sistema de provas pode ser simulado por um sistema de provas interativo considerando-se que Artur abre mão de sua primeira mensagem.

Vamos mostrar nesta seção que um certo problema que não se sabe se está em NP está em IP(2). Trata-se do Problema Grafo Não-Isomorfismo, que consiste em decidir se dois grafos dados não são isomorfos.

Teorema 9. Grafo Não-Isomorfismo *está em* IP(2).

Prova. Seja (G_1, G_2) uma instância do problema, onde os grafos dados são $G_1 = (V, E_1)$ e $G_2 = (V, E_2)$. O protocolo entre Artur e Merlin está descrito na Figura III.3.

(1) Artur escolhe aleatoriamente $c \in \{1,2\}$ e uma permutação aleatória π de V.
(2) Artur computa o grafo $\pi(G_c) = G_3 = (V, E_3)$, onde E_3 é definido por $\{\pi(u), \pi(v)\} \in E_3$ se e somente se $\{u, v\} \in G_c$.
(3) Artur descreve G_3 na fita F_V;
(4) Merlin, depois de ler G_3, escreve na fita F_P o valor de c.
(5) Se Merlin erra no Passo 4 o valor de c, Artur rejeita; caso contrário, Artur aceita.

Figura III.3. Protocolo para Grafo Não-Isomorfismo

Se os grafos G_1 e G_2 não são isomorfos, então Merlin 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 pode acertar no Passo 4.

Por outro lado, se G_1 e G_2 são isomorfos, então devido à escolha aleatória da permutação π, a probabilidade de G_3 ser $\pi(G_1)$ é igual à probabilidade de ser $\pi(G_2)$. Neste caso, Merlin, desconhecendo o valor de c, irá errar no Passo 4 com probabilidade $1/2$. Para obtermos o limitante desejado de $3/4$ na probabilidade de acerto de Artur, o protocolo acima deve ser repetido em paralelo 2 vezes. $\square$

No protocolo acima, é fundamental o fato de Merlin não ter acesso aos bits aleatórios de Artur, ou mais precisamente, aos valores de c e

π. Porém, veremos a seguir que tal fato não é tão importante quanto parece.

1.3. Propriedades das classes IP e AM. Vamos mencionar, sem provas, duas propriedades importantes das classes IP e AM.

A primeira delas esclarece uma questão mencionada anteriormente, a saber, o quão importante é o fato de Merlin ter ou não acesso aos bits aleatórios de Artur.

TEOREMA 10. *Para todo polinômio $Q(n)$, temos*

$$\mathrm{IP}(Q(n)) \subseteq \mathrm{AM}(Q(n)+2).$$

A segunda propriedade responde a seguinte pergunta. Será que o número de vezes que Artur e Merlin trocam mensagens é relevante na definição dessas classes? A resposta a essa pergunta é um tanto surpreendente.

TEOREMA 11. *Seja $t\colon \mathbb{N} \to \mathbb{N}$ uma função com $t(n) \geq 2$ para todo n. Então $\mathrm{AM}(t(n)) = \mathrm{AM}(2t(n))$.*

Em particular, esse teorema nos diz que a hierarquia das classes $\mathrm{AM}(k)$, para k finito, colapsa. Ou seja, $\mathrm{AM}(2) = \mathrm{AM}(k)$ para qualquer inteiro k. Note que o teorema não implica que a classe AM, na qual não se limita o número de trocas de mensagens, colapse. Pelo Teorema 10, os mesmos comentários se aplicam à classe IP.

2. coNP $\subset$ IP

Provaremos nesta seção que se uma linguagem L admite certificados polinomiais de pertinência, então a afirmativa $x \notin L$ admite uma prova interativa. Formalmente, coNP $\subset$ IP. Na próxima seção provaremos algo muito mais forte, a saber, que PSPACE $\subset$ IP. A presente seção é mais um aquecimento para que possamos adquirir maior familiaridade com provas interativas, e também para ver um dos componentes da demonstração de que PSPACE $\subset$ IP, a saber, o 'Protocolo LFKN' de Lund, Fortnow, Karloff, e Nisan [LFKN92].

Um problema completo para a classe coNP é o de decidir se o número cromático $\chi(G)$ de um dado grafo G é ≥ 4, isto é, se G não admite uma coloração de seus vértices com no máximo 3 cores de forma que vértices adjacentes recebam cores distintas (para o conceito de completude veja

Seção 2 do Capítulo V). Note que claramente este problema pertence à classe coNP, já que se G admite uma tal coloração, então exibindo uma destas colorações certificamos que $\chi(G) \leq 3$.

Seja dado um grafo G. Exibimos abaixo um jogo Artur–Merlin em que Merlin sempre vence se $\chi(G) \geq 4$, e o Rei Artur tem alta chance de vencer caso $\chi(G) \leq 3$, independentemente dos lances de Merlin. Começamos escolhendo um inteiro positivo N bastante grande em relação ao tamanho de G, e escolhemos um polinômio $p(X)$ de grau 5 de coeficientes racionais tal que $p(0) = 0$ e $p(t) = 1$ se $t \in \{\pm 1, \pm 2\}$. Suponha que o conjunto de vértices de G é $\{1, \ldots, n\}$, e ponha

$$q(X_1, \ldots, X_n) = \prod p(X_i - X_j),$$

onde o produto se estende sobre todos pares $i < j$ com ij uma aresta de G. Note que o polinômio $q(X_1, \ldots, X_n)$ tem grau total $d \leq 5|E(G)|$. No que segue, usamos as cores 0, 1, e 2 para colorir os vértices de G. Ponha $H = \{0, 1, 2\}$. Então $\mathbf{x} \in H^n$ corresponde naturalmente a uma coloração dos vértices de G, e

$$q(\mathbf{x}) = \begin{cases} 1 & \text{se } \mathbf{x} \text{ é uma coloração própria de } G \\ 0 & \text{caso contrário.} \end{cases} \tag{1}$$

Desta forma, o número de colorações próprias de G é

$$q_0 = \sum_{\mathbf{x} \in H^n} q(\mathbf{x}),$$

e é o objetivo de Merlin convencer o Rei Artur de que o valor q_0 desta soma é nulo. Claramente, a definição explícita de q_0 é uma soma sobre um conjunto de tamanho exponencial ('experimente todas as possibilidades'), e assim não podemos esperar que o Rei Artur possa ingenuamente computar q_0, já que ele é polinomialmente limitado.

Para cada $1 \leq i \leq n$, consideremos o polinômio racional

$$q_i(X_1, \ldots, X_i) = \sum_{x_{i+1} \in H} \cdots \sum_{x_n \in H} q(X_1, \ldots, X_i, x_{i+1}, \ldots, x_n) \tag{2}$$

de i indeterminadas. Note que, trivialmente, temos

$$q_{i-1}(X_1, \ldots, X_{i-1}) = \sum_{x_i \in H} q_i(X_1, \ldots, X_{i-1}, x_i) \tag{3}$$

para todo $1 \leq i \leq n$. O jogo de Artur–Merlin é dado na Figura III.4.

> Merlin declara inicialmente que $q_0 = \hat{q}_0 = 0$.
> Para cada $i = 1, \ldots, n$, alternam-se os seguintes lances.
> (1) Merlin fornece um polinômio racional $\hat{q}_i(X_i)$ de grau $\leq d$ ao Rei Artur, e declara que $\hat{q}_i(X_i) = q_i(\rho_1, \ldots, \rho_{i-1}, X_i)$;
> (2) O Rei Artur verifica se $\hat{q}_{i-1}(\rho_{i-1}) \neq \sum_{x_i \in H} \hat{q}_i(x_i)$, e REJEITA se este for o caso e o jogo termina. Caso contrário, o Rei escolhe uniformemente ao acaso um inteiro ρ_i satisfazendo $1 \leq \rho_i \leq N$, e fornece este valor a Merlin.
>
> Finalmente, se $\hat{q}_n(\rho_n) = q(\rho_1, \ldots, \rho_n)$, o Rei Artur ACEITA, e caso contrário o Rei REJEITA.

FIGURA III.4. Protocolo LFKN

Assuma que $q_0 = 0$ e descrevamos a estratégia de vitória para Merlin. Simplesmente, basta que os lances de Merlin no passo (1) do jogo sejam, de fato, $\hat{q}_i(X_i) = q_i(\rho_1, \ldots, \rho_{i-1}, X_i)$ para todo $1 \leq i \leq N$. Note que, então, a identidade (3) garante que o Rei Artur nunca rejeitará, e assim ele terá convencido o Rei de que $q_0 = 0$ e, assim, de que $\chi(G) \geq 4$, como queríamos.

Suponha agora que Merlin está tentando certificar uma afirmativa falsa; isto é, admitamos que $q_0 \neq 0$. O nosso objetivo é obter uma estimativa superior para a probabilidade de que o Rei Artur acabe aceitando que $q_0 = 0$ ou, equivalentemente, que $\chi(G) \geq 4$. Assuma, portanto, que o Rei acabou devolvendo ACEITA.

Suponha que $\hat{q}_1(X_1)$ passou o *teste de consistência* no passo (2) do jogo, isto é, suponha que $\hat{q}_1(0) + \hat{q}_1(1) + \hat{q}_1(2) = 0$. Como estamos assumindo que $q_1(0) + q_1(1) + q_1(2) = q_0 \neq 0$, temos que os polinômios $\hat{q}_1(X_1)$ e $q_1(X_1)$ diferem. Agora notemos que, como Merlin declarou que $\hat{q}_1(X_1) = q_1(X_1)$, ele declarou implicitamente que $q_1(\rho_1) = \hat{q}_1(\rho_1)$. Como ambos $\hat{q}_1(X_1)$ e $q_1(X_1)$ são de grau no máximo d, sabemos que eles coincidem para no máximo d valores de X_1. Assim, a probabilidade de que seja correto o valor $\hat{q}_1(\rho_1)$ declarado por Merlin para $q_1(\rho_1)$ é de no máximo d/N. Suponha que $\hat{q}_1(\rho_1) \neq q_1(\rho_1)$. Consideraremos o caso em que há acidentalmente igualdade aqui como 'azar', e contabilizaremos a probabilidade correspondente na estimativa superior para a probabilidade de o Rei Artur ser 'ludibriado'.

Observemos agora que a situação é a seguinte. Temos um polinômio racional $q(\rho_1, X_2, \ldots, X_n)$ com $n-1$ indeterminadas e Merlin declarou um valor errôneo para a soma

$$q_1(\rho_1) = \sum_{x_2 \in H} \cdots \sum_{x_n \in H} q(\rho_1, x_2, \ldots, x_n), \tag{4}$$

a saber, $\hat{q}_1(\rho_1) \neq q_1(\rho_1)$. Assim, estamos numa situação análoga àquela da qual partimos: no começo do jogo, Merlin havia declarado o valor errôneo 0 para a soma $q_0 = \sum_{\mathbf{x} \in H^n} q(\mathbf{x})$. Entretanto, temos agora *uma indeterminada a menos*, e a soma é sobre o 'hipercubo' H^{n-1} de dimensão *um menor*.

Prosseguimos agora por indução, sempre assumindo que os valores declarados $\hat{q}_i(\rho_i)$ $(1 \leq i \leq n)$ para as somas intermediárias

$$\sum_{x_{i+1} \in H} \cdots \sum_{x_n \in H} q(\rho_1, \ldots, \rho_i, x_{i+1}, \ldots, x_n)$$

são incorretos. Analisemos o que acontece ao fim das n rodadas do jogo. Temos, neste ponto, os inteiros $\rho_1, \ldots, \rho_n$ e o polinômio $\hat{q}_n(X_n)$. Este polinômio é, de acordo com Merlin, o polinômio

$$q_n(\rho_1, \ldots, \rho_{n-1}, X_n). \tag{5}$$

No último teste de consistência do Rei Artur no protocolo, ele verifica se

$$\hat{q}_n(\rho_n) = q_n(\rho_1, \ldots, \rho_n). \tag{6}$$

Como estamos admitindo que o Rei Artur aceitou a afirmativa de Merlin, a igualdade (6) verificou-se, o que ocorre com probabilidade no máximo d/N, já que $\hat{q}_n(X_n)$ *não* é o polinômio dado em (5).

Concluímos assim que a probabilidade de Merlin ludibriar o Rei Artur é de no máximo

$$\sum_{0 \leq i \leq n} \left(1 - \frac{d}{N}\right)^i \frac{d}{N} \leq \frac{d(n+1)}{N}. \tag{7}$$

Basta agora tomar N suficientemente grande para que $d(n+1)/N \leq 1/4$. Por exemplo, podemos tomar $N = 20n^3$, onde, recordamos, n é o número de vértices em G.

Assim certificamos a afirmativa $\chi(G) \geq 4$ através de uma prova interativa. Como mencionado acima, esta afirmativa é completa para a classe coNP, e portanto temos o seguinte resultado.

TEOREMA 12. coNP $\subset$ IP.

Note que o resultado acima mostra que o problema de se certificar a *não* existência de um circuito hamiltoniano em um dado grafo pode ser resolvido através de um sistema de provas interativo, já que este problema pertence à classe coNP. Lembre que este foi um dos exemplos discutidos no Capítulo I (o 'problema do banquete pacífico').

Em realidade, o Protocolo LFKN pode ser usado para certificar o *número* de colorações próprias que um dado grafo G tem com k cores, onde k é qualquer inteiro fixo. Isto implica algo muito mais forte que o Teorema 12, embora este resultado seja mais 'técnico'.

Suponha que uma máquina de Turing M tenha acesso a um oráculo que é capaz de responder perguntas do tipo 'quanto é $f(x)$?', onde f é uma função em #P (cf. Seção 3 do Capítulo II). O custo de cada consulta ao oráculo é aqui, por definição, unitário. A classe de linguagens que são reconhecidas em tempo polinomial por tais máquinas de Turing M é conhecida como a classe $\mathrm{P}^{\#\mathrm{P}}$. O Protocolo LFKN fornece o seguinte resultado de Lund, Fortnow, Karloff, e Nisan [LFKN92].

TEOREMA 13. $\mathrm{P}^{\#\mathrm{P}} \subset \mathrm{IP}$.

Por um resultado de Toda [Tod89], a hierarquia polinomial (veja, por exemplo, Garey e Johnson [GJ76]) está contida em $\mathrm{P}^{\#\mathrm{P}}$. Assim temos do resultado acima o seguinte corolário imediato.

COROLÁRIO 14. *Toda linguagem na hierarquia polinomial admite um sistema interativo de prova.*

Veremos na seção seguinte o resultado definitivo sobre o poder das provas interativas.

3. IP = PSPACE

A intenção ao se definir a classe AM foi a de se definir classes de linguagens um pouco maiores que NP. Nas palavras de Babai e Moran [BM88], "a hierarchy of complexity classes 'just above NP'." O resultado que descrevemos a seguir, que IP = PSPACE, foi uma surpresa para todos os pesquisadores da área. O poder de uma fonte de bits aleatórios num sistema de provas tinha sido subestimado.

Uma possível justificativa para tal fato é a seguinte. Existem classes de complexidade definidas de forma similar à classe P, com a diferença que agora os algoritmos têm acesso a uma fonte de bits aleatórios e se requer que a resposta seja apenas *probabilisticamente* correta. Esses algoritmos são chamados de algoritmos do tipo *Monte Carlo.* Classes de linguagens reconhecíveis por tais algoritmos foram definidas nos anos 70. Desde então, todos os esforços empreendidos na direção de se provar que essas classes contenham propriamente a classe P ou se igualem a classe NP foram em vão. Com isso, a comunidade passou a duvidar do poder computacional de elementos aleatórios em procedimentos algorítmicos.

Antes da prova que IP = PSPACE, precisamos fazer umas observações sobre a classe PSPACE. Podemos definir o similar da redução de Karp (Definição 46) para a classe PSPACE. Com essa nova redução podemos definir problemas completos para PSPACE, e provar o seguinte teorema.

TEOREMA 15. *O Problema* QBF *é* PSPACE-*completo.*

(Para a definição do Problema QBF veja Apêndice A.)

Para mostrar que IP = PSPACE comentaremos inicialmente que IP $\subseteq$ PSPACE. A seguir, mostraremos que QBF está em IP. Como o Problema QBF é completo para PSPACE, isso implica que PSPACE $\subseteq$ IP, encerrando a prova.

TEOREMA 16. IP = PSPACE.

PROVA. Não é muito difícil ver que IP $\subseteq$ PSPACE. Para uma máquina de Turing M' reconhecer uma linguagem $L \in$ IP usando espaço polinomial, basta que M' simule, para cada possível conteúdo y da fita F_P e cada conteúdo τ da fita de bits aleatórios, a computação de V. A máquina M' pára com resposta SIM se e somente se para algum dos y e para qualquer τ, o verificador V pára com resposta SIM. Caso contrário, M' pára com resposta NÃO.

Portanto, a parte difícil da prova é mostrar que PSPACE $\subseteq$ IP. Para tanto, vamos mostrar que um problema completo para PSPACE, o problema QBF, está em IP.

A entrada do QBF é uma fórmula booleana quantificada

$$(Q_1x_1)(Q_2x_2)\dots(Q_kx_k)B(x_1,x_2,\dots,x_k),$$

onde Q_i é ou $\forall$ ou $\exists$ e B é uma fórmula booleana (sem quantificadores).

Para cada fórmula booleana $B(x_1,x_2,\dots,x_k)$ podemos fazer corresponder um polinômio $b(x_1,x_2,\dots,x_k)$, conhecido como a *aritmetização* da fórmula B, onde $\alpha\wedge\beta$ é substituído por $\alpha\beta$, $\neg\alpha$ por $1-\alpha$ e $\alpha\vee\beta$ por $\alpha\star\beta=\alpha+\beta-\alpha\beta=1-(1-\alpha)(1-\beta)$. Observe que os valores de B e b coincidem para argumentos booleanos, isto é, interpretando 1 como verdadeiro e 0 como falso.

Seja $P(x,x_1,x_2,\dots,x_k)$ um polinômio. Definimos os seguintes polinômios a partir de P:

$$\begin{aligned}
(A_xP)(x_1,x_2,\dots,x_k) &= P(0,x_1,x_2,\dots,x_k)P(1,x_1,x_2,\dots,x_k),\\
(E_xP)(x_1,x_2,\dots,x_k) &= P(0,x_1,x_2,\dots,x_k)\star P(1,x_1,x_2,\dots,x_k),\\
(R_xP)(x,x_1,x_2,\dots,x_k) &= P \bmod (x^2-x)\\
&\qquad\text{(isto é, todas as ocorrências de } x^n \text{ com}\\
&\qquad n>1 \text{ são substituídas por } x).
\end{aligned}$$

Nos polinômios A_xP e E_xP a variável x está ausente, enquanto R_xP tem as mesmas variáveis de P. Note que P e R_xP coincidem em argumentos booleanos.

Seja $S(x_1,x_2,\dots,x_k)$ um polinômio sobre um corpo finito C de cardinalidade $|C|$. Assuma que exista um sistema de provas interativo α em IP permitindo que Merlin convença Artur que $S(u_1,u_2,\dots,u_k)=v$ com probabilidade 1 para qualquer entrada $u_1,u_2,\dots,u_k,\ v\in C$ quando isso é verdadeiro, e com probabilidade menor que ε quando isso é falso. Seja U o polinômio obtido de S por uma das operações A_x, E_x ou R_x. Seja d o grau de x em S e suponha este valor conhecido por Artur. Vamos construir um sistema de provas interativo β permitindo a Merlin convencer Artur de que $U(c_1,c_2,\dots,c_t)=e$ com probabilidade 1 para quaisquer $c_1, c_2, \ldots, c_t$, e quando isso é verdadeiro e com probabilidade menor que $\varepsilon + d/|C|$ quando isso é falso.

Descrevemos agora a construção do sistema de provas interativo β.

Caso A. $U(y_1, \ldots, y_t) = A_x S(x, y_1, \ldots, y_t)$. Merlin quer convencer Artur de que $U(c_1, c_2, \ldots, c_t) = e$. O protocolo está descrito na Figura III.5.

(1) Merlin envia a Artur os coeficientes do polinômio $s(x) = S(x, c_1, \ldots, c_t)$.
(2) Se o grau de s excede d ou $s(0)s(1) \neq e$, Artur REJEITA.
(3) Caso contrário, Artur envia a Merlin um elemento aleatório de $r \in C$.
(4) Usando o sistema de provas α, Merlin tenta convencer Artur de que $S(r, c_1, \ldots, c_t) = s(r)$.

FIGURA III.5. Protocolo para o Caso A

Caso E. $U(y_1, \ldots, y_t) = E_x S(x, y_1, \ldots, y_t)$. Troca-se, no caso anterior, $s(0)s(1)$ por $s(0) \star s(1)$.

Caso R. $U(x, y_1, \ldots, y_t) = R_x S(x, y_1, \ldots, y_t)$. Merlin quer convencer Artur de que $U(f, c_1, c_2, \ldots, c_t) = e$. O protocolo está descrito na Figura III.6.

Merlin pode enganar Artur ou durante α (com probabilidade menor que ε) ou se polinômios diferentes $s(x)$ e $S(x, c_1, \ldots, c_t)$ coincidem no ponto aleatório r (probabilidade menor que $d/|C|$).

