Classes de Complexidade

Nelma Moreira
Departamento de Ciência de Computadores
Faculdade de Ciências, Universidade do Porto
email: nam@dcc.fc.up.pt

Última revisão: 2016

Conteúdo

Lista de Figuras

Capítulo 1

Preliminares

1.1 Ordens de Grandeza

Nesta secção são revistas algumas noções sobre ordens de grandeza de funções (reais).

Definição 1.1 *Sendo* $f : \mathbb{N} \to \mathbb{R}$ *e* $g : \mathbb{N} \to \mathbb{R}$

1. $f(n)$ *é* $O(g(n))$ *se*
$$\exists c \in \mathbb{R}^+ \exists m \forall n \geq m \mid f(n) \mid \leq c \mid g(n) \mid$$
(diz-se que f *é da ordem de* g*, isto é,* f *cresce como* g *ou mais lentamente)*
2. $f(n)$ *é* $\Omega(g(n))$ *se* $g(n)$ *é* $O(f(n))$
3. $f(n)$ *é* $\Theta(g(n))$ *se* $f(n)$ *é* $O(g(n))$ *e* $g(n)$ *é* $O(f(n))$
4. $f(n)$ *é* $o(g(n))$ *se* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$*, isto é,* $g(n)$ *cresce mais rapidamente do que* $f(n)$*.*

Se $f(n)$ é $O(p(n))$ e $p(n)$ é uma função polinomial então diz-se que $f(n)$ é de ordem polinomial.

Teorema 1.1 *Se* $f(n)$ *é* $o(g(n))$ *então* $f(n)$ *é* $O(g(n))$*.*

Exercício 1.1.1 Mostra o teorema anterior.

Exercício 1.1.2 Mostra as seguintes igualdades:

a) $f(n) = O(f(n))$

b) $c.O(f(n)) = O(f(n))$ para qualquer c constante

c) $O(f(n) + g(n)) = O(f(n)) + O(g(n))$

d) $O(O(f(n))) = O(f(n))$

e) $O(f(n))O(g(n)) = O(f(n)g(n))$

f) $O(f(n))g(n) = f(n)O(g(n))$

Exemplo 1.1

- $3n^3 + 2n^2 + 5$ *é* $O(n^3)$ *(ou* $O(n^4)$ *mas não* $O(n^2)$*.*
- *Se* $p(n)$ *tem termo de ordem maior* k *então* $p(n)$ *é* $O(n^k)$*.*
- $O(n^2) + O(n) = O(n^2)$
- $O(\log n)$ *não depende da base* $b > 1$*.*
- $3n \log n + 4n \log \log n + 5 = O(n \log n)$*.*
- $2^{cn} = 2^{O(n)}$
- $2^{O(\log n)} = O(n^{O(1)})$*.*

Exemplo 1.2

- $\sqrt{n} = o(n)$
- $n = o(n \log \log n)$
- $n \log \log n = o(n \log n)$
- $n \log n = o(n^2)$
- $n^2 = o(n^3)$

Exercício 1.1.3 Mostra que:

a) $a_m n^m + \ldots + a_0$ é $\Theta(n^m)$, para $a_m > 0$. Sugestão: Para mostrar que é $\Omega(n^m)$ considere $c = \frac{2}{a_m}$ e $n_0 = mh$ com $h = (2/a_m)max(|a_{m-1}|, \ldots, |a_0|)$

b) n^α é $o(n^\beta)$, se $0 \leq \alpha < \beta$.

c) n^α é $o(\beta^n)$, se $\beta > 1$. Concluir que para qualquer polinómio $p(n)$ e $\beta > 1$, $p(n)$ é $O(\beta^n)$.Sugestão: Considera $\beta = (1 + \epsilon)$ e a expansão do binómio de Newton.

d) para qualquer polinómlio $p(n)$ e constante c existe um inteiro n_0 tal que, $\forall\, n > n_0,\ 2^{cn} > p(n)$. Calcula esse n_0 para n^2 e $c = 1$ e para $100n^{100}$ e $c = \frac{1}{100}$. Conclua que $p(n)$ não é $\Omega(c^n)$, $c > 1$

e) $\log n$ é $O(n)$ e $o(n^\alpha)$, $\alpha > 0$

f) $\forall a, b > 1,\ \log_a n$ é $\Theta(\log_b)$. Sugestão: Recorda que para $n > 0$, $\log_a n$ é y tal que $a^y = n$.

g) $\forall \alpha > 1,\ \alpha^n$ é $o(n!)$

Exercício 1.1.4 Mostra que Θ é uma relação de equivalência.

Exercício 1.1.5 Sendo $f(n)$ e $g(n)$ qualquer par de funções a seguir apresentadas, determina se $f(n)$ é $O(g(n))$, $f(n)$ é $\Omega(g(n))$ ou $f(n)$ é $\Theta(g(n))$

$$n^2 \quad n^3 \quad n^{\log n} \quad 2^n \quad n^n \quad 2^{2^n} \quad n^2 \log n$$

1.2 Codificações Razoáveis

Definição 1.2 *Duas codificações c_1 e c_2, dizem-se* polinomialmente relacionadas*, se existem polinómios p e p' tal que, sendo x_1 e x_2 as representações dum mesmo objecto x, respectivamente em c_1 e c_2, se tem $\mid x_1 \mid \leq p(\mid x_2 \mid)$ e $\mid x_2 \mid \leq p'(\mid x_1 \mid)$.*

Exercício 1.2.1 Considera o problema de codificar um inteiro não-negativo m numa base $b \geq 2$. Mostre que

(i) os comprimentos das codificações em bases diferentes estão polinomialmente (até linearmente, $O(n)$) relacionadas entre si.

(ii) a codificação em unário não está polinomialmente relacionada com a codificação numa base $b \geq 2$.

Exercício 1.2.2 Considera o problema de codificar um grafo não dirigido $G = (V, E)$. Dois processos são:

- matriz de adjacências representada por $|V|^2$ bits
- lista de ramos: a cada vértice associar a lista dos pontos de chegada dos ramos que dele partem

(i) Define mais rigorosamente as codificações mencionadas

(ii) Dá exemplo de um grafo codificado das duas formas

(iii) Mostra que os comprimentos das duas codificações estão polinomialmente (quadraticamente) relacionados entre si

Exercício 1.2.3 Supondo o alfabeto $\Sigma = \{0, 1, -, [,], (,), , \}$ inventa uma codificação genérica para representar

- um inteiro em binário

- a referência ao n-ésimo elemento dum conjunto, sequência, etc...
- uma sequência de elementos

Usando as codificações anteriores diz como representar: um conjunto (finito) de objectos, um grafo, uma função finita e um número racional.

Capítulo 2

P e NP

Bibliografia [AB09, Cap 1., 2. 3.], [GJ79, Cap. 1–3,5,7.1,7.2], [Pap94, Cap. 8, 9, 10 e 17], [], e [BC94, Cap. 4, 5 e 7]

2.1 Classes de complexidade

O conjunto das linguagens decidíveis pode ser dividida em classes de complexidade, que caracterizam os limites dos recursos computacionais usados para as decidir. Uma *classe de complexidade* é especificada pelo modelo de computação – vamos considerar máquinas de Turing com k-fitas –, pelo modo de computação – determinístico ou não determinístico –, pelo um recurso – p.e., tempo ou espaço – e por um limite, isto é, uma função de $\mathbb{N}$ em $\mathbb{N}$. Uma classe de complexidade é então um conjunto $C(f(n))$ definido por:

$$\begin{aligned} C(f(n)) \quad = \{L \mid\ & L \text{ é decidida por uma máquina de Turing } M \text{ do modo} \\ & \text{adequado, tal que para qualquer } x, \text{ e } |x| = n, M \text{ gasta} \\ & \text{no máximo } f(|x|) \text{ unidades do respectivo recur¡so}\} \end{aligned}$$

Uma função $f(n)$ para ser limite de complexidade tem de ser total e não decrescente. No caso, de funções de complexidade temporal deve ainda ser tal que exista uma MT M_f de k-fitas que, para quaisquer dados x, pare em exactamente $f(|x|)$ passos. Esta característica permite a simulação de *relógios* em máquinas de Turing. Equivalentemente dizemos que f é *tempo-construível* se $f(n) \geq n$ e existe uma MT que calcula a função $n \mapsto \lfloor f(n) \rfloor$ no máximo em $f(n)$ passos. Restrições semelhantes se impõem para funções de complexidade de espaço.

2.1.1 Classe P

Seja M uma máquina de Turing (determinística) e $x \in \Sigma^\star$. O tempo requerido por M em x, $T_M(x)$, é o número de passos da computação de M para dados x.[1] Então, a *complexidade temporal* (ou

[1] Se a computação não parar $T_M(x) = \infty$.

tempo de execução) de M é dada por:

$$T_M(n) = max\{t \mid \exists x \in \Sigma^\star,\ |x| = n \text{ e } t = T_M(x)\}$$

Se $T_M(n)$ é $O(f(n))$, diz-se que M opera em tempo limitado por $f(n)$. Se $f(n)$ é um polinómio dizemos que M é polinomial.

Vamos considerar máquinas de Turing com k-fitas, *multi-fita* sendo uma de leitura, uma de escrita e as restantes de leitura/escrita. Recordar que, se uma máquina de Turing M decidir uma linguagem L, então podemos ter $F = \{s_y, s_n\}$ ou $F = \{s_h\}$ (de paragem, para cálculo de funções). Formalmente temos

Definição 2.1 *Uma máquina de Turing com k-fitas MT, $k \geq 2$ é um tuplo*

$$M = (S, \Sigma, \Gamma, s_0, \Box, \rhd, \delta, \{s_y, s_n, s_h\}),$$

onde Γ é o alfabeto das fitas, $\Sigma \subseteq \Gamma$ alfabeto de entrada ,$s_0 \in S$ é o estado inicial. O símbolo $\Box \in \Gamma$ é símbolo branco e $\rhd \in \Gamma$ é símbolo inicial. A função de transição $\delta : S \times \Gamma^k \longrightarrow S \times \Gamma^{k-1} \times \{\leftarrow, \rightarrow, -\}^k$ descreve as regras de M. Sendo $s \in S$, $\sigma_i \in \Gamma$ e $m_i \in \{\leftarrow, \rightarrow, -\}^k$ temos

$$\delta(s, (\sigma_1, \sigma_2, \ldots, \sigma_k)) = (s', (\sigma'_2, \ldots, \sigma'_k), (m_2, \ldots, m_k)).$$

Se algum dos σ_i é $\rhd$ então m não pode ser $\leftarrow$ e $\sigma'_i = \rhd$. No início supomos que os dados $x = x_1 \ldots x_n$ estão na primeira fita a seguir a $\rhd$, e a cabeça da primeira fita está em x_1. As restantes fitas estão em branco e só com o símbolo $\rhd$ e as cabeças estão em $\rhd$. A máquina aceita se entrar em s_y, rejeita se entrar em s_n. Se entrar em s_h pará e a saída é o conteúdo da fita k é o resultado. Esta fita poderá ser só de escrita.

Exercício 2.1.1 *Define formalmente as noções de configuração, um movimento ($\xrightarrow{M}$) e aceitação de uma MT multi-fita M com dados x.*

Exercício 2.1.2 *Mostra que uma máquina de Turing com uma fita de leitura/escrita semi-infinita simula em $O(T^2(n))$ uma máquina de Turing multi-fita com complexidade temporal $T(n)$.*

Para estimar a complexidade computacional de um problema não serão consideradas constantes aditivas ou multiplicativas e a notação O será usada. O teorema seguinte justifica a escolha de MT multi-fitas para o estudo da complexidade temporal.

Teorema 2.1 *Seja L uma linguagem aceite por uma MT M com complexidade temporal $f(n)$. Para qualquer $c > 0$, L é aceite por uma MT M' que executa em $cf(n) + n + 2$.*

Dem. Existe um inteiro m, dependente de c e M, tal que se Γ é o alfabeto de M, M' usa um alfabeto $\Gamma' = \Gamma \cup \Gamma^m$ de tal modo que m células da fita de M correspondam (comprimidas)

a uma só célula em M'. E deste modo m passos de M correspondem a um passo de M'. No ínicio M' tem de ler os dados x de M e comprimi-los. Se M só tinha uma fita M' tem de ter duas. Primeiro M' lê os dados x em blocos de m símbolos $(\sigma_1, \ldots, \sigma_m)$ (usando os estados para recordar esses símbolos) e escreve o símbolo resultante na segunda fita. Depois M' simula m passos de M (um estágio) em no máximo 6 passos. Cada estado de M' é da forma $(s, j_1, \ldots, j_k)$ onde s é o estado de M no início do estágio e cada $j_i < m$ é a posição exacta de cada cabeça de M no bloco de m símbolos e k o número de fitas. Se em M a cabeça i estiver no l-ésimo símbolo, então $j_i = l \mod m$ e a cabeça da fita $i+1$ de M' está no símbolo $\lceil \frac{l}{m} \rceil$. Nos quatro primeiros passos M' move todas as cabeças para uma posição para esquerda, duas para a direita e uma para esquerda. Neste processo regista no estado qual os símbolos à esquerda e à direita do símbolo corrente (todos blocos de m símbolos). Para tal, o conjunto de estados terá de incluir o conjunto $S \times \{1, \ldots, m\} \times \Gamma^{3mk}$. Com a informação recolhida M' simula m passos de M em no máximo dois passos (dois se tiver de considerar um dos símbolos vizinhos, isto é cabeça tem de ir também para o vizinho). M' aceita se M aceita. No total tempo gasto por M' com dados x é no máximo $|x| + 2 + 6\lceil \frac{|f(x)}{m} \rceil$. Logo, basta tomar $m = \lceil \frac{6}{c} \rceil$. $\square$

Dada uma função $f(n)$, define-se a classe de complexidade DTIME($f(n)$) por:

$$\text{DTIME}(f(n)) \quad = \quad \{L \mid L \text{ é decidível por uma TM multi-fita } M \text{ tal que } T_M(n) \text{ é } O(f(n))\}$$

A classe P é definida como o conjunto de linguagens que são decidíveis em tempo polinomial por uma MT, isto é:

$$\mathsf{P} \quad = \quad \bigcup_{k \geq 1} \text{DTIME}(n^k)$$

2.1.2 Classe NP

Seja $N = (S, \Sigma, \Gamma, s_0, \Delta, \Box, \rhd, F)$ uma máquina de Turing não-determinística (MTN) com 1-fita, onde

$$\Delta \subseteq (S \times \Gamma) \times (S \times \Gamma \times \{\leftarrow, \rightarrow\})$$

Recorde que um passo de computação entre duas configurações é definido por uma relação $xsy \xrightarrow{M} x's'y'$ se e só se existe $((s, \alpha), s', \alpha', d) \in \Delta$ que torne a transição possível. Então uma computação de N (em n passos, $\xrightarrow{N}{}^{n}$) com dados $x \in \Sigma^\star$, $\Sigma \subseteq \Gamma$, pode ser vista como uma árvore de computação. Cada caminho da árvore iniciado na raiz é um *caminho de computação*. Diz-se que N aceita $x \in \Sigma^\star$, se a árvore de computação de N com dados x incluí pelo menos um caminho de computação que termina num estado de aceitação. Isto é, N aceita x se existe uma sequência de escolhas não-determinísticas que leva a um estado de aceitação. Note-se que x só é rejeitado se *todas* as sequências de escolhas (caminhos de computação) conduzirem a um estado de rejeição.

Note ainda a assimetria entre a aceitação e a rejeição por estas máquinas, que não existia nas máquinas de Turing determinísticas. Seja $L(N) = \{x \in \Sigma^\star \mid N \text{ aceita } x\}$. Se $L \subseteq \Sigma^\star$, N decide L se $L = L(N)$.

De modo análogo funciona uma máquina de Turing não determinística N com k-fitas. Se N aceita x, o tempo associado é definido pelo número de passos do menor caminho de computação que leva a um estado de aceitação, isto é:

$$T_N(x) \quad = \quad min\{t \mid N \text{ aceita } x \text{ com um caminho de computação em } t \text{ passos}\}$$

e, a complexidade temporal de N é a função:

$$T_N(n) \quad = \quad max\{\{1\} \cup \{m \mid \exists\, x \in L(N),\ |x| = n \text{ e } m = T_N(x)\}\}$$

Note-se, que $T_M(n) = 1$ se não há nenhum x, com comprimento n que seja aceite por N. Se $T_N(n)$ é $O(f(n))$, diz-se que N opera em tempo limitado por $f(n)$ e se $f(n)$ é um polinómio dizemos que N é polinomial.

Seja,

$$\text{NTIME}(f(n)) \quad = \quad \{L \mid L \text{ é decidível por uma MTN } N \text{ tal que } T_N(n) \text{ é } O(f(n))\}$$

Teorema 2.2 *Se* $L \in \text{NTIME}(f(n))$ *então existe* $c > 1$ *tal que* $L \in \text{DTIME}(c^{f(n)})$.

Dem. Seja N uma MTN que executa em tempo $f(n)$ e seja $d > 1$ o máximo das escolhas possíveis de $\Delta(s, \sigma)$ para qualquer estado s e símbolo σ. Então qualquer sequência finita de escolhas pode ser representada por uma sequência de dígitos de 1 a d.

Uma MT M com 3 fitas pode simular N. A primeira contém a sequência de entrada. A segunda, gera sequências de dígitos de 1 a d dum modo sistemático. Por exemplo, por ordem crescente de comprimento e por ordem lexicográfica se de igual comprimento: 1, ..., d, 11,...1d, ..., 21, ..., 2d, ..., 111, ..., ddd, ...

Para cada sequência gerada na segunda fita, M copia a sequência de entrada para a terceira fita e simula N na terceira fita, usando a sequência da segunda fita para decidir os movimentos de N. Nota que cada sequência de escolhas corresponde a uma sequência de configurações de N. Deste modo M simula sucessivamente todas as sequências de k movimentos de N, $k = 1, 2, \ldots$, 1,2,11,12,21, 111,112,113,121,122,211,212,213,1111, ... M explora a árvore de configurações de N em largura. Se N atinge um estado de aceitação, então M aceita. Se nenhuma sequência de escolhas com um mesmo tamanho t conduz a N aceitar e embora todas terminem, M também não aceita. O número de sequências de escolhas é majorado por $\sum_{t=1}^{f(n)} d^t = O(d^{f(n)+1})$. O tempo de gerar e considerar cada escolha é $O(2^{f(n)})$. Assim o tempo total requerido por M é $O(d^{f(n)+1})O(2^{f(n)}) = O(c^{f(n)})$, para alguma constante $c > 1$. □

A classe NP é definida com o conjunto de linguagens que são decidíveis em tempo polinomial por uma MTN, isto é:

$$\mathsf{NP} \;=\; \bigcup_{k\geq 1} \mathrm{NTIME}(n^k)$$

Corolário 2.1 $\mathsf{P} \subseteq \mathsf{NP}$

2.2 Problemas, Linguagens e Complexidade

Para resolver um problema por meios computacionais é necessário codificá-lo numa dada linguagem e escrever um algoritmo (um programa, nessa mesma linguagem), isto é, um método para resolver o problema num número finito de passos. Vamos considerar que cada problema pode ser descrito por um conjunto de parâmetros e de relações entre eles, com as quais se expressam as condições de solução do problema. Uma instância I dum problema Π, é obtida especificando valores para cada um dos parâmetros. Chamaremos instância genérica à descrição dum problema em termos dos seus parâmetros, ou apenas instância se na enunciação do mesmo. Por exemplo, considere-se um *caixeiro viajante* que tem de visitar todas as cidades de um dado país e voltar à cidade donde partiu, e pretende fazê-lo pelo caminho mais curto. Podemos enunciar este problema clássico da combinatória, em termos dum conjunto finito de elementos e de propriedades desses elementos:

(TSO) Dado um conjunto de cidades e as distâncias entre elas existe uma permutação das cidades que minimize a distância total?

Instância: $C = \{c_1, c_2, \ldots c_n\}$ e $d : C \times C \longrightarrow \mathbb{N}$ distância

Questão: Qual é a permutação $\pi : C \longrightarrow C$ que minimiza

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

Uma instância do problema é por exemplo $C = \{Porto, Aveiro, Viana, Espinho\}$ e $d(P, A) = 60$, $d(P, V) = 60$, $d(P, E) = 20$, $d(A, V) = 120$, $d(A, E) = 40$, $d(V, E) = 80$. A sequência $< V, P, E, A >$ minimiza a distância total, $d_T = 120$.

Pretende-se normalmente obter um algoritmo que (se existir!) seja o mais eficiente. A eficiência dum algoritmo pode-se medir em:

- tempo de execução em função dos dados
- espaço gasto para a obtenção da solução do problema.

Para cada problema a complexidade temporal vai depender da máquina usada e da codificação escolhida. Contudo, cada problema pode ser classificado do ponto de vista computacional, como:

- insolúvel
- tratável
- intratável

Esta classificação é válida independentemente da máquina ou modelo computacional usado (máquina de Turing, RAM's, etc) e da codificação escolhida, desde que seja "razoável", isto é, que satisfaça as seguintes condições:

1. A codificação duma instância I, deve ser concisa e não conter símbolos desnecessários.
2. Os números inteiros devem ser representados numa base maior ou igual a 2.

Podemos supor que os números inteiros se representam em binário; os conjuntos pela sequência dos seus elementos (codificados) e os grafos pela sequência de vértices seguida dos pares de vértices para cada ramo (ou pela matriz de adjacência). Neste caso, se um dado é um número n o seu comprimento será $\lceil \log_2 n \rceil$, se é um conjunto de n elementos o seu comprimento é n e se é um grafo o seu comprimento é o número de vértices mais o de ramos.

Considerem-se os seguintes problemas e procuremos resolvê-los !

1. Problema da Paragem

   Instância: Um algoritmo e um conjunto de dados para esse algoritmo.

   Questão: Esse algoritmo pára para esse conjunto de dados (ou entra num ciclo infinito)?

   Resposta: Não existe nenhum algoritmo que resolva este problema. É um problema insolúvel.

2. Procura dum elemento numa sequência.

   Instância: X, $V[i]$, $1 \leq i \leq n$

   Questão: Existe i tal que $X = V[i]$?

   Resposta: Existem algoritmos polinomiais que o resolvem, por exemplo:

   $k \leftarrow 1$
   $V[i+1] \leftarrow X$
   while $V[k] \neq X$ **do**
   $k \leftarrow k + 1$
   end while
   if $k \neq n$ **then** PRINT("Sim")
   else PRINT("Não")

end if

A complexidade deste algoritmo é $O(n)$, considerando o número de comparações que serão necessárias no "pior caso".

3. (HC) Existência de ciclo hamiltoniano

Instância: Seja $G = (V, E)$ um grafo dirigido.

Questão: Existe um ciclo $< (X_1, X_2)(X_2, X_3) \ldots (X_{n-1}, X_n)(X_n, X_1) >$, $X_i \in V$, $(X_i, X_{i+1}) \in E$ e e $n = |V|$, tal que todos os X_i são diferentes?

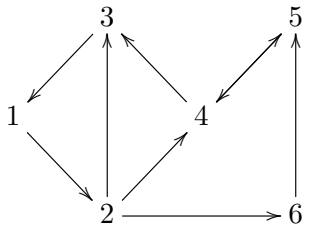
Resposta: Não se conhece nenhum algoritmo polinomial que resolva este problema. Um algoritmo que corre em tempo exponencial no número de vértices é o seguinte:

```
i ← 1
for i ≤ n do
    marca[i] ← False
end for
k ← 0
if PROCURA(1) then PRINT("Sim")
else PRINT("Não")
end if
function PROCURA(i)
    k ← k + 1
    marca[i] ← True
    for all (i, j) ∈ E do
        if j = 1 ∧ k = n then return True
        end if
        if ¬marca[j] then
            if PROCURA(j) then return True
            end if
        end if
    end for
    k ← k − 1
    marca[i] ← False return False
end function
```

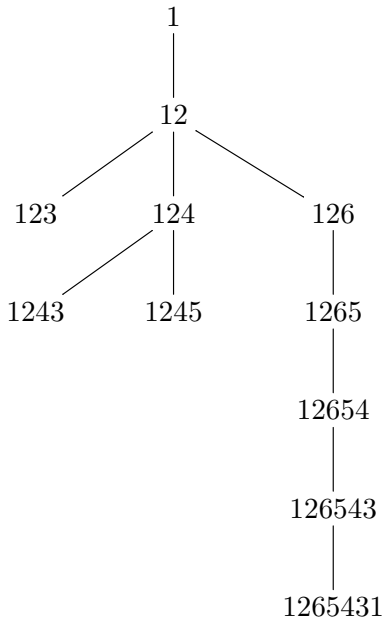
Exemplo 2.1 *Seja* $G = (V, E)$ *onde*

$$\begin{aligned} V &= \{1, 2, 3, 4, 5, 6\} \\ E &= \{(1,2), (2,3), (3,1), (2,4), (2,6), (4,3), (4,5), (5,4), (6,5)\} \end{aligned}$$

com o seguinte diagrama:



Neste caso, temos a seguinte árvore de procura, se $X_1 = 1$:



Se G é um grafo completo, o tempo de execução é $O(n)$ (caso melhor) mas no pior caso o algoritmo tem complexidade $O(n!)$.

Exercício 2.2.1 Escreve um programa para cada um dos problemas 2 e 3 e obtenha o tempo de execução para vários valores dos parâmetros. Comenta os resultados.

Um problema é *tratável* se existe um algoritmo polinomial que o resolve. Se todos os algoritmos conhecidos para resolver um problema forem exponenciais o problema diz-se *intratável*. Isto porque o tempo de execução dum algoritmo exponencial torna-se rapidamente insuportável (quase "infinito") para comprimentos pequenos dos dados. O quadro seguinte ilustra a diferença entre as complexidades temporais citadas. Para valores crescentes do comprimento dos dados, n, indica o tempo de execução, tomando para base o microsegundo.

Complexidade temporal	Valores de n			
	10	20	40	60
n	10 μs	20 μs	40 μs	60 μs
n^2	0.1 ms	0.4 ms	1.6 ms	3.6 ms
n^5	0.1 s	3.2 s	1.7 minutos	13 minutos
2^n	10 ms	1 s	12.7 dias	366 séculos
10^n	2.7 horas	3.1×10^4 séculos	3.1×10^{24} séculos	

É importante notar que esta situação não se modifica significativamente se as máquinas usadas forem muito mais rápidas. Considerando aumentos de 100 e 1000 vezes na velocidade dum processador podemos ver pelo quadro seguinte, que o tamanho máximo dos problemas resolvidos num dado tempo por um algoritmo exponencial não é praticamente alterado (principalmente em comparação com o que sucede com os algoritmos polinomiais, mesmo de grau elevado).

Complexidade temporal	Máximo comprimento dos dados para 1 hora de CPU		
	Computador actual	Computador 100x rápido	Computador 1000x rápido
n	X_1	$100X_1$	$1000X_1$
n^2	X_2	$10X_2$ n	$31.6X_2$
n^5	X_3	$2.5X_3$	$3.98X_3$
2^n	X_4	$X_4 + 6.64$	$X_4 + 9.97$
10^n	X_5	$X_5 + 2$	$X_5 + 3$

Note-se contudo que um algoritmo exponencial não significa que para todas as instâncias demore um tempo que é exponencial nos dados, mas sim que no pior caso pode demorar esse tempo. Isto permite na prática usar algoritmos exponenciais para resolver alguns problemas (ex: método simplex para resolver problemas de programação linear).

Facto: Existe uma grande família de problemas para os quais só se conhecem algoritmos exponenciais - essencialmente problemas de optimização e decisão que exigem procuras exaustivas em determinados conjuntos.

A seguir apresentámos alguns problemas dos quais uns pertencem a essa classe e outros para os quais são conhecidos algoritmos polinomiais para os resolver.

(EC) Dado um grafo $G = (V, E)$, G tem um ciclo euleriano, isto é, que passe por todos os ramos uma só vez?

(HC) Dado um grafo $G = (V, E)$, G tem um ciclo hamiltoniano, isto é, que passe por todos os vértices uma só vez ?

(PRIMO) Dado n inteiro, n é primo?

(SET SPLIT) Dada uma família C de subconjuntos de um conjunto S, existe uma partição de S, $S = S_1 \cup S_2$, e tal que $\forall\ S' \in C,\ S' \cap S_1 \neq \emptyset$ e $S' \cap S_2 \neq \emptyset$

(PAR) (Partição) Dado um conjunto A e uma função $s : A \longrightarrow \mathbb{N}$, existe $A' \subset A$ tal que

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$$

(VC) (Cobertura por vértices) Dado um grafo $G = (V, E)$ e $K \leq \mid V \mid$ existe $V' \subseteq V$, $\mid V' \mid \leq K$ e $\forall (a, b) \in E$, $a \in V'$ ou $b \in V'$.

(GI) (Isomorfismo de grafos) Dados dois grafos dirigidos $G_i = (V_i, E_i)$, $i = 1, 2$ decidir se existe um isomorfimo de grafos entre eles, i.e $f : V_1 \longrightarrow V_2$ bijecção tal que $\forall (a, b) \in E_1, (f(a), f(b)) \in E_2$?

(LP) (Programação Linear) Dada uma matriz de inteiros A e dois vectores de inteiros b e c, existe um vector real x tal que $Ax \leq b$ e maximize cx ?

(IP) Como o anterior mas x inteiro (ou até com valores em $0, 1$).

(DIOF) Dada uma equação de coeficientes inteiros, existem soluções inteiras?

(3DM) (Emparelhamento 3 dimensional) Dado um conjunto $M \subseteq W \times X \times Y$, onde $|W| = |X| = |Y| = q$ e disjuntos, existe um subconjunto $M' \subseteq M$, tal que $|M'| = q$ e nenhum elemento de M' tem uma "coordenada"comum?

(SAT) Um literal é uma variável lógica ou a sua negação. Uma cláusula é uma disjunção de literais. Dado um conjunto de cláusulas (representando a sua conjunção) $C = \{c_1, c_2, ..., c_m\}$ com literais num conjunto finito U de variáveis lógicas, existe uma atribuição de valores de verdade às variáveis de tal modo que todas as cláusulas de C sejam simultaneamente satisfeitas?

(3SAT) Análogo ao anterior com todas as cláusulas com 3 literais ?

(2SAT) Análogo ao anterior com todas as cláusulas com 2 literais ?

(K-COLOR) Dado um grafo $G = (V, E)$ e um inteiro k, pode-se colorir G com k cores, isto é, existe uma função $\Phi : \mathbb{N} \longrightarrow \mathbb{Z}_k$, tal que, se $\{u, v\} \in E$, então $\Phi(u) \neq \Phi(v)$?

2.2.1 Problemas de decisão

Vamos começar por analisar um tipo particular de problemas: os problemas de decisão. Um problema de decisão Π é um problema que apenas tem duas soluções possíveis: ou *sim* ou *não*. O conjunto das suas instâncias D_Π é divido em dois subconjuntos S_Π e N_Π, correspondentes respectivamente, às instâncias para as quais a resposta é *sim* e às instâncias para as quais a resposta é *não*. Estes problemas podem-se converter facilmente em problemas de reconhecimento de linguagens e, portanto, é possível estudar a sua complexidade computacional formalmente.

Dado um conjunto finito de símbolos Σ (alfabeto), uma linguagem L é um subconjunto de $\Sigma^\star$. Seja Σ um conjunto finito de símbolos e $c : D_\Pi \longrightarrow \Sigma^\star$, uma codificação de instâncias dum problema de decisão Π em $\Sigma^\star$. A *linguagem associada a* Π para a codificação c é:

$$\begin{aligned} L(\Pi, c) \quad = \quad & \{x \in \Sigma^\star \mid x = c(\mathsf{I}) \text{ onde } \mathsf{I} \text{ é uma instância para a qual} \\ & \qquad \text{a resposta de } \Pi \text{ é } \textit{sim}, \mathsf{I} \in S_\Pi\} \end{aligned}$$

Resolver o problema Π equivale a reconhecer $L(\Pi, c)$ usando um modelo computacional, por exemplo, uma máquina de Turing. Mais que isso, se as codificações usadas forem razoáveis podemos falar apenas na linguagem L associada a um problema Π. Isto é, se c e c' forem duas codificações razoáveis de um problema Π as linguagens $L(\Pi, c')$ e $L(\Pi, c)$ terão as mesmas propriedades (formais, computacionais).

Dado que a complexidade temporal dum algoritmo é uma função do comprimento dos dados, isto é, de cada instância do problema, é necessário obter esta medida independentemente das codificações. Assim, para cada problema de decisão Π associámos, independentemente da codificação, uma função comprimento $comp : D_\Pi \longrightarrow \mathbb{N}$ nas instâncias I de Π que é *relacionada polinomialmente*[2] com o comprimento da sequência de símbolos de qualquer codificação razoável de I. Isto é, para qualquer codificação c de Π existem das funções polinomiais p e p' tal que se $\mathsf{I} \in D_\Pi$ e $x = c(\mathsf{I})$ então $comp(\mathsf{I}) \leq p(|x|)$ e $|x| \leq p'(comp(\mathsf{I}))$.

Por exemplo, no problema de decisão do *caixeiro viajante*, (TSP), (ver página 25) o comprimento duma instância genérica I é:

$$comp(I) \quad = \quad n + \lceil \log_2 B \rceil + max\{\log_2 d(c_i, c_j) \mid c_i,\, c_j \in C\}$$

Exercício 2.2.2 Escolhe uma codificação para o problema TSP e verifica que a função *comp* se *relaciona polinomialmente* com essa codificação.

Podemos então considerar que todos os problemas são codificados numa mesma codificação razoável. Isto, permite dum modo informal comparar as propriedades computacionais de dois problemas sem ter de os exprimir em termos de cada uma das suas codificações. Basta saber que existe uma. Exemplo duma codificação seria considerar as sequências finitas sobre o o conjunto

[2] Lembrar que, para um dado problema, apenas estamos interessados na existência de algoritmos polinomiais para o resolver. Ou ainda mais eficientes.

$\Sigma = \{0,1\}$. Então, qualquer problema de decisão pode ser identificado com uma linguagem $L \in \{0,1\}$. Normalmente, para escrever algoritmos numa dada linguagem de programação, usam-se codificações menos minimalistas. Ver Exercício 1.2.3 da Secção 1.2.

Informalmente podemos falar na *complexidade* de um problema e dizer que um problema *pertence* a uma dada classe de complexidade. Isto é, identificar um problema de decisão com a linguagem associada. Toda a teoria a seguir apresentada tem como base esta equivalência e o que será dito para problemas e algoritmos é exprimível formalmente em termos de linguagens e modelos computacionais.

A menos referência em contrário, os problemas considerados serão problemas de decisão. Note-se que dado um problema de optimização – em que não só se pretende saber se um problema tem solução, mas também qual é essa solução (óptima) – é sempre possível formular um problema correspondente de decisão. O problema de decisão é tratável se o de optimização o for, isto é, se o de decisão é intratável o de optimização também o é (o problema de decisão não é mais difícil que o correspondente de optimização). O inverso também é verdade para muitos casos. Consideremos o seguinte exemplo:

(CIIQUE MÁXIMO) Um subgrafo completo maximal dum grafo G chama-se clique. O tamanho dum clique é o seu número de vértices.

Instância: Um grafo $G = (V, E)$ não dirigido.

Questão: Qual o tamanho do maior clique de G ?

O problema de decisão associado é:

(CLIQUE)

Instância: Um grafo $G = (V, E)$ não dirigido e um inteiro $k \leq |V|$

Questão: G contém algum clique com tamanho maior ou igual a k ?

Seja CLID(G, k) um algoritmo que resolve o problema de decisão CLIQUE. Se $|V| = n$, então o tamanho do maior clique pode ser determinado aplicando CLID(G, k) para $k = n, n-1, n-2, \ldots$ até que o resultado desse algoritmo seja *sim*. Se CLID tem complexidade $f(n)$, então algoritmo para determinar o maior clique tem complexidade $n \times f(n)$. Inversamente, se o tamanho do maior clique pode ser determinado em tempo $g(n)$ então o problema de decisão pode ser resolvido em tempo $g(n)$.

Exercício 2.2.3 *Dado um conjunto U de variáveis lógicas uma atribuição de valores para U é uma função $t : U \longrightarrow \{0,1\}$. Se $t(u) = 1$, $u \in U$, dizemos que u é* verdade *em t, caso contrário dizemos que a variável u é* falsa *em t. Se $u \in U$ então u e $\overline{u}$ são literais em U. O literal u é* verdade *em t sse a variável u é* verdade *em t. O literal $\overline{u}$ é* verdade *em t sse a variável u é* falsa *em t. Uma*

cláusula em U é um conjunto (finito) de literais em U e é satisfeita por uma atribuição de valores, t, sse pelo menos um dos seus elementos é verdade para t. Considera, então o seguinte problema:

(SATA)

Instância: *Seja $C = \{c_1, c_2, \ldots, c_m\}$ um conjunto de cláusulas com literais num conjunto finito $U = \{u_1, u_2, \ldots, u_n\}$*

Questão: *Determinar uma atribuição de valores para U que satisfaça simultaneamente todas as cláusulas de C?*

Mostra que se existir um algoritmo polinomial $\mathsf{SAT}(C)$ que determine se um conjunto de cláusulas é satisfazível então o problema SATA também é resolúvel em tempo polinomial.

Sugestão:

Vamos supor que em C ocorrem todos os elementos de U. Basta então construir um algoritmo que utilize $\mathsf{SAT}()$ n vezes, de cada vez para um C' que resulta de C substituindo sucessivamente cada uma das variáveis por o valor 0 ou 1:

$C' \leftarrow C$
for all $u[i] \in U$ ***do***
  $C'' \leftarrow$ *obtido de* C' *substituindo* 0 *pelos* $u[i]$ *e* 1 *por cada* $\overline{u}[i]$
  if $\mathsf{SAT}(C'') = 1$ ***then***
    $t(u[i]) \leftarrow 0$
    $C' \leftarrow C''$
  else
    $t(u[i]) \leftarrow 1$
    $C' \leftarrow$ *obtido de* C' *substituindo* 1 *por* $u[i]$ *e substituindo* 0 *por* $\overline{u}[i]$
  end if
end for

2.2.2 Problemas em P

A noção de algoritmo que usamos até agora e que é a habitual, pressupõe que cada instrução é univocamente determinada, isto é, para cada conjunto de dados existe uma só solução. Chamemos a estes algoritmos - que correspondem a máquinas de Turing determinísticas que param sempre ou a programas executáveis por um computador - algoritmos determinísticos. Podemos então identificar a classe P com:

$$\mathsf{P} = \{\text{conjunto de todos os problemas de decisão que são resolúveis por um algoritmo determinístico em tempo polinomial}\}$$

Da lista de problemas dada (EC), (2SAT) e (2-COLOR) pertencem a P.