Seja $F = (Q_1 x_1)(Q_2 x_2) \ldots (Q_k x_k) B(x_1, x_2, \ldots, x_n)$ uma fórmula booleana quantificada. Considere a aritmetização $b(x_1, x_2, \ldots, x_n)$ de B e aplique, seqüencialmente, operações conforme descritas na Figura III.7. Nessa figura $q_i x = A_x$ ou E_x conforme $Q_i = \forall$ ou $\exists$. Após essas operações obtemos uma constante igual a 0 ou 1, dependendo da validade da fórmula F. Se a constante é 1, Merlin pode convencer Artur desse fato. Para tanto, o trabalho inicial de Merlin é aplicar no polinômio b as operações descritas acima que resultaram na constante 1. Após essa fase, Merlin precisa convencer Artur de que cada

(1) Merlin envia a Artur os coeficientes do polinômio $s(x) = S(x, c_1, \dots, c_t)$.
(2) Se o grau de s excede d ou $s(0) + (s(1) - s(0))f \neq e$, Artur REJEITA.†
(3) Caso contrário, Artur envia a Merlin um elemento aleatório de $r \in C$.
(4) Usando os sistema de provas α, Merlin tenta convencer Artur de que $S(r, c_1, \dots, c_t) = s(r)$.

†Observe que $s(0) + (s(1) - s(0))f$ é o valor de $s(x) \bmod (x^2 - x)$ em f.

FIGURA III.6. Protocolo para o Caso R

operação foi feita corretamente. Isso é feito seguindo a seqüência de operações de 'baixo para cima.' Em cada passo Merlin usa o sistema β para convencer Artur de que a operação foi aplicada corretamente. Finalmente, a igualdade $b(u_1, u_2, \dots, u_n) = v$ precisa ser verificada para algum $u_1, u_2, \dots, u_n$, v. Artur pode fazer isso sozinho pois a fórmula B é conhecida. A probabilidade de erro não excede

$$\frac{(\text{número de operações } A, E, R) \times (\text{grau máximo})}{|C|}.$$

Se o comprimento da fórmula F é l, então o número de operações é $O(l^2)$. O grau máximo de b não excede l e as operações R reduzem o grau a 1, sendo que posteriormente os graus são no máximo 2. Se C tem mais do que l^4 elementos, a probabilidade de erro tende a 0 quando $l \to \infty$. Podemos usar $C = \mathbb{Z}/p\mathbb{Z}$, onde p é um primo de comprimento logarítmico em l. O primo p pode ser escolhido por Artur, pois o teste de primalidade é trivial para números desse tamanho. □

4. Notas Bibliográficas

A classe IP foi definida por Goldwasser, Micali, e Rackoff [GMR85], e a classe AM por Babai [Bab85]. Os trabalhos independentes apresentando essas classes aparecem simultaneamente numa mesma conferência. Um ano depois, Goldwasser e Sipser [GS86] provam o Teorema 10 mostrando a equivalência dessas classes. O Teorema 11 é

$$\boxed{\begin{array}{l} R_{x_1}, R_{x_2}, \ldots, R_{x_n}, \\ q_n x_n, \\ R_{x_1}, R_{x_2}, \ldots, R_{x_{n-1}}, \\ q_{n-1} x_{n-1}, \\ \ldots \\ R_{x_1}, R_{x_2}, \\ q_2 x_2, \\ R_{x_1}, \\ q_1 x_1, \end{array}}$$

FIGURA III.7. Operações sobre b

de Babai e Moran [BM88]. Goldreich discute em [Gol94] vários sistemas de provas, salientando a importância dos elementos aleatórios em todos eles.

Os resultados da Seção 2 são todos devidos a Lund, Fortnow, Karloff, e Nisan. Em [LFKN92], estes autores dão uma prova de que o valor do permanente de uma matriz pode ser certificado através de uma prova interativa. Como provado por Valiant [Val79], esta função é completa para a classe #P, e assim segue o Teorema 13, o resultado central da Seção 2. A prova alternativa aqui apresentada é devida a Babai [Bab94].

O Teorema 15 é de Stockmeyer [Sto76]. O resultado IP = PSPACE é devido a Shamir [Sha92]. A demonstração original deste teorema segue de perto Lund, Fortnow, Karloff, e Nisan [LFKN92]. Veja Babai [Bab90] para uma discussão muito interessante sobre a história deste resultado. A demonstração simplificada dada acima é devida a Shen [She92].

CAPÍTULO IV

PROVAS VERIFICÁVEIS PROBABILISTICAMENTE

Neste capítulo discutiremos a demonstração do resultado central de Arora, Lund, Motwani, Sudan, e Szegedy [ALM$^+$92b]. Lembremos rapidamente o que diz este resultado. Seja L um linguagem NP, ou seja, uma linguagem cujos elementos $x \in L$ admitem *certificados de pertinência* y_x simples e curtos. Intuitivamente, isto significa que para que 'provemos' que $x \in L$, basta apresentarmos este certificado y_x como uma demonstração deste fato. Sendo simples e curta, esta prova pode ser eficientemente verificada em tempo polinomial.

Informalmente, o resultado de Arora *et al.* diz que se $L \in \mathrm{NP}$, então todo elemento $x \in L$ também admite um certificado de pertinência Π_x *transparente*. Para que fiquemos convencidos de que $x \in L$, basta lermos um número $O(1)$ de caracteres de Π_x. Os caracteres a serem lidos são escolhidos aleatoriamente, mas o número de bits aleatórios usados para se sortear estes caracteres é $O(\log n)$, onde como de usual $n = |x|$.

1. O Sistema de Provas PCP

Aqui descrevemos precisamente o sistema de provas em que se baseiam as demonstrações transparentes. Neste capítulo, sempre teremos $\Sigma = \{0, 1\}$ e, sem perda de generalidade, consideramos linguagens $L \subset \Sigma^*$ apenas.

No sistema de provas PCP (*Probabilistically Checkable Proofs*), o *verificador* V é uma máquina de Turing determinística de complexidade de tempo polinomial que recebe como entrada uma palavra $x \in \Sigma^*$

e que tem acesso a uma seqüência de bits aleatórios $\tau \in \Sigma^*$. Este verificador também tem acesso a uma palavra $\Pi = \pi_1\pi_2 \ldots \pi_{|\Pi|} \in \Sigma^*$ através de um oráculo. Uma vez dadas a entrada x e a seqüência de bits aleatórios τ, o verificador V 'sabe' quais caracteres π_i de Π ele vai querer acessar durante a sua computação; em outras palavras, uma vez dadas x e τ, o verificador V pode pedir ao oráculo que lhe dê, de uma vez, todos os caracteres $\pi_{i_1}, \ldots, \pi_{i_\ell}$ de que V necessitará, não sendo o valor destes caracteres levado em conta na decisão de quais caracteres serão lidos. Assim, a seqüência $i_1, \ldots, i_\ell$ é determinada apenas com base em x e τ, e portanto dizemos algumas vezes que o verificador V é *não-adaptativo*. No fim de sua computação, V deve dizer se aceita ou não a entrada x; a saída, denotada por $V(x, \tau, \Pi)$, será ACEITA ou REJEITA.

Sejam $r(n)$ e $q(n)$ duas funções reais definidas sobre os inteiros não negativos n. Um verificador V é dito ser $(r(n), q(n))$-*restrito* se existem duas funções inteiras $\hat{r}(n) = O(r(n))$ e $\hat{q}(n) = O(q(n))$ tais que V, com qualquer entrada x de comprimento $|x| = n$ e qualquer seqüência de bits aleatórios τ, faz uso em sua computação de não mais do que os primeiros $\hat{r}(n)$ bits de τ, e pede ao oráculo não mais que $\hat{q}(n)$ caracteres π_i de Π. No que segue, escrevemos $\mathbb{P}_\tau$ para denotar probabilidade com respeito à seqüência de bits aleatórios τ. Formalmente, embora os nossos verificadores sempre usem apenas um segmento inicial finito de τ, podemos assumir que τ tem comprimento infinito.

DEFINIÇÃO 17. *A linguagem $L \subset \Sigma^*$ está em $\mathrm{PCP}(r(n), q(n))$ se e só se existe um verificador $(r(n), q(n))$-restrito V para o qual valem as seguintes duas afirmativas.*

(*i*) *Para todo $x \in L$, existe uma prova $\Pi = \Pi_x \in \Sigma^*$ tal que*

$$\mathbb{P}_\tau\{V(x, \tau, \Pi) = \text{ACEITA}\} = 1. \tag{8}$$

(*ii*) *Para todo $x \notin L$ e para todo $\Pi \in \Sigma^*$, temos*

$$\mathbb{P}_\tau\{V(x, \tau, \Pi) = \text{ACEITA}\} < 1/4. \tag{9}$$

Note que na definição acima, o valor da constante $1/4$ não tem um papel muito importante. Substituindo-se aquele valor por qualquer constante $\theta < 1$, obteríamos para $\mathrm{PCP}(r(n), q(n))$ a mesma classe de linguagens. O motivo é simples: um verificador para o qual temos

$$\mathbb{P}_\tau\{V(x, \tau, \Pi_x) = \text{ACEITA}\} < \theta \tag{10}$$

pode ser transformado em um que satisfaz (9) acima por iteração.

Por conveniência, se Φ_1 e Φ_2 são duas classes de funções reais definidas sobre os inteiros não negativos, poremos

$$\mathrm{PCP}(\Phi_1, \Phi_2) = \bigcup_{f \in \Phi_1, g \in \Phi_2} \mathrm{PCP}(f, g).$$

Como de usual, $\mathrm{poly}(n)$ denota o conjunto de todos os polinômios $p(n)$.

Quando sistemas de provas PCP estão sob consideração, usamos o termo genérico *provas transparentes* para os certificados Π da Definição 17. Caso mais precisão seja necessária, quando uma linguagem em $\mathrm{PCP}(r(n), q(n))$ esteja sob consideração, referiremo-nos a Π_x como em (i) acima como uma *prova*, *demonstração*, ou *certificado* $(r(n), q(n))$*-transparente*.

O resultado central deste capítulo, devido a Arora, Lund, Motwani, Sudan, e Szegedy [ALM$^+$92b], pode ser formulado como segue.

Teorema 18. $\mathrm{PCP}(\log n, 1) = \mathrm{NP}$.

Já existem hoje versões mais finas do que o resultado acima. Mencionamos apenas duas delas. Para enunciar o primeiro refinamento, introduzimos a seguinte notação. Dizemos que uma linguagem $L \subset \Sigma^*$ pertence a classe $\mathrm{PCP}'(r(n), q(n))$ se existe um verificador associado V tal que todo $x \in L$ de comprimento $|x| = n$ admite uma prova transparente Π reconhecida por V usando $O(r(n))$ bits aleatórios e acessando *em média* não mais que $q(n)$ caracteres de Π. Lembre que os caracteres de Π a serem acessados no processo de verificação são definidos de acordo com x, a entrada, e τ, a seqüência de bits aleatórios. Assim, o número de caracteres a serem lidos é uma variável aleatória; a média acima refere-se às seqüências aleatórias τ. O seguinte refinamento do Teorema 18 segue de resultados provados por Bellare, Golwasser, Lund, e Russel [BGLR93, BGLR94].

Teorema 19. $\mathrm{PCP}'(\log n, 100) = \mathrm{NP}$.

Não é difícil de deduzir do resultado acima que não só o número *esperado* de caracteres da prova a serem lidos pode ser limitado por uma constante *universal*, mas também podemos garantir que *nunca* leremos mais do que uma certa quantidade constante de caracteres da prova.

O segundo refinamento do Teorema 18 é o seguinte resultado provado por Polishchuk e Spielman [PS94], em que o comprimento dos certificados transparentes é investigado.

Teorema 20. *Seja $\varepsilon > 0$ uma constante arbitrária e suponha que $L \in \mathrm{NP}$. Suponha que $x \in L$ e y é um certificado polinomial da pertinência de x à linguagem L. Ponha $n = |x| + |y|$. Então existe uma prova $(\log n, 1)$-transparente da pertinência de x à L de comprimento $O(n^{1+\varepsilon})$.*

A demonstração de Arora *et al.* do Teorema 18 fornece provas transparentes de comprimento superquadrático, e o resultado acima nos diz que provas transparentes de comprimento quase-linear existem.

Fechamos esta seção com o seguinte comentário. Podemos pensar no verificador V como sendo uma máquina de Turing que, dados x e τ, constrói em tempo polinomial um circuito booleano $C_\tau = C_{\tau,x}$ que tem variáveis de entrada $y = y_1 \dots y_\ell$. Quando o oráculo fornece os caracters $q_{i_1} \dots q_{i_\ell}$ da prova Π a V, o resultado $V(x, \tau, \Pi)$ da computação de V é o mesmo que o resultado que obtemos quando fornecemos a entrada $y = q_{i_1} \dots q_{i_\ell}$ a $C_\tau = C_{\tau,x}$. Esta formalização é usada por Arora *et al.* em [ALM+92b]. Aqui seguimos Arora *et al.* [ALM+92a] e Hougardy, Prömel, e Steger [HPS94], que adotam uma formalização mais combinatória.

2. A Estrutura da Demonstração

Descrevemos aqui a estrutura da demonstração do Teorema 18. Comecemos observando que a inclusão $\mathrm{PCP}(\log n, 1) \subset \mathrm{NP}$ é bastante simples. De fato, o resultado genérico abaixo vale. Lembre que denotamos por $\mathrm{NTIME}(f(n))$ a classe de linguagens reconhecidas por sistemas de prova conforme a Definição 4 da Seção 2.1 do Capítulo II. Como de usual, se Φ é uma classe de funções, pomos

$$\mathrm{NTIME}(\Phi) = \bigcup_{f \in \Phi} \mathrm{NTIME}(f).$$

Em particular, $\mathrm{NTIME}(\mathrm{poly}(n)) = \mathrm{NP}$.

Lema 21. $\mathrm{PCP}(r(n), q(n)) \subset \mathrm{NTIME}\left(\mathrm{poly}(n2^{r(n)})\right)$.

Prova. As seguintes observações simples são suficientes para provar o nosso lema. Suponha que temos uma linguagem $L \subset \Sigma^*$ que pertence à classe $\mathrm{PCP}(r(n), q(n))$, e seja V um verificador $(r(n), q(n))$-restrito associado. Seja dado $x \in L$ e suponha que Π é uma prova $(r(n), q(n))$-transparente da pertinência de x à L. Uma vez fixada a seqüência aleatória de bits τ, o comportamento de V é determinístico. Assim, basta provar que é possível simular o comportamento de V para *todos* os possíveis τ através de um sistema NTIME de provas em que o verificador leva tempo $O\left(\mathrm{poly}(n2^{r(n)})\right)$ no total. Note que há apenas $2^{O(r(n))}$ seqüências τ a serem consideradas. Cada uma delas define $O(q(n))$ posições da prova Π a serem examinadas. Assim, não mais que $O(q(n)2^{O(r(n))})$ entradas de Π serão examinadas no processo inteiro. Construa, a partir de Π, uma seqüência Π' contendo todos os caracters de Π 'relevantes' para o processo de verificação executado por V. Simulando o comportamento de V sobre todos os possíveis τ, usando Π' como a fita do provador no sistema NTIME de provas, podemos decidir se x pertence ou não a L. De fato, x pertence a L se e só se V aceita x para *todo* τ. Claramente, o verificador no sistema NTIME de provas leva não mais que tempo $n^{O(1)}2^{O(r(n))}$ para executar a simulação descrita acima. □

Segue do Lema 21 que $\mathrm{PCP}(\log n, 1) \subset \mathrm{NP}$. O 'conteúdo' do Teorema 18 é que a inclusão reversa vale. Seja $L \in \Sigma^*$ uma linguagem NP. Para demonstrarmos que $L \in \mathrm{PCP}(\log n, 1)$, devemos (i) descrever como obter uma prova transparente Π_x para todo $x \in L$ e (ii) descrever um verificador $(\log n, 1)$-restrito V associado que reconhece estas demonstrações como tais, e que, se $x \notin L$, não pode ser ludibriado com probabilidade $\geq 1/4$ por 'demonstrações incorretas' de que $x \in L$.

Suponha que $L \subset \Sigma^*$ é uma linguagem em NP e que $x \in L$. Então sabemos que existe um certificado polinomial $y_x \in \Sigma^*$ para a pertinência de x em L. Para demonstrarmos que $L \in \mathrm{PCP}(r(n), q(n))$ para $r(n)$ e $q(n)$ dados, *codificaremos* y_x através de uma função cuidadosamente construída $E\colon \Sigma^* \to \Sigma^*$ de forma a obter uma prova transparente $\Pi_x = E(x)$ para a pertinência de x em L.

Basicamente, a demonstração de que $\mathrm{NP} \subset \mathrm{PCP}(\log n, 1)$ é composta de três passos.

(1) Exibe-se primeiro uma codificação $E\colon \Sigma^* \to \Sigma^*$ e um verificador $(\mathrm{poly}(n), 1)$-restrito associado V que demonstram conjuntamente que
$$\mathrm{NP} \subset \mathrm{PCP}(\mathrm{poly}(n), 1).$$
(2) Exibe-se então uma codificação $E'\colon \Sigma^* \to \Sigma^*$ e um verificador associado $(\log n, \mathrm{poly}(\log n))$-restrito V' que provam que
$$\mathrm{NP} \subset \mathrm{PCP}(\log n, \mathrm{poly}(\log n)).$$
(3) Mostra-se finalmente que as codificações E e E' e os verificadores V e V' acima podem ser 'compostos' de forma a se obter uma codificação E'' e um verificador $(\log n, 1)$-restrito V'', demonstrando-se assim que $\mathrm{NP} \subset \mathrm{PCP}(\log n, 1)$.

No resto deste capítulo, discutimos os três passos acima.

3. A Demonstração

3.1. Certificados transparentes I. Objetivamos provar aqui que $\mathrm{NP} \subset \mathrm{PCP}(\mathrm{poly}(n), 1)$. Seja $L \subset \Sigma^*$ uma linguagem em NP. Precisamos provar que todo $x \in L$ admite um certificado $(\mathrm{poly}(n), 1)$-transparente Π_x e precisamos contruir um verificador $(\mathrm{poly}(n), 1)$-restrito associado V para reconhecer tais certificados.

Note que os certificados Π_x de que trataremos aqui têm uma característica comum com os certificados cuja existência está implícita no Teorema 18. De fato, os certificados Π_x não precisam ser lidos inteiros. Muito pelo contrário, para que o verificador fique satisfeito de que Π_x é um certificado válido da afirmativa '$x \in L$', basta que ele leia um número $O(1)$ de entradas de Π_x.

Por outro lado, cabe ressaltar que os verificadores desta subseção exigem acesso a um número *polinomial* de bits aleatórios, e não apenas logarítmico. Veremos que este fato é uma conseqüência do comprimento dos certificados Π_x que construiremos. De fato, os Π_x que construiremos abaixo terão tamanho exponencial no tamanho de x. Note que isto implica que, se quisermos acessar uma entrada de Π_x escolhida uniformemente ao acaso, precisamos fazer uso de uma quantidade polinomial de bits aleatórios.

Concluímos com uma observação que nos porá na direção certa para começarmos a demonstração da inclusão $\mathrm{NP} \subset \mathrm{PCP}(\mathrm{poly}(n), 1)$. Recordemos o resultado clássico que afirma que 3SAT é um problema

NP-completo (veja o Teorema 53 da Seção 2 do Capítulo V). Desta forma, para mostrarmos a inclusão acima, basta que provemos que toda instância satisfatível x de 3SAT admite um certificado $(\mathrm{poly}(n), 1)$-transparente e que construamos um verificador associado apropriado. O certificado que construiremos abaixo será $(n^3, 1)$-transparente.

3.1.1. *A aritmetização.* O primeiro passo para demonstrarmos que 3SAT $\in \mathrm{PCP}(n^3, 1)$ é a *aritmetização* deste problema. Esta idéia apareceu na demonstração do resultado de que IP = PSPACE (Teorema 16 do Capítulo III). De fato, uma idéia básica lá foi a de tratar fórmulas booleanas como polinômios—a idéia aqui é a mesma. Seja φ uma instância de 3SAT com m cláusulas e n variáveis. Assim φ é uma família de m cláusulas C_i $(1 \leq i \leq m)$, onde cada cláusula C_i é uma família de três literais $\ell_j^{(i)}$ $(j \in \{1, 2, 3\})$. Cada literal é uma variável x_k ou a sua negação $\neg x_k$ $(1 \leq k \leq n)$. A interpretação de φ é que ela representa a fórmula booleana

$$(\ell_1^{(1)} \vee \ell_2^{(1)} \vee \ell_3^{(1)}) \wedge \cdots \wedge (\ell_1^{(m)} \vee \ell_2^{(m)} \vee \ell_3^{(m)}). \tag{11}$$

O problema 3SAT consiste em descobrir se φ admite uma atribuição de valores $a\colon U = \{x_k\colon 1 \leq k \leq n\} \to \{\mathrm{V}, \mathrm{F}\}$ tal que $\varphi(a) = \mathrm{V}$. Um argumento simples mostra que podemos nos restringir aqui a instâncias 3SAT com $m = n$. No que segue, sempre admitiremos isto.