Exercício 2.2.4 *Encontra um algoritmo determinístico polinomial para cada um desses problemas.* Sugestão:

a) EC *Para um grafo (não dirigido) ter um ciclo euleriano tem de ser conexo e todos os vértices terem grau par. Estas duas condições podem ser implementadas em tempo polinomial.*

b) 2SAT *Sendo* $U = \{u_1, \ldots, u_n\}$ *e* C *o conjunto de cláusulas, construir o grafo (dirigido)* $G = (V, E)$*, onde* $V = \{u_1, \ldots, u_n, \overline{u_1}, \ldots, \overline{u_n}\}$ *e sendo* l_i *igual a* u_i *ou* $\bar{u}_i$*,* $(l_i, l_j) \in E$ *se* $\{\overline{l_i}, l_j\} \in C$ *(isto é,* $u_i \Rightarrow l_j$*). Mostra que* C *é satisfazível se e só se as componentes fortemente conexas de* G *não contêm simultaneamente uma variável e a sua negação.*

c) 2-COLOR *Para colorir um grafo com* 2 *cores vamos supor que o grafo é conexo (senão aplicámos o algoritmo a cada componente conexa). O seguinte algoritmo é* $O(n^3)$*, onde* n *é a ordem do grafo:*

```
colorir o vértice 1
f ← True
while f do                                  ▷ enquanto houver vértices para colorir
    f ← False
    i ← 1
    for i ≤ n do                            ▷ pelo menos um vérice é colorido
        if ¬COLORIR(i) then
            f ← True
        end if
    end for
end while
PRINT(sim)
function COLORIR(i)
    if i está colorido then return True
    end if
    if i não tem vizinhos coloridos then return False      ▷ i tem no máximo n − 1 vizinhos
    else
        if os vizinhos têm cores contraditórias then PRINT(não)
        else
            colorir i com a cor alternativa return True
        end if
    end if
end function
```

Também estão em P os problemas LP e PRIMO (2005).

2.2.3 Problemas em NP

Considere-se o problema do *caixeiro viajante* expresso como um problema de decisão:

(TSP) Dado um conjunto de cidades, as distâncias entre elas, e um limite B, existe um itinerário pelas cidades cuja distância total é menor ou igual a B ?

Instância: $C = \{c_1, c_2, \ldots c_n\}$, $d : C \times C \longrightarrow \mathbb{N}$ distância e $B \in \mathbb{N}$

Questão: Existe uma permutação $\pi : C \longrightarrow C$ tal que

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}) \leq B$$

Como se disse não se conhece nenhum algoritmo polinomial que resolva este problema. Contudo, suponhamos que temos um mecanismo que nos gera uma possível solução. Podemos verificar se ela realmente é uma solução, bastando para isso calcular o seu comprimento e compará-lo com B! E isto pode ser feito em tempo polinomial.

O mecanismo mencionado acima pode ser simulado por um algoritmo que permita a existência de instruções cujo resultado não é univocamente determinado mas sim limitado a um conjunto finito de possibilidades: algoritmo não-determinístico de verificação (equivale a uma máquina de Turing não-determinística). Um tal algoritmo termina com insucesso (resposta *não*) se e só se nenhum conjunto de escolhas conduz a uma resposta *sim* (caso em que, o algoritmo termina com sucesso). Se existir algum conjunto de escolhas que conduza a uma solução de sucesso, esse conjunto é sempre gerado, isto é, em qualquer instrução que envolva uma escolha é escolhido um elemento correcto dum desses conjuntos. Para tal efeito serão usadas a função e as instruções seguintes:

- ESCOLHA(S) escolha arbitrária de um elemento de S
- SUCESSO indica que o algoritmo termina com a resposta *sim*
- INSUCESSO indica que o algoritmo termina com a resposta *não*

Supõe-se que cada uma destas instruções é executada em tempo constante, isto é, $O(1)$. Por exemplo, o problema da procura dum elemento numa sequência pode ser *resolvido* não-deterministicamente pelo seguinte algoritmo:

```
i ←ESCOLHA(1...n)
if V[i] = X then
    PRINT(sim)
    SUCESSO
else
    PRINT(não)
    INSUCESSO
```

end if

Neste caso se existir algum $V[i]$ igual a X, um desses i é gerado na primeira instrução e o algoritmo termina em sucesso, caso contrário termina em insucesso. Mais precisamente, comparando com a noção clássica de algoritmo, o algoritmo *não resolve*[3] o problema, apenas *verifica* se uma possível solução é efectivamente solução.

Exemplo 2.2 *Para mostrar que TSP está em* NP *basta considerar o seguinte algoritmo não-determinístico:*

$i \leftarrow 1$
while $S \neq \emptyset$ ***do***
  $c_i \leftarrow$ESCOLHA*(S)*
  $i \leftarrow i + 1$
  $S \leftarrow S \setminus \{c_i\}$
end while
if $\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}) \leq B$ ***then***
  PRINT*(sim)*
  SUCESSO
else
  PRINT*(não)*
  INSUCESSO
end if

Exercício 2.2.5 *Para cada um dos seguintes problemas escreve um algoritmo não-determinístico:*

1. *Ordenar uma sequência de inteiros positivos.*
2. *(HC) Existência de ciclo hamiltoniano num grafo (dirigido ou não).*
3. *(CLIQUE)*

Admitindo processamento paralelo, podemos interpretar um algoritmo não-determinístico do seguinte modo: Sempre que existe uma escolha o algoritmo produz várias cópias de si mesmo, uma para cada uma das possíveis alternativas. Estas cópias são executadas em simultâneo. Se uma chega a um resultado de SUCESSO todas as outras terminam. Se uma cópia chega a um resultado de INSUCESSO só essa cópia termina.

O tempo requerido por um algoritmo não-determinístico para um dado conjunto de dados é o menor número de passos necessários para chegar a um resultado de SUCESSO. Se tal não for possível o tempo requerido é $O(1)$. A sua complexidade é $O(f(n))$ se para todos os dados de comprimento n, $n \geq n_0$ que conduzem a um resultado de sucesso, o,tempo requerido é no máximo $cf(n)$, para

[3]Isto porque não é indicado um método para efectuar cada uma das escolhas.

c e n_0 constantes. Por exemplo, o algoritmo não-determinístico dado para procurar um elemento numa sequência é $O(1)$.

Um algoritmo não-determinístico diz-se *polinomial* se a sua complexidade é $O(p(n))$, para algum polinómio, $p(n)$. Formalmente podemos identificar estes algoritmos com máquinas de Turing não-deterministicas polinomiais e os problemas correspondentes com linguagens da classe NP. Temos:

$$\mathsf{NP} = \{\text{conjunto de todos os problemas de decisão que são resolúveis por um algoritmo não-determinístico polinomial}\}$$

Exercício 2.2.6 *Determina a complexidade temporal de cada um dos algoritmos encontrados no Exercício 2.2.5.*

Exercício 2.2.7 *Obtem um algoritmo não-determinístico de $O(n)$ que determine se existe um subconjunto de n números a_i, $1 \leq i \leq n$ cuja soma seja M.*

A noção de *verificação polinomial* duma solução pode ainda formalizar-se em termos de *testemunhas concisas* duma linguagem.

Teorema 2.3 *Uma linguagem $L \subseteq \Sigma^\star$ pertence a* NP *se e só se existe uma linguagem $R_L \subseteq \Sigma^\star \times \Sigma^\star$ que pertence a* P *e uma função polinomial p tal que:*

$$L \;=\; \{x \mid \exists y.\langle x, y\rangle \in R_L \text{ e } |y| \leq p(|x|)\}$$

Para cada $x \in L$, y diz-se uma testemunha *concisa (ou polinomial) da $x \in L$.*

Dem. ($\Leftarrow$) Suponhamos que $L = \{x \mid \exists y.\langle x, y\rangle \in R_L \text{ e } |y| \leq p(|x|)\}$ e $R_L \in \mathsf{P}$ e $p(n)$ é polinomial. Então L é decidida pela seguinte MTN N: Com dados x, N gera uma sequência y de comprimento no máximo $p(|x|)$ e depois usa a TM que decide R_L para testar (verificar) em tempo polinomial se $\langle x, y\rangle \in R_L$. Se $\langle x, y\rangle \in R_L$ N aceita x caso contrário rejeita.[4]

($\Rightarrow$) Seja $L \in \mathsf{NP}$. Então existe uma MTN N que decide L e cuja complexidade temporal é limitada por uma função polinomial $p(n)$. Para cada $x \in \Sigma^\star$, cada caminho de computação (sequência de escolhas de Δ) de N com dados x pode ser codificado numa sequência y de símbolos de Σ tal que $|y| \leq p(|x|)$. A linguagem R_L é definida do seguinte modo: um par $\langle x, y\rangle$ pertence a R_L se y codifica um caminho duma computação de aceitação de N com dados x. É fácil ver que $R_L \in \mathsf{P}$ e que $x \in L$ se e só se $\exists y, (\langle x, y\rangle \in R_L \text{ e } |y| \leq p(|x|)$. $\square$

Alternativamente, podemos ainda considerar uma variante de máquinas de Turing determinísticas: máquinas de Turing com "escolhas" (de "adivinhação") polinomiais.

Considera a seguinte definição duma máquina de Turing com "escolhas" (de "adivinhação"), ver [GJ79, Cap. 2.3] ou [AB09].

[4] Note que o número de sequências y é finito.

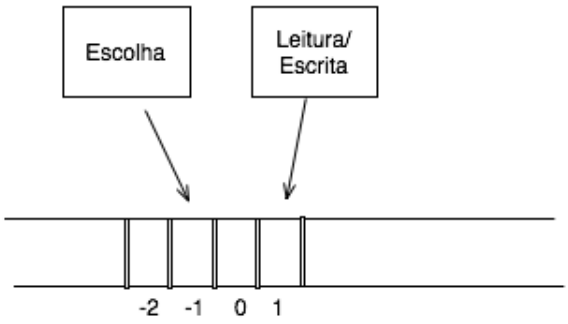


Figura 2.1: Uma MTV com 2 cabeças: uma escolhe e outra é de leitura/escrita.

Definição 2.2 *Uma MTV, M, é uma máquina de Turing determinística com fita duplamente infinita à qual foi adicionado um módulo de* escolha *com apenas uma cabeça de escrita. A computação duma máquina MTV para dados $x \in \Sigma^*$ é feita em dois estágios:*

Escolha *Inicialmente os dados x estão escritos na fita a partir da posição 1 até $|x|$, os restantes em branco ($\Box$), a cabeça de leitura-escrita na posição 1, a cabeça de escrita na célula -1 e o controlo finito desactivado (i.e. não está em nenhum estado). O módulo de* escolha *dirige então a cabeça de escrita, ou escrevendo um símbolo de Γ e movendo-se para a esquerda, ou parando; neste último caso este módulo fica inactivo e entra-se no estado s_0. Nota que este módulo escreveu um qualquer $s \in \Gamma^\star$.*

Verificação *A computação procede exactamente como numa máquina de Turing determinística*

A computação pára se é atingido um estado final e diz-se que é uma computação de aceitação se pára num estado de aceitação. Dizemos que M aceita x se existir pelo menos uma computação de aceitação para x.

Exercício 2.2.8 Mostra que uma máquina MTV é equivalente (polinomialmente) a uma MTN. Sugestão: Considera a demonstração do Teorema 2.3.

2.2.4 Relação entre P e NP

Já se viu que $\mathsf{P} \subseteq \mathsf{NP}$ (Figure 2.2) Pelo que foi dito no final da secção anterior, um algoritmo não-determinístico é essencialmente constituído por dois passos:

- escolher uma solução

- verificar se é solução

Se é polinomial isso equivale a dizer que a *verificação* da solução do problema pode ser feita em tempo polinomial. Temos assim que a distinção entre as classes P e NP corresponde diferença entre a resolução dum problema em tempo polinomial e a sua verificação em tempo polinomial.

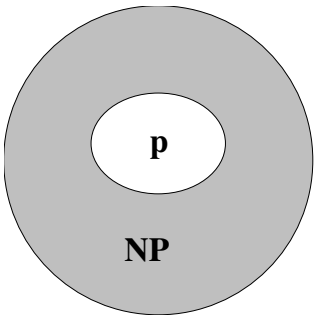


Figura 2.2: Relação entre P e NP

Será que NP $\subseteq$ P ? Até agora ainda ninguém soube responder a esta pergunta.

Exercício 2.2.9 *Mostrar que se* $\Pi \in$ NP *então existe um polinómio* $p(n)$ *tal que* Π *pode ser resolvido por um algoritmo determinístico com complexidade* $O(2^{p(n)})$.

2.2.5 Reduções entre problemas

Um problema A reduz a um problema B se dado um algoritmo para resolver B podemos construir um algoritmo que resolva A. Este conceito de redutibilidade é central na teoria da computabilidade e também vai ser fulcral na teoria dos problemas NP-completos.

Facto: Muitos dos problemas para os quais não se conhecem algoritmos polinomiais, reduzem-se polinomialmente uns aos outros de tal forma que se um deles tiver um algoritmo polinomial todos os outros têm.

Formalmente temos:

Definição 2.3 *Uma transformação polinomial duma linguagem* $L_1 \subseteq \Sigma_1^\star$ *numa linguagem* $L_2 \subseteq \Sigma_2^\star$ *é uma função* $f : L_1 \longrightarrow L_2$ *que satisfaz as seguintes condições:*

1. *Existe uma máquina de Turing MT polinomial que calcula* f
2. *Para todo o* $x \in \Sigma_1^\star$, $x \in L_1$ *se e só se* $f(x) \in L_2$

Em termos de problemas, dizemos que um problema A *reduz-se polinomialmente* a um problema B, $A \leq_m^p B$, se e só se existe um algoritmo determinístico polinomial, F, que transforma cada instância $X \in A$ numa instância $F(X) \in B$, tal que X tem solução se e só se $F(X)$ tiver (Figura 2.3).

Proposição 2.1 *Se* $L_1 \leq_m^p L_2$ *e* $L_2 \in$ P *então* $L_1 \in$ P.

Proposição 2.2 *A relação* $\leq_m^p$ *é transitiva.*

Exercício 2.2.10 *Mostrar as Proposições 2.1 e 2.2.*

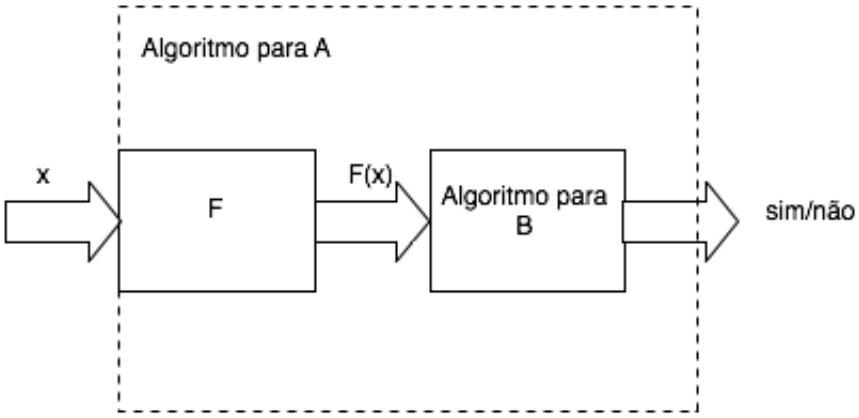


Figura 2.3: Redução de A a B.

Duas linguagens L_1 e L_2 são *polinomialmente equivalentes* se $L_1 \leq_m^p L_2$ e $L_2 \leq_m^p L_1$. Análogamente, se definem problemas *polinomialmente equivalentes*. A classe P é uma classe de equivalência dessa relação, formada pelas linguagens mais *fáceis*. Vejamos alguns exemplos de reduções polinomiais.

Exemplo 2.3 *HC*$\leq_m^p$ *TSP*

(HC)

Instância: *Um grafo não dirigido* $G = (V, E)$ *e* $|V| = m$

Questão: *G tem um ciclo hamiltoniano?*

(TSP)

Instância: *Um conjunto finito* $C = \{c_1, c_2, \ldots c_n\}$ *de cidades,* $d : C \times C \longrightarrow \mathbb{N}$ *distância e* $B \in \mathbb{N}$.

Questão: *Existe uma permutação* $\pi : C \longrightarrow C$ *tal que*

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}) \leq B$$

A transformação polinomial, f, é definida do seguinte modo: Seja $G = (V, E)$ *e* $|V| = m$ *uma instância de HC. A instância correspondente de TSP é dada por:*

a) $C = V$

b) Para cada par $v_i, v_j \in C$, $d(v_i, v_j) = 1$ *se* $(v_i, v_j) \in E$ *e* $d(v_i, v_j) = 2$, *caso contrário.*

c) $B = m$

Vamos ver duma forma informal que f é uma transformação polinomial de HC para TSP.

1. *Para construir as $m(m-1)/2$ distâncias $d(v_i, v_j)$, é necessário apenas verificar se $(v_i, v_j) \in E$. Isto pode ser feito em tempo polinomial.*

2. *Vejamos que G tem um ciclo hamiltoneano se e só se existe uma permutação π cuja soma das distâncias é menor que B. Suponhamos que $\langle v_1, \ldots, v_m \rangle$ é um ciclo Hamiltoneano de G. Então, $\langle v_1, \ldots, v_m \rangle$ é uma volta em $f(G)$ com distância total $m=B$. Inversamente, suponhamos que $\langle v_1, ..., v_m \rangle$ é uma volta em $f(G)$ com distância total $\leq B$. Pela construção de d, por $B=m$, e pelo facto de exactamente m cidades serem visitadas, a distância entre duas cidades consecutivas tem de ser 1. Então, todos os pares (v_i, v_{i+1}), $1 \leq i \leq m$ e (v_1, v_m) pertencem a E e constituem um ciclo hamiltoniano para G.*

Concluímos então que se existir um algoritmo polinomial para TSP, então também podemos construir um algoritmo determinístico polinomial para HC e se HC for intratável, TSP também o é.

Genericamente se $\Pi_1 \leq_m^p \Pi_2$ dizemos que Π_2 é *pelo menos tão difícil quanto* Π_1.

Consideremos agora uma redução entre dois problemas de dois domínios diferentes: lógica e teoria de grafos.