A *aritmetização* $\mathcal{C} = \mathcal{C}(\varphi)$ de φ que consideraremos aqui é a seguinte: $\mathcal{C}$ será um vetor $(\mathcal{C}_1(x), \ldots, \mathcal{C}_n(x))$ de n polinômios $\mathcal{C}_i(x)$ $(1 \leq i \leq n)$ sobre as variáveis $x = (x_1, \ldots, x_n)$. Aqui usamos a mesma notação para as variáveis booleanas de φ e para as indeterminadas de $\mathcal{C}$. O polinômio $\mathcal{C}_i(x)$ é a aritmetização da negação $\neg C_i$ de C_i no sentido encontrado na demonstração do Teorema III.16.

Por exemplo, se

$$\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_4), \tag{12}$$

então

$$\mathcal{C}(\varphi) = \big((1 - x_1)x_2x_3, x_2(1 - x_3)(1 - x_4), x_1x_2(1 - x_4)\big). \tag{13}$$

Note que há, no total, quatro possibilidades para cada um dos polinômios $\mathcal{C}_i(x)$, a menos das indeterminadas envolvidas.

No que segue, sempre trabalhamos no corpo $\mathbb{F}_2 = \mathrm{GF}(2)$. Uma observação imediata mas importante é que $a\colon U \to \{\mathrm{V}, \mathrm{F}\}$ é uma atribuição que satisfaz φ se e só se o vetor em $\mathbb{F}_2^n$ naturalmente associado a a é tal que $\mathcal{C}(a) = 0 \in \mathbb{F}_2^n$. Desta forma, podemos convencer alguém de que φ é satisfatível convencendo-o de que $\mathcal{C}(a) = 0$ para algum vetor $a \in \mathbb{F}_2^n$. A idéia agora é que nós conhecemos $a \in \mathbb{F}_2^n$ tal que $\mathcal{C}(a) = 0$, e que queremos escrever uma prova transparente deste fato.

3.1.2. *Discussão informal.* O contexto em que trabalhamos aqui é o seguinte. Temos uma instância φ de 3SAT com n variáveis x_i $(1 \leq i \leq n)$ e n cláusulas C_i $(1 \leq i \leq n)$. A aritmetização de φ é $\mathcal{C} = \mathcal{C}(\varphi) = (\mathcal{C}_i(x))_{1\leq i\leq n}$, onde cada $\mathcal{C}_i(x)$ é um polinômio nas indeterminadas $x = (x_i)_{1\leq i\leq n}$. Mais precisamente, cada $\mathcal{C}_i(x)$ é um polinômio com no máximo três indeterminadas e é, a menos das indeterminadas, um dos seguintes polinômios: $P_1 = xyz$, $P_2 = xy(1-z)$, $P_3 = x(1-y)(1-z)$, $P_4 = (1-x)(1-y)(1-z)$. Finalmente, supomos que temos uma atribuição de valores a para as variáveis de φ que satisfaz φ e, assim, que temos $a = (a_i)_{1\leq i\leq n} \in \mathbb{F}_2^n$ tal que $\mathcal{C}(a) = 0$.

Consideremos $\langle \mathcal{C}(a), r\rangle$ para um vetor $r = (r_i)_{1\leq i\leq n} \in \mathbb{F}_2^n$, onde

$$\langle u, v\rangle = \langle (u_i)_{1\leq i\leq n}, (v_i)_{1\leq i\leq n}\rangle = \sum_{1\leq i\leq n} u_i v_i$$

denota o produto escalar canônico em $\mathbb{F}_2^n$. Note que

$$\langle \mathcal{C}(a), r\rangle = c + \sum_{i\in S_1(r)} a_i + \sum_{(i,j)\in S_2(r)} a_i a_j + \sum_{(i,j,k)\in S_3(r)} a_i a_j a_k, \quad (14)$$

onde $c = c_r \in \mathbb{F}_2$, $S_1(r) \subset [n]$, $S_2(r) \subset [n]^2$, e $S_3(r) \subset [n]^3$ são univocamente determinados por r apenas e, em particular, independem de a. Acima, como de costume, $[n] = \{1, \ldots, n\}$. Desta forma, se temos acesso aos valores de

$$\sum_{i\in S_1} a_i, \qquad \sum_{(i,j)\in S_2} a_i a_j, \qquad \sum_{(i,j,k)\in S_3} a_i a_j a_k \quad (15)$$

para todos os possíveis conjuntos $S_1 \subset [n]$, $S_2 \subset [n]^2$, e $S_3 \subset [n]^3$, o valor de $\langle \mathcal{C}(a), r\rangle$ pode ser facilmente obtido para qualquer $r \in \mathbb{F}_2^n$. O certificado transparente $\Pi = \Pi_\varphi$ de que φ é satisfatível será composto dos valores das somas em (15) para todos os possíveis S_1, S_2, e S_3.

Como deve agir o verificador para interpretar este certificado? A idéia é simples: para que ele fique satisfeito com o fato de que $\mathcal{C}(a) = 0$, basta-lhe calcular $\langle \mathcal{C}(a), r\rangle$ para vários $r \in \mathbb{F}_2^n$ escolhidos ao acaso. Note que, devido a (14), cada um destes cálculos pode ser feito consultando-se o certificado Π_φ apenas três vezes. Ademais, cada um destes produtos escalares é trivialmente nulo caso $\mathcal{C}(a) = 0$. O verificador deve aceitar φ se e só se todos estes produtos forem de fato nulos.

Consideremos agora o caso em que φ *não é satisfatível*, e que o verificador recebeu algum $\Pi \in \Sigma^*$ como um certificado da satisfatibilidade de φ. O primeiro passo do verificador é sempre fazer uma análise de coerência de Π. Isto é, ele deve se satisfazer de que Π tem o 'formato' correto. Mais especificamente, ele deve se certificar de que Π de fato apresenta as somas em (15) para algum $a \in \mathbb{F}_2^n$. Uma vez convencido deste fato, o verificador irá então calcular $\langle \mathcal{C}(a), r\rangle$ para vários $r \in \mathbb{F}_2^n$ escolhidos ao acaso, usando as somas em (15).

A observação simples mas crucial neste ponto é que, como $\mathcal{C}(a)$ não é nulo, o produto $\langle \mathcal{C}(a), r\rangle$ tem probabilidade $1/2$ de ser não nulo quando escolhemos $r \in \mathbb{F}_2^n$ ao acaso. Assim, se o verificador calcula $\langle \mathcal{C}(a), r\rangle$ para três vetores $r \in \mathbb{F}_2^n$ escolhidos independentemente, a probabilidade de todos os produtos escalares serem nulos é $1/8$. Em outras palavras, o verificador será ludibriado com probabilidade no máximo $1/8$. Devemos agora formalizar a discussão acima.

3.1.3. *Codificações lineares.* Como na discussão informal acima, suponha que $\mathcal{C}(a) = 0$. Comecemos observando que as somas em (15) podem ser codificadas como três aplicações lineares. De fato, ponha $b_{ij} = a_i a_j$ e $c_{ijk} = a_i a_j a_k$ $(1 \le i,\ j,\ k \le n)$, e

$$b = a \circ a = (b_{ij})_{1\le i,j\le n} \in \mathbb{F}_2^{n^2}, \quad c = a \circ a \circ a = (c_{ijk})_{1\le i,j,k\le n} \in \mathbb{F}_2^{n^3}, \tag{16}$$

e considere as aplicações

$$A\colon \mathbb{F}_2^n \to \mathbb{F}_2, \qquad B\colon \mathbb{F}_2^{n^2} \to \mathbb{F}_2, \qquad C\colon \mathbb{F}_2^{n^3} \to \mathbb{F}_2, \tag{17}$$

dadas por

$$A(x) = \langle a, x\rangle, \qquad B(y) = \langle b, y\rangle, \qquad C(z) = \langle c, z\rangle \tag{18}$$

para todo $x \in \mathbb{F}_2^n$, $y \in \mathbb{F}_2^{n^2}$, e $z \in \mathbb{F}_2^{n^3}$. O certificado transparente Π_φ de que φ é satisfatível será então o conjunto das três aplicações lineares homogêneas A, B e C, dadas como uma seqüência 0–1 de comprimento $2^n + 2^{n^2} + 2^{n^3}$. Mais precisamente, pensamos em Π_φ como um elemento de Σ^* daquele comprimento, com cada um de seus caracteres indexados por um elemento de $\mathbb{F}_2^n \cup \mathbb{F}_2^{n^2} \cup \mathbb{F}_2^{n^3}$.

Devemos agora descrever um verificador $(n^3, 1)$-restrito V que reconhece tais certificados Π_φ.

3.1.4. *O verificador.* Suponha que temos uma instância φ de 3SAT como na discussão informal acima. Temos ainda $\Pi \in \Sigma^*$ de comprimento $2^n + 2^{n^2} + 2^{n^3}$, fornecida a nós como um certificado da satisfatibilidade de φ como descrito acima. O nosso objetivo é descrever um processo que verifica se Π é de fato um certificado 'correto'.

O nosso procedimento é composto de três passos, que podem ser descritos, a grosso modo, como segue.

1. Verificamos primeiro se Π codifica três aplicações lineares A, B, e C como em (17). Caso descubramos que este não é o caso, rejeitamos φ e o procedimento termina.
2. Verificamos agora se os vetores a, b, e c associados às aplicações lineares A, B, e C satisfazem as relações de consistência (16). Caso descubramos que elas não são satisfeitas, o procedimento termina rejeitando φ.
3. Finalmente, convencidos de que Π codifica um vetor $a \in \mathbb{F}_2^n$ como esperado, verificamos probabilisticamente se $\mathcal{C}(a) = 0$. A saber, verificamos se $\langle \mathcal{C}(a), r \rangle = 0$ para um número suficientemente grande de $r \in \mathbb{F}_2^n$ escolhidos independentemente ao acaso. Caso todos os produtos escalares calculados sejam nulos, aceitamos φ e, caso contrário, rejeitamos φ.

Tendo em mente a discussão informal acima, deve ser claro que o Passo 3 pode ser executado sem problemas por um verificador $(n^3, 1)$-restrito. Passamos agora a discutir os Passos 1 e 2.

3.1.5. *O teste de linearidade.* Descrevemos aqui um procedimento surpreendentemente simples para implementar o Passo 1 acima do nosso processo de verificação. Dado que Π tem comprimento $2^n + 2^{n^2} + 2^{n^3}$, podemos assumir que Π representa três funções

$$\widetilde{A}\colon \mathbb{F}_2^n \to \mathbb{F}_2, \qquad \widetilde{B}\colon \mathbb{F}_2^{n^2} \to \mathbb{F}_2, \qquad \widetilde{C}\colon \mathbb{F}_2^{n^3} \to \mathbb{F}_2. \tag{19}$$

Note que, trivialmente, se devemos consultar apenas $O(1)$ entradas de Π, não há como verificar se $\widetilde{A}$, $\widetilde{B}$, e $\widetilde{C}$ são 'exatamente' lineares. O que fazemos é verificar se elas são *próximas* a funções lineares. No que segue, todas as nossas funções lineares serão homogêneas.

Sejam f e g duas funções definidas sobre um conjunto finito Ω, e seja $0 \leq \delta \leq 1$ um número real. Dizemos que f e g são *δ-próximos* se $f(\omega) = g(\omega)$ para pelo menos $(1-\delta)|\Omega|$ elementos $\omega \in \Omega$. Equivalentemente, tomando-se a distribuição uniforme sobre Ω, temos que f e g são δ-próximos se

$$\mathbb{P}_{\omega \in_{\mathrm{R}} \Omega}\{f(\omega) = g(\omega)\} \geq 1 - \delta,$$

onde escrevemos $\mathbb{P}_{\omega \in_{\mathrm{R}} \Omega}$ ou simplesmente $\mathbb{P}_\omega$ para denotar probabilidade sobre o espaço uniforme Ω. No que segue, escrevemos $\omega \in_{\mathrm{R}} \Omega$ quando ω é escolhido uniformemente ao acaso em Ω.

O que é feito no Passo 1 do processo de verificação consiste em descobrir se $\widetilde{A}$, $\widetilde{B}$, e $\widetilde{C}$ são δ-próximos a funções lineares A, B, e C, para algum $\delta > 0$ cuidadosamente escolhido. Esta verificação pode ser feita com base no seguinte resultado.

Teorema 22. *Seja $0 \leq \delta < 1/3$ uma constante, e seja $\widetilde{f}\colon \mathbb{F}_2^n \to \mathbb{F}_2$ uma função tal que*

$$\mathbb{P}_{x,y}\{\widetilde{f}(x+y) \neq \widetilde{f}(x) + \widetilde{f}(y)\} \leq \delta/2,$$

onde x, $y \in \mathbb{F}_2^n$ são independentemente escolhidos uniformemente ao acaso. Então existe uma única função linear homogênea $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ que é δ-próxima à função $\widetilde{f}$.

Prova. Defina a função $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ pondo

$$f(x) = \begin{cases} 0 & \text{se } \mathbb{P}_y\{\widetilde{f}(x+y) - \widetilde{f}(y) = 0\} \\ & \qquad\qquad \geq \mathbb{P}_y\{\widetilde{f}(x+y) - \widetilde{f}(y) = 1\} \\ 1 & \text{caso contrário.} \end{cases}$$

Provamos abaixo que esta funcão f é δ-próxima à $\widetilde{f}$ e que ela é linear homogênea. Suponha inicialmente por absurdo que f e $\widetilde{f}$ não são δ-próximas. Então temos que $\mathbb{P}_x\{f(x) \neq \widetilde{f}(x)\} > \delta$. Por outro lado, pela definição de f, temos que $\mathbb{P}_y\{f(x) = \widetilde{f}(x+y) - \widetilde{f}(y)\} \geq 1/2$ para todo $x \in \mathbb{F}_2^n$. Deduzimos que

$$\mathbb{P}_{x,y}\{\widetilde{f}(x+y) \neq \widetilde{f}(x) + \widetilde{f}(y)\} > \delta/2,$$

o que contradiz a hipótese do nosso lema. Assim concluímos que f e $\widetilde{f}$ são de fato δ-próximas. Provemos agora a linearidade de f.

Para todo $a \in \mathbb{F}_2^n$, ponha

$$p_a = \mathbb{P}_x\{f(a) = \widetilde{f}(a+x) - \widetilde{f}(x)\}.$$

Observamos acima que, claramante, $p_a \geq 1/2$. Afirmamos agora que $p_a \geq 1-\delta$ para todo $a \in \mathbb{F}_2^n$. Para provar esta afirmativa, fixe a e observe que temos

$$\mathbb{P}_{x,y}\{\widetilde{f}(x+a) + \widetilde{f}(y) \neq \widetilde{f}(x+a+y)\} \leq \delta/2,$$

e

$$\mathbb{P}_{x,y}\{\widetilde{f}(x) + \widetilde{f}(a+y) \neq \widetilde{f}(x+a+y)\} \leq \delta/2,$$

e assim

$$\mathbb{P}_{x,y}\{\widetilde{f}(a+x) - \widetilde{f}(x) \neq \widetilde{f}(a+y) - \widetilde{f}(y)\} \leq \delta.$$

Portanto, temos que

$$\begin{aligned} 1-\delta &\leq \mathbb{P}_{x,y}\{\widetilde{f}(a+x) - \widetilde{f}(x) = \widetilde{f}(a+y) - \widetilde{f}(y)\} \\ &= \sum_{z \in \{0,1\}} \mathbb{P}_x\{\widetilde{f}(a+x) - \widetilde{f}(x) = z\}^2 = p_a^2 + (1-p_a)^2. \end{aligned}$$

Lembrando que $p_a \geq 1/2$, temos que

$$1-\delta \leq p_a^2 + (1-p_a)^2 \leq p_a^2 + p_a(1-p_a) = p_a,$$

como afirmamos acima. Usamos agora este fato para demonstrar que f é linear.

Fixe a, $b \in \mathbb{F}_2^n$. Temos pela afirmativa acima que

$$\begin{aligned} p_a = \mathbb{P}_x\{f(a) + f(b) + \widetilde{f}(x) \neq \widetilde{f}(a+x) + f(b)\} &\leq \delta, \\ p_b = \mathbb{P}_x\{f(b) + \widetilde{f}(a+x) \neq \widetilde{f}(a+b+x)\} &\leq \delta, \\ p_{a+b} = \mathbb{P}_x\{\widetilde{f}(a+b+x) \neq f(a+b) + \widetilde{f}(x)\} &\leq \delta. \end{aligned}$$

Segue que

$$\mathbb{P}_x\{f(a) + f(b) + \widetilde{f}(x) = f(a+b) + \widetilde{f}(x)\} \geq 1 - 3\delta > 0. \quad (20)$$

Como o evento considerado no lado esquerdo de (20) é *independente* de x, temos que a probabilidade correspondente é 0 ou 1, e daí segue o resultado. □

Um teste de linearidade imediato do Teorema 22 é dado na Figura IV.8.

Escolha x, $x' \in_{\mathrm{R}} \mathbb{F}_2^n$, e verifique se $\widetilde{A}(x+x') = \widetilde{A}(x) + \widetilde{A}(x')$. Escolha y, $y' \in_{\mathrm{R}} \mathbb{F}_2^{n^2}$, e verifique se $\widetilde{B}(y+y') = \widetilde{B}(y) + \widetilde{B}(y')$. Escolha z, $z' \in_{\mathrm{R}} \mathbb{F}_2^{n^3}$, e verifique se $\widetilde{C}(z+z') = \widetilde{C}(z) + \widetilde{C}(z')$.

FIGURA IV.8. O TESTE DE LINEARIDADE

Naturalmente, se uma das igualdades no TESTE DE LINEARIDADE não for satisfeita, o verificador deve rejeitar Π. Por outro lado, segue do Teorema 22 que se uma das funções $\widetilde{A}$, $\widetilde{B}$, e $\widetilde{C}$ não for δ-próxima a uma função linear, então a probabilidade de o teste correspondente no TESTE DE LINEARIDADE falhar é maior que $\delta/2$. Iterando-se este teste um número suficientemente grande de vezes, podemos aumentar esta probabilidade para qualquer $\theta < 1$. Note que, desde que $\theta < 1$ seja uma constante real fixa, basta iterar o teste acima um número $O(1)$ de vezes.

TEOREMA 23. *Seja $0 < \delta < 1/3$ uma constante fixa. Então existe um inteiro $k = k(\delta) \geq 1$ que depende apenas de δ para o qual vale o seguinte. Se não existem $a \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2^{n^2}$, ou $c \in \mathbb{F}_2^{n^3}$ tais que as funções $\widetilde{A}$ e A, $\widetilde{B}$ e B, e $\widetilde{C}$ e C são δ-próximas, onde $A(x) = \langle a, x\rangle$ ($x \in \mathbb{F}_2^n$), $B(x) = \langle b, y\rangle$ ($y \in \mathbb{F}_2^{n^2}$), e $C(z) = \langle c, z\rangle$ ($z \in \mathbb{F}_2^{n^3}$), então, iterando-se o* TESTE DE LINEARIDADE *k vezes, a probabilidade de o teste falhar é de pelo menos $1 - \delta$.*

3.1.6. *O teste de consistência.* Suponha agora que $\widetilde{A}$, $\widetilde{B}$, e $\widetilde{C}$ passaram pelo TESTE DE LINEARIDADE. Assim podemos assumir que existem funções lineares A, B, e C como no Teorema 23. Sejam $a \in \mathbb{F}_2^n$, $b \in \mathbb{F}_2^{n^2}$, e $c \in \mathbb{F}_2^{n^3}$ como naquele teorema. O nosso objetivo agora é verificar se estes vetores satisfazem o critério de consistência dado por (16). Como anteriormente, se x, $x' \in \mathbb{F}_2^n$, escrevemos $x \circ x'$ para o vetor $y = (y_{ij})_{1\leq i,j\leq n}$ com $y_{ij} = x_i x_j$ $(1 \leq i$, $j \leq n)$. Ademais, se $y = (y_{jk})_{1\leq j,k\leq n} \in \mathbb{F}_2^{n^2}$, escrevemos $x \circ y$ para o vetor $z = (z_{ijk})_{1\leq i,j,k\leq n}$ com coordenadas $z_{ijk} = x_i y_{jk}$ $(1 \leq i$, j, $k \leq n)$. Note agora que

$$A(x)A(x') = B(x \circ x') \tag{21}$$

para todo x, $x' \in \mathbb{F}_2^n$ se e só se $b = a \circ a$, e que

$$A(x)B(y) = C(x \circ y) \tag{22}$$

para todo $x \in \mathbb{F}_2^n$ e todo $y \in \mathbb{F}_2^{n^2}$ se e só se $c = a \circ b$. As identidades (21) e (22) são candidatas naturais para a verificação da consistência entre os vetores a, b, e c. Naturalmente, um verificador $(n^3, 1)$-restrito não pode verificar estas identidades para *todos* os x, x', e y. Entretanto, a seguinte observação é suficiente para se resolver este problema.

LEMA 24. *Suponha que A, B, e C são funções lineares dadas por a, b, e c como acima, mas que $b \neq a \circ a$ e $c \neq a \circ b$. Então*

$$\mathbb{P}_{x,x'}\{A(x)A(x') \neq B(x \circ x')\} \geq 1/4, \tag{23}$$

onde x, $x' \in_{\mathrm{R}} \mathbb{F}_2^n$, e

$$\mathbb{P}_{x,y}\{A(x)B(y) \neq C(x \circ y)\} \geq 1/4, \tag{24}$$

onde $x \in_{\mathrm{R}} \mathbb{F}_2^n$ e $y \in_{\mathrm{R}} \mathbb{F}_2^{n^2}$.

PROVA. Provemos (23). Por conveniência, considere $a \circ a \in \mathbb{F}_2^{n^2}$ e $b = (b_{ij}) \in \mathbb{F}_2^{n^2}$ como matrizes $n \times n$ da forma natural. Temos assim $A(x)A(x') = \langle x, (a \circ a)x' \rangle$ e $B(x \circ x') = \langle x, bx' \rangle$ para quaisquer x, $x' \in \mathbb{F}_2^n$. Usando o fato de que dois funcionais lineares distintos sobre $\mathbb{F}_2^n$ concordam em exatamente metade dos pontos de $\mathbb{F}_2^n$, temos do fato de que $b \neq a \circ a$ que

$$\mathbb{P}_{x'}\{(a \circ a)x' \neq bx'\} \geq 1/2,$$

e também que

$$\mathbb{P}_x\{\langle x, (a \circ a)x' \rangle \neq \langle x, bx' \rangle\} = 1/2$$

para todo $x' \in \mathbb{F}_2^n$ tal que $(a \circ a)x' \neq bx'$. Daí segue (23). A desigualdade (24) segue de forma análoga. $\square$

O uso do Lema 24 apresenta, entretanto, uma dificuldade. Lembre que não temos acesso às funções A, B, e C, mas às funções δ-próximas respectivas $\widetilde{A}$, $\widetilde{B}$, e $\widetilde{C}$. Em particular, embora seja imediato que

$$\mathbb{P}_{x,x'}\{A(x)A(x') \neq \widetilde{A}(x)\widetilde{A}(x')\} \leq 2\delta$$

para x, $x' \in_{\mathrm{R}} \mathbb{F}_2^n$, não podemos em geral afirmar que

$$\mathbb{P}_{x,x'}\{B(x \circ x') \neq \widetilde{B}(x \circ x')\} \leq 2\delta,$$

já que $x \circ x'$ não é um elemento uniformemente distribuído em $\mathbb{F}_2^{n^2}$ quando x, $x' \in_{\mathrm{R}} \mathbb{F}_2^n$. Este problema pode ser contornado com o uso de funções 'auto-corretoras'. Defina as funções *aleatórias*

$$\text{SC-}\widetilde{A}\colon \mathbb{F}_2^n \to \mathbb{F}_2, \qquad \text{SC-}\widetilde{B}\colon \mathbb{F}_2^{n^2} \to \mathbb{F}_2, \qquad \text{SC-}\widetilde{C}\colon \mathbb{F}_2^{n^3} \to \mathbb{F}_2$$

como dadas na Figura IV.9.

SC-$\widetilde{A}(x)$:	Escolha $r \in_{\mathrm{R}} \mathbb{F}_2^n$, e devolva $\widetilde{A}(x+r) - \widetilde{A}(r)$.
SC-$\widetilde{B}(y)$:	Escolha $r \in_{\mathrm{R}} \mathbb{F}_2^{n^2}$, e devolva $\widetilde{B}(y+r) - \widetilde{B}(r)$.
SC-$\widetilde{C}(z)$:	Escolha $r \in_{\mathrm{R}} \mathbb{F}_2^{n^3}$, e devolva $\widetilde{C}(z+r) - \widetilde{C}(r)$.

FIGURA IV.9. As Funções Auto-Corretoras

A observação simples que torna estas funções importantes para nós é a seguinte. Lembre que $\widetilde{A}$, $\widetilde{B}$, e $\widetilde{C}$ são δ-próximas às funções lineares A, B, e C.

LEMA 25. *Para todo $x \in \mathbb{F}_2^n$, $y \in \mathbb{F}_2^{n^2}$, e $z \in \mathbb{F}_2^{n^3}$, temos*

$$\mathbb{P}\{\,\text{SC-}\widetilde{A}(x) = A(x)\} \geq 1 - 2\delta, \quad \mathbb{P}\{\,\text{SC-}\widetilde{B}(y) = B(y)\} \geq 1 - 2\delta,$$

e

$$\mathbb{P}\{\,\text{SC-}\widetilde{C}(z) = C(z)\} \geq 1 - 2\delta.$$

Naturalmente, as probabilidades no lema acima referem-se aos valores das funções aleatórias SC-$\widetilde{A}$, SC-$\widetilde{B}$, e SC-$\widetilde{C}$. Note agora que se $\widetilde{A}$, $\widetilde{B}$, e $\widetilde{C}$ são δ-próximas a A, B, e C com a, b, e c tais que $b \neq a \circ a$ e $c \neq a \circ b$, então, pelos Lemas 24 e 25, temos que

$$\mathbb{P}_{x,x'}\{\,\text{SC-}\widetilde{A}(x)\,\text{SC-}\widetilde{A}(x') \neq \text{SC-}\widetilde{B}(x \circ x')\} \geq 1/4 - 6\delta$$

para x, $x' \in_{\mathrm{R}} \mathbb{F}_2^n$, e

$$\mathbb{P}_{x,y}\{\,\text{SC-}\widetilde{A}(x)\,\text{SC-}\widetilde{B}(y) \neq \text{SC-}\widetilde{B}(x \circ y)\} \geq 1/4 - 6\delta,$$

para $x \in_{\mathrm{R}} \mathbb{F}_2^n$, $y \in_{\mathrm{R}} \mathbb{F}_2^{n^2}$. A discussão acima sugere o teste de consistência para as aplicações lineares A, B, e C descrito na Figura IV.10.

(1) Escolha x, $x' \in_{\mathrm{R}} \mathbb{F}_2^n$, e verifique se SC-$\widetilde{A}(x)$ SC-$\widetilde{A}(x')$ = SC-$\widetilde{B}(x \circ x')$.
(2) Escolha $x \in_{\mathrm{R}} \mathbb{F}_2^n$ e $y \in_{\mathrm{R}} \mathbb{F}_2^{n^2}$, e verifique se SC-$\widetilde{A}(x)$ SC-$\widetilde{B}(y)$ = SC-$\widetilde{C}(x \circ y)$.

Figura IV.10. O Teste de Consistência

Podemos agora enunciar formalmente o resultado que permite ao verificador executar os Passos 1 e 2 do processo de verificação.

Lema 26. *Seja $0 < \delta < 1/24$ uma constante fixa. Então existe um inteiro $k = k(\delta) \geq 1$ que depende apenas de δ para o qual vale o seguinte. Se não existe $a \in \mathbb{F}_2^n$ tal que as funções $\widetilde{A}$ e A, $\widetilde{B}$ e B, e $\widetilde{C}$ e C são δ-próximas, onde $A(x) = \langle a, x\rangle$ $(x \in \mathbb{F}_2^n)$, $B(x) = \langle a \circ a, y\rangle$ $(y \in \mathbb{F}_2^{n^2})$, e $C(z) = \langle a \circ a \circ a, z\rangle$ $(z \in \mathbb{F}_2^{n^3})$, então, iterando-se o* Teste de Linearidade *e o* Teste de Consistência *k vezes cada, a probabilidade de um destes testes falhar é de pelo menos $1 - \delta$.*

(1) Escolha $r \in_{\mathrm{R}} \mathbb{F}_2^n$ e calcule $c = c_r$, $S_1 = S_1(r) \in \mathbb{F}_2^n$, $S_2 = S_2(r) \in \mathbb{F}_2^{n^2}$, e $S_3 = S_3(r) \in \mathbb{F}_2^{n^3}$ como em (14).
(2) Verifique se $c + \widetilde{A}(S_1) + \widetilde{B}(S_2) + \widetilde{C}(S_3) = 0$.

FIGURA IV.11. O TESTE DE SATISFATIBILIDADE

3.1.7. *O teste de satisfatibilidade.* Resta-nos descrever o processo pelo qual executamos o Passo 3 do algoritmo de verificação. Este passo é, na verdade, bastante simples. Considere o procedimento dado na Figura IV.11.

O resultado formal que nos garante a 'correção' do TESTE DE SATISFATIBILIDADE é o seguinte.

LEMA 27. *Seja $0 < \delta < 1$ uma constante fixa. Então existe um inteiro $k = k(\delta) \geq 1$ que depende apenas de δ para o qual vale o seguinte. Se $a \in \mathbb{F}_2^n$ é tal que $\mathcal{C}(a) \neq 0$, mas as funções $\widetilde{A}$ e A, $\widetilde{B}$ e B, e $\widetilde{C}$ e C são δ-próximas, onde $A(x) = \langle a, x\rangle$ $(x \in \mathbb{F}_2^n)$, $B(x) = \langle a \circ a, y\rangle$ $(y \in \mathbb{F}_2^{n^2})$, e $C(z) = \langle a \circ a \circ a, z\rangle$ $(z \in \mathbb{F}_2^{n^3})$, então, iterando-se o* TESTE DE SATISFATIBILIDADE *k vezes, a probabilidade de um destes testes falhar é de pelo menos $1 - \delta$.*

Os Lemas 26 e 27 fornecem conjuntamente o seguinte resultado.

TEOREMA 28. *Existe uma constante absoluta $k \geq 1$ tal que a iteração do* TESTE DE LINEARIDADE, *do* TESTE DE CONSISTÊNCIA, *e do* TESTE DE SATISFATIBILIDADE *k vezes cada, e aceitando φ se e só se nenhum destes testes detecta inconsistência na prova Π apresentada, constitui um verificador $(n^3, 1)$-restrito para* 3SAT.

Devido à NP-completude de 3SAT, temos o seguinte resultado.

COROLÁRIO 29. $\mathrm{NP} \subset \mathrm{PCP}(\mathrm{poly}(n), 1)$.

3.1.8. *Codificação de soluções.* Por motivos técnicos, precisamos detalhar uma característica do verificador $(n^3, 1)$-restrito que descrevemos acima. A idéia aqui é a seguinte. A demonstração transparente Π_φ acima da satisfatibilidade de φ é uma seqüência que descreve as três funções lineares A, B, e C. Podemos pensar em A como sendo uma codificação do vetor a, que é por sua vez uma representação de uma atribuição que satisfaz φ. Será importante na Secão 3.3 abaixo pensar em A como uma codificação de uma 'solução' do problema de satisfazer φ, e em B e C como uma demonstração de que A é de fato uma tal codificação.

Formalmente, precisamos introduzir *soluções probabilisticamente verificáveis*. Dizemos que uma relação $R \subset \Sigma^* \times \Sigma^*$ é uma p-*relação* se (i) existe um polinômio p tal que $|y| \leq p(|x|)$ para todo $(x, y) \in R$, e (ii) o predicado '$(x, y) \in R$' pode ser verificado em tempo polinomial em $|x| + |y|$. Para maiores detalhes, veja a discussão imediatamente após a Definição 5 do Capítulo II.[1]

Por exemplo, seja

$$R_{3\text{SAT}} = \{(x, y) \in \Sigma^* \times \Sigma^* \colon \ x \text{ é uma instância } \varphi \text{ de } 3\text{SAT codificada e } y \text{ codifica uma atribuição que satisfaz } \varphi\}.$$

Então $R_{3\text{SAT}}$ é uma p-relação.

Uma função $E\colon \Sigma^* \to \Sigma^*$ é uma *codificação* se o comprimento $|E(x)|$ de $E(x)$ depende apenas do comprimento $|x|$ de x e, ademais, se $E(x)$ e $E(x')$ diferem em no mínimo metade de seus caracteres sempre que $|x| = |x'|$ mas $x \neq x'$. Note que a aplicação

$$E_0 \colon x \in \Sigma^* \mapsto (\langle x, y\rangle \colon y \in \mathbb{F}_2^{|x|}) \in \Sigma^* \tag{25}$$

é uma codificação. Note ainda que na prova transparente Π_φ discutida acima, podemos identificar A com $E_0(a)$.

Um *verificador de soluções* é um verificador que tem acesso não só a uma prova $\Pi \in \Sigma^*$, mas também a uma *solução* $s \in \Sigma^*$, que ele também acessa via um oráculo. Por motivos técnicos, será ainda necessário exigir que tanto a solução s como a prova Π fornecidas ao

[1]Note que usamos aqui por conveniência uma terminologia levemente diferente.

verificador de soluções venham segmentadas em blocos de algum comprimento fixo. Tais blocos devem ter início em posições que são congruentes a 1 módulo o comprimento dos blocos. Em outras palavras, tanto a solução s como a prova Π devem ser concatenações de segmentos de um comprimento fixo e, ademais, em cada acesso a s ou Π, o verificador recebe do oráculo um destes segmentos que constituem estas seqüências.

Um verificador de soluções é dito ser $(r(n), q(n), b(n))$-*restrito* se existem funções inteiras $\hat{r}(n) = O(r(n))$, $\hat{q}(n) = O(q(n))$, e $\hat{b}(n) = O(b(n))$ para as quais vale o seguinte. Com uma entrada x de comprimento n, o verificador usa no máximo $\hat{r}(n)$ bits aleatórios e acessa no total não mais que $\hat{q}(n)$ blocos da solução s e da prova Π, onde assumimos que ambos s e Π são segmentados com blocos de comprimento $\hat{b}(n)$. Novamente, o verificador é assumido ser *não-adaptativo*, isto é, uma vez definidas a entrada x e a seqüência aleatória τ, os blocos de s e Π a serem lidos ficam determinados, independentemente da solução s e da prova Π apresentadas. Seja $V(x, \tau, s, \Pi) \in \{\text{ACEITA}, \text{REJEITA}\}$ a saída do nosso verificador de soluções quando a entrada é x, a seqüência de bits aleatórios é τ, a solução é s, e a prova é Π.

DEFINIÇÃO 30. *Seja $R \subset \Sigma^* \times \Sigma^*$ uma* p-*relação e $E \colon \Sigma^* \to \Sigma^*$ uma codificação. Então o par (R, E) pertence a* $\mathrm{PCS}(r(n), q(n), b(n))$ *se e só se existe um verificador de soluções $(r(n), q(n), b(n))$-restrito V tal que*

(*i*) *Para todo x, $y \in \Sigma^*$ com $(x, y) \in R$, existe uma prova $\Pi_{x,y} \in \Sigma^*$ tal que*

$$\mathbb{P}_\tau\{V(x, \tau, E(y), \Pi_{x,y}) = \text{ACEITA}\} = 1. \tag{26}$$

(*ii*) *Para todo x e $s \in \Sigma^*$ tal que s não é $1/4$-próximo a uma codificação $E(y)$ de algum $y \in R(x)$, temos*

$$\mathbb{P}_\tau\{V(x, \tau, s, \Pi) = \text{ACEITA}\} < 1/4 \tag{27}$$

para todo $\Pi \in \Sigma^*$.

O seguinte teorema segue imediatamente das discussões desta seção.

TEOREMA 31. *Seja* $E_0\colon \Sigma^* \to \Sigma^*$ *a codificação dada em* (25). *Então*

$$(R_{3\mathrm{SAT}}, E_0) \in \mathrm{PCS}(n^3, 1, 1).$$

Finalmente, note que o seguinte resultado segue de uma rápida comparação das definições.

LEMA 32. *Seja* $R \subset \Sigma^* \times \Sigma^*$ *uma* p-*relação e* $E\colon \Sigma^* \to \Sigma^*$ *uma codificação. Seja* $L \subset \Sigma^*$ *a linguagem naturalmente dada por* R, *de forma que* $x \in L$ *se e só se* $R(x) \neq \emptyset$. *Então se*

$$(R, E) \in \mathrm{PCS}(r(n), q(n), b(n)),$$

temos que $L \in \mathrm{PCP}(r(n), q(n)b(n))$.

3.2. Certificados transparentes II. As técnicas de que fizemos uso na Seção 3.1 ilustram vários ingredientes da demonstração do Teorema 18. Entretanto, a codificação E_0, em que se baseiam todos os resultados principais daquela seção, tem um defeito fundamental: o comprimento de $E_0(x)$ é exponencial no comprimento de x. Desta forma, para interpretar soluções codificadas por E_0, qualquer verificador precisa de um número polinomial de bits aleatórios.

Na demonstração do Teorema 18, Arora *et al.* usam além de E_0 uma outra codificação, uma codificação *polinomial* e não apenas linear como E_0. Esta codificação, E_1, já havia sido utilizada com sucesso em trabalhos anteriores de Arora e Safra [AS92b] e outros (veja as notas bibliográficas abaixo). A codificação $E_1\colon \Sigma^* \to \Sigma^*$ é muito mais 'econômica' na quantidade de bits aleatórios que são necessários, embora eles sejam razoavelmente custosos em termos do número de caracteres da solução e da prova que precisam ser examinados. Esta codificação E_1 é tal que se temos uma instância satisfatível φ de 3SAT e $a \in \mathbb{F}_2^n$ satisfaz φ, então $E_1(a)$ tem comprimento $O(n^3)$.

Como mencionado acima, enquanto que E_0 é uma codificação baseada em funcionais lineares, E_1 é baseada em codificações polinomiais. Os resultados algébricos envolvidos no estudo de E_1 são mais sofisticados, embora ainda elementares. Não entraremos em detalhes aqui; restringiremo-nos a enunciar dois resultados principais sobre E_1. O primeiro é o seguinte.

TEOREMA 33. $(R_{3\mathrm{SAT}}, E_1) \in \mathrm{PCS}(\log n, 1, \mathrm{poly}(\log n))$.

Introduzimos agora uma notação para que possamos enunciar o segundo resultado sobre E_1. Seja q um inteiro positivo. A codificação $E_1^{(q)}\colon (\Sigma^q)^* \to \Sigma^*$ é definida por

$$E_1^{(q)}(x) = E_1(x_1)\dots E_1(x_q) \tag{28}$$

para todo $x = x_1 \dots x_q$, onde $|x_1| = \cdots = |x_q|$. Note que, estritamente falando, não devemos nos referir a $E_1^{(q)}$ como uma codificação, já que ela não está definida para $x \in \Sigma^*$ de comprimento não divisível por q. Entretanto, como estaremos sempre codificando atribuições $a \in \mathbb{F}_2^n$ que satisfazem uma dada instância φ de 3SAT, podemos assumir que o número de variáveis n em φ é divisível por qualquer inteiro q dado de antemão. O segundo resultado principal sobre E_1 é o seguinte.

TEOREMA 34. *Para todo inteiro positivo q, temos*

$$(R_{3\text{SAT}}, E_1^{(q)}) \in \mathrm{PCS}(\log n, 1, \mathrm{poly}(\log n)).$$

A demonstração do Teorema 18 usará basicamente apenas estes dois resultados sobre E_1. Apenas para completude, enunciamos o resultado abaixo, que é um lema intermediário na demonstração do Teorema 33. Note por outro lado que o Lema 32 diz que o resultado abaixo segue imediatamente do Teorema 33.

COROLÁRIO 35. $\mathrm{NP} \subset \mathrm{PCP}(\log n, \mathrm{poly}(\log n))$.

Terminamos esta seção enunciando um resultado sobre E_0 no mesmo espírito do Teorema 34. Defina $E_0^{(q)}\colon (\Sigma^q)^* \to \Sigma^*$ de forma análoga à definição de $E_1^{(q)}$. Temos então a seguinte generalização do Teorema 31.

TEOREMA 36. *Para todo inteiro positivo q, temos*

$$(R_{3\text{SAT}}, E_0^{(q)}) \in \mathrm{PCS}(n^3, 1, 1).$$

3.3. Provas transparentes recursivas. Aqui completamos a demonstração do Teorema 18. Os ingredientes principais serão os Teoremas 31, 33, 34, e 36. A idéia fundamental é que é possível combinar características de E_0 e E_1 através de uma codificação 'recursiva'. Suponha que, como acima, temos uma instância satisfatível φ de 3SAT, e que $a \in \mathbb{F}_2^n$ satisfaz φ.

Lembremos o que significa o Teorema 33. Ponha $s = E_1(a)$. Aquele teorema diz que há uma prova $\Pi \in \Sigma^*$ tal que as seguintes afirmativas são válidas. Tanto s quanto Π são segmentados, com cada segmento de comprimento $\hat{b}(n) = O(\text{poly}(\log n))$. Ademais, existe um verificador de soluções V que usa $\hat{r}(n) = O(\log n)$ bits aleatórios, e acessa $\hat{q}(n) = O(1)$ blocos de s e de Π para decidir se aceita ou não φ.