Exemplo 2.4 *3SAT* $\leq_m^p$ *VC*

(3SAT)

Instância: *Seja $C = \{c_1, c_2, \ldots, c_m\}$ um conjunto de cláusulas com literais num conjunto finito U de variáveis lógicas, $\mid U \mid = n$, e tal que cada cláusula tem exactamente 3 literais.*

Questão: *Existe uma atribuição de valores de verdade para U que satisfaz simultaneamente todas as cláusulas de C?*

(VC)

Instância: *Um grafo não dirigido $G = (V, E)$ e $K \leq |V|$ inteiro positivo.*

Questão: *Existe um subconjunto $V' \subseteq V$ tal que $|V'| \leq K$ e para cada ramo $\{u, v\} \in E$, $u \in V'$ ou $v \in V'$? (a V' chama-se* cobertura por vértices *de G)*

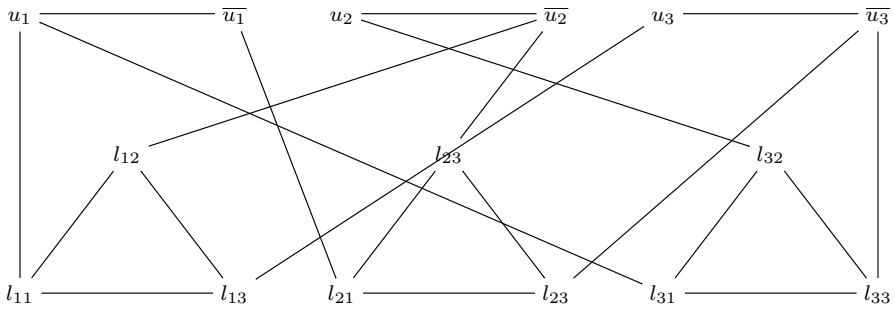
Construção da transformação : *Suponhamos uma instância de 3SAT. Vamos construir uma instância de VC indicando os conjuntos V e E. Para cada $u_i \in U$, $\{u_i, \overline{u_i}\} \subseteq V$ e $\{u_i, \overline{u_i}\} \in E$. Note-se que uma cobertura por vértices deve conter pelo menos um dos vértices destes ramos. Para cada $c_j \in C$ adicionar a V três vértices correspondentes a cada um dos literais – l_{j1}, l_{j2}, l_{j3} – que*

ocorre em c_j e adicionar a E os ramos que unem estes vértices, de modo a formar um triângulo para cada cláusula (ver figura). Qualquer cobertura por vértices dever conter pelo menos 2 dos vértices de cada triângulo. Finalmente, para cada cláusula constroem-se ramos que unem cada literal l_{jk}, $k = 1, 2, 3$, ao literal correspondente, u_i ou $\overline{u_i}$. Seja $K = n + 2m$ e $G = (V, E)$, onde

$$\begin{aligned} V &= U \cup \overline{U} \cup \bigcup_{i=1}^{m} \{l_{i1}, l_{i2}, l_{i3}\}, \\ E &= \bigcup_{i=1}^{n} \{\{u_i, \overline{u_i}\}\} \cup \bigcup_{i=1}^{m} \{\{l_{i1}, l_{i2}\}, \{l_{i1}, l_{i3}\}, \{l_{i2}, l_{i3}\}\} \cup \bigcup_{i=1}^{m} \bigcup_{j=1}^{3} \{\{l_{ij}, \mu\}\}, \end{aligned}$$

onde μ designa o literal correspondente u_k ou $\overline{u_k}$, para algum k, $1 \leq k \leq n$. Temos que $|V| = 2n + 3m$ e $|E| = n + 6m$. Toda esta construção pode ser feita em tempo polinomial.

Considerando a instância de 3SAT, $U = \{u_1, u_2, u_3\}$ e $C = \{(u_1, \overline{u_2}, u_3), (\overline{u_1}, \overline{u_2}, \overline{u_3}), (u_1, u_2, \overline{u_3})\}$ temos o seguinte grafo G: associado a VC, com $K = 11$ e $G = (V, E)$:



Falta ver que C é satisfeita se e só se G tem uma cobertura por vértices de tamanho no máximo K. Suponhamos que $V' \subseteq V$ é uma cobertura por vértices para G. Pelo que se disse acima V' contem exactamente um de cada $\{u_i, \overline{u_i}\}$ e 2 vértices l_{jk} correspondentes a cada cláusula. Então a seguinte atribuição de valores de verdade satisfaz C, $t : U \longrightarrow \{0, 1\}$:

$$t(u_i) = \begin{cases} 1 & \text{se } u_i \in V' \\ 0 & \text{se } \overline{u_i} \in V' \end{cases}$$

Exercício 2.2.11 Verificar que t satisfaz C.

Inversamente suponhamos que $t : U \longrightarrow \{0, 1\}$ é uma atribuição de valores de verdade que satisfaz C. A cobertura para V é a seguinte: $u_i \in V'$ se $t(u_i) = 1$ e $\overline{u_i} \in V'$ se $t(u_i) = 0$. Isto assegura que para cada cláusula uma dos ramos que unem l_{ji} aos literais u_i ou $\overline{u_i}$ fique coberta por V'. Basta então, adicionar a V' dois dos vértices da cada triângulo que não correspondem a esse literal.

Exercício 2.2.12 Verificar que V' é uma cobertura por vértices de G e $|V'| \leq K$.

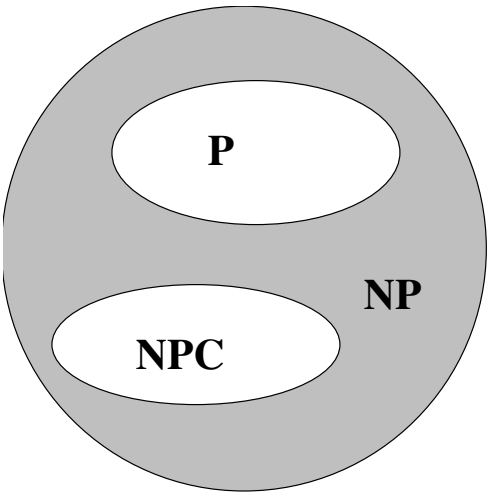


Figura 2.4: Classe NP, se P $\neq$ NP.

2.3 Problemas NP-completos

Definição 2.4 *Seja C uma classe de complexidade. Uma linguagem L é* C-hard *se $L' \leq_m^p L$, para toda $L' \in \mathsf{C}$. Se $L \in \mathsf{C}$ então L é completa para C (C-completa). Uma classe C é fechada para reduções se sempre que $L' \leq_m^p L$ e $L \in \mathsf{C}$, então $L' \in \mathsf{C}$.*

As classes P e NP são fechadas para reduções. Em particular, uma linguagem L é NP-completa se $L \in \mathsf{NP}$ e para todo $L' \in \mathsf{NP}$ tem-se que $L' \leq_m^p L$. Em termos de problemas, um problema Π diz-se NP-completo se $\Pi \in \mathsf{NP}$ e para todo $\Pi' \in \mathsf{NP}$ tem-se que $\Pi' \leq_m^p \Pi$.

Sela NPC a classe dos problemas NP-completos. Pela Proposição 2.1 os problemas NP-completos são os mais difíceis em NP. Se um problema em NP é intratável então todos os problemas em NPC também o são. Se um problema NP-completo tiver um algoritmo determinístico polinomial então todos os problemas em NP tem também um, isto é, P =NP! Por outro lado, se P $\neq$ NP e $\Pi \in$ NPC então $\Pi \in \mathsf{NP} \setminus \mathsf{P}$. Note-se ainda que todos os problemas na classe NPC são polinomialmente equivalentes. Se P $\neq$ NP então a classe NP divide-se como indicado na Figura 2.4.

No entanto, não parece claro como se pode mostrar que um problema é NP-completo: para cada problema em NP (que são em número infinito) é necessário arranjar uma redução polinomial para esse problema. A proposição seguinte mostra que se existir um problema NP-completo é mais fácil provar que outro problema é NP-completo.

Proposição 2.3 *Se L_1 e L_2 pertencem a* NP*, L_1 é* NP*-completa e $L_1 \leq_m^p L_2$ então L_2 é* NP*-completa.*

Exercício 2.3.1 Mostrar a Proposição 2.3.

Cook mostrou que SAT é NP-completo. Dado que para cada problema Π em NP existe um algoritmo polinomial não-determinístico A que o resolve, Cook [Coo71] provou que é possível obter

em tempo polinomial a partir desse algoritmo (genérico) e duma instância I do problema Π, um conjunto de cláusulas C, tal que C é satisfeito se e só se A termina com sucesso para essa instância I. Levin [Lev73] independentemente mostrou um teorema semelhante.

Teorema 2.4 *SAT é* NP*-completo.*

Dem. Temos que mostrar:

(i) $\mathsf{SAT} \in \mathsf{NP}$

(ii) $\forall \Pi \in \mathsf{NP} \implies \Pi \leq_m^p \mathsf{SAT}$

Seja $U = \{u_1, u_2, \ldots, u_n\}$ um conjunto de variáveis lógicas e $C = \{c_1, c_2, \ldots, c_m\}$ um conjunto de cláusulas com literais de U. Então, o seguinte algoritmo não-determinístico termina com sucesso se e só se C é satisfeito:

$i \leftarrow 1$
for $i \leq n$ **do**
  $u[i] \leftarrow$ ESCOLHA$((0,1))$
end for
$i \leftarrow 1$
for $i \leq m$ **do**
  if $c[i] = 0$ **then** INSUCESSO
  end if
end for
SUCESSO

O tempo de execução deste algoritmo é $O(n)$ para escolher uma atribuição de valores de verdade para as variáveis, mais $O(m)$ para verificar se cada cláusula de C é satisfeita, portanto é proporcional ao comprimento dos dados. Logo, $\mathsf{SAT} \in \mathsf{NP}$. Seja $L_{\mathsf{SAT}} = L[\mathsf{SAT}, c]$ a linguagem associada a SAT para uma dada codificação c. Temos que mostrar

$$\forall L \in \mathsf{NP},\ L \leq_m^p L_{SAT}.$$

Cada $L \in \mathsf{NP}$ é decidida por uma MTV polinomial (ver Definição 2.2). Seja, então, M uma MTV polinomial arbitrária, $M = (S, \Sigma, \Gamma, \Box, s_0, \{s_y, s_n\}, \delta)$ tal que $L(M) = L$ e seja $T_M(n) = p(n)$, com $p(n) \geq n,\ \forall\, n \in \mathbb{N}$. A transformação genérica F_L será definida de $\Sigma^\star$ em instâncias de SAT (mais propriamente Σ_{SAT}), tal que para $x \in \Sigma^\star$, $x \in L$ se e só se $F_L(x)$ é satisfazível. Isto é, $x \in \Sigma$ é aceite por M se e só se $F_L(x)$ é um conjunto de cláusulas satisfazível. A ideia é "simular" a computação de M em termos de variáveis lógicas. Se $x \in \Sigma^\star$ é aceite por M então existe uma computação que dado x termina no estado s_y e que tanto o número de passos na fase de verificação como o número de símbolos na sequência *adivinhada* são limitados por $p(n)$, sendo $n = |x|$. Cada computação deste tipo envolve no máximo as

posições de $-p(n)$ a $p(n)+1$. Recorde-se que cada configuração, ou descrição instantânea de M, fica caracterizada pelo conteúdo das células da fita, pelo estado corrente e a célula lida pela cabeça. Como não há mais de $p(n)$ passos na fase de verificação, existem no máximo $p(n)+1$ configurações a considerar. Podemos, então, associar a cada computação um número limitado de variáveis lógicas e uma atribuição de valores de verdade para elas. Numerem-se os elementos de S s_0, $s_1 = s_y$, $s_2 = s_n$, s_3, ..., s_r, onde $r = |S| - 1$, e os elementos de Γ, $a_0 = \square$, a_1, ..., a_v, onde $v = |\Gamma| - 1$. Consideram-se 3 tipos de variáveis:

Variável	Variação	*Significado*
$S[i,k]$	$0 \le i \le p(n)$ $0 \le k \le r$	no passo i M está no estado s_k
$H[i,j]$	$0 \le i \le p(n)$ $-p(n) \le j \le p(n)+1$	no passo i a cabeça está na célula j
$A[i,j,k]$	$0 \le i \le p(n)$ $-p(n) \le j \le p(n)+1$ $0 \le k \le v$	no passo i o conteúdo da célula j é o símbolo a_k

Uma computação de M induz uma atribuição de valores de verdade às variáveis acima definidas duma maneira óbvia, com a convenção de que se o programa terminar em $t \le p(n)$ passos, a configuração mantém-se a mesma de $t+1$ a $p(n)$. No passo 0 o conteúdo da fita tem os dados x, escritos nas células de 1 a n e a "advinha" w escrita nas células de -1 a $-|w|$, com as restantes células em branco[5]. A transformação F_L constrói uma conjunto de cláusulas envolvendo estas variáveis de tal modo que a atribuição de valores de verdade satisfaz esse conjunto sse é a atribuição de valores de verdade induzida por uma computação que aceita x cuja fase de verificação demora $p(n)$ ou menos passos e cuja sequência "adivinhada"tem comprimento no máximo $p(n)$. Tem-se

$x \in L$ $\Leftrightarrow$ existe uma computação de M que aceita x

$\Leftrightarrow$ existe uma computação de M que aceita x com $p(n)$ ou menos passos na fase de verificação e com uma sequência adivinhada w de comprimento exactamente $p(n)$

$\Leftrightarrow$ existe uma atribuição de valores de verdade que satisfaz o conjunto de cláusulas em $F_L(x)$

As cláusulas em $F_L(x)$ podem ser divididas em 6 grupos:

Grupo	Restrições impostas	Cláusulas nesse Grupo

[5]Note que uma qualquer atribuição de valores de verdade para estas variáveis não corresponde provavelmente a nenhuma computação de M.

G_1	No passo i, M está exactamente num só estado.	$\{S[i,0], S[i,1], \ldots, S[i,r]\}$ $0 \leq i \leq p(n)$ $\{\overline{S[i,j]}, \overline{S[i,j']}\}$ $0 \leq i \leq p(n)$ $0 \leq j < j' \leq r$
G_2	No passo i a cabeça está exactamente numa só célula da fita.	$\{H[i,-p(n)], H[i,-p(n)+1], \ldots, H[i,p(n)+1]\}$ $0 \leq i \leq p(n)$ $\{\overline{H[i,j]}, \overline{H[i,j']}\}$ $0 \leq i \leq p(n)$ $-p(n) \leq j < j' \leq p(n)+1$
G_3	No passo i, cada célula contém exactamente um símbolo de Γ.	$\{A[i,j,0], A[i,j,1], \ldots, A[i,j,v]\}$ $0 \leq i \leq p(n)$ $-p(n) \leq j \leq p(n)+1$ $\{\overline{A[i,j,k]}, \overline{A[i,j,k']}\}$ $0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)+1$ $0 \leq k < k' \leq v$
G_4	No passo 0, a computação está na configuração inicial para a fase de verificação para dados $x = a_{k_1} \cdots a_{k_n}$.	$\{S[0,0]\}, \{H[0,1]\}, \{A[0,0,0]\},$ $\{A[0,1,k_1]\}, \ldots, \{A[0,n,k_n]\},$ $\{A[0,n+1,0]\}, \ldots, \{A[0,p(n)+1,0]\}$
G_5	No passo $p(n)$, M entrou no estado s_y.	$\{S[p(n),1]\}$
G_6	Para cada i, $0 \leq i < p(n)$, a configuração de M em $i+1$ resulta duma única aplicação de δ dada a configuração em i.	(Ver texto)

É fácil ver que as cláusulas dos grupos G_1 a G_5 cumprem a função especificada para cada grupo e que uma atribuição de valores de verdade para elas corresponde a uma computação que aceita x. Apenas falta especificar as cláusulas do grupo G_6. Consideramos dois subgrupos.

1. Este subgrupo garante que se a cabeça não está no passo i na célula j então o seu símbolo

não é mudado de i para $i+1$. As cláusulas são da forma[6]:

$$\{H[i,j], \overline{A[i,j,l]}, A[i+1,j,l]\} \quad \begin{array}{l} 0 \leq i < p(n) \\ -p(n) \leq j \leq p(n)+1 \\ 0 \leq l \leq v \end{array}$$

2. Este subgrupo garante que as modificações duma configuração para a seguinte estão de acordo com a função de transição δ. Para cada (i,j,k,l), $0 \leq i < p(n)$, $-p(n) \leq j \leq p(n)+1$, $0 \leq k \leq r$ e $0 \leq l \leq v$ tem-se as seguintes 3 cláusulas[7] :

$$\{\overline{H[i,j]}, \overline{S[i,k]}, \overline{A[i,j,l]}, H[i+1, j+D]\}$$
$$\{\overline{H[i,j]}, \overline{S[i,k]}, \overline{A[i,j,l]}, S[i+1,k']\}$$
$$\{\overline{H[i,j]}, \overline{S[i,k]}, \overline{A[i,j,l]}, S[i,j,l']\}$$

onde se $s_k \in S \setminus \{s_y, s_n\}$ então os valores de D, k' e l' são tais que $\delta(s_k, a_l) = (s_{k'}, a_{l'}, D)$ e se $s_k \in \{s_y, s_n\}$ então $D = 0$, $k' = k$ e $l' = l$.

O número de cláusulas por grupo é:

Grupo	Número de cláusulas	
G_1	$(p(n)+1)(1+r(r+1)/2)$	
G_2	$(p(n)+1)(1+u(u+1)/2)$	onde $u = 2(p(n)+1)$
G_3	$2(p(n)+1)^2(1+v(v+1)/2)$	
G_4	$p(n)+4$	
G_5	1	
G_6	$2(p(n)+1)^2(v+1) + 6(p(n))(p(n)+1)(r+1)(v+1)$	

Se $x \in L$ então existe uma computação de M com dados x de tamanho $p(n)$ ou menos, e esta computação, dada a interpretação das variáveis, impõe uma atribuição de valores de verdade que satisfaz todas as cláusulas de $C = G_1 \cup G_2 \cup G_3 \cup G_4 \cup G_5 \cup G_6$.

Inversamente, a construção de C é tal que qualquer atribuição de valores de verdade que satisfaça C corresponde a uma computação de M que aceita x. Segue que $F_L(x)$ é satisfazível se e só se $x \in L$.

Apenas resta ver que, para uma linguagem L, a transformação $F_L(x)$ pode ser construída a partir de x em tempo limitado por uma função polinomial em $n = |x|$. Dado L e M uma MTN que decide L em tempo $p(n)$ tem de se construir o conjunto de variáveis U e o conjunto de cláusulas C. Para ver que a construção da transformação é limitada polinomialmente, basta ver que $comp[F_L(x)]$ é limitado superiormente por um polinomial em n. Para este problema

[6]Isto é, $\neg H[i,j] \rightarrow (A[i,j,l] \rightarrow A[i+1,j,l])$.

[7]Isto é $S[i,k] \ \wedge \ H[i,j] \wedge A[i,j,l] \rightarrow (H[i+1,j+D] \wedge S[i+1,k'] \ \wedge \ A[i+1,j,l'])$

(SAT), dada uma codificação razoável, $comp[F_L(x)] = |U|.|C|$. Note-se que cada cláusula não pode conter mais de $2|U|$ literais e o número de símbolos necessários para descrever um literal é um factor da ordem $\log|U|$. Dado que r e v são fixos e não dependem de x, tem-se $|U| = O(p(n)^2)$ e $|C| = O(p(n)^2)$. Então, $comp[F_L(x)] = O(p(n)^4)$. □

Assim, usando a Proposição 2.3, para mostrar que um problema Π é NP-completo basta:

a) Mostrar que $\Pi \in$ NP

b) Selecionar um problema $\Pi' \in$ NPC

c) Construir uma transformação F de Π' em Π

d) Mostrar que F é uma redução polinomial

Proposição 2.4 *3SAT é NP-completo.*

Dem. 1. 3SAT $\in$ NP Seja $U = \{u_1, u_2, \ldots, u_n\}$ um conjunto de variáveis lógicas e $C = \{c_1, c_2, \ldots, c_m\}$ um conjunto de cláusulas com 3 literais de U. Então, o seguinte algoritmo não-determinístico termina com sucesso se e só se C é satisfeito:

```
i ← 1
for i ≤ n do
    u[i] ← ESCOLHA((0, 1))
end for
i ← 1
for i ≤ m do
    if c[i] = 0 then INSUCESSO
    end if
end for
SUCESSO
```

O tempo de execução deste algoritmo é $O(n)$ para escolher uma atribuição de valores de verdade para as variáveis, mais $O(m)$ para verificar se cada cláusula de C é satisfeita, portanto é proporcional ao comprimento dos dados. Logo, 3SAT $\in$ NP.