Suponha que a seqüência de bits aleatórios τ é fornecida ao nosso verificador V. Dado que os blocos a serem lidos dependem apenas de φ e de τ, podemos assumir neste ponto que V simplesmente aceita ou rejeita φ com base nos $\hat{q}(n) = O(1)$ blocos de s e Π a serem lidos. Esta discussão mostra que a saída de V, após fixo τ, é simplesmente computada por uma máquina de Turing determinística $M_\tau = M_{\tau,\varphi}$ com base em uma entrada segmentada $\pi = \pi_\tau = \pi_1 \dots \pi_q$, onde $q = \hat{q}(n) = O(1)$ e $|\pi_i| = \hat{b}(n) = O(\text{poly}(\log n))$ para todo $1 \leq i \leq q$.

Precisamos agora recordar um teorema clássico de Cook [Coo71].

TEOREMA 37. *Seja M uma máquina de Turing de complexidade de tempo polinomial. Então existem polinômios $p_1(n)$ e $p_2(n)$ e uma máquina de Turing determinística M' para os quais vale o seguinte. Para todo inteiro $n \geq 0$, a máquina M' computa em tempo no máximo $p_1(n)$ uma instância*

$$\Phi(y, z) = \Phi_M(y, z)$$

de 3SAT *com variáveis $y = (y_1, \dots, y_n)$ e $z = (z_1, \dots, z_{p_2(n)})$ tal que M aceita a entrada $x = x_1 \dots x_n \in \Sigma^*$ se e só a fórmula booleana $\Psi(z) = \Psi_x(z) = \Phi(x, z)$ obtida a partir de $\Phi(y, z)$ pondo-se $y = x$ é satisfatível.*

Seja $\Phi_\tau(y, z) = \Phi_{\tau,\varphi}(y, z)$ a instância de 3SAT dada pelo teorema de Cook para a nossa máquina $M_\tau = M_{\tau,\varphi}$ acima. Sabemos que V aceitará φ se e só se M_τ aceita $\pi = \pi_\tau$. Pela definição de $\Phi_\tau(y, z)$, isto ocorre se e só se $\Psi_\tau(z) = \Phi_\tau(\pi_\tau, z)$ é satisfatível. Sem perda de generalidade, podemos assumir que $|z| = \hat{b}(n)$. Como estamos admitindo que φ é satisfatível, sabemos que $\Psi_\tau(z) = \Phi_\tau(\pi_\tau, z)$ é satisfatível, e podemos assim incorporar na nossa prova transparente de que φ é satisfatível uma prova transparente de que, para todo τ, a fórmula booleana $\Psi_\tau(z) = \Phi_\tau(\pi_\tau, z)$ é satisfatível.

Para tanto, usamos $E_1^{(q+1)}$ para codificar

$$(\pi, z_0) = (\pi_\tau, z_0) = (\pi_1, \dots, \pi_q; z_0),$$

onde z_0 é tal que $\Psi_\tau(z_0) = \Phi_\tau(\pi_\tau, z_0) = \mathrm{V}$, e invocamos o Teorema 34 para obter uma prova transparente $\Pi = \Pi_\tau$ do fato de que

$$E_1^{(q+1)}(\pi_\tau, z_0) = (E_1(\pi_1), \ldots, E_1(\pi_q); E_1(z_0)) \tag{29}$$

codifica realmente uma solução para a instância $\Psi_\tau(z) = \Phi_\tau(\pi_\tau, z)$ de 3SAT. Note que o número de variáveis na fórmula booleana $\Psi_\tau(z) = \Phi_\tau(\pi_\tau, z)$ é apenas $O(\mathrm{poly}(\log n))$. Assim, o Teorema 34 garante que existe um verificador para interpretar (29) que é

$$(\log\log n, 1, \mathrm{poly}(\log\log n))\text{-restrito.}$$

A formalização das idéias acima, que exige certas observações técnicas adicionais não mencionadas, fornece a seguinte versão refinada do Teorema 33.

TEOREMA 38. $(R_{3\mathrm{SAT}}, E_1) \in \mathrm{PCS}(\log n, 1, \mathrm{poly}(\log\log n))$.

Note que o resultado acima foi obtido, essencialmente, 'compondo-se' E_1 recursivamente e usando os Teoremas 33 e 34. Compondo-se agora E_1 com E_0 e usando os Teoremas 38 e 36, completamos a demonstração do Teorema 18.

De fato, agora usamos E_0 para codificar as soluções das instâncias $\Psi_\tau(z) = \Phi_\tau(\pi_\tau, z)$ de 3SAT acima e, devido ao Teorema 38, podemos agora assumir que a variável z tem comprimento $\mathrm{poly}(\log\log n)$. O Teorema 36 fornece então um verificador que usa $\mathrm{poly}(\log\log n)$ bits aleatórios apenas, e examina somente uma quantidade $O(1)$ de entradas da solução e da prova apresentadas. Podemos concluir portanto que

$$(R_{3\mathrm{SAT}}, E_1) \in \mathrm{PCS}(\log n, 1, 1).$$

Claramente, basta agora usar o fato de que 3SAT é NP-completo e o Lema 32 para concluir que $\mathrm{NP} \subset \mathrm{PCP}(\log n, 1)$. Isto completa a demonstração do Teorema 18.

4. Notas Bibliográficas

O sistema de provas PCP foi introduzido por Arora e Safra [AS92b], e este trabalho foi imediatamente seguido pelo resultado definitivo de Arora, Lund, Motwani, Sudan, e Szegedy [ALM+92b]. Mais amplamente falando, a linha de pesquisa destes trabalhos foi sugerida por vários resultados na área de provas interativas (cf. Capítulo III). Em particular, a investigação do poder de sistemas interativos que contam com *vários* provadores foi importante. Dois resultados maximais sobre estes sistemas são devidos a Feige e Lovász [FL92] e Bellare, Goldwasser, Lund, e Russel [BGLR93, BGLR94]. Para uma ótima discussão sobre estes sistemas e a sua relação com os sistemas PCP, veja Babai [Bab94].

Os resultados que foram sucessivamente refinados até a obteção do Teorema 18 são devidos a Babai, Fortnow, e Lund [BFL91], Babai, Fortnow, Levin, e Szegedy [BFLS91], Feige, Goldwasser, Lovász, Safra, e Szegedy [FGL$^+$91], e Arora e Safra [AS92b]. Em particular, Arora *et al.* [ALM$^+$92b] atribuem o Teorema 33 ao conjunto destes trabalhos. A técnica de composição de codificações e dos respectivos verificadores (cf. Seção 3.3 acima) é devida a Arora e Safra [AS92b] e a Arora, Lund, Motwani, Sudan, e Szegedy [ALM$^+$92b].

Os Teoremas 31 e 36 são devidos a Arora *et al.* [ALM$^+$92b], e tiveram origem em resultados anteriores sobre verificação de programas e auto-correção [BLR90, Lip91]. O Teorema 22 é devido a Blum, Luby, e Rubinfeld [BLR90]. Funções auto-corretoras e, mais genericamente, a idéia de que programas podem verificar seus próprios resultados são o assunto de Blum e Kannan [BK89], Lipton [Lip91], e Blum, Luby, e Rubinfeld [BLR90].

Além dos artigos originais [ALM$^+$92a, ALM$^+$92b], trabalhos que contém provas completas do Teorema 18 incluem, entre outros, Sudan [Sud92], Arora [Aro94], e Hougardy, Prömel, e Steger [HPS94]. Este capítulo deve muito à excelente exposição de Hougardy, Prömel, e Steger [HPS94].

CAPÍTULO V

COMPLEXIDADE DE PROBLEMAS DE APROXIMAÇÃO

Nas classes P e NP são colocados problemas de *decisão,* ou seja, problemas com resposta SIM ou NÃO. Em conexão com alguns problemas de decisão, aparecem os 'problemas de otimização.'

Um *problema de otimização* $\mathcal{P}$ é um problema ou de maximização ou de minimização e consiste de três partes:

(1) um conjunto I de *instâncias* de $\mathcal{P}$;
(2) para cada instância $x \in I$, um conjunto finito $S(x)$ de *candidatos à solução* para x;
(3) uma função m que atribui a cada instância $x \in I$ e cada candidato à solução $y \in S(x)$ um número racional positivo $m(x, y)$, chamado de *valor* do candidato à solução y.

Se $\mathcal{P}$ é um problema de minimização (resp., maximização), então uma *solução ótima* para uma instância $x \in I$ é um candidato $y^* \in S(x)$ tal que, para todo $y \in S(x)$, temos $m(x, y^*) \leq m(x, y)$ (resp., $m(x, y^*) \geq m(x, y)$). Usaremos a notação $\mathrm{OTIM}(x)$ para denotar o valor $m(x, y^*)$ de uma solução ótima para x.

Por exemplo, o problema de otimização MAX-3SAT relacionado com o Problema 3SAT é o de encontrar uma atribuição que maximize o número de cláusulas satisfeitas. O problema de otimização relacionado com o Problema CIRCUITO HAMILTONIANO poderia ser o de encontrar no grafo dado um circuito, não necessariamente hamiltoniano, mas contendo o maior número possível de vértices. A outros problemas, porém, não correspondem problemas de otimização.

Para os problemas de otimização, faz sentido pensarmos em soluções aproximadas. Uma *solução aproximada* é um candidato à solução cujo valor não é muito diferente do valor ótimo.

DEFINIÇÃO 39. *Seja $\mathcal{P}$ um problema de otimização, x uma instância de $\mathcal{P}$ e y um candidato à solução para x. A* razão de aproximação *de y em relação a x é definido por*

$$R(x,y) = \max\left\{\frac{m(x,y)}{\mathrm{OTIM}(x)}, \frac{\mathrm{OTIM}(x)}{m(x,y)}\right\}.$$

Note que de acordo com essa definição a razão de aproximação é sempre maior ou igual a 1.

DEFINIÇÃO 40. *Seja A um algoritmo para um problema $\mathcal{P}$ que, dada uma instância x, devolve um candidato à solução $A(x)$. Seja r uma função $r\colon \mathbb{N} \to [1,\infty)$. Dizemos que A obtém* grau de aproximação r *para $\mathcal{P}$ se, para toda instância x,*

$$R(x, A(x)) \leq r(|x|).$$

Note que de acordo com essa definição a função r somente pode assumir valores maiores ou iguais a 1. Por simplicidade, um algoritmo como na definição acima é dito *aproximar* o problema $\mathcal{P}$ com razão r

1. As Classes NPO, APX e EAP

O sentido deste estudo de aproximabilidade é o de investigar como se comportam problemas NP-completos quando, ao invés de buscarmos uma resposta SIM ou NÃO, estamos interessados em soluções aproximadas. A definição a seguir captura essa noção: ao invés de considerarmos problemas de otimização em geral, vamos nos concentrar em problemas de otimização que tenham características de problemas em NP.

DEFINIÇÃO 41. *Um problema de otimização $\mathcal{P}$ está na classe* NPO *se as seguintes condições são satisfeitas:*

(1) *o conjunto das instâncias de $\mathcal{P}$ é reconhecível em tempo polinomial no tamanho da instância;*
(2) *para x uma instância de $\mathcal{P}$, o problema de pertinência ao conjunto dos candidatos à solução de x é solúvel em tempo polinomial em $|x|$;*

(3) *dados uma instância x e um candidato y à solução de x, o valor $m(x,y)$ de y é computável em tempo polinomial.*

Durante todo este capítulo, todo problema de otimização deve ser entendido como um problema em NPO.

DEFINIÇÃO 42. *Um problema $\mathcal{P}$ em* NPO *está na classe* APX *se existe algum $\varepsilon > 0$ para o qual existe um algoritmo polinomial que aproxime o problema com grau de aproximação $r \leq 1 + \varepsilon$.*

DEFINIÇÃO 43. *Dizemos que $\mathcal{P}$ em* NPO *tem um* esquema de aproximação polinomial, *se para cada $\varepsilon > 0$ existe um algoritmo polinomial A_ε que aproxime $\mathcal{P}$ com grau de aproximação $r \leq 1+\varepsilon$. Dizemos que $\mathcal{P}$ está na classe* EAP *se ele tem um esquema de aproximação polinomial.*

Pelas definições, trivialmente temos que EAP $\subseteq$ APX $\subseteq$ NPO. Vamos mostrar que todas as inclusões são próprias, a menos que P = NP, o que tornaria as três classe iguais.

O exemplo clássico de um problema em EAP cujo problema de decisão correspondente é NP-completo (veja Secão 2), é o Problema MOCHILA MÁXIMA. A prova deste teorema se encontra na Seção 4.

TEOREMA 44. *O Problema* MOCHILA MÁXIMA *está em* EAP.

Vamos mostrar mais a frente (Teorema 67), que o Problema MAX-3SAT não está em EAP, a menos que P = NP. Provamos também na Seção 4 o seguinte teorema.

TEOREMA 45. *O Problema* MAX-3SAT *está em* APX.

Finalmente, o Problema CLIQUE MÁXIMO, que claramente está em NPO, não pode estar em APX, a menos que P = NP. É o que mostramos no Teorema 68. (Este resultado é, na verdade, bastante mais forte.)

2. Completude para Classes de Complexidade

O conceito de problemas *completos* é bastante importante nas classes de complexidade. Por um lado mostra-se que existem problemas *completos* na classe, isto é, problemas que são tão difíceis quanto qualquer outro da mesma classe. Isso implica que, havendo uma melhora substancial no método de solução de um problema completo, esse método pode se estender a todos os outros problemas da classe. Por outro lado, estabelecem-se métodos, chamados de *reduções*, para se transformar um problema em outro. Essas reduções objetivam caracterizar outros problemas da classe como completos, ou mesmo caracterizar problemas como tão *difíceis* quanto qualquer problema da classe. Por exemplo, se um problema $\mathcal{P}_1$ é completo e pode ser convenientemente reduzido a um problema $\mathcal{P}_2$, então o problema $\mathcal{P}_2$, se pertencer à classe, também é completo.

Nesta seção definimos e exemplificamos tais conceitos.

2.1. Classe NP. Definimos a seguir uma redução conveniente para problemas da classe NP, conhecida como redução de Karp.

DEFINIÇÃO 46. *Uma* redução de Karp *de uma linguagem* $L_1 \in \Sigma_1^*$ *para uma linguagem* $L_2 \in \Sigma_2^*$ *é uma função* $f\colon \Sigma_1^* \to \Sigma_2^*$ *computável em tempo polinomial tal que para qualquer* $x \in \Sigma_1^*$,

$$x \in L_1 \Leftrightarrow f(x) \in L_2.$$

Usamos a notação $L_1 \leq_{\text{Karp}} L_2$ para denotar a existência de uma redução de Karp de L_1 para L_2. A propriedade fundamental de uma redução de Karp é que se $L_1 \leq_{\text{Karp}} L_2$ e existe um algoritmo *polinomial* para decidir o problema de pertinência em L_2, o problema de pertinência em L_1 pode ser resolvido em tempo polinomial usando o algoritmo para L_2 e a função f.

DEFINIÇÃO 47. *Uma linguagem L é* NP-completa *se $L \in$* NP *e para qualquer $L' \in$* NP, *temos $L' \leq_{\text{Karp}} L$.*

O mesmo conceito é importante para a classe coNP.

DEFINIÇÃO 48. *Uma linguagem L é* coNP-completa *se $L \in$* coNP *e para qualquer $L' \in$* coNP, *temos $L' \leq_{\text{Karp}} L$.*

Os teoremas a seguir, típicos em teoria da complexidade, são aplicações dos conceitos dados acima e consistem de resultados clássicos para a classe NP.

Teorema 49. *Se* $L_1 \leq_{\text{Karp}} L_2$ *e* $L_2 \leq_{\text{Karp}} L_3$, *então* $L_1 \leq_{\text{Karp}} L_3$.

Teorema 50. *Se* $L_1 \leq_{\text{Karp}} L_2$ *e* $L_2 \in \mathrm{P}$, *então* $L_1 \in \mathrm{P}$.

Teorema 51. *Se* L_1 *é* NP*-completa,* $L_2 \in \mathrm{NP}$, *e* $L_1 \leq_{\text{Karp}} L_2$, *então* L_2 *é* NP*-completa.*

Teorema 52. *Se uma linguagem* NP*-completa está em* P, *então*

$$\mathrm{P} = \mathrm{NP}\,.$$

Teorema 53. *Os seguintes problemas são* NP*-completos.*

(1) 2sat.
(2) 3sat.
(3) Satisfatibilidade.
(4) Clique.
(5) Circuito Hamiltoniano.
(6) Coloração.
(7) Passeio do Caixeiro Viajante.
(8) Mochila.
(9) Corte em Grafos.
(10) Árvore de Steiner.

A seguir damos o conceito de linguagens difíceis. Note que uma linguagem classificada como difícil para uma classe não precisa necessariamente estar na classe.

Definição 54. *Uma linguagem* L *é* NP-difícil *se para qualquer* $L' \in \mathrm{NP}$, *temos* $L' \leq_{\text{Karp}} L$.

Equivalentemente, uma linguagem L é NP-difícil se para alguma linguagem NP-completa L', temos $L' \leq_{\text{Karp}} L$.

2.2. Classes APX **e** EAP**.** Embora a redução de Karp seja apropriada para problemas em NP, elas nada nos diz em relação à aproximação de problemas de otimização, pois a redução de Karp não necessariamente preserva soluções aproximadas. Portanto, para as classes APX e EAP vamos definir a L-*redução* (L de linear), redução esta que preserva aproximabilidade.

Definição 55. *Uma* L-redução *de um problema de otimização* $\mathcal{P}_1$ *para um problema de otimização* $\mathcal{P}_2$ *é uma quádrupla* (f, g, α, β) *de dois algoritmos polinomiais* f *e* g *e duas constantes* α, $\beta > 0$ *satisfazendo as seguintes condições:*

(1) *Para toda instância* x_1 *de* $\mathcal{P}_1$, $x_2 = f(x_1)$ *é uma instância de* $\mathcal{P}_2$ *satisfazendo* $\mathrm{OTIM}_{\mathcal{P}_2}(x_2) \leq \alpha\, \mathrm{OTIM}_{\mathcal{P}_1}(x_1)$.
(2) *Para todo* y_2, *candidato à solução de* x_2 *com* $c_2 = m_{\mathcal{P}_2}(x_2, y_2)$, $y_1 = g(y_2)$ *é um candidato à solução de* x_1 *com* $c_1 = m_{\mathcal{P}_1}(x_1, y_1)$ *satisfazendo*

$$|c_1 - \mathrm{OTIM}(x_1)| \leq \beta |c_2 - \mathrm{OTIM}(x_2)|.$$

Usamos a notação $\mathcal{P}_1 \leq_{\mathrm{L}} \mathcal{P}_2$ para denotar a existência de uma L-redução de $\mathcal{P}_1$ para $\mathcal{P}_2$. A L-redução é ilustrada na Figura V.12.

Definição 56. *Um problema de otimização* $\mathcal{P}$ *é* APX-completo *se* $\mathcal{P} \in \mathrm{APX}$ *e para qualquer* $\mathcal{P}' \in \mathrm{APX}$, *vale que* $\mathcal{P}' \leq_{\mathrm{L}} \mathcal{P}$.

Os teoremas a seguir são os equivalentes, para a classe APX, aos vistos na seção anterior para a classe NP. Em todos os teoremas abaixo, estamos supondo que os $\mathcal{P}_i$ são problemas de otimização que pertencem à classe NPO.

Teorema 57. *Se* $\mathcal{P}_1 \leq_{\mathrm{L}} \mathcal{P}_2$ *e* $\mathcal{P}_2 \leq_{\mathrm{L}} \mathcal{P}_3$, *então* $\mathcal{P}_1 \leq_{\mathrm{L}} \mathcal{P}_3$.

Teorema 58. *Se* $\mathcal{P}_1 \leq_{\mathrm{L}} \mathcal{P}_2$ *e* $\mathcal{P}_2 \in \mathrm{EAP}$, *então* $\mathcal{P}_1 \in \mathrm{EAP}$.

Teorema 59. *Se* $\mathcal{P}_1 \leq_{\mathrm{L}} \mathcal{P}_2$ *e* $\mathcal{P}_2 \in \mathrm{APX}$, *então* $\mathcal{P}_1 \in \mathrm{APX}$. *Mais precisamente, se existe um algoritmo polinomial* A *para* $\mathcal{P}_2$ *com grau de aproximação* $r_A \leq 1 + \varepsilon$, *então existe um algoritmo polinomial* B *para* $\mathcal{P}_1$ *com grau de aproximação* $r_B \leq 1 + \alpha\beta\varepsilon$.