2. Seleccionámos SAT (é o único problema NP-completo que conhecemos.)

3. Vamos transformar SAT em 3SAT. Seja $U = \{u_1, u_2, \ldots, u_n\}$ um conjunto de variáveis lógicas e $C = \{c_1, c_2, \ldots, c_m\}$ um conjunto de cláusulas duma instância de SAT. Vamos construir um conjunto de cláusulas C' com 3 literais num conjunto U' de variáveis tal que C' é satisfeito se e só se C o é. Cada cláusula $c_i \in C$ é transformada num conjunto C'_i de cláusulas com 3 literais do conjunto U mais variáveis adicionais (usadas só para as cláusulas de C'_i) dum conjunto U'_i. A estrutura de C'_i e U'_i dependem do número de literais k, em c_i, de acordo com a seguinte tabela:

k	c_i	U'_i	C'_i
1	(l_1)	$\{u_i^1, u_i^2\}$	$\{(l_1, \overline{u_i^1}, \overline{u_i^2}), (l_1, u_i^1, \overline{u_i^2}), (l_1, \overline{u_i^1}, u_i^2), (l_1, u_1^1, u_i^2)\}$
2	(l_1, l_2)	$\{u_i^1\}$	$\{(l_1, l_2, u_i^1), (l_1, l_2, \overline{u_i^1})\}$
3	(l_1, l_2, l_3)	$\emptyset$	$\{(l_1, l_2, l_3)\}$
≥ 4	$(l_1, l_2, \ldots, l_k)$	$\{u_i^1, \ldots, u_i^{k-3}\}$	$\{(l_1, l_2, \overline{u_i^1}), (u_i^1, l_3, \overline{u_i^2}), (u_i^2, l_4, \overline{u_i^3}),$ $\ldots, (u_i^{k-4}, l_{k-2}, \overline{u_i^{k-3}}), (u_i^{k-3}, l_{k-1}, l_k)\}$

e

$$U' = U \cup \bigcup_{i=1}^{m} U'_i$$
$$C' = \bigcup_{i=1}^{m} C'_i$$

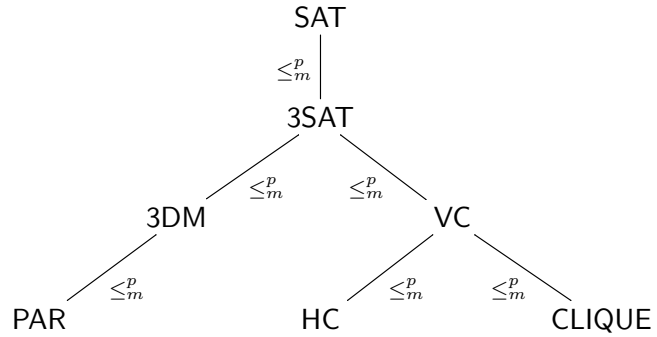
4. Vejamos que C' é satisfeita se e só se C o é. Suponhamos $t : U \longrightarrow \{0,1\}$ uma atribuição de valores de verdade que satisfaz C. Vejamos como t pode ser estendida a uma atribuição de valores de verdade para C', $t' : U' \longrightarrow \{0,1\}$. Para isso basta ver quais os valores de t' para cada elemento de U'_i (e para cada caso verificar que as cláusulas em C' são todas satisfeitas). Dizemos que o conjunto U' é do tipo k se corresponder a uma cláusula em C com k literais, de acordo com a tabela anterior. Se U'_i for do tipo 1 ou 2 qualquer valor pode ser dado aos seus elementos, por exemplo $t'(u_i^j) = 1$, $u_i^j \in U_i$. Se U' for do tipo 3 é vazio e portanto não é preciso fazer nada. Se U'_i for do tipo k, $k \geq 4$, então pelo menos um dos literais de c_i tem $t(l_m) = 1$, para algum $1 \leq m \leq k$. Se for l_1 ou l_2 então fazemos $t'(u_i^j) = 1$ para $1 \leq j \leq k-3$. Se for l_k ou l_{k-1} fazemos $t'(u_i^j) = 0$ para $1 \leq j \leq k-3$. Caso contrário, $t'(u_i^j) = 0$ para $1 \leq j \leq m-2$ e $t'(u_i^j) = 1$ para $m-1 \leq j \leq k-3$.

   Inversamente, se t' é uma atribuição de valores de verdade para C' é fácil ver que a restrição de t' a U deve satisfazer C.

5. Note-se que o número máximo de cláusulas em C' é limitado por um polinómio em mn e portanto o tamanho duma instância de 3SAT é limitado por um polinómio no tamanho da instância de SAT. Isto é, a construção da transformação pode ser feita em tempo polinomial. Donde $\mathsf{SAT} \leq_m^p \mathsf{3SAT}$, e portanto 3SAT é NP-completo.

□

Em 1972 Karp [Kar72] publica uma lista com 21 problemas que mostra serem NP-completos, indicando as reduções entre eles. Dessa lista salientámos 6 (consultar [GJ79, Cap 3]), que já foram referidos anteriormente, e o diagrama seguinte mostra uma sequência de reduções que pode ser usada para a sua demonstração:



Pelo diagrama e pela redução atrás apresentada podemos concluir que o problema de decisão do *caixeiro viajante* é NP-completo: HC é NP-completo e HC $\leq_m^p$ TSP.

Proposição 2.5 *HC é* NP*-completo.*

Dem.

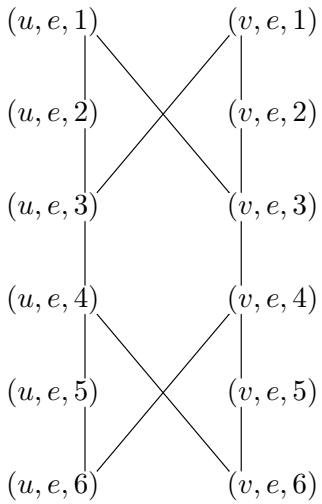
a) HC $\in$ NP porque um algoritmo não determinístico apenas tem de escolher uma ordenação dos vértices e verificar em tempo polinomial que o conjunto de ramos correspondentes está contido em E. Podemos considerar o seguinte algoritmo:

$i \leftarrow 1$
$S = V$
while $S \neq \emptyset$ **do**
  $v_i \leftarrow$ESCOLHA(S)
  $i \leftarrow i + 1$
  $S \leftarrow S \setminus \{v_i\}$
end while
if $\{v_1, v_m\} \notin E$ **then** INSUCESSO
end if
$i \leftarrow 1$
for $i < m$ **do**
  if $\{v_i, v_{i+1}\} \notin E$ **then** INSUCESSO
  end if
  $i \leftarrow i + 1$
end for
SUCESSO

b) Seleccionámos VC

c) Vamos transformar VC em HC. Seja $G = (V, E)$ e $K \leq |V|$ uma instância de VC. Temos de construir um grafo $G' = (V', E')$ tal que G' tem um ciclo Hamiltoneano se e só se G tem uma cobertura por vértices de tamanho no máximo K. Primeiro, o grafo G'

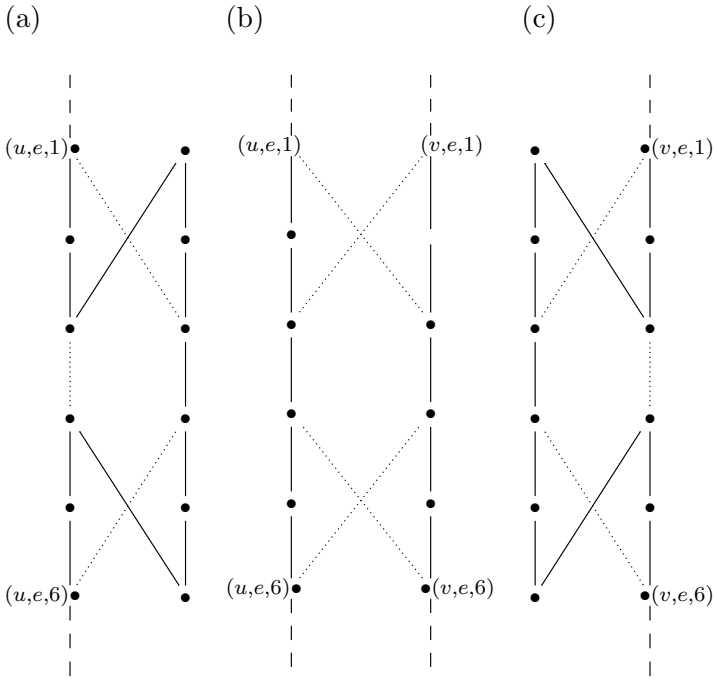
tem K vértices selectores $a_1, a_2, \ldots, a_K$, que serão usados para seleccionar K vértices do conjunto V. Depois, para cada ramo em E, G' contém uma componente de "teste de cobertura"que será usada para assegurar que pelo menos uma das extremidades desse ramo é um dos K vértices seleccionado. A componente para um ramo $e = \{u, v\} \in E$ é:



com

$$
\begin{aligned}
V'_e &= \{(u,e,i),(v,e,i) : 1 \leq i \leq 6\} \\
E'_e &= \{(u,e,i),(u,e,i+1)\},\{(v,e,i),(v,e,i)\} \mid 1 \leq i \leq 5\} \\
&\quad \cup\{\{(u,e,3),(v,e,1)\},\{(v,e,3),(u,e,1)\}\} \\
&\quad \cup\{\{(u,e,6),(v,e,4)\},\{(v,e,6),(u,e,4)\}\}
\end{aligned}
$$

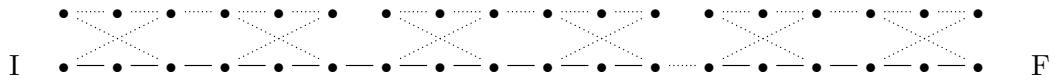
Cada V'_e tem 12 vértices e cada E'_e tem 14 ramos. Na construção final, os únicos vértices desta componente que entrarão noutros ramos são os vértices $(u, e, 1)$,$(v, e, 1)$, $(u, e, 6)$ e $(v, e, 6)$. Isto implica que qualquer ciclo Hamiltoneano tem de passar pelos ramos de E'_e de uma das seguintes maneiras:



Ramos adicionais serão usados na construção para juntar pares de componentes de "teste de cobertura" ou para juntar componentes a um vértice seleccionador. Para cada $v \in V$, ordenem-se os ramos incidentes $e_{v[1]}$, $e_{v[2]}$, ..., $e_{v[gr(v)]}$, onde $gr(v)$ é o grau de v, isto é, o número de ramos que incidem em v. Todas as componentes de "teste de cobertura" correspondentes a estes ramos (tendo v como extremidade) são unidas pelos seguintes ramos:

$$E'_v \quad = \quad \{\{(v, e_{v[i]}, 6), (v, e_{v[i+1]}, 1)\} \mid \ 1 \leq i \leq gr(v)\}$$

Cria-se um único caminho em G' que incluí exactamente os vértices da forma (v, y, z) tal que



onde $I = (v, e_{v[1]}, 1)$, $F = (v, e_{v[gr(v)]}, 6)$ e cada vértice dessa linha é da forma $(v, e_{v[i]}, j)$ para $1 \leq j \leq 6$ e $1 \leq i \leq gr(v)$. Finalmente, adicionam-se ramos para unir o primeiro e o último vértice de cada um destes caminhos a todos os seleccionadores a_1, a_2, ..., a_K. Estes ramos são:

$$E'' \quad = \quad \{\{a_i, (v, e_{v[1]}, 1)\}, \{a_i, (v, e_{v[gr(v)]}, 6)\} \mid 1 \leq i \leq K, \ v \in V\}$$

O grafo $G' = (V', E')$ é então definido por:

$$\begin{aligned} V' &= \{a_i \mid 1 \leq i \leq K\} \cup (\bigcup_{e \in E} V'_e)\}) \\ E' &= (\bigcup_{e \in E} E'_e) \cup (\bigcup_{v \in V} E'_v) \cup E'' \end{aligned}$$

d) É fácil verificar que G' pode ser construído em tempo polinomial. Vejamos que G' tem um ciclo Hamiltoneano se e só se G tem uma cobertura por vértices de tamanho no máximo K. Suponhamos que $\langle v_1, v_2, \ldots, v_n \rangle$ onde $n = |V'|$ é um ciclo Hamiltoneano. Considere-se qualquer porção do ciclo que comece num vértice de $\{a_1, a_2, \ldots, a_K\}$, termine num vértice desse conjunto e que não passe por nenhum outro elemento desse conjunto. Dadas as restrições da forma como um ciclo Hamiltoneano pode passar por uma componente de "teste de cobertura", esta porção do ciclo deve passar por um conjunto dessas componentes correspondentes exactamente aos ramos de E que incidem num vértice particular $v \in V$. Cada componente é atravessada por um dos modos referidos e nenhum vértice de outra componente é encontrado. Então os K vértices de $\{a_1, a_2, \ldots, a_K\}$ dividem o ciclo Hamiltoneano em caminhos, cada um correspondente a um vértice distinto $v \in V$. Dado que o ciclo Hamiltoneano deve incluir todos os vértices de todas as componentes de *teste de cobertura*, e como os vértices duma componente para $e \in E$ só podem ser atravessados por um caminho correspondente a uma extremidade de e, todos os ramos de E têm de ter uma extremidade nesse conjunto de K vértices seleccionados. Então, esse conjunto forma uma cobertura por vértices de tamanho K para G.

Inversamente, suponhamos $V^\star \subseteq V$ uma cobertura por vértices de G e $|V^\star| \leq K$. Podemos supor que $|V^\star| = K$, pois podem sempre adicionar-se vértices. Sejam $v_1, \ldots, v_K$ os elementos de $V^\star$. Escolhemos os ramos no ciclo Hamiltoneano de G' da seguinte forma. Da componente de "teste de cobertura" representante de cada $e = \{u, v\} \in E$, escolher os ramos correspondentes a uma das três maneiras, (a), (b) ou (c), como podem ser percorridas, consoante $\{u, v\} \cap V^\star$ é igual a $\{u\}$, $\{u, v\}$ ou $\{v\}$. Note-se que uma dessas condições tem de se verificar porque $V^\star$ é uma cobertura de G. Em seguida escolhe-se todos os ramos em E'_{v_i} para $1 \leq i \leq K$. Finalmente, os ramos:

$$\begin{aligned} \{a_i, (v_i, e_{v_i[1]}, 1)\}, &\qquad 1 \leq i \leq K \\ \{a_{i+1}, (v_i, e_{v_i[gr(v_i)]}, 6)\}, &\qquad 1 \leq i < K \\ \{a_1, (v_K, e_{v_K[gr(v_K)]}, 6)\}. & \end{aligned}$$

Deixa-se ao leitor a verificação de que este conjunto de ramos corresponde a um ciclo Hamiltoneano.

□

Proposição 2.6 *3DM é* NP*-completo.*

Dem. Este problema é uma variante a três dimensões do problema dos casamentos estáveis (bi-dimensional) que pode ser resolvido em tempo polinomial. O problema de decisão é:

3DM

Instância: Seja M um conjunto $M \subseteq W \times X \times Y$, onde $|W| = |X| = |Y| = q$ e disjuntos

Questão: Existe um subconjunto $M' \subseteq M$, tal que $|M'| = q$ e nenhum elemento de M' tem uma "coordenada"comum (um *matching*/emparelhamento 3 dimensional) ?

a) 3DM $\in$ NP, dado que basta escolher um conjunto $M' \subseteq M$ com $|M'| = q$ e depois verificar se respeita as condições. Isto pode ser feito em tempo polinomial.

b) Vamos ver que 3SAT $\leq_m^p$ 3DM. Seja uma instância de 3SAT com $C = \{c_1, \ldots, c_m\}$, $U = \{u_1, \ldots, u_n\}$ e cada $c_j = (l_{1j}, l_{2j}, l_{3j})$ com l_{ij} um u_k ou $\overline{u_k}$. É necessário construir W, X, Y com $|W| = |X| = |Y|$ e $M \subseteq W \times Y \times Z$ tal que M tem um emparelhamento se e só se C é satisfazível. O conjunto de triplos M é constituído por três grupos. O primeiro estabelece uma valorização para U, T_i, $1 \leq i \leq n$. Para cada $u_i \in U$ consideramos elementos $u_i[j], \overline{u_i}[j] \in W$ que também vão ser considerados noutros triplos e elementos internos $a_i[j] \in X$ e $b_i[j] \in Y$, para $1 \leq j \leq m$. Temos

$$\begin{aligned} T_i^t &= \{(\overline{u_i}[j], a_i[j], b_i[j]) \mid 1 \leq j \leq m\} \\ T_i^f &= \{(u_i[j], a_i[j+1], b_i[j]) \mid 1 \leq j \leq m\} \cup \{(u_i[m], a_i[1], b_i[m])\} \\ T_i &= T_i^t \cup T_i^f \end{aligned}$$

Um emparelhamento M' tem de conter exactamente m triplos de T_i: ou todos de T_i^t ou todos de T_i^f. Isto define uma valorização para U tal que $t(u_i) = 1$ sse $M' \cap T_i = T_i^t$. O segundo grupo testa a satisfabilidade de cada cláusula $c_j \in C$. Tem elementos internos $s_1[j] \in X, s_2[j] \in Y$ e elementos de $\{[u_i[j], \overline{u_i}[j] \mid 1 \leq i \leq n\}$, que determinam os literais em c_j, $1 \leq j \leq m$.

$$C_j = \{(u_i[j], s_1[j], s_2[j]) \mid u_i \in c_j\} \cup \{(\overline{u_i}[j], s_1[j], s_2[j]) \mid \overline{u_i} \in c_j\}$$

Qualquer emparelhamento $M' \subseteq M$ contém exactamente um triplo de C_j. Isto pode ser feito se para $u_i \in c_j$ ($\overline{u_i} \in c_j$) $u_i[j] \notin M' \cap T_i$ ($\overline{u_i}[j] \notin M' \cap T_i$), isto é sse a valorização determinada por M' satisfaz c_j. O restante grupo corresponde os u_i ainda não utilizados. O grupo G contem elementos internos $g_1[k] \in X, g_2[k] \in Y$, para $1 \leq k \leq m(n-1)$ e elementos $u_i[j], \overline{u_i}[j] \in W$ ainda não usados.

$$G = \{(u_i[j], g_1[k], g_2[k]) \mid 1 \leq k \leq m(n-1), 1 \leq i \leq n, 1 \leq j \leq m\}$$

Cada par $g_1[k], g_2[k]$ será emparelhado com um $u_i[j]$ ou $\overline{u_i}[j]$ que não ocorre em $M' \setminus G$. Existem $m(n-1)$ elementos destes. Sempre que $M \setminus G$, satisfaz as condições duma valorização de C, G apenas garante a extensão desse subconjunto a um emparelhamento de M. Temos em resumo:

$$\begin{aligned} W &= \{u_i[j], \overline{u_i}[j] \mid 1 \leq i \leq n, 1 \leq j \leq m\} \\ X &= \{a_i[j] \mid 1 \leq i \leq n, 1 \leq j \leq m\} \cup \{s_1[j] \mid 1 \leq j \leq m\} \\ &\quad \cup \{g_1[j] \mid 1 \leq j \leq m(n-1)\} \\ Y &= \{b_i[j] \mid 1 \leq i \leq n, 1 \leq j \leq m\} \cup \{s_2[j] \mid 1 \leq j \leq m\} \\ &\quad \cup \{g_2[j] \mid 1 \leq j \leq m(n-1)\} \\ M &= \bigcup_{i=1}^{n} T_i \bigcup \bigcup_{j=1}^{m} C_j \bigcup G \end{aligned}$$

e $|M| = 2mn + 3m + 2m^2n(n-1)$.

c) É fácil verificar que M pode ser construído em tempo polinomial. Vejamos que C é satisfazível se e só se M tem um emparelhamento. Por construção, se M tem um emparelhamento, C é satisfazível. Seja $t : U \longrightarrow \{0,1\}$ uma valorização que satisfaz C. Um emparelhamento M' para M é construído do seguinte modo: para cada $c_j \in C$ seja $z_j \in \{u_i, \overline{u_i} \mid 1 \leq i \leq n\} \cap c_j$ tal que $t(z_j) = 1$. Então,

$$M' = \left(\bigcup_{t(u_i)=1} T_i^t \right) \bigcup \left(\bigcup_{t(u_i)=0} T_i^f \right) \bigcup \left(\bigcup_{j=1}^{m} \{(z_j[j], s_1[j], s_2[j]) \mid 1 \leq j \leq m\} \right) \bigcup G'$$

onde G' tem os restantes $u_i[j]$ e $\overline{u_i}[j]$ e todos os $g_1[k], g_2[k]$.

□

Proposição 2.7 *PAR é* NP*-completo.*

Dem. Seja o problema de decisão PAR

Instância: Dados um conjunto A e uma função $s : A \longrightarrow \mathbb{N}$.

Questão: Existe $A' \subseteq A$ tal que

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a).$$

a) PAR está em NP dado que basta escolher A' e verificar se a igualdade é satisfeita.

b) Selecionamos 3DM

c) Vamos mostrar que 3DM $\leq_m^p$ PAR. Sejam $W = \{w_1, \ldots, w_q\}$, $X = \{x_1, \ldots, x_q\}$ e $Y = \{y_1, \ldots, y_q\}$ e $M = \{m_1, \ldots, m_k\} \subseteq W \times X \times Y$ uma instância de 3DM. Temos de construir um conjunto A e uma função s tal que A contém um subconjunto A' com $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$ se e só se M tem um emparelhamento M'. Tomámos $|A| = k+2$ e $\{a_i \mid 1 \leq i \leq k\} \subseteq A$ com cada a_i associado a m_i. Para $s(a_i)$ consideramos a sua representação em binário $3qp$ bits onde $p = \lceil \log_2 (k+1) \rceil$. Os primeiros qp bits correspondem a Y $(y_1, \ldots, y_q)$ (sendo o primeiro bloco de p bits para y_q), os seguintes (de $pq+1$ a $2pq$) a X $(x_1, \ldots, x_q)$ e os mais significativos a W $(w_1, \ldots w_q)$, de $2qp+1$ a $3qp$. Cada grupo de p bits chamámos *zona* e corresponde a um dos elementos de um dos conjuntos W, X ou Y. Se $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)})$ então $s(a_i)$ em binário tem um 1 nas posições mais à direita das zonas $w_{f(i)}$, $x_{g(i)}$ e $y_{h(i)}$ e 0 em todas as restantes. Significa que,

$$s(a_i) = 2^{p(3q-f(i))} + 2^{p(2q-g(i))} + 2^{p(q-h(i))}.$$

A função s pode ser construída em tempo polinomial. Somando todos os possíveis valores numa zona obtemos no máximo $2^p - 1 = k$. Isto garante que para qualquer $A' \subseteq A$, em $\sum_{a \in A'} s(a)$ (em binário) não haverá transporte duma zona para outra. Seja

$$B = \sum_{j=0}^{3q-1} 2^{pj},$$

cuja representação em binário tem um 1 no bit mais à direita de cada zona. Então qualquer $A' \in \{a_i \mid 1 \leq i \leq k\}$ tem

$$\sum_{a \in A'} s(a) = B$$

se e só se M tiver um emparelhamento $M' = \{m_i \mid a_i \in A'\}$. São necessários ainda mais dois elementos b_1, b_2 em A, tal que $s(b_1) = 2\sum_{i=0}^{k} s(a_i) - B$ e $s(b_2) = \sum_{i=0}^{k} s(a_i) + B$. Este valores necessitam no máximo de $3pq+1$ bits pelo que também podem ser calculados em tempo polinomial.

d) Vamos ver que a resposta é sim para esta instância calculada de PAR se e só se a instância de 3DM tiver um emparelhamento. Para que exista $A' \subseteq A$ que verifique a igualdade é necessário que o valor das somas seja $2\sum_{i=0}^{k} s(a_i)$ e b_1 e b_2 não estejam os dois nem em A' nem em $A \setminus A'$. No subconjunto que contiver b_1 a soma correspondente os restantes elementos (todos a_i) tem de ser B, e como vimos acima então M tem um emparelhamento. Inversamente, se $M' \subseteq M$ é um emparelhamento então o conjunto $A' = \{b_1\} \cup \{a_i \mid m_i \in M'\}$ satisfaz PAR.

□

Exercício 2.3.2 Mostrar que os seguintes problemas são NP-completos:

1. CLIQUE

   Sugestão: VC $\leq_m^p$ CLIQUE, sabendo que se $G = (V, E)$ é um grafo e $V' \subseteq V$ então V' é uma cobertura por vértices de G sse $V \setminus V'$ é um clique no grafo complementar de G, G^c onde $G^c = (V, E^c)$ e $E^c = \{(u, v) : u, v \in V \text{ e } (u, v) \notin E\}$.

2. Conjunto Independente

   Instância: Dado um grafo $G = (V, E)$ e $K \leq |V|$.

   Questão: Existe com $V' \subseteq V$, um conjunto independente. i.e. $\forall u, v \in V', \{u, v\} \notin V'$ e $|V'| \geq K$. Sugestão: Se V' é uma cobertura por vértices então $V \setminus V'$ é um conjunto independente.

3. 4SAT

   Sugestão: 3SAT $\leq_m^p$ 4SAT

4. 3-COLOR

   Sugestão: 3SAT $\leq_m^p$ 3COLOR. Idea: Suponhamos uma instância de 3SAT com $U = \{u_1, u_2, \ldots, u_n\}$ um conjunto de variáveis lógicas e $C = \{c_1, c_2, \ldots, c_m\}$ um conjunto de cláusulas com 3 literais de U. Constrói-se o seguinte grafo: sela T um triângulo e associa-se a cada vértice uma das cores possíveis $\{0, 1, 2\}$. Para cada variável u_i constrói-se o ramo $\{u_i, \overline{u_i}\}$ e liga-se cada um desses vértices ao vértice 2 do triângulo T (assim u_i terá de ser colorido com uma cor diferente da de $\overline{u_i}$, correspondendo a uma atribuição de valores de verdade para U). Para cada uma das m cláusulas com 3 literais X_j, Y_j, Z_j constrói-se um grafo C_j correspondendo à fórmula $F = (X_j = 1 \vee Y_j = 1 \vee Z_j = 1)$ e liga-se ao vértice 1 de T. Liga-se finalmente cada X_j, Y_j, Z_j ao vértice u_i ou $\overline{u_i}$ correspondente. Seja G o grafo assim obtido. Verifica que se pode colorir G com 3 cores sse C é satisfeito. Desenha o diagrama de G e começa por ver que C_j é 3-colorável sse F é satisfeita. Finalmente verifica que a transformação pode ser feita em tempo polinomial.

Uma maneira simples de mostrar que um problema Π é NP-completo é obter uma restrição de Π que corresponda a um problema NP-completo, isto é, mostrando que Π contém como caso especial um problema NP-completo.

Exemplo 2.5 *SUBGRAFO ISOMORFO*

Instância: *Dois grafos não dirigidos $G = (V, E)$ e $G' = (V', E')$.*

Questão: *G contem um subgrafo isomorfo a G', isto é,*

$$\exists V_1 \subseteq V, \exists E_1 \subseteq E \exists f : V' \longrightarrow V_1 \text{ bijecção, tal que } \forall (a, b) \in E', (f(a), f(b)) \in E_1$$

Se G' for um grafo completo, este problema tem como restrição o problema do CLIQUE. Logo é NP*-completo.*

Exercício 2.3.3 Mostrar por restrição que os seguintes problemas são NP-completos:

1. KNAPSACK (Problema do "saco de viagem")

   Instância: Um conjunto finito U, $s : U \longrightarrow \mathbb{N}$ (tamanho) e $v : U \longrightarrow \mathbb{N}$ (valor), $B \in \mathbb{N}$ (tamanho máximo) e $K \in \mathbb{N}$ (valor objectivo).

   Questão: Existe um subconjunto $U' \subset U$ tal que $\sum_{u \in U'} s(u) \leq B$ e $\sum_{u \in U'} v(u) \geq K$?

   Sugestão: Restrição a PAR com $\forall u \in U, s(u) = v(u)$ e $B = K = 1/2 \sum_{u \in U} s(u)$.

2. Caminho mais longo

   Instância: Um grafo $G = (V, E)$ e um inteiro positivo $K \leq |V|$.

   Questão: G tem um caminho de tamanho maior ou igual a K que não repete nenhum vértice?

   Sugestão: Restringir a HC com $K = |V|$.

3. (X3C) Cobertura exacta

   Instância: Um conjunto X com $|X| = 3q$ e uma colecção C de subconjuntos de X com 3 elementos.

   Questão: É verdade que C contém uma cobertura exacta de X, isto é, uma sub- colecção $C' \subseteq C$ tal que cada elemento de X ocorre num e num só elemento de C'?

   Sugestão: Restringir a 3DM considerando M um conjunto desordenado $W \cup X \cup Z$.

4. Conjuntos Disjuntos

   Instância: Dada a colecção C de conjuntos finitos e um inteiro positivo, $K \leq |C|$.

   Questão: C contém K conjuntos disjuntos? Sugestão: Restringir a X3C

Outras técnicas são (mais complicadas)

- Substituição local: como feito para 3SAT, cada componente duma instÂncia de um problema é modificada para a ncia do outro.
- Desenho de componentes: como feito para VC, HC e 3DM.

A classe dos problemas NP-completos inclui muitos problemas para os quais se fizeram muitos esforços para encontrar algoritmos polinomiais. O facto de que se um deles tiver um algoritmo polinomial todos os outros têm, torna-os os "mais difíceis"da classe NP e reforça a conjectura de que eles não pertencem classe P e portanto $\mathsf{P} \neq \mathsf{NP}$. Assim se se mostra que um novo problema é NP-completo não vale muito a pena procurar um algoritmo polinomial para ele mas sim tentar resolvê-lo por outros processos!

Ao tentar provar que um dado problema é NP-completo duas coisas podem acontecer:

1. não se conseguir provar que pertence a NP (um caso trivial é se o problema não é de decisão).

2. não se conseguir provar que é completo

No primeiro caso podemos ainda conseguir provar que é completo, mostrando que esse problema se pode transformar polinomialmente num problema NP-completo - e sendo assim só ser resolvido em tempo polinomial se $\mathsf{P} = \mathsf{NP}$. Neste caso o problema diz-se NP-hard (é pelo menos tão difícil como os NP-completos). No segundo caso se se mostrar que é NP (mas não P) então é um candidato da classe $\mathsf{NP}-\mathsf{P}-\mathsf{NPC}$ que chamaremos NPI – problemas de dificuldade "intermédia". Teoricamente se $\mathsf{P} \neq \mathsf{NP}$ mostra-se que esta classe é não vazia e que existe uma infinidade de classes de equivalência entre P e NP.

Teorema 2.5 ([Lad75]) *Seja B uma linguagem recursiva tal que $B \notin \mathsf{P}$. Então existe uma linguagem $D \in \mathsf{P}$ tal que a linguagem $A = D \cap B$ não pertence a P, $A \leq_m^p B$ mas $B \not\leq_m^p A$.*

Suponhamos que $\mathsf{P} \neq \mathsf{NP}$. Seja $B \in \mathsf{NPC}$ então $B \notin \mathsf{P}$ e $A \in \mathsf{NP}$. É verdade que $A \leq_m^p B$ mas $B \not\leq_m^p A$ pelo que A não pode ser NP-completa. Como $A \notin \mathsf{P}$, concluímos que $A \in \mathsf{NPI}$.

Problema candidato a pertencer a NPI é o Isomorfismo de grafos (GI). Está em NP mas não se sabe se é NP-completo ou P.

2.4 Classe coNP e Estrutura de NP

A classe dos problemas P é fechada em relação complementação. Dado um problema de decisão Π, o conjunto de soluções do problema complementar, Π^c, corresponde às soluções cuja resposta para o problema Π é *não*. Isto é, $S_{\Pi^c} = D_\Pi \setminus S_\Pi$.

Se Π tem um algoritmo determinístico polinomial, um algoritmo para Π^c obtém-se trocando as respostas *sim* por *não* (continuando a ser determinístico e polinomial). Como já se viu, o mesmo não acontece com a classe dos problemas NP. Nem sempre se pode provar que se $\Pi \in \mathsf{NP}$, então $\Pi^c \in \mathsf{NP}$. Por exemplo, o complementar do problema de decisão do "caixeiro viajante"(TSP) pode-se formular nos seguintes termos: Dado um conjunto de cidades, a distância inter-cidades e um limite B, é verdade que não existe *nenhum* circuito por todas as cidades com comprimento menor ou igual a B? A resposta a esta questão necessita da verificação de todos (ou quase todos)

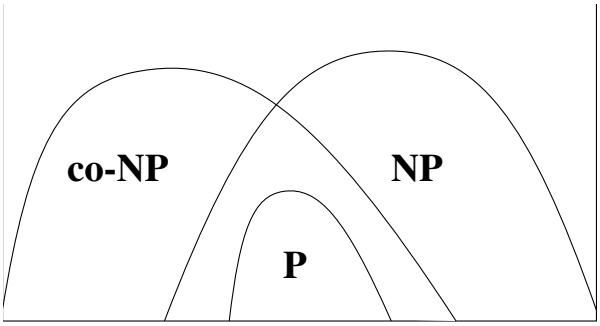


Figura 2.5: NP e coNP

os circuitos correctos, o que não parece ser possível usando um algoritmo não-determinístico em tempo polinomial!

Em termos formais, não parece que este problema tenha para cada instância, uma *testemunha concisa*. Seja a classe

$$\mathsf{coNP} \quad = \quad \{\Pi^c \mid \Pi \in \mathsf{NP}\}$$

Ou em termos de linguagens, dado um alfabeto Σ:

$$\mathsf{coNP} \quad = \quad \{\Sigma^\star \setminus L(\text{ou } L^c) \mid L \subseteq \Sigma^\star \text{ e } L \in \mathsf{NP}\}$$

Teorema 2.6 *Uma linguagem $L \subseteq \Sigma^\star$ pertence a* coNP *se e só se existe uma linguagem $R_L \subseteq \Sigma^\star \times \Sigma^\star$ que pertence a* P *e uma função polinomial p tal que:*

$$L \quad = \quad \{x \mid \forall y.\langle x, y\rangle \in R_L \ e \ |y| \leq p(|x|)\}$$

Notar que se um problema é NP-completo então o seu complementar é coNP-completo. Dado que existem muitos problemas em coNP que parecem não estar em NP, podemos conjecturar que $\mathsf{NP} \neq \mathsf{coNP}$. Note-se que se esta conjectura for verdadeira implica que $\mathsf{P} \neq \mathsf{NP}$. Porquê? Como se viu $\mathsf{P} \in \mathsf{coNP} \cap \mathsf{NP}$. Nota que o contrário não é verdade: pode ser que $\mathsf{P} = \mathsf{NP}$ e $\mathsf{coNP} \neq \mathsf{NP}$. Na Figura 2.5 tem-se a relação entre estas classes. O resultado seguinte relaciona os problemas NP-completos com esta conjectura.

Proposição 2.8 *Se existe um problema* NP*-completo Π, tal que $\Pi^c \in \mathsf{NP}$, então $\mathsf{NP} = \mathsf{coNP}$.*

Exercício 2.4.1 Mostra a Proposição 2.8.

Se $\mathsf{P} \neq \mathsf{NP}$ e $\mathsf{NP} \neq \mathsf{coNP}$ a estrutura para NP é a apresentada na Figura 2.6.

Exercício 2.4.2 Considera o problema complementar de 3SAT:

$\mathsf{3SAT}^c$

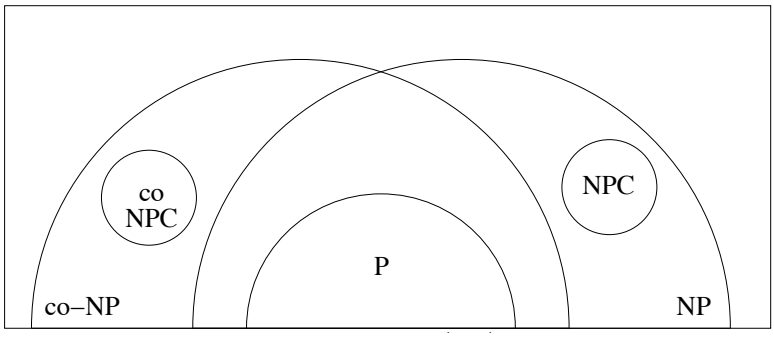


Figura 2.6: Estrutura de NP

Instância: Um conjunto U de variáveis lógicas e um conjunto C de cláusulas, cada uma com 3 literais.

Questão: Não existe nenhuma atribuição de valores de verdade às variáveis que satisfaça C?

Mostra que se tivéssemos um algoritmo polinomial para decidir este problema, todos os problemas de NP poderiam ser decididos polinomialmente. Comenta em relação completitude desse problema.

Exemplo 2.6 *Considera o seguinte problema: TAUT*

Instância: *Seja E uma expressão booleana com conectivas negação, disjunção e disjunção, para além de variáveis boolenas*

Questão: *A expressão E é verdadeira para qualquer valorização das variáveis?*

Este problema é coNP*-completo. Dada uma fórmula E podemos escolher uma valorização para as variáveis e verificar se a fórmula não é satisfeita (i.e se o seu valor é* 0*). Isto prova que TAUT está em* coNP*. Para ver que é completo, tem que se ver que qualquer problema em* coNP *se reduz a TAUT. Seja* $\Pi \in$ coNP*, então* $\Pi^c \in$ NP *e então* $\Pi^c \leq_m^p$ SAT*. Seja F a redução, então se considerarmos a redução complementar* F^c *essa é uma redução polinomial de* Π *para* TAUT*. Logo* $\Pi \leq_m^p$ TAUT*.*

2.5 Redução de Turing e Problemas de Procura

Vamos definir uma forma mais geral de redutibilidade polinomial, que pode ser aplicada a uma classe mais geral de problemas: problemas de procura em que para cada instância I, a resposta (em vez de ser apenas *sim* ou *não*) pode ser um elemento dum conjunto finito de soluções ou *não* (se o dito conjunto for vazio). Reparar que esta classe contém os problemas de decisão e os problemas de optimização.

Um problema de procura Π consiste num conjunto D_Π de instâncias, e para cada $I \in D_\Pi$ num conjunto finito $S_\Pi[I]$ de soluções de I. O problema Π tem solução para a instância I se $S_\Pi[I \neq \emptyset$.

Um algoritmo resolve o problema Π, se dado uma instância I tem como resposta *não* se $S_\Pi[\mathsf{I}] = \emptyset$ ou se tem como saída uma solução s pertencente a $S_\Pi[\mathsf{I}]$.

Por exemplo, o problema do caixeiro viajante tem como soluções todas as voltas de comprimento mínimo, e um algoritmo resolve o problema se para cada instância indicar uma das possíveis soluções.

Um problema de decisão Π pode ser formulado como um problema de procura definindo $S_\Pi[\mathsf{I}] = \{sim\}$ se $\mathsf{I} \in S_\Pi$ e $S_\Pi[\mathsf{I}] = \emptyset$, caso contrário.