Teorema 60. *Se* $\mathcal{P}_1$, $\mathcal{P}_2 \in \mathrm{APX}$, $\mathcal{P}_1$ *é* APX-*completo e* $\mathcal{P}_1 \leq_{\mathrm{L}} \mathcal{P}_2$, *então* $\mathcal{P}_2$ *é* APX-*completo.*

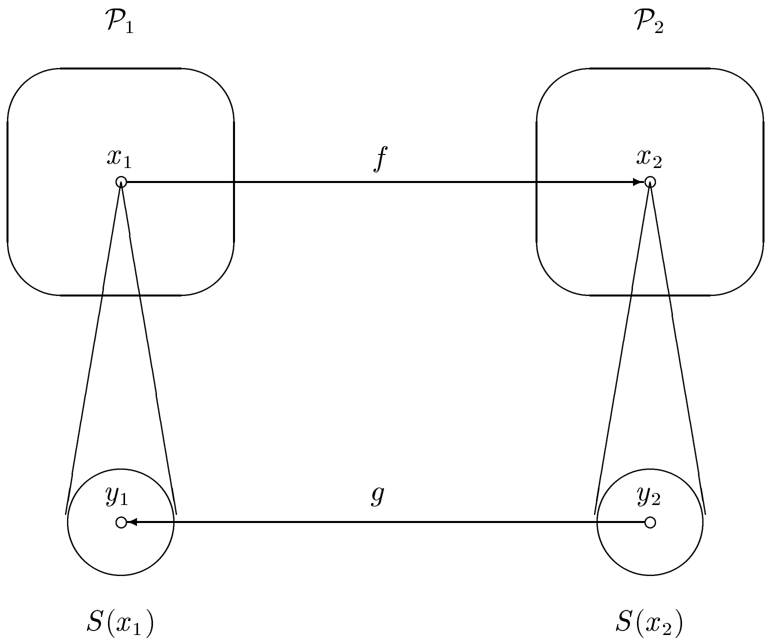

FIGURA V.12. L-redução

TEOREMA 61. *Se* $\mathcal{P}_1 \leq_{\mathrm{L}} \mathcal{P}_2$*,* $\mathcal{P}_1$ *é* APX*-completo e* $\mathcal{P}_2 \in$ EAP*, então* EAP = APX.

Veremos na Seção 3 que, a menos que P = NP, temos EAP $\neq$ APX.

TEOREMA 62. *Os seguintes problemas são* APX*-completos.*

(1) MAX-SAT.
(2) MAX-2SAT.
(3) MAX-3SAT.
(4) CORTE MÁXIMO.
(5) k-COLORAÇÃO MÁXIMA.
(6) SUPER STRING MÍNIMO.
(7) ÁRVORE MÍNIMA DE STEINER.

DEFINIÇÃO 63. *Um problema de otimização* $\mathcal{P}$ *é* APX-difícil *se para qualquer* $\mathcal{P}' \in$ APX*, temos* $\mathcal{P}' \leq_{\mathrm{L}} \mathcal{P}$*.*

Equivalentemente, um problema de otimização $\mathcal{P}$ é APX-difícil se para algum problema APX-completo $\mathcal{P}'$, temos $\mathcal{P}' \leq_{\mathrm{L}} \mathcal{P}$.

3. Resultados de Inaproximabilidade

Quando queremos mostrar que um problema de decisão $\mathcal{P}$ é intratável computacionalmente, mostramos que ele é NP-completo ou NP-difícil. Como não se acredita que tais problemas possam ser resolvidos em tempo polinomial nos modelos computacionais convencionais, isso é uma evidência de que $\mathcal{P}$ é intratável computacionalmente. Para se provar que $\mathcal{P}$ é intratável, mostra-se que $\mathcal{P}' \leq_{\mathrm{Karp}} \mathcal{P}$ para algum $\mathcal{P}'$ NP-completo. A partir dos anos 70 proliferaram na literatura provas com esse estilo. Mas essas construções raramente preservavam *aproximabilidade.* Ou seja, se construíssemos um algoritmo polinomial que obtivesse uma solução aproximada, digamos, com um grau de aproximação de $11/10$ para o problema de otimização relacionado com $\mathcal{P}$, em geral nada podemos afirmar sobre a obtenção de uma solução aproximada para $\mathcal{P}'$.

Logo após o aparecimento dos conceitos das classes P e NP, já se sabia que a obtenção de soluções aproximadas para problemas de otimização relacionados com alguns problemas NP-completos é computacionalmente tão difícil quanto a obtenção da solução exata. Para exemplificar, citamos os seguintes teoremas.

TEOREMA 64. *Se* P $\neq$ NP*, então não existe um algoritmo polinomial para o Problema* COLORAÇÃO MÍNIMA *com grau de aproximação menor que* 2.

TEOREMA 65. *Se* P $\neq$ NP*, então não existe um algoritmo polinomial para o Problema* PASSEIO DO CAIXEIRO VIAJANTE MÍNIMO *com grau de aproximação menor que* ∞.

Os teoremas acima mostram em particular que o Problema COLORAÇÃO MÍNIMA não pode estar em EAP, e o Problema PASSEIO DO CAIXEIRO VIAJANTE MÍNIMO não pode estar em APX, a menos que P $=$ NP.

Para outros problemas, resultados similares aparecem de forma isolada, indicando a impossibilidade de certas aproximações. Em 1989, Papadimitriou e Yannakakis [PY88] definem duas classes para problemas de otimização: as classes MAXNP e MAXSNP. Eles definem reduções entre problemas de otimização e provam a existência de problemas completos para a classe MAXSNP. A característica dessas classes é que se existir um algoritmo polinomial que obtenha uma solução arbitrariamente próxima da solução ótima de um problema completo para uma das classes, esse algoritmo pode ser usado para obter aproximações arbitrariamente próximas do ótimo para todos os outros problemas nessa classe.

Em 1992, como corolário dos resultados relativos a provas verificáveis probabilisticamente, Arora *et al.* [ALM$^+$92b] mostram que os problemas completos para a classe MAXSNP não podem ser arbitrariamente aproximados, a menos que P = NP.

Mais recentemente, Ausiello, Crescenzi e Protasi [ACP94] definem de forma diferente classes de problemas de aproximação que se mostram equivalentes às mencionadas acima. Essas definições são as que estamos usando nestas notas.

3.1. Resultados negativos para a classe APX**.** Iremos mostrar que, a menos que P = NP, nenhum problema APX-difícil pode ter um esquema de aproximação polinomial e, portanto, não pode pertencer à classe EAP. Uma conseqüência desse fato é que, para cada problema APX-difícil $\mathcal{P}$, existe uma constante $\varepsilon > 0$ tal que, a menos que P = NP, nenhum algoritmo polinomial para $\mathcal{P}$ pode obter grau de aproximação menor que $1 + \varepsilon$. O teorema é o seguinte.

TEOREMA 66. *A menos que* P = NP*, temos* APX $\neq$ EAP.

Para provar o teorema, basta mostrar que, a menos que P = NP, existe um problema em APX que não pode pertencer a EAP. O problema apropriado para esse objetivo é o MAX-3SAT. A parte crucial da prova usa o resultado principal discutido no Capítulo IV, ou seja, que $\mathrm{PCP}(\log n, 1) = \mathrm{NP}$.

TEOREMA 67. *Não existe um esquema de aproximação polinomial para* MAX-3SAT*, a menos que* P = NP.

PROVA. Seja $L \subseteq \Sigma^*$ uma linguagem em NP. Vamos mostrar que, dado um esquema de aproximação polinomial para MAX-3SAT, o problema de pertinência para a linguagem L pode ser resolvido em tempo polinomial. Como a prova vale para qualquer $L \in$ NP, decorre que P = NP.

Do Teorema 18 temos que existe um verificador $(\log n, 1)$-restrito V para L. Dado $x \in \Sigma^*$ vamos descrever como construir em tempo polinomial uma instância S_x para MAX-3SAT tal que

$$\begin{aligned} x \in L &\;\Rightarrow\; S_x \text{ é satisfatível} \\ x \notin L &\;\Rightarrow\; \begin{cases} \text{somente uma fração constante} \\ \text{(independente de } x\text{) das cláusulas} \\ \text{de } S_x \text{ pode ser satisfeita.} \end{cases} \end{aligned}$$

Com isso, fica fácil mostrar que se existe um esquema de aproximação polinomial para MAX-3SAT, então podemos decidir a pertinência de x em L e segue que P = NP.

Lembre-se de que no esquema PCP$(\log n, 1)$ o verificador V usa o oráculo um número constante de vezes para acessar a prova. A cada seqüência de bits aleatórios τ com $|\tau| = \hat{r}(x) = O(\log n)$, usada pelo verificador, corresponde um conjunto de endereços na prova. Para cada um desses endereços, vamos associar uma variável booleana da seguinte forma: para o endereço i associe a variável booleana b_i. Vamos considerar que cada prova Π corresponde à uma atribuição às variáveis b_i: o valor V é atribuído à variável b_i se o i-ésimo bit de Π for 1; o valor F é atribuído a b_i se o i-ésimo bit for 0. Para cada um dos $2^{|\tau|} = 2^{\hat{r}(x)} = 2^{O(\log n)}$ possíveis valores de τ, seja S_τ a fórmula booleana com variáveis b_i que expressa quais Π são aceitos por V quando a entrada é x. Note que S_τ tem tamanho constante pois somente um número constante de bits de Π são acessados. Seja S'_τ a fórmula booleana S_τ expressa com no máximo 3 variáveis por cláusula[1] (*i.e.*, na forma 3SAT). Como cada S'_τ tem tamanho constante, cada S'_τ tem um número constante, digamos $\leq k$, de cláusulas. Seja S_x o conjunto de todas as cláusulas S'_τ. Pela definição de sistemas PCP temos que

$$\begin{aligned} x \in L &\;\Rightarrow\; \exists \Pi_x \text{ tal que } S_x \text{ é satisfatível} \\ x \notin L &\;\Rightarrow\; \forall \Pi \text{ no máximo } 1/4 \text{ das } S'_\tau \text{ são} \\ &\qquad\qquad \text{satisfeitas simultaneamente.} \end{aligned}$$

[1] É fato bem conhecido que qualquer fórmula pode ser expressa, a custa de um certo aumento do número de variáveis e cláusulas, como a conjunção de cláusulas com no máximo 3 variáveis por cláusula. Veja Fatos 74 e 75.

Como cada S'_τ tem no máximo k cláusulas, segue que no caso de $x \notin L$ no máximo

$$1 - \frac{3}{4k}$$

das cláusulas de S_x são satisfeitas simultaneamente.

Se nos fosse dado um esquema de aproximação polinomial para MAX-3SAT, poderíamos obter um algoritmo polinomial para o problema com grau de aproximação menor que

$$\left(1 - \frac{3}{4k}\right)^{-1} = \frac{4k}{4k-3}.$$

Observando que k é uma constante que não depende de x, com esse algoritmo poderíamos separar o caso em que $x \in L$ do caso em que $x \notin L$. No primeiro caso S_x é satifatível e o algoritmo nos devolveria uma atribuição satisfazendo mais do que $1 - 3/4k$ das cláusulas de S_x. No segundo caso no máximo $1 - 3/4k$ das cláusulas de S_x podem ser satisfeitas simultaneamente. □

3.2. A impossibilidade de se aproximar o clique máximo. O resultado da seção anterior tira a esperança de encontrarmos esquemas polinomiais de aproximação para vários problemas de otimização. Porém, outras conseqüências podem ser tiradas do fato que $\mathrm{PCP}(\log n, 1) = \mathrm{NP}$. É o caso do seguinte teorema extremamente forte que iremos provar.

TEOREMA 68. *Existe um $\varepsilon > 0$ para o qual o Problema* CLIQUE MÁXIMO *não pode ser aproximado em tempo polinomial com grau de aproximação menor que n^ε, onde n é o número de vértices do grafo dado, a menos que* $\mathrm{P} = \mathrm{NP}$.

Antes de provar esse teorema, provaremos uma versão mais fraca.

TEOREMA 69. *O Problema* CLIQUE MÁXIMO *não pode ser aproximado em tempo polinomial com grau de aproximação menor que* 4, *a menos que* $\mathrm{P} = \mathrm{NP}$.

PROVA. Como de costume, para um grafo G, denote por $\omega(G)$ a cardinalidade de um clique máximo de G. Seja L uma linguagem em

NP sobre um alfabeto Σ. Fixe $x \in \Sigma^*$ com $|x| = n$. Vamos mostrar como construir eficientemente um grafo G_x e uma função f tais que

$$x \in L \quad \Rightarrow \quad \omega(G_x) = f(n) \tag{30}$$

$$x \notin L \quad \Rightarrow \quad \omega(G_x) < \frac{1}{4} f(n). \tag{31}$$

Então, um algoritmo para o Problema Clique Máximo com grau de aproximação 4 pode ser usado para decidir pertinência em L.

Do Teorema 18 temos que existe um verificador $(\log n, 1)$-restrito V para L. Seja $\hat{r}(n) = O(\log n)$ o número de bits aleatórios usados por V e $\hat{q}(n) = O(1)$ o número de consultas feitas às provas. Considere o vetor $(\tau, b_1, b_2, \ldots, b_{\hat{q}(n)})$, onde τ é uma seqüência de $\hat{r}(n)$ bits e cada $b_i \in \{0, 1\}$ para todo i. Definimos como o conjunto de vértices de G_x os vetores $(\tau, b_1, b_2, \ldots, b_{\hat{q}(n)})$ tais que se V usa τ como os bits aleatórios e recebe, como respostas às $\hat{q}(n)$ consultas à prova, os bits $b_1, b_2, \ldots, b_{\hat{q}(n)}$, então V aceita x. Dois vértices

$$v_1 = (\tau, b_1, b_2, \ldots, b_{\hat{q}(n)}) \quad \text{e} \quad v_2 = (\tau', b'_1, b'_2, \ldots, b'_{\hat{q}(n)})$$

são adjacentes se existir uma prova Π consistente com v_1 e v_2. Ou seja, se b_i e b'_j corresponderem a um mesmo bit consultado, então $b_i = b'_j$.

Vamos estimar $|V(G_x)|$, o número de vértice de G_x. Como existem $2^{\hat{r}(n)}$ possíveis τ, e para cada um deles $2^{\hat{q}(n)}$ possíveis valores para os b_i, temos que $|V(G_x)| \leq 2^{\hat{r}(n)} 2^{\hat{q}(n)} = 2^{O(\log n) + O(1)}$, o que é polinomial em n. Para saber se um candidato a vértice é realmente um vértice em G_x, basta verificar se V aceita x usando a informação contida no candidato a vértice. Isso pode ser feito em tempo polinomial simulando-se o procedimento de verificação que V executa. Como podemos também verificar facilmente se dois vértices são adjacentes, segue que podemos construir G_x em tempo polinomial.

Fixe uma prova Π qualquer. Então,

$$\begin{aligned} \omega(G_x) &\geq |\{\tau : V(x, \tau, \Pi) = \text{ACEITA}\}| \\ &= 2^{\hat{r}(n)} \mathbb{P}_\tau(V(x, \tau, \Pi) = \text{ACEITA}). \end{aligned}$$

Por outro lado, se C é um clique em G_x então existe uma prova Π consistente com todos os vértices de C. Em particular, existe uma prova Π_0 consistente com um clique de tamanho $\omega(G_x)$. Portanto,

$$\begin{aligned}\omega(G_x) &\leq |\{\tau\colon V(x,\tau,\Pi_0) = \text{ACEITA}\}| \\ &= 2^{\hat{r}(n)}\mathbb{P}_\tau(V(x,\tau,\Pi_0) = \text{ACEITA}).\end{aligned}$$

Das duas equações acima temos que

$$\omega(G_x) = 2^{\hat{r}(n)} \max_{\Pi} \mathbb{P}_\tau(V(x,\tau,\Pi) = \text{ACEITA}).$$

Mas, pela definição de sistemas PCP,

$$\max_{\Pi} \mathbb{P}_\tau\left(V(x,\tau,\Pi) = \text{ACEITA}\right) \begin{cases} = 1 & \text{se } x \in L \\ < 1/4 & \text{se } x \notin L. \end{cases}$$

Portanto, mostramos que G_x e $f(n) = 2^{\hat{r}(n)}$ são de acordo com as equações (30) e (31). □

Seja $\mathrm{PCP}(r(n), q(n), \varepsilon)$ a classe de linguagens definida da mesma forma que $\mathrm{PCP}(r(n), q(n))$, exceto que o limite superior $1/4$ para a probabilidade do verificador ser ludibriado é trocado por ε. Como já observado (veja Capítulo III),

$$\mathrm{PCP}(r(n), q(n), \varepsilon) = \mathrm{PCP}(r(n), q(n))$$

se ε é uma constante. A razão disso é que podemos repetir em paralelo um número conveniente (constante) de vezes o processo de verificação e diminuir arbitrariamente a probabilidade de erro.

Portanto, podemos reescrever a prova do Teorema 69 com ε no lugar de $1/4$, obtendo o seguinte resultado.

TEOREMA 70. *Para todo* $\varepsilon > 0$ *o Problema* CLIQUE MÁXIMO *não pode ser aproximado em tempo polinomial com grau de aproximação menor que* $1 + \varepsilon$, *a menos que* $\mathrm{P} = \mathrm{NP}$.

Pensando da mesma forma, poderíamos tentar provar o Teorema 68. Repetindo o esquema acima, teríamos que repetir o processo de verificação k vezes, com k satisfazendo $(1/4)^k \leq n^{-\varepsilon}$. Isso implica que $k = \Omega(\varepsilon \log n)$ e assim teríamos que usar $\Theta(\log^2 n)$ bits aleatórios ao invés de $O(\log n)$. Ademais, agora $\Theta(\log n)$ consultas seriam necessárias ao invés de $O(1)$, mostrando que $\mathrm{NP} \subseteq \mathrm{PCP}(\log^2 n, \log n, n^{-\varepsilon})$. Com isso, o número de vértices de G_x seria

$$|V(G_x)| \leq 2^{\hat{r}(n)}2^{\hat{q}(n)} = 2^{O(\log^2 n + \log n)},$$

o que não é satisfatório pois esse número não é limitado por nenhum polinômio em n. Porém, veremos na próxima seção que usando outros métodos bastante poderosos poderemos mostrar que

$$\mathrm{NP} \subseteq \mathrm{PCP}(\log n, \log n, n^{-\varepsilon}),$$

o que prova o Teorema 68.

3.3. $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, \log n, n^{-\varepsilon})$**.** A dificuldade em se usar o resultado $\mathrm{PCP}(\log n, 1) = \mathrm{NP}$ para provar a impossiblidade de um grau de aproximação de n^{ε} para o Problema Clique Máximo decorre do fato de termos que recorrer a $O(\log^2 n)$ bits aleatórios. Porém, vamos mostrar como $O(\log n)$ bits aleatórios podem gerar $O(\log^2 n)$ bits *pseudo-aleatórios*, que podem ser usados em um procedimento de verificação $(\log n, 1)$-restrito.

A idéia é a seguinte. Para uma constante d, vamos escolher adequadamente um grafo d-regular G com 2^r vértices rotulados com todos os número binários de 0 a $2^r - 1$. Vamos agora descrever um passeio aleatório em G. (Para detalhes sobre passeios aleatórios veja Apêndice B.) Inicialmente uma partícula é colocada num vértice de G escolhido uniformemente ao acaso. No passeio aleatório o próximo vértice a ser visitado pela partícula é escolhido uniformemente ao acaso dentre os d adjacentes ao vértice atual. Vamos escolher, a cada c vértices visitados, o rótulo do vértice que acaba de ser visitado como uma seqüência de r bits pseudo-aleatórios, onde c é uma constante adequada. Visitando-se um total de ct vértices, obtemos os tr bits pseudo-aleatórios que serão usados. Note que o custo de se visitar ct vértices desta forma é de

$$r + \lceil \log_2 d \rceil (ct - 1) = r + O(t)$$

bits (genuinamente) aleatórios. O próximo lema nos dá a justificativa desse método.

No lema abaixo, $\mathcal{G}_{d,\rho}$ é a família de grafos d-regulares com a seguinte propriedade. Se G é membro de $\mathcal{G}_{d,\rho}$ e A é a sua matriz de adjacência multiplicada por $1/d$, então todos menos o maior dos auto-valores de A são positivos e menores do que $1-\rho$.

Lema 71. *Seja $G=(V,E)\in\mathcal{G}_{d,\rho}$. Considere um passeio aleatório* P *em G com início em um vértice escolhido uniformemente ao acaso. Então, existe um inteiro $c=c_\rho$ tal que, para qualquer subconjunto C de V com $|C|\leq|V|/4$, a probabilidade do trecho inicial de* P *de comprimento kc visitar a cada c passos algum vértice de C é de no máximo $(3/4)^k$.*

Prova. Seja $G=(V,E)$ de acordo com o enunciado do teorema, com $V=\{1,2,\ldots,n\}$. Vamos usar o fato bem conhecido (veja, por exemplo, Lovász [Lov93, Exercício 11.4]) que o maior auto-valor de uma matriz de adjacência de um grafo d-regular é d. Segue daí que o maior auto-valor de A, a matriz de adjacência de G multiplicada por $1/d$, é 1. Sejam $1=\lambda_0>\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_m>0$ os auto-valores de A. Como $\lambda_1\leq 1-\rho$, existe um inteiro c tal que $\lambda_1^c\leq 1/4$. Seja $p^{(t)}=(p_1^{(t)},p_2^{(t)},\ldots,p_n^{(t)})\in\mathbb{R}^n$ o vetor de probabilidades com $p_i^{(t)}$ sendo a probabilidade do vértice i estar sendo visitado no tempo t. Como o primeiro vértice de G é escolhido ao acaso, o vetor $p^{(0)}$ é o vetor cujas entradas são todas $1/n$.