Formalmente, dado um alfabeto um problema de procura vai corresponder a uma relação binária em Σ^+. A uma relação binária R em Σ^+ pode-se associar uma família de funções $f : \Sigma^+ \longrightarrow \Sigma^+$ tal que para todo $x \in \Sigma^+$, $f(x) = \epsilon$ se não existe nenhum $y \in \Sigma^+$ tal que $(x, y) \in R$ ou $f(x) = y$ se algum y existe. Diz-se que f realiza R. Uma TM M resolve R se a função f_M calculada por M realiza R.

Dado um problema de procura Π e uma codificação c sobre $\Sigma^\star$ associámos a relação $R[\Pi, c]$ definida por:

$$R[\Pi, c] \quad = \quad \{(x, y) \mid x = c(\mathsf{I}),\ \mathsf{I} \in D_\Pi \text{ e } y = c(s),\ s \in S_\Pi[\mathsf{I}]\}$$

O problema Π com codificação c é solúvel em tempo polinomial se existe uma TM polinomial que resolve $R[\Pi, c]$.

Exemplo 2.7 *Podemos identificar uma linguagem $L \subseteq \Sigma^+$ com a relação binária em Σ^+, tal que $R = \{(x, y) \mid x \in \Sigma^+ \text{ and } x \in L\}$ e onde y é um elemento fixo de Σ^+. Podemos concluir que um problema de decisão é um problema de procura.*

Podemos ainda definir uma classe especial de problemas de procura relacionados directamente com os problemas em NP e em P.

Seja $L \in \mathsf{NP}$. Pelo Teorema 2.3 existe uma relação R_L tal que para todo x, existe y tal que $R_L(x, y)$ se e só se $x \in L$. O problema de procura associado a L, FL é o seguinte: dado x, procurar y tal que $R_L(x, y)$ se tal y existe e se não existir retornar *não*.

A classe dos problemas de procura associados com linguagens em NP denota-se por FNP. O subconjunto de problemas associados com linguagens em P, denota-se FP.

Exemplo 2.8 $SATA \in \mathsf{FNP}$

É fácil concluir que $\mathsf{FP} = \mathsf{FNP}$ se e só se $\mathsf{P} = \mathsf{NP}$.

2.5.1 Redução de Turing

Em termos de problemas, uma redução polinomial de Turing dum problema Π num problema Π' é um algoritmo determinístico A que resolve Π usando uma subrotina S que resolve Π', e tal que se

S é um algoritmo polinomial para Π' então A é um algoritmo polinomial para Π. Diz-se que Π *é Turing redutível a* Π', e representa-se por $\Pi \leq_t \Pi'$.

Em termos de linguagens corresponde a existir uma máquina de Turing determinística com oráculo Π' que reconhece Π em tempo polinomial.

Definição 2.5 *Uma máquina de Turing com oráculo (MTO) corresponde a uma máquina de Turing básica (com uma fita)* $O = (S, \Gamma, \Sigma, \square, \triangleright, s_0, \delta, F)$ *com as seguintes características adicionais:*

- *possuí uma fita adicional – a fita de oráculo – e uma cabeça de escrita/leitura que opera nessa fita.*
- *o conjunto de estados* S *inclui dois estados especiais* s_c*, estado de consulta do oráculo e* s_r*, estado de continuação da computação.*
- *A função de transição é* $\delta : S \setminus F \cup \{s_c\} \times \Gamma \times \Sigma \longrightarrow S \times \Gamma \times \Sigma \times \{\leftarrow, \rightarrow\} \times \{\leftarrow, \rightarrow\}$

Um passo de computação duma MTO é análogo ao de uma TM básica (considerando as duas fitas) excepto quando o controlo finito se encontra no estado s_c. Neste estado, a computação depende duma função de oráculo $g : \Sigma^\star \longrightarrow \Sigma^\star$. Seja $y \in \Sigma^\star$ a sequência de símbolos nas células de 1 a $|y|$ da fita de oráculo e seja $g(y) = z$. então, *num* passo de computação a fita de oráculo é modificada para conter a sequência $z \in \Sigma^\star$ nas células de 1 a $|z|$ e brancos nas restantes células. A cabeça de oráculo fica a ler a célula 1 e os estado passa a ser s_r. A fita normal e sua cabeça não são alteradas neste passo.

Exercício 2.5.1 Descreve formalmente um passo de computação duma MTO.

Exercício 2.5.2 Generaliza uma MTO para k-fitas normais.

Uma MTO O com função de oráculo g associada designa-se por O_g. As noções de computação, função calculada f_O^g e complexidade $T_{O_g}(n)$ são definidas de modo idêntico ao das TM básicas.

Definição 2.6 *Sejam* R *e* R' *duas relações binárias em* $\Sigma^\star$*. Uma* redução polinomial de Turing *de* R *em* R'*,* $R \leq_t R'$*, é uma MTO* O *tal que para toda a função* $g : \Sigma^\star \longrightarrow \Sigma^\star$ *que realiza* R'*,* O_g *é uma MTO polinomial e a função* f_O^g *calculada por* O_g *realiza* R*.*

Proposição 2.9 *A relação* $\leq_t$ *é transitiva.*

Exercício 2.5.3 Mostra a Proposição 2.9.

Proposição 2.10 *Se* Π *e* Π' *são problemas de decisão então:*

$$\Pi \leq_m^p \Pi' \implies \Pi \leq_t \Pi'$$

Dem. A transformação F da redução polinomial $\leq_m^p$ define um algoritmo para resolver Π usando uma subrotina para resolver Π': dada uma instância de Π, constrói uma instância $F(X)$ de Π', aplica a subrotina a $F(X)$ e tem como resultado a resposta dada pela subrotina (pois X tem solução sse $F(X)$ tem solução). $\square$

Exercício 2.5.4 Enuncia e demonstra a Proposição 2.10 em termos de linguagens, relações e máquinas de Turing com oráculo.

Definição 2.7 *Uma relação R é* NP-hard *se existe uma linguagem $L \in \mathsf{NPC}$ tal que $L \leq_t R$.*

Um problema de procura Π diz-se NP-hard *se existe um problema* NP*-completo Π' tal que $\Pi' \leq_t \Pi$.*

Em particular, como já visto, um problema de decisão Π é NP-hard se $\forall \Pi' \in \mathsf{NP}$, $\Pi' \leq_t \Pi$ e é NP-completo se além disso $\Pi \in \mathsf{NP}$.

Exercício 2.5.5 Mostrar que dado um problema $\Pi \in \mathsf{NP}$ e o seu problema complementar Π^c, se tem $\Pi \leq_t \Pi^c$ e vice-versa. Explica porque não parece para muitos problemas que se possa ter $\Pi^c \leq_m^p \Pi$.

Pelo exercício anterior concluímos que se Π é NP-completo ou NP-hard então Π^c é NP-hard. Vimos que um problema de decisão D não é "mais difícil"que o correspondente de optimização O, vejamos que muitas vezes também não é "mais fácil"! Isto é não só $D \leq_t O$ mas também $O \leq_t D$. Se o problema D é NP-completo então o problema de optimização O pode ser resolvido em tempo polinomial se e só se $\mathsf{P} = \mathsf{NP}$.

Exemplo 2.9 *Consideremos o problema do* caixeiro viajante *mais uma vez. Comecemos por definir um problema intermédio:*

(TSE)

Instância: *Um conjunto $C = \{c_1, c_2, \ldots, c_m\}$ de cidades, $d : C \times C \longrightarrow \mathbb{N}$ distância, $B \in \mathbb{N}$ e uma volta* parcial $\Theta = \langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(k)} \rangle$ *de k cidades distintas, $1 \leq k \leq m$.*

Questão: *Pode-se estender Θ a uma volta completa $\langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(k)}, c_{\pi(k+1)}, \ldots c_{\pi(m)} \rangle$ com comprimento total menor ou igual a B?*

Este problema é NP *(mostra!) e como TSP é completo, tem-se que* TSE $\leq_t$ TSP*. Seja (TSO) o problema de optimização (página 15), resta ver que* TSO $\leq_t$ TSE *e por transitividade vem* TSO $\leq_t$ TSP.

Considerem-se instâncias de TSE e TSO para C e d comuns. Suponhamos que $\mathsf{S}(C, d, \Theta, B)$ é uma subrotina que resolve TSE. Seja $B^\star$ a volta óptima para essa instância de TSO e suponhamos que essa volta óptima começa em c_1. É óbvio que $B^\star$ tem como limite inferior $B_{MIN} = m$ e como

limite superior $B_{MAX} = mx$ *onde* $x = max\{d(c_i, c_j) \mid (c_i, c_j) \in C \times C\}$. *Podemos então usar uma pesquisa binária para determinar* $B^\star$ *que chama* $\mathsf{S}(C, d, \langle c_1 \rangle, B)$ *para vários valores de* B, *no máximo* $\lceil \log_2 B \rceil$ *vezes:*

$B_{MIN} \leftarrow m$
$B_{MAX} \leftarrow mx$
while $B_{MAX} - B_{MIN} \neq 1$ ***do***
  $B \leftarrow \lceil \frac{1}{2}(B_{MAX} + B_{MIN}) \rceil$
  if $\mathsf{S}(C, d, \langle c_1 \rangle, B) = sim$ ***then***
    $B_{MAX} \leftarrow B$
  else
    $B_{MIN} \leftarrow B$
  end if
end while
$B^\star \leftarrow B_{MIN}$

Conhecido $B^\star$ *determina-se uma volta óptima usando a subrotina* S. *Para isso constroem-se sequências* Θ *que possam ser estendidas a voltas óptimas. Dado* $\langle c_1 \rangle$ *e como é estendível, existe* $c_j \in C \setminus \{c_1\}$ *tal que* $\langle c_1, c_j \rangle$ *é uma volta parcial estendível. Podemos encontrar* c_j *no máximo em* $m - 2$ *chamadas a* $\mathsf{S}[C, d, \langle c_1, c_j \rangle, B^\star]$. *Então uma volta óptima pode ser construída usando* S *no máximo* $(m-1)(m-2)/2$ *vezes. Considerando o comprimento dos dados,* $n = m + \lceil \log_2 B_{MAX} \rceil$, *o algoritmo anterior fornece uma redução polinomial de Turing entre* TSO *e* TSE, *como pretendido.*

Note-se que basta provar que um dado problema é Turing–redutível a um problema em NP para saber que ele não é "mais difícil"que um problema NP-completo.

Definição 2.8 *Um problema* Π *é* NP-easy *se existe um problema* $\Pi' \in \mathsf{NP}$ *tal que* $\Pi \leq_t \Pi'$.

Exercício 2.5.6 Mostrar que os problemas de optimização (ou de procura de solução) associados com os seguintes problemas NP-completos são NP-easy:

1. SAT
2. VC
3. HC

Concluímos assim que a restrição feita aos problemas de decisão na teoria apresentada não provocou perda de generalidade, pois a maior parte dos problemas cujos correspondentes de decisão são NP-completos, são NP-easy e portanto só serão resolvidos em tempo polinomial se $\mathsf{P} = \mathsf{NP}$!

Exercício 2.5.7 Considera o problema de decisão seguinte:

K-ésimo Conjunto Maior

Instância: Dado um conjunto A, uma medida $s : A \longrightarrow \mathbb{Z}^+$, e dois inteiros não negativos $B \leq \sum_{a \in A} s(a)$ e $K \leq 2^{|A|}$.

Questão: Existem pelo menos K subconjuntos $A' \subseteq A$ tal que $\sum_{a \in A'} s(a) \leq B$?

Mostra que PAR $\leq_t$ K-ésimo Conjunto Maior. Será que K-ésimo Conjunto Maior $\in$ NP?

Sugestão: Considera uma subrotina $S[A, s, B, K]$ para K-ésimo Conjunto Maior. Se $\sum_{a \in A'} s(a)$ não é divisível por 2, a resposta é *não*. Senão, determina por pesquisa binária o número L^* de subconjuntos $A' \subseteq A$ satisfazendo $\sum_{a \in A'} s(a) \leq b$, com $b = \sum_{a \in A} s(a)/2$. Depois basta chamar $S[A, s, b-1, L^*]$.

2.6 A Classe DP

Consideremos mais uma variante do *caixeiro viajante*.

ETSP

Instância: Um conjunto $C = \{c_1, c_2, \ldots, c_m\}$ de cidades, $d : C \times C \longrightarrow \mathbb{N}$ distância, e um inteiro B.

Questão: O comprimento da volta mais curta é B?

Usando uma variante de HC considerando um caminho (HP), pode-se mostrar que HP $\leq_m^p$ ETSP (tomar $B = n + 1$). Como HP está em NPC, podemos concluir que TSP $\leq_m^p$ ETSP.

Exercício 2.6.1 Formaliza o raciocinio do parágrafo anterior.

Mas será que ETSP $\in$ NP? Como certificar que o custo óptimo é B? Do mesmo modo parece difícil determinar que B não é o custo óptimo. Portanto, também não podemos garantir que está em coNP.

Mas ETSP pode ser considerado como uma linguagem que é a interseção duma linguagem em NP (TSP) e uma linguagem em coNP: a linguagem TSPC onde a questão é saber se o custo óptimo é pelo menos B. Isto é, $\mathsf{I} \in S_{\mathsf{ETSP}}$ se e só se $\mathsf{I} \in S_{\mathsf{TSP}}$ e $\mathsf{I} \in S_{\mathsf{TSPC}}$.

Definição 2.9 *Uma linguagem L está na classe* DP *se e só se existem duas linguagens $L_1 \in$ NP e $L_2 \in$ coNP tal que $L = L_1 \cap L_2$.*

É evidente que DP não é NP $\cap$ coNP (esta última interseção é no dominio das classes).

Vamos ver que a classe DP também tem problemas completos. Seja

SAT-UNSAT

Instância: Dados dois conjuntos de cláusulas com 3 literais por cláusula C e C'.

Questão: É verdade que C é satisfazível e C' não é satisfazível

Teorema 2.7 *SAT-UNSAT é* DP*-completo.*

Dem. SAT-UNSAT $\in$ DP, porque sendo $L_1 = \{(C, C') \mid C \text{ é satisfazível}\}$ e $L_2 = \{(C, C') \mid C' \text{ não é satisfazível}\}$, tem-se que SAT-UNSAT $= L_1 \cap L_2$. Para a completude. Seja $L \in$ DP, temos que mostrar que $L \leq_m^p$ SAT-UNSAT. Existem $L_1 \in$ NP e $L_2 \in$ coNP tal que $L = L_1 \cap L_2$. Existe uma redução R_1 de L_1 a SAT e uma redução R_2 de L_2 a UNSAT. Seja $R(x) = (R_1(x), R_2(x))$. Então $R(x)$ é uma instância "sim"para SAT-UNSAT sse $R_1(x)$ é satisfazível e $R_2(x)$ não é satisfazível o que é verdade sse $x \in L_1$ e $x \in L_2$, isto é $x \in L$. □

Também se pode mostrar que:

Teorema 2.8 *ETSP é* DP*-completo.*

A classe DP é uma classe de linguagens que pode ser decidido por uma MTO especial. A máquina faz duas perguntas a um oráculo SAT e aceita se e só se a primeira resposta é *sim* e a segunda é *não*.

2.7 Hierarquia polinomial

Nesta secção vamos estudar problemas de decisão que são NP-hard mas que parecem não ser NP-easy. Podemos generalizar a noção de redução de Turing permitindo que o algoritmo usado seja não-determinístico, o que equivale, em termos de linguagens, a considerar máquinas de Turing não-determinísticas com oráculo. Temos por uma lado uma escolha e por outro uma consulta ao oráculo. Dados dois problemas Π e Π', se existir um algoritmo não-determinístico polinomial que resolva Π usando uma subrotina que resolva Π', indica-se $\Pi \leq_t^N \Pi'$.

Vamos ver um problema que é NP-hard mas que não parece ser NP-easy e para o qual se pode encontrar uma redução $\leq_t^N$.

Exemplo 2.10 *Considera o seguinte problema*

(EME)

Instância: *Uma expressão booleana E com literais num conjunto de variáveis U, constantes V e F e conectivas lógicas $\vee$, $\wedge$, $\neg$ e $\rightarrow$. E ainda, $K \in \mathbb{N}$.*

Questão: *Existe uma expressão booleana E' com K ou menos literais e tal que E' é equivalente a E?*

Este problema é NP-hard *porque* SAT $\leq_t$ EME *(Verifica). No entanto, não parece fácil mostrar que ele é Turing redutível a um problema de* NP*. Contudo, usando um algoritmo não-determinístico*

com um "oráculo"podemos reduzir este problema ao problema da "satisfação de expressões booleanas":

(SBE)

Instância: *Uma expressão booleana E com literais num conjunto de variáveis U, constantes* 1 *e* 0 *e conectivas lógicas* $\vee$, $\wedge$, $\neg$ *e* $\rightarrow$.

Questão: *Existe uma atribuição de valores de verdade a U que satisfaz E?*

Este problema contém em particular o problema SAT, *portanto é* NP*-completo. Seja uma instância de* EME *com dados U, E e K, suponha-se B_K o conjunto das expressões booleanas com K ou menos literais e seja, ainda,* $\mathsf{S}[E,U]$ *uma subrotina que resolve* SBE. *O seguinte algoritmo estabelece a redução,* $\mathsf{EME} \leq_t^N \mathsf{SBE}$*:*

$E' \leftarrow$ESCOLHA(B_K)
if $S[\neg((E \rightarrow E') \wedge (E' \rightarrow E)), U] =$ *não* ***then*** SUCESSO
else
INSUCESSO
end if

Concluímos então que EME *pertence a uma classe mais ampla de problemas aparentemente* mais difíceis *do que os da classe* NP.

Se $L \leq_t L'$ podemos escrever $L \in \mathsf{P}^{L'}$ onde

$$\mathsf{P}^{L'} = \{L(O) \mid O \text{ é uma MTO com oráculo } L'\}.$$

Do maneira análoga podemos definir $\mathsf{NP}^{L'}$. Sendo Y uma classe de complexidade, as classes P^{Y} e NP^{Y} são definidas do modo seguinte:

$$\begin{aligned} \mathsf{P}^{\mathsf{Y}} &= \{L \mid \exists L' \in \mathsf{Y} \text{ e } L \leq_t L'\} = \bigcup_{L' \in \mathsf{Y}} \mathsf{P}^{L'} \\ \mathsf{NP}^{\mathsf{Y}} &= \{L \mid \exists L' \in \mathsf{Y} \text{ e } L \leq_t^N L'\} = \bigcup_{L' \in \mathsf{Y}} \mathsf{NP}^{L'} \end{aligned}$$

Se L é $\leq_m^p$-completa para a classe Y então $\mathsf{P}^{\mathsf{Y}} = \mathsf{P}^{\mathsf{L}}$ e $\mathsf{NP}^{\mathsf{Y}} = \mathsf{NP}^{\mathsf{L}}$.

Exercício 2.7.1 *Mostra que* $\mathsf{SAT} \leq_t \mathsf{EME}$.

P^{NP} é a classe dos problemas a que chamamos anteriormente NP-easy. O problema EME pertence a $\mathsf{NP}^{\mathsf{NP}}$ ($=\mathsf{NP}^{\mathsf{SAT}}$).

Em termos de classes para problemas de procura, podemos concluir que concluir TSP pertence à classe $\mathsf{FP}^{\mathsf{NP}}$ (e até é completo para esta classe). Muito problemas de optimização pertencem também a esta classe.