É fácil de se verificar que $Ap^{(0)}=p^{(1)}$, e em geral $A^tp^{(0)}=p^{(t)}$. Note simplesmente que A é a matriz de transição da cadeia de Markov definida pelo passeio aleatório.

Seja C um subconjunto qualquer de V com $|C|\leq n/4$. Seja N a matriz com zero em todas as entradas, exceto na diagonal, onde $N_{ii}=1$ se e só se $i\in C$. Então o vetor $NA^cp^{(0)}$ nos dá, para cada vértice $v\in C$, a probabilidade do passeio estar visitando v após c passos. O vetor $NA^c(NA^cp^{(0)})$ nos dá a probabilidade do passeio visitar vértices de C no passo c e no passo $2c$. Em geral, $(NA^c)^kp^{(0)}$ nos dá a probabilidade de um passeio aleatório de comprimento kc atingir a cada c passos um vértice de C.

Vamos denotar a seguir a norma ℓ_1 de um vetor v por $|v|$ (*i.e.*, $|v|=\sum_1^n|v_i|$) e a norma ℓ_2 por $\|v\|$ (*i.e.*, $\|v\|=\sqrt{\sum_1^n v_i^2}$).

Claramente $|(NA^c)^kp^{(0)}|$ é a probabilidade do passeio de comprimento kc atingir a cada c passos um vértice de C. Para limitar essa probabilidade vamos inicialmente limitar $|NA^cp|$, onde p é um vetor qualquer. Para tanto, é mais fácil trabalharmos com a norma ℓ_2. Escreva p como a soma de dois vetores ortogonais v e w, onde v é múltiplo de $p^{(0)}$. Note que $p^{(0)}$ é um auto-vetor associado ao auto-valor $\lambda_0 = 1$ e w é combinação linear de auto-vetores associados aos auto-valores $\lambda_1, \lambda_2, \ldots, \lambda_m$. Assim,

$$\|NA^c v\| = \|Nv\| = \sqrt{\sum_{N_{ii}=1} v_i^2} \leq \sqrt{\frac{1}{4}\sum_{i=1}^{n} v_i^2} = \frac{1}{2}\|v\| \tag{32}$$

e

$$\|NA^c w\| \leq \|A^c w\| \leq \lambda_1^c \|w\| \leq \frac{1}{4}\|w\|. \tag{33}$$

Usando a desigualdade triangular obtemos

$$\begin{aligned} \|NA^c p\| &= \|NA^c(v+w)\| \leq \|NA^c v\| + \|NA^c w\| \\ &\leq \frac{1}{2}\|v\| + \frac{1}{4}\|w\| \leq \frac{1}{2}\|p\| + \frac{1}{4}\|p\| \leq \frac{3}{4}\|p\|. \end{aligned}$$

Fazendo uso k vezes da desigualdade acima, temos

$$\|(NA^c)^k p^{(0)}\| \leq \left(\frac{3}{4}\right)^k \|p^{(0)}\| = \left(\frac{3}{4}\right)^k \frac{1}{\sqrt{n}}.$$

Agora, lembrando que para qualquer vetor $v \in \mathbb{R}^n$ temos $|v| \leq \sqrt{n}\|v\|$, obtemos

$$|(NA^c)^k p^{(0)}| \leq \sqrt{n}\|(NA^c)^k p^{(0)}\| \leq \sqrt{n}\left(\frac{3}{4}\right)^k \frac{1}{\sqrt{n}} = \left(\frac{3}{4}\right)^k,$$

como queríamos demonstrar. □

A prova da existência e construtibilidade em tempo polinomial de infinitos membros da família $\mathcal{G}_{d,\rho}$ é tratado no teorema abaixo.

TEOREMA 72. *Existem constantes α, $\rho > 0$ tais que se n é uma potência de 2 suficientemente grande, então existe um grafo $G = (V, E)$ com n vértices com as seguintes propriedades:*

1. *o grau de cada vértice é 7;*
2. *o segundo auto-valor λ_1 de $A(G)$ é menor que $1 - \rho$;*
3. *dado n, o grafo G pode ser construído em tempo polinomial em n por uma máquina de Turing.*

Veremos mais adiante que o fato de termos G em $\mathcal{G}_{d,\rho}$ com $d = O(1)$ no teorema acima é importante. Esses grafos são conhecidos como *grafos expansores esparsos*. Informalmente, um grafo é um expansor se a vizinhança de qualquer subconjunto de seus vértices é relativamente grande. O 'fator de expansão' de um grafo G é intimamente ligado ao segundo auto-valor de G. Para detalhes veja o Capítulo 9 de Alon e Spencer [AS92a]. O problema da construção de grafos expansores com grau fixo aparece antes da descoberta de sua utilização como auxiliares na geração de bits pseudo-aleatórios. Embora a existência de tais grafos pode ser facilmente provada probabilisticamente, a construção explícita e eficiente de tais grafos é muito mais difícil. Para um breve apanhado do desenvolvimento de técnicas para a construção de expansores veja Seção 5.

Usando o Lema 71 e o Teorema 72, podemos provar agora o seguinte.

Teorema 73. *NP $\subseteq$ PCP$(\log n, \log n, n^{-\varepsilon})$.*

Prova. Seja L uma linguagem em NP e $\varepsilon > 0$ um número real fixo. Seja V um verificador $(\log n, 1)$-restrito para L. Para provar o teorema vamos, a partir de V, construir um verificador $(\log n, \log n)$-restrito V_ε. Seja $\hat{r}(n) = O(\log n)$ o número de bits aleatórios usados por V. Seja x uma palavra cuja pertinência em L deseja-se decidir. Inicialmente, o verificador V_ε constrói um grafo $G \in \mathcal{G}_{d,\rho}$ $(d = 7)$ com $2^{\hat{r}(|x|)}$ vértices, de acordo com Teorema 72. Seja c um inteiro satisfazendo $(1-\rho)^c \leq 1/4$. Seja $k = \lceil \varepsilon \log_{4/3} n \rceil$.

Usando $O(\log n)$ bits aleatórios, o verificador V_ε faz um passeio aleatório em G de comprimento ck, coletando a cada c passos os rótulos dos vértices visitados. Note que cada passo do passeio aleatório usa somente $O(1)$ bits aleatórios, pois o grafo tem grau $O(1)$. Esses rótulos são usados como bits aleatórios para V_ε repetir k vezes o processo de verificação de V. Se alguma vez V REJEITA, então V_ε REJEITA. Caso contrário, V_ε ACEITA.

A probabilidade de V dar uma resposta errada (ou seja V ACEITA e $x \notin L$) é limitada por $1/4$. Ou seja, no máximo $1/4$ dos rótulos dos vértices de G podem fazer V fornecer uma resposta incorreta. Seja C o conjunto dos vértices de G cujos rótulos correspondem a respostas erradas de V. Usando então o Lema 71, a probabilidade dos k vértices visitados estarem todos em C, ou seja, a probabilidade de todas as k

simulações de V devolverem ACEITA erroneamente, é, no máximo,

$$\left(\frac{3}{4}\right)^k \leq \left(\frac{3}{4}\right)^{\varepsilon \log_{4/3} n} = n^{-\varepsilon},$$

como queríamos. $\square$

4. Demonstrações dos Teoremas 44 e 45

Os Teoremas 44 e 45, já enunciados, serão provados a seguir.

TEOREMA 44. *O Problema* MOCHILA MÁXIMA *está em* EAP.

PROVA. Precisamos mostrar que para qualquer $\varepsilon > 0$ existe um algoritmo polinomial com grau de aproximação no máximo $1 + \varepsilon$.

Lembremos que uma instância x do problema consiste de um conjunto $U = \{u_1, u_2, \ldots, u_n\}$, duas funções t, $v \colon U \to \mathbb{Z}$ e um inteiro CAP. Nosso objetivo é obter um subconjunto $U' \subseteq U$ que maximize

$$m(x, U') = \sum_{u \in U'} v(u), \quad \text{sujeito a} \quad \sum_{u \in U'} t(u) \leq \mathrm{CAP}\,.$$

Definimos a *densidade* de um $u \in U$ como $d(u) = v(u)/t(u)$.

Vamos descrever um procedimento que obtém uma solução aproximada do problema. Fixe um inteiro $k \geq 1$. Considere todos os subconjuntos de U com no máximo k elementos. Para cada um desses subconjuntos A, verifique se $\sum_{u \in A} t(u) \leq \mathrm{CAP}$. Se esse for o caso, tente incluir em A mais elementos de U. Os elementos são considerados para inclusão por ordem decrescente de densidade e colocados em A se e somente se a capacidade CAP da mochila não é violada com a inclusão. Chame o conjunto assim definido de U'. A solução aproximada é a que maximiza $m(x, U')$ sobre todas as possíveis escolhas de A.

Seja $\widehat{U}$ uma solução ótima para o problema de maximização. Seja $\mathrm{OTIM} = m(x, \widehat{U})$. Se $|\widehat{U}| \leq k$ então o subconjunto $\widehat{U}$ é explicitamente considerado pelo procedimento e neste caso a solucão aproximada é igual à solução ótima. Vamos assim supor que $|\widehat{U}| > k$.

Seja $B \subseteq U$ o conjunto dos k elementos de $\widehat{U}$ de maior valor. Seja $\overline{U} \subseteq U$ o candidato à solução obtido pelo procedimento descrito acima quando o conjunto B é considerado. Vamos mostrar que

$$\frac{\text{OTIM}}{m(x, \overline{U})} \leq 1 + 1/k.$$

Sejam $u_{i_1}, u_{i_2}, \ldots, u_{i_r}$ todos os elementos de $\widehat{U} \setminus \overline{U}$ ordenados por ordem decrescente de densidade, ou seja, $d(u_{i_1}) \geq d(u_{i_2}) \geq \cdots \geq d(u_{i_r})$. Se $r = 0$ então $\overline{U}$ é uma solução ótima. Vamos supor então que $r \geq 1$. O elemento u_{i_1} não é um dos k elementos de maior valor em $\widehat{U}$, e portanto

$$v(u_{i_1}) \leq \text{OTIM}/(k+1). \tag{34}$$

Seja $T = \overline{U} \cap \widehat{U}$. Sejam $u_{j_1}, u_{j_2}, \ldots, u_{j_s}$ os elementos de $\overline{U} \setminus \widehat{U}$ com densidades maiores ou iguais a $d(u_{i_1})$ ordenados por ordem decrescente de densidades. Ou seja,

$$d(u_{j_1}) \geq d(u_{j_2}) \geq \cdots \geq d(u_{j_s}) \geq d(u_{i_1}).$$

Não pode ser o caso que $s = 0$, pois, por construção, a razão que levou u_{i_1} a não ser colocado em $\overline{U}$ é que essa inclusão violaria a capacidade da mochila. Como a inclusão é feita por ordem decrescente de densidade, algum elemento fora de $\widehat{U}$, de densidade maior que $d(u_{i_1})$, já teria que estar em $\overline{U}$ quando u_{i_1} foi considerado pelo procedimento. Portanto,

$$\sum_{u \in T} t(u) + \sum_{p=1}^{s} t(u_{j_p}) + t(u_{i_1}) > \text{CAP}.$$

Por outro lado, temos que

$$\sum_{u \in T} t(u) + \sum_{p=1}^{r} t(u_{i_p}) \leq \text{CAP},$$

pois $\widehat{U} = T \cup \bigcup_{p=1}^{r} \{u_{i_p}\}$. Segue das duas equações acima que

$$\sum_{p=1}^{s} t(u_{j_p}) + t(u_{i_1}) > \sum_{p=1}^{r} t(u_{i_p}),$$

implicando que

$$\sum_{p=1}^{s} t(u_{j_p}) > \sum_{p=2}^{r} t(u_{i_p}).$$

Como $d(u_{j_s}) \geq d(u_{i_1}) \geq d(u_{i_p})$, para todo $1 \leq p \leq r$, temos que $v(u_{i_p}) \leq d(u_{j_s})t(u_{i_p})$. Desse fato e da equação acima, segue que

$$\sum_{p=2}^{r} v(u_{i_p}) \leq d(u_{j_s}) \sum_{p=2}^{r} t(u_{i_p}) \leq d(u_{j_s}) \sum_{p=1}^{s} t(u_{j_p}) \leq \sum_{p=1}^{s} v(u_{j_p}).$$

Portanto,

$$\begin{aligned} \mathrm{OTIM} &= \sum_{u\in T} v(u) + v(u_{i_1}) + \sum_{p=2}^{r} v(u_{i_p}) \\ &\leq \sum_{u\in T} v(u) + v(u_{i_1}) + \sum_{p=1}^{s} v(u_{j_p}) \\ &\leq \sum_{u\in \overline{U}} v(u) + v(u_{i_1}) = m(x,\overline{U}) + v(u_{i_1}). \end{aligned}$$

Usando (34) obtemos,

$$\mathrm{OTIM} \leq m(x,\overline{U}) + v(u_{i_1}) \leq m(x,\overline{U}) + \mathrm{OTIM}/(k+1),$$

ou

$$\frac{\mathrm{OTIM}}{m(x,\overline{U})} \leq 1 + 1/k.$$

Escolhendo $k = \lceil 1/\varepsilon \rceil$ obtemos a aproximação desejada, de acordo com a Definição 43.

Só nos falta mostrar que o procedimento que obtém a aproximação pode ser executado em tempo polinomial. Observe que o número dos subconjuntos $A \subseteq U$ com no máximo k elementos não excede n^k. A construção de U' a partir de A pode ser feita em tempo $O(nB)$, onde B é o número de bits do maior número que aparece na instância do problema, dentre os tamanhos, valores e a capacidade CAP. Podemos gerar sistematicamente todos os subconjuntos com no máximo k elementos através de um processo que nos fornece o próximo subconjunto a partir do anterior. Cada passo dessa construção pode ser feito em tempo $O(k \log n)$. Portanto, o algoritmo todo pode ser executado em tempo $O(n^k \times nB \times k \log n)$ ou $O(Bn^{k+1} \log n)$ considerando-se k constante. Em outros termos, uma vez fixado ε, o algoritmo com grau de aproximação $1 + \varepsilon$ roda em tempo $O(Bn^{\lceil 1/\varepsilon \rceil + 1} \log n)$. $\square$

TEOREMA 45. *O Problema* MAX-3SAT *está em* APX.

PROVA. Lembre-se de que uma instância do Problema MAX-3SAT é uma coleção de cláusulas $C = \{c_1, c_2, \dots, c_m\}$ sob um conjunto finito U de variáveis, com $|c_i| = 3$ para $1 \leq i \leq m$. Seja $U = \{u_1, u_2, \dots, u_n\}$ o conjunto de variáveis.

Para provar que o problema está em APX, basta mostrar um $\varepsilon > 0$ e um algoritmo A que aproxime o problema com grau de aproximação $r \leq 1 + \varepsilon$. Vamos mostrar um algoritmo com grau de aproximação no máximo 2.

O algoritmo é bastante simples. Inicialmente considere o conjunto de cláusulas $C_1 \subseteq C$ nas quais a variável u_1 aparece. Atribua o valor V ou F a u_1 de forma a maximizar o número de cláusulas satisfeitas em C_1. Pelo menos metade dessas cláusulas podem ser satisfeitas.

Num passo i defina o conjunto de cláusulas

$$C_i \subseteq C \setminus (C_1 \cup C_2 \cup \dots \cup C_{i-1})$$

nas quais a variável u_i aparece. Atribua o valor V ou F a u_i de forma a maximizar o número de cláusulas satisfeitas em C_i. Pelo menos metade dessas cláusulas podem ser satisfeitas.

Podemos facilmente constatar que a atribuição acima definida satisfaz pelo menos metade de todas as cláusulas de U, e, portanto, o valor ótimo para o problema é no máximo 2 vezes maior que o valor obtido pela atribuição. $\square$

5. Notas Bibliográficas

Os pilares para o estudo das classes de complexidade em computação foram lançados no início dos anos 70 por Stephen Cook [Coo71], com as noções das classes P e NP e a prova de que o Problema SATISFATIBILIDADE é NP-completo, que ficou conhecida como o Teorema de Cook. A redução de Karp aparece em [Kar91]. Enquanto isso, atrás da cortina de ferro, muito antes da proliferação do uso da *Internet* e do correio eletrônico, L. Levin [Lev72] provou independentemente uma variante do Teorema de Cook e define um outro tipo de redução, que passou a ser conhecida como redução de Levin. Para uma coleção detalhada de problemas NP-completos o leitor poderá consultar Garey e Johnson [GJ79], onde encontrará uma excelente introdução ao assunto.

O Teorema 64 que mostra que o Problema MOCHILA MÁXIMA está em EAP é de Sahni [Sah75]. Dentre os primeiros resultados mostrando a impossibilidade de certas aproximações estão os Teoremas 64 e 65 de Garey e Johnson [GJ76] e Sahni e Gonzales [SG76] respectivamente.

Conforme já mencionado, para as definições de classes de problemas de aproximação estamos seguindo as definições de Ausiello *et al.* [ACP94], posteriores ao trabalho pioneiro de Papadimitriou e Yannakakis [PY88].

A conexão entre provas interativas e a impossibilidade de se aproximar cliques em grafos foi estabelecida por Feige, Goldwasser, Lovász, Safra e M. Szegedy [FGL$^+$91]. O primeiro trabalho estabelecendo essa conexão, que de certa forma passou desapercebido, é de Condon [Con93].

As técnicas usadas para mostrar como $O(\log n)$ bits aleatórios podem gerar $O(\log^2 n)$ bits pseudo-aleatórios são de Cohen e Wigderson [CW89] e Impagliazzo e Zuckerman [IZ89], inspirados em Ajtai, Komlós e Szemerédi [AKS87].

O problema da construção de grafos expansores com grau fixo aparece antes da utilização destes como auxiliares na geração de bits pseudo-aleatórios. Embora a existência de tais grafos pode ser facilmente provada probabilisticamente, a construção explícita e eficiente de tais grafos é muito mais difícil. A primeira construção foi dada por Margulis [Mar75], usando a teoria da representação de grupos de Lie. Uma versão melhorada é obtida em Gabber e Galil [GG81] e se baseia em análise de Fourier comutativa. O Teorema 72 é mencionado em [AKS87] e é baseado na construção de [GG81]. Posteriormente, a conexão entre auto-valores e propriedades expansivas de grafos, que aparece no teorema acima, é estabelecida em Alon e Milman [AM85], e Alon [Alo86], servindo também como base para melhores estimativas de λ_1. Isso possibilitou que, em trabalhos similares, Margulis [Mar88] e Lubotzky, Phillips e Sarnak [LPS88] melhorassem bastante as propriedades expansivas obtidas. Para a construção de grafos com λ_1 pequeno foi usada, entre outros, a teoria dos números, incluindo resultados sobre uma conjectura de Ramanujan sobre o número de representações de inteiros por certas formas quadráticas.

APÊNDICE A

LISTA DE PROBLEMAS

Neste apêndice estão listadas as definições dos vários problemas mencionados no decorrer do trabalho. O estilo da apresentação é o usado por Garey e Johnson [GJ79] (problemas de decisão) e por Crescenzi e Kann [CK94] (problemas de otimização). Estas duas referências trazem listagens e classificações quanto à complexidade de centenas de problemas.

1. Problemas de Decisão

Árvore de Steiner
Instância: Um grafo G, um subconjunto S de seus vértices e um inteiro positivo k.
Problema: Existe um subgrafo conexo de G contendo todos os vértices de S e tendo menos do que k arestas?

Passeio do Caixeiro Viajante
Instância: Um conjunto finito $C = \{c_1, c_2, \ldots, c_m\}$ de "cidades," uma "distância" $d(c_i, c_j) \in \mathbb{N}$ para cada par de cidades c_i, $c_j \in C$, e um inteiro positivo L
Problema: Existe um passeio de *Caixeiro Viajante*, *i.e.*, um passeio que visite todas as cidades e retorne à cidade de origem, de comprimento no máximo L ?

Circuito Hamiltoniano
Instância: Um grafo G.

Problema: Existe um circuito hamiltoniano em G, *i.e.*, um circuito que passa exatamente uma vez por cada vértice de G?

CLIQUE
Instância: Um grafo $G = (V, E)$ e um inteiro k.
Problema: O grafo G tem um clique de tamanho k, *i.e.*, existe $C \subseteq V$ com $|C| = k$ tal que quaisquer dois vértices distintos em C são adjacentes?

COLORAÇÃO
Instância: Um grafo $G = (V, E)$ e um inteiro k.
Problema: O grafo G tem uma k-coloração própria, *i.e.*, existe uma função $f\colon V \to \{1, 2, \ldots, k\}$ tal que $f(v) \neq f(w)$ sempre que $vw \in E$?

CORTE EM GRAFOS
Instância: Um grafo $G = (V, E)$ e um inteiro positivo k.
Problema: Existe uma partição de V em V_1 e V_2 tais que o número de arestas ligando vértices de V_1 a vértices de V_2 é pelo menos k?

QBF
Instância: Uma fórmula booleana quantificada

$$F = (Q_1 x_1)(Q_2 x_2) \ldots (Q_n x_n) B(x_1, x_2, \ldots, x_n),$$

onde B é uma fórmula booleana sem quantificadores e cada Q_i é ou $\exists$ ou $\forall$.
Problema: É F verdadeira?

GRAFO NÃO-ISOMORFISMO
Instância: Dois grafos G_1 e G_2.
Problema: G_1 e G_2 são não-isomorfos? (Dizemos que dois grafos $H_1 = (V, E_1)$ e $H_2 = (V, E_2)$ *são isomorfos* se existe uma função bijetora $f : V \to V$ tal que $uv \in E_1$ se e somente se $f(u)f(v) \in E_2$.)

MOCHILA
Instância: Um conjunto finito U, para cada $u \in U$ um *tamanho* inteiro $t(u)$ e um *valor* inteiro $v(u)$, uma *capacidade* inteira CAP e um inteiro c.
Problema: Existe um subconjunto $U' \subseteq U$ tal que $\sum_{u \in U'} t(u) \leq \mathrm{CAP}$ (a soma dos tamanhos dos elementos em U' é no máximo a capacidade

da mochila) e $\sum_{u \in U'} v(u) \geq c$ (o valor total colocado na mochila é pelo menos c)?

PRIMALIDADE
Instância: Um número inteiro positivo n.
Problema: É n um número primo?

2SAT
Instância: Uma coleção de cláusulas $C = \{c_1, c_2, \ldots, c_m\}$ sobre um conjunto finito U de variáveis, com $|c_i| = 2$ para $1 \leq i \leq m$ e um inteiro positivo k.
Problema: Existe uma atribuição às variaveis de U que satisfaça pelo menos k cláusulas?

3SAT
Instância: Uma coleção de cláusulas $C = \{c_1, c_2, \ldots, c_m\}$ sobre um conjunto finito U de variáveis, com $|c_i| = 3$ para $1 \leq i \leq m$.
Problema: Existe uma atribuição para U que satisfaça a todas as cláusulas de C?

SATISFATIBILIDADE
Instância: Uma coleção de cláusulas $C = \{c_1, c_2, \ldots, c_m\}$ sobre um conjunto finito U de variáveis.
Problema: Existe uma atribuição para U que satisfaça a todas as cláusulas de C?

2. Problemas de Otimização

ÁRVORE MÍNIMA DE STEINER
Instância: Um grafo G e um subconjunto S de seus vértices.
Solução: Uma subgrafo conexo T contendo todos os vértices de S.
Função objetivo: Número de arestas em T.

PASSEIO DO CAIXEIRO VIAJANTE MÍNIMO
Instância: Um conjunto finito $C = \{c_1, c_2, \ldots, c_m\}$ de "cidades," uma "distância" $d(c_i, c_j) \in \mathbb{N}$ para cada par de cidades c_i, $c_j \in C$, e um inteiro positivo L
Solução: Um passeio de Caixeiro Viajante, *i.e.*, um passeio que visite

todas as cidades em C e retorne à cidade de origem.
Função objetivo: Comprimento total do passeio.

Coloração Mínima
Instância: Um grafo $G = (V, E)$.
Solução: Uma k-coloração própria de G, *i.e.*, uma função

$$f : V \to \{1, 2, \ldots, k\}$$

tal que $f(v) \neq f(w)$ sempre que $vw \in E$.
Função objetivo: O inteiro k.

k-coloração Máxima
Instância: Um grafo G.
Solução: Um subgrafo H de G que tenha uma k-coloração própria. (Aqui, $k \geq 2$ é uma constante, não fazendo parte da entrada.)
Função objetivo: O número de arestas de H.

Clique Máximo
Instância: Um grafo $G = (V, E)$.
Solução: Um clique de G, *i.e.*, um subconjunto $C \subseteq V$ tal que quaisquer dois vértices distintos em C são adjacentes.
Função objetivo: Cardinalidade de C.

Corte Máximo
Instância: Um grafo $G = (V, E)$.
Solução: Subconjuntos X do conjunto de vértices de G.
Função objetivo: Número de arestas entre X e $V \setminus X$.

Max-2sat
Instância: Uma coleção de cláusulas $C = \{c_1, c_2, \ldots, c_m\}$ sobre um conjunto finito U de variáveis, com $|c_i| = 2$ para $1 \leq i \leq m$.
Solução: Uma atribuição A às variáveis de U.
Função objetivo: O número de cláusulas satisfeitas por A.

Max-3sat
Instância: Uma coleção de cláusulas $C = \{c_1, c_2, \ldots, c_m\}$ sobre um conjunto finito U de variáveis, com $|c_i| = 3$ para $1 \leq i \leq m$.
Solução: Uma atribuição A às variáveis de U.
Função objetivo: O número de cláusulas satisfeitas por A.

MAX-SAT
Instância: Uma coleção de cláusulas $C = \{c_1, c_2, \ldots, c_m\}$ sobre um conjunto finito U de variáveis.
Solução: Uma atribuição A às variáveis de U.
Função objetivo: O número de cláusulas satisfeitas por A.

MOCHILA MÁXIMA
Instância: Um conjunto finito U, para cada $u \in U$ um *tamanho* inteiro $t(u)$ e um *valor* inteiro $v(u)$, e uma *capacidade* inteira CAP.
Solução: Um subconjunto $U' \subseteq U$ tal que $\sum_{u \in U'} t(u) \leq \text{CAP}$ (a soma dos tamanhos dos elementos em U' é no máximo a capacidade da mochila).
Função objetivo: O valor $\sum_{u \in U'} v(u)$ (a soma dos valores dos elementos colocados na mochila).

SUPER STRING MÍNIMO
Instância: Um conjunto S de palavras.
Solução: Uma palavra w contendo cada palavra de S como segmento, *i.e.*, para qualquer $s \in S$, existem w_1, w_2, tais que $w = w_1 s w_2$
Função objetivo: Comprimento de w.

APÊNDICE B

DEFINIÇÕES

A intenção deste apêndice é relembrar rapidamente o leitor de algumas definições básicas que são usadas nestas notas. Mais detalhes podem ser vistos em livros clássicos de teoria dos grafos e combinatória como os livros [BM79, Bol79] e [Lov93]. O restante das definições são dadas durante o desenvolvimento dos tópicos. Para tais definições, consulte o Índice Remissivo.

1. Notação Assintótica

Seja f e g duas funções $f, g : \mathbb{N} \to \mathbb{R}$. Escrevemos que

$$f = O(g)$$

(f é *da ordem* de g) se existe uma constante $c > 0$ tal que para todo n suficientemente grande temos que $|f(n)| \leq c|g(n)|$.

Escrevemos que

$$f = \Omega(g)$$

se $g = O(f)$.

Escrevemos que

$$g = \Theta(f)$$

se $f = O(g)$ e $g = O(f)$.

2. Terminologia da Teoria dos Grafos

Um *grafo* G consiste de um conjunto finito não-vazio $V(G)$ de *vértices* e um conjunto finito $E(G)$ de pares não-ordenados de vértices, chamados *arestas*. Uma aresta $\{x, y\}$ de um grafo G é denotado, por comodidade, tanto por xy como por yx. Os *extremos* de uma aresta são os vértices contidos nela. Escrevemos $G = (V(G), E(G))$. Dizemos que uma aresta *liga* o par de vértices que são seus extremos. Dois vértices são *adjacentes* se são ligados por alguma aresta do grafo. Se v é um extremo de uma aresta f, dizemos que f e v são *incidentes* ou *adjacentes*.

O número de arestas incidentes a um vértice v é o *grau* de v; denotamos esta quantidade por $\mathrm{grau}(v)$. Um grafo G é *k-regular* se todos os seus vértices têm grau k.

Dado um grafo G, uma função $\varphi\colon V(G) \to K$ é muitas vezes dita ser uma *coloração* dos vértices de G pelas *cores* em K. Se $|K| = k$, dizemos que φ é uma *k-coloração*. A coloração φ é *própria* se vértices adjacentes em G recebem cores distintas, *i.e.*, se para toda aresta xy de G temos $\varphi(x) \neq \varphi(y)$. O *número cromático* $\chi(G)$ de um grafo G é o menor inteiro k para o qual existe uma k-coloração própria de G.

Um *clique* em um grafo G é um subgrafo completo de G, *i.e.*, um subgrafo cujos vértices são todos dois-a-dois adjacentes. Um clique de um grafo G é *máximo* se não há em G um clique com uma quantidade maior de vértices. O número de vértices em um clique máximo de G é denotado por $\omega(G)$.

Um *passeio* em um grafo é uma seqüência de vértices $v_0, v_1, \ldots, v_k$ tal que v_{i-1} e v_i são adjacentes para todo $1 \leq i \leq k$. Um *caminho* é um passeio sem vértice repetidos. Um *circuito* é um passeio fechado, *i.e.*, com $v_0 = v_k$, e sem vértices repetidos (a menos de $v_0 = v_k$). Um circuito *hamiltoniano* é um circuito que contém ('passa') por todos os vértices do grafo. Um grafo é *hamiltoniano* se ele contém um circuito hamiltoniano.

A *matriz de adjacência* de um grafo G com vértices $v_1, v_2, \ldots, v_n$ é a matriz

$$A_G = (a_{ij})_{i,j=1}^{n},$$

onde a_{ij} é o número de arestas ligando v_i a v_j. Note que esta matriz é simétrica.

Seja $G = (V, E)$ um grafo com $V = \{1, 2, \ldots, n\}$. Um *passeio aleatório* em G é uma seqüência de vértices $(X_t)_0^\infty$ visitados por uma partícula que começa num vértice especificado e visita outros vértices de acordo com a seguinte regra de transição: se a partícula está num vértice i no tempo t ou, formalmente, $X_t = i$, então no tempo $t+1$ ela se move a um vértice adjacente a i escolhido uniformemente ao acaso. Pode ser facilmente verificado que a seqüência (X_t) é uma cadeia de Markov com matriz de transição $A = (a_{ij})_{i,j=1}^n$ com

$$a_{ij} = \begin{cases} 1/\operatorname{grau}(j) & \text{se } i \text{ e } j \text{ são adjacentes em } G \\ 0 & \text{caso contrário.} \end{cases}$$

3. Terminologia da Lógica

Seja $U = \{u_1, u_2, \ldots, u_n\}$ um conjunto de variáveis booleanas. Uma *atribuição* para U é uma função $t \colon U \to \{\mathrm{V}, \mathrm{F}\}$. Se $t(u) = \mathrm{V}$ dizemos que u é 'verdadeira' sob a atribuição. Se $t(u) = \mathrm{F}$ dizemos que u é 'falsa.' Dizemos que u e $\bar{u} = \neg u$ são *literais* sob U. O literal u é verdadeiro sob t se e somente se $t(u) = \mathrm{V}$. O literal $\neg u$ é verdadeiro sob t se e somente se $t(u) = \mathrm{F}$.

Uma *cláusula* sobre U é um conjunto de literais sobre U. Uma cláusula representa a disjunção daqueles literais e é *satisfeita* por uma atribuição se e somente pelo menos um de seus membros é verdadeiro sob a atribuição. Uma coleção de cláusulas é *satisfatível* se existe uma atribuição para U que satisfaça todas as suas cláusulas simultaneamente.

Uma *fórmula atômica* é um literal ou uma das *constantes* V ou F. Uma *fórmula booleana* é uma fórmula atômica ou é da forma $\neg A$, $(A \vee B)$ ou $(A \wedge B)$, onde A e B são fórmulas. Uma fórmula é *satisfatível* se existe uma atribuição para a qual a fórmula é verdadeira.

Os seguintes fatos elementares são bem conhecidos.

FATO 74. *Dada uma fórmula A, existe uma coleção de cláusulas C de tamanho polinomial no comprimento de A tal que A é satisfatível se e somente se C o é.*

FATO 75. *Dada uma coleção de cláusulas C, existe uma outra coleção de cláusulas D, com no máximo três literais por cláusula e de* *tamanho polinomial no tamanho de C, tal que C é satisfatível se e somente se D o é.*

REFERÊNCIAS

[ACP94] G. Ausiello, P. Crescenzi, and M. Protasi, *Approximate solution of NP optimization problems*, Tech. Report SI/RR 94/03, Università di Roma "La Sapienza", 1994.

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi, *Deterministic simulation in $LOGSPACE$*, Proc. 19th Ann. ACM Symp. on Theory of Computing, 1987, pp. 132–140.

[ALM+92a] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *On the intractability of approximation problems (early draft)*, Manuscrito, 1992.

[ALM+92b] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science, 1992, pp. 14–23.

[Alo86] N. Alon, *Eigenvalues, geometric expanders, sorting in rounds, and Ramsey theory*, Combinatorica **6** (1986), no. 3, 207–219.

[AM85] N. Alon and V. Milman, *λ_1, isoperimetric inequalities for graphs, and superconcentrators*, Journal of Combinatorial Theory, Series B **38** (1985), 73–88.

[Aro94] S. Arora, *Probabilistic Checking of Proofs and Hardness of Approximation Problems*, Ph.D. thesis, U. of California at Berkeley, 1994.

[AS92a] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley and Sons, Inc., 1992.

[AS92b] S. Arora and S. Safra, *Probabilistic checking of proofs*, Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science, 1992, pp. 2–13.

[Bab85] L. Babai, *Trading group theory for randomness*, Proc. 17th Ann. ACM Symp. on Theory of Computing, 1985, pp. 421–429.

[Bab90] L. Babai, *E-mail and the unexpected power of interaction*, Proc. 5th IEEE Symp. on Structure in Complexity Theory, 1990, pp. 30–44.

[Bab94] L. Babai, *Transparent proofs and limits to approximation*, Tech. Report TR-94-07, University of Chicago, 1994, To appear in the *Proc. First Europ. Congr. Math.*, Birkhäuser.

[BFL91] L. Babai, L. Fortnow, and C. Lund, *Non-deterministic exponential time has two-prover interactive protocols*, Computational Complexity **1** (1991), 3–40.

[BFLS91] L. Babai, L. Fortnow, L.A. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, Proc. 23rd Ann. ACM Symp. on Theory of Computing, 1991, pp. 21–31.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russel, *Efficient probabilistically checkable proofs and applications to approximation*, Proc. 25th Ann. ACM Symp. on Theory of Computing, 1993, pp. 294–304.

[BGLR94] M. Bellare, S. Goldwasser, C. Lund, and A. Russel, *Efficient probabilistically checkable proofs and applications to approximation*, Proc. 26th Ann. ACM Symp. on Theory of Computing, 1994, Errata, p. 820.

[BK89] M. Blum and S. Kannan, *Designing programs that check their work*, Proc. 21st Ann. ACM Symp. on Theory of Computing, 1989, pp. 86–97.

[BLR90] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, Proc. 22nd Ann. ACM Symp. on Theory of Computing, 1990, pp. 73–83.

[BM79] J. Bondy and U. Murty, *Graph Theory with Applications*, North-Holland, New York, 1979.

[BM88] L. Babai and S. Moran, *Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes*, Journal of Computer and System Sciences **36** (1988), 254–276.

[Bol79] B. Bollobás, *Graph Theory—an Introductory Course*, Graduate Texts in Mathematics, vol. 63, Springer-Verlag, New York, 1979.

[CK94] P. Crescenzi and V. Kann, *A compendium of NP optimization problems*, 1994, Versão Preliminar. Disponível por `ftp` anônimo de `nada.kth.se`.

[Con93] A. Condon, *The complexity of the max word problem and the power of one-way interactive proof systems*, Computational Complexity **3** (1993), 292–305.

[Coo71] S. Cook, *The complexity of theorem-proving procedures*, Proc. 3rd Ann. ACM Symp. on Theory of Computing, 1971, pp. 151–158.

[CR73] S.A. Cook and R.A. Reckhow, *Time bounded random access machines*, Journal of Computer and System Sciences **7** (1973), no. 4, 354–375.

[CW89] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science, 1989, pp. 14–19.

[Edm65a] J. Edmonds, *Minimum partition of a matroid into independent sets*, Journal of Research of the National Bureau of Standards (B) **69** (1965), 67–72.

[Edm65b] J. Edmonds, *Paths, trees, and flowers*, Canadian J. of Mathematics **17** (1965), 449–467.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Approximating clique is almost NP-complete*, Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science, 1991, pp. 2–12.

[FL92] U. Feige and L. Lovász, *Two-prover one-round proof systems: their power and their problems*, Proc. 24th Ann. ACM Symp. on Theory of Computing, 1992, pp. 733–744.

[GG81] O. Gabber and Z. Galil, *Explicit constructions of linear-sized superconcentrators*, Journal of Computer and System Sciences **22** (1981), 407–420.

[GJ76] M.R. Garey and D.S. Johnson, *The complexity of near-optimal graph coloring*, Journal of the ACM **23** (1976), 43–49.

[GJ79] M.R. Garey and D.S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-completeness*, W. H. Freeman and Company, 1979.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof-systems*, Proc. 17th Ann. ACM Symp. on Theory of Computing, 1985, pp. 291–304.

[Gol94] O. Goldreich, *Probabilistic proof systems (a survey)*, Electronic Colloquium on Computational Complexity, 1994, Report Nr.: TR94–008.

[GS86] S. Goldwasser and M. Sipser, *Private coins versus public coins* *in interactive proof systems*, Proc. 18th Ann. ACM Symp. on Theory of Computing, 1986.

[HPS94] S. Hougardy, H.J. Prömel, and A. Steger, *Probabilistically checkable proofs and their consequences for approximation algorithms*, Discrete Mathematics **136** (1994), 175–223.

[IZ89] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science, 1989, pp. 248–253.

[Kar72] R.M. Karp, *Reducibility among combinatorial problems*, Complexity in Computer Computations (New York) (R.E. Miller and J.W. Thatcher, eds.), Plenum Press, 1972, pp. 85–103.

[Kar91] R. Karp, *An introduction to randomized algorithms*, Disc. Applied Math. **34** (1991), 165–201.

[Lev72] L. Levin, *Problemas universais de busca*, Problemy Peredachi Informatsii **9** (1972), no. 3, 265–266, em russo.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, Journal of the ACM **39** (1992), 859–868.

[Lip91] R. Lipton, *New directions in testing*, Distributed computing and cryptography, American Mathematical Society, 1991, pp. 191–202.

[Lov93] L. Lovász, *Combinatorial Problems and Exercises*, 2nd ed., North-Holland, New York, 1993.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica **8** (1988), 261–277.

[LSS+77] C. Lucchesi, Imre Simon, Istvan Simon, Janos Simon, and T. Kowaltowski, *Aspectos Teóricos da Computação*, 11º Colóquio Brasileiro de Matemática, Poços de Caldas, 1977.

[Mar75] G.A. Margulis, *Explicit construction of concentrators*, Problems of Information Transmission **9** (1975), 325–332.

[Mar88] G.A. Margulis, *Combinatorial schemes and their application to the design of expanders and concentrators*, Problems of Information Transmission **24** (1988), 39–46.

[Mat93] Yuri M. Matiyasevich, *Hilbert's Tenth Problem*, MIT Press, Cambridge MA, 1993.

[Pra75] V. Pratt, *Every prime has a succint certificate*, SIAM Journal on Computing **4** (1975), 214–220.

[PS94] A. Polishchuk and D.A. Spielman, *Nearly-linear size holographic proofs*, Proc. 26th Ann. ACM Symp. on Theory of Computing, 1994, pp. 194–203.

[PY88] C.H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, Proc. 20th Ann. ACM Symp. on Theory of Computing, 1988, pp. 229–234.

[Sah75] S. Sahni, *Approximate algorithms for* 0/1 *knapsack problem*, Journal of the ACM **22** (1975), 115–124.

[SG76] S. Sahni and T. Gonzales, *P-complete approximation problems*, Journal of the ACM **23** (1976), 555–565.

[Sha92] A. Shamir, $IP = PSPACE$, Journal of the ACM **39** (1992), 869–877.

[She92] A. Shen, $IP = PSPACE$: *simplified proof*, Journal of the ACM **39** (1992), 878–880.

[Sim75] Janos Simon, *On some central problems in computational complexity*, Ph.D. Thesis, Cornell University, Computer Science Tech Report TR 75–224, 1975.

[Sip92] M. Sipser, *The history and status of the P versus NP question*, Proc. 24th Ann. ACM Symp. on Theory of Computing, 1992, pp. 603–618.

[Sto76] L. Stockmeyer, *The polynomial time hierarchy*, Theoretical Computer Science **3** (1976), 1–22.

[Sud92] M. Sudan, *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, Ph.D. thesis, U. of California at Berkeley, 1992.

[Tod89] S. Toda, *On the computational power of PP and* $\oplus P$, Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science, 1989, pp. 514–519.

[Tra84] B.A. Trakhtenbrot, *A survey of Russian approaches to Perebor (brute-force search) algorithms*, Annals of the history of computing **6** (1984), 384–400.

[Tur36] A.M. Turing, *On computable numbers with an application to the Entscheidungsproblem*, Proc. London Math. Soc. **2** (1936), 230–265, A correction, *ibid.*, **43**, pp. 544–546.

[Val79] L. Valiant, *The complexity of computing the permanent*, Theoretical Computer Science **8** (1979), 189–201.

ÍNDICE REMISSIVO