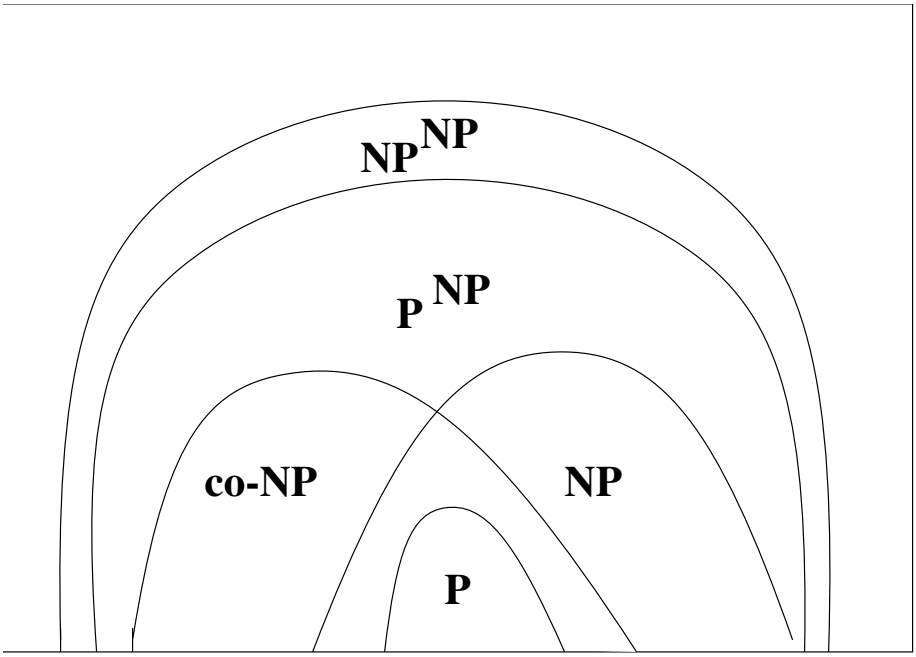


Figura 2.7: Classes P, NP, coNP, P^{NP} e $\mathsf{NP}^{\mathsf{NP}}$

Exercício 2.7.2 Considera o problema seguinte: EE

Instância: Duas expressões booleanas E e E' com literais num conjunto de variáveis U, constantes V e F e conectivas lógicas $\vee$, $\wedge$, $\neg$ e $\rightarrow$.

Questão: E e E' são equivalentes?

(i) Mostra que EE é coNP.

(ii) Concluí, novamente, que $\mathsf{EME} \in \mathsf{NP}^{\mathsf{NP}}$ usando a alínea anterior.

Se $\mathsf{P} \neq \mathsf{NP}$ a Figura 2.7 mostra a relação entre as classes P, NP, coNP, P^{NP} e $\mathsf{NP}^{\mathsf{NP}}$.
Este processo de definir novas classes pode ser estendido indefinidamente, produzindo classes de dificuldade aparentemente crescente, [MS72]. Esta hierarquia de classes chama-se *hierarquia polinomial* e é definida do seguinte modo[8]:

$$\begin{array}{lcl} \Sigma_0^{\mathsf{p}} = \Pi_0^{\mathsf{p}} = \Delta_0^{\mathsf{p}} = \mathsf{P} \\ \Delta_k^{\mathsf{p}} = \mathsf{P}^{\Sigma_{k-1}^{\mathsf{p}}} \\ \Sigma_k^{\mathsf{p}} = \mathsf{NP}^{\Sigma_{k-1}^{\mathsf{p}}} \\ \Pi_k^{\mathsf{p}} = \mathsf{co}\Sigma_k^{\mathsf{p}} \end{array}$$

[8]Podemos omitir o p sempre que não houver confusão ou então escrever $\Sigma_i\mathsf{P}$, etc.

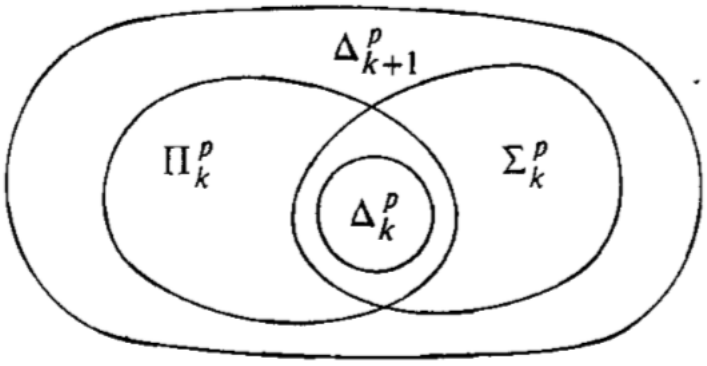


Figura 2.8: Hierarquia Polinomial. PH

Define-se, ainda,

$$\mathsf{PH} = \bigcup_{k \in \mathbb{N}} \Sigma^p_k.$$

Em particular, $\Sigma^{\mathsf{p}}_1 = \mathsf{NP}^{\mathsf{P}} = \mathsf{NP}$, $\Pi^{\mathsf{p}}_1 = \mathsf{coNP}$, $\Delta^{\mathsf{p}}_1 = \mathsf{P}$, $\Sigma^{\mathsf{p}}_2 = \mathsf{NP}^{\mathsf{NP}}$ e $\Delta^{\mathsf{p}}_2 = \mathsf{P}^{\mathsf{NP}}$. Notar que $\Pi^{\mathsf{p}}_{\mathsf{k}} = \mathsf{coNP}^{\Sigma^{\mathsf{p}}_{k-1}}$.

Como se viu EME pertence a Σ^{p}_2, embora não se saiba se $\Sigma^{\mathsf{p}}_1 = \Sigma^{\mathsf{p}}_2$, ou se existe $k \geq 0$ tal que $\Sigma^{\mathsf{p}}_{\mathsf{k}} \neq \Sigma^{\mathsf{p}}_{\mathsf{k+1}}$. A relação de inclusão entre as várias classes é dada pelo diagrama da Figura 2.8. A classe DP está contida em Δ^{p}_2.

Para um dado problema de decisão Π, como situá-lo na hierarquia? Vamos apresentar um resultado que permite determinar um majorante do menor k tal que $\Pi \in \Sigma^{\mathsf{p}}_{\mathsf{k}}$.

Recordemos que um problema está em NP se tiver uma *testemunha concisa*. Esta noção pode ser estendida para qualquer problema de decisão Π, isto é, qualquer linguagem L.

Dado um alfabeto Σ uma relação k-ária em $\Sigma^\star$ é reconhecida em tempo polinomial se existir uma TM que reconhece precisamente os tuplos $(x_1, \ldots, x_k) \in R$, isto é, tal que $R(x_1, \ldots, x_k)$ se verifica.

Podemos então dizer que:

- $L \in \mathsf{P}$ se e só se existe $R_1 \in \Sigma^\star$ reconhecível polinomialmente tal que $L = \{x \mid R_1(x)\}$
- $L \in \mathsf{NP}$ se e só se existe $R_2 \in \Sigma^\star \times \Sigma^\star$ reconhecível polinomialmente e uma função polinomial p tal que $L = \{x \mid \exists y, |y| \leq p(|x|) \text{ e } R_2(x, y)\}$
- $L \in \mathsf{coNP}$ se e só se existe $R^c_2 \in \Sigma^\star \times \Sigma^\star$ reconhecível polinomialmente e uma função polinomial p tal que $L = \{x \mid \forall y, |y| \leq p(|x|) \text{ e } R^c_2(x, y)\}$

Os resultados seguintes são de Wrathall [Wra76] (ver [Pap94, Sec. 17.2]).

Proposição 2.11 *Seja $L \subseteq \Gamma^\star$ uma linguagem sobre um alfabeto Γ e ($|\Gamma| \geq 2$) e $i \geq 1$. A linguagem $L \in \Sigma^{\mathsf{p}}_{\mathsf{k}}$ se e só se existe uma relação reconhecida em tempo polinomial tal que a linguagem $\{\langle x, y\rangle \mid R(x, y)\} \in \Pi^{\mathsf{p}}_{\mathsf{k}-1}$ e existe p tal que $L = \{x \mid \exists y,\ |y| \leq p(|x|)\ e\ R(x, y)\}$.*

Dem. A prova segue por indução em k. Para $k = 1$ é imediato. Suponhamos que o resultado é válido para $k > 1$ e suponhamos que R existe. Temos de mostrar que $L \in \Sigma^{\mathsf{p}}_{\mathsf{k}}$. Isto é, exibir uma *MTO* O máquina de Turing não determinística limitada em tempo polinomial com um oráculo em $\Sigma^{\mathsf{p}}_{\mathsf{k}-1}$, tal que $L = L(O)$. A máquina O com dados x adivinha y e pergunta ao oráculo se $(x, y) \notin R$. No outro lado suponhamos que $L \in \Sigma^{\mathsf{p}}_{\mathsf{k}}$. Temos que mostrar que R existe. Sabemos que L pode ser decidida por uma MTO O não determinística com oráculo $L' \in \Sigma^{\mathsf{p}}_{\mathsf{k}-1}$. Por indução, existe uma relação S reconhecida em $\Pi^{\mathsf{p}}_{\mathsf{k}-2}$ tal que $z \in L'$ sse existe w (e um polinómio o com $|w| \leq p(|z|)$) tal que $(z, w) \in S$. Podemos obter um certificado sucinto para cada $z \in L$, sabendo que existe uma computação de aceitação de $O^{L'}$ para x. Seja y a string que representa esse computação. Como $O^{L'}$ tem várias chamadas ao oráculo L', z_i, umas podendo ter resposta "sim"outras "não". Para cada "sim", o certificado y inclui também o certificado w_i tal que $(z_i, w_i) \in S$. A definição de R é a seguinte: $(x, y) \in R$ se e só se y representa uma computação de $O^{L'}$ com dados x, juntamente com os certificados w_i para cada pergunta z_i com resposta "sim"na computação. Temos que mostrar que testar se $(x, y) \in R$ pode ser feito em $\Pi^{\mathsf{p}}_{\mathsf{k}-1}$.Testar se os passos de $O^{L'}$ são correctos pode ser feito em tempo polinomial. Depois temos de verificar se $(z_i, w_i) \in S$, sendo estes pares em número polinomial. Cada teste pode ser feito em $\Pi^{\mathsf{p}}_{\mathsf{k}-2}$ e portanto em $\Pi^{\mathsf{p}}_{\mathsf{k}-1}$. Para todas as perguntas com resposta negativa, z'_i, é necessário testar que $z'_i \notin L'$. Mas isto é um pergunta em $\Pi^{\mathsf{p}}_{\mathsf{k}-1}$. Assim $(x, y) \in R$ se e só se diversas perguntas em $\Pi^{\mathsf{p}}_{\mathsf{k}-1}$ tem a resposta "sim"e isto pode ser feito como uma única computação em $\Pi^{\mathsf{p}}_{\mathsf{k}-1}$. $\square$

Corolário 2.2 *Seja $L \subseteq \Gamma^\star$ uma linguagem sobre um alfabeto Γ e ($|\Gamma| \geq 2$) e $i \geq 1$. $L \in \Pi^{\mathsf{p}}_{\mathsf{k}}$ se e só se existe uma relação reconhecida em tempo polinomial tal que a linguagem $\{\langle x, y\rangle \mid R(x, y)\} \in \Sigma^{\mathsf{p}}_{\mathsf{k}-1}$ e existe p tal que $L = \{x \mid \forall y,\ |y| \leq p(|x|)\ e\ R(x, y)\}$.*

Iterando temos,

Teorema 2.9 *Seja $L \subseteq \Gamma^\star$ uma linguagem sobre um alfabeto Γ e($|\Gamma| \geq 2$). Para todo $k \geq 1$ $L \in \Sigma^{\mathsf{p}}_{\mathsf{k}}$ se e só se existem funções polinomiais $p_1, \ldots, p_k$ e uma relação R reconhecida em tempo polinomial tal que:*

$$\begin{aligned} x \in L \quad sse \quad & \exists y_1 \in \Gamma^\star |y_1| \leq p_1(|x|) \\ & \forall y_2 \in \Gamma^\star |y_2| \leq p_2(|x|) \\ & \ldots \\ & Q y_k \in \Gamma^\star |y_k| \leq p_k(|x|) \\ & R(x, y_1, \ldots, y_k), \end{aligned}$$

onde Q em y_k é $\exists$ ou $\forall$ consoante k é ímpar ou par.

Usando o Teorema 2.9 para provar que $\mathsf{EME} \in \Sigma_2^{\mathsf{p}}$, considere-se a relação R, com $R(x, y_1, y_2)$ se e só se x é $\langle E, K\rangle$ (instância) onde E é uma expressão booleana e $K \in \mathbb{N}$, y_1 é uma expressão booleana E' com no máximo K literais, e y_2 é uma atribuição de valores às variáveis $t : U \longrightarrow \{0, 1\}$ que satisfaz $E \Leftrightarrow E'$:

$$\exists E' \forall t, t(E) = t(E').$$

As classes $\Pi_{\mathsf{k}}^{\mathsf{p}}$ também podem ser caracterizadas de modo análogo, bastando no Teorema 2.9 trocar todos os quantificadores $\exists$ por $\forall$ e vice-versa.

A noção de completitude pode ser definida para cada k e para cada um dos tipos de classes da hierarquia, caracterizando os problemas "mais difíceis"de cada classe. Para cada classe é possível construir um problema $\Sigma_{\mathsf{k}}^{\mathsf{p}}$-completo e portanto torna-se mais fácil provar a completitude de outros problemas.

Na prática só se conhecem alguns problemas (além dos construídos teoricamente) candidatos a $\Sigma_2^{\mathsf{p}} \setminus (\Sigma_1^{\mathsf{p}} \cup \Sigma_1^{\mathsf{p}})$ ou mesmo Σ_2^{p}-completos.

Uma questão importante é saber se $\Sigma_k^p \subset \Sigma_{k+1}^p$. Um resultado interessante é o seguinte:

Proposição 2.12 *Se para algum k, $\Sigma_{\mathsf{k}}^{\mathsf{p}} = \Pi_{\mathsf{k}}^{\mathsf{p}}$ então $\Sigma_{\mathsf{j}}^{\mathsf{p}} = \Pi_{\mathsf{j}}^{\mathsf{p}} = \Delta_j^p = \Sigma_{\mathsf{k}}^{\mathsf{p}}$ para todo o $j \geq k$.*

Exercício 2.7.3 Mostrar a Proposição 2.12. Sugestão: Mostra que $\Sigma_{\mathsf{k}}^{\mathsf{p}} = \Pi_{\mathsf{k}}^{\mathsf{p}}$ implica $\Sigma_{\mathsf{k+1}}^{\mathsf{p}} = \Sigma_{\mathsf{k}}^{\mathsf{p}}$.

Isto é, a hierarquia pode colapsar a partir de certa ordem e portanto não ser infinita. Em particular, se $\mathsf{P} = \mathsf{NP}$, então $\Sigma_{\mathsf{j}}^{\mathsf{p}} = \mathsf{P}$, para todo $j \geq 0$. Temos também que se $\mathsf{NP} = \mathsf{coNP}$ então $\mathsf{NP} = \mathsf{PH}$. Se $\Sigma_0^{\mathsf{p}} \neq \Sigma_{\mathsf{k}}^{\mathsf{p}}$ para algum $k > 0$, então $\mathsf{P} \neq \mathsf{NP}$. Temos então que qualquer problema da hierarquia só pode ser resolvido em tempo polinomial se e só se $\mathsf{P} = \mathsf{NP}$ (e portanto embora possa não ser NP-easy na prática tem as mesmas consequências). Contudo, o estudo destas classes pode ajudar a estabelecer a conjectura de que $\mathsf{P} \neq \mathsf{NP}$.

2.7.1 Problemas completos para níveis de PH

PH não tem problemas completos a menos que colapse no primeiro nível.

Proposição 2.13 *Se existir uma linguagem L que seja PH-completa, então existe um $i \geq 1$ tal que $\mathsf{PH} = \Sigma_i^p$.*

Dem. Seja i tal que $L \in \Sigma_i^p$. Qualquer problema em PH reduz-se polinomialmente a L. Mas então qualquer linguagem de PH está em Σ_i^p. Donde, $\mathsf{PH} \subseteq \Sigma_i^p$. □

Considere-se o seguinte problema: QSAT_i

Instância: Seja $i \geq 1$ e uma fórmula Booleana quantificada

$$\exists y_1 \forall y_2 \exists y_3 \cdots Q_i y_i . \phi(y_1, \ldots, y_i),$$

onde cada y_j representa uma sequência de variáveis Booleanas e ϕ é uma expressão Booleana envolvendo os y_j.

Questão: A fórmula é verdade? Isto é, existem valorizações das variáveis Booleanas que a satisfaçam?

Proposição 2.14 *Para qualquer $i \geq 1$, o problema QSAT_i é Σ_i^p-completo.*

Dem. (Idea da prova) O problema está em Σ_i^p. Vamos supor que ϕ está em forma normal conjuntiva (instância de SAT). Seja $L \in \Sigma_i^p$. Então, pelo Teorema 2.9, L é da forma

$$\{x \mid \exists y_1 \forall y_2 \cdots Q_i y_i R(x, y_1, \ldots, y_i)\},$$

onde R é uma relação reconhecida em tempo polinomial. Pelo teorema de Cook, existem ϕ_x^1, ϕ_x^2 (instâncias de SAT e com x fixo), tal que:

$$\begin{array}{rcl} R(x, y_1, \ldots, y_i) & \text{sse} & \exists z . \phi_x^1(y_1, \ldots, y_i, z) \\ \neg R(x, y_1, \ldots, y_i) & \text{sse} & \exists z . \neg \phi_x^2(y_1, \ldots, y_i, z). \end{array}$$

Se $Q_i = \exists$ então

$$\begin{array}{rcl} x \in L & \Leftrightarrow & \exists y_1 \forall y_2 \cdots \exists_i \exists z . \phi_x^1(y_1, \ldots, y_i, z) \\ & \Leftrightarrow & \phi_x^1 \in \mathsf{QSAT}_i. \end{array}$$

E se $Q_i = \forall$ então

$$\begin{array}{rcl} x \in L & \Leftrightarrow & \exists y_1 \forall y_2 \cdots \forall y_i . R(x, y_1, \ldots, y_i) \\ & \Leftrightarrow & \neg(\forall y_1 \exists y_2 \cdots \exists y_i . \neg R(x, y_1, \ldots, y_i)) \\ & \Leftrightarrow & \neg(\forall y_1 \exists y_2 \cdots \forall y_i . \exists z . \neg \phi_x^2(y_1, \ldots, y_i, z) \\ & \Leftrightarrow & \exists y_1 \forall y_2 \cdots \exists y_i \forall z . \phi_x^2(y_1, \ldots, y_i, z) \\ & \Leftrightarrow & \phi_x^2 \in \mathsf{QSAT}_i. \end{array}$$

□

2.8 Exercícios de Revisão sobre Classes de Complexidade e NP-completitude

Exercício 2.8.1 Caminhos Hamiltonianos
Sejam respectivamente HPN e HPE os problemas de decisão "existência de um caminho hamiltoniano qualquer (entre vértices distintos)" e "existência de um caminho hamiltoniano entre dois vértices (distintos) especificados". Note que, no primeiro caso, uma instância é um grafo não dirigido $G = (V, E)$ enquanto no segundo caso uma instância é da forma $G = (V, E)/x/y$ onde G é um grafo não dirigido e $x, y \in V$ são os vértices especificados.

1. Mostra que HPE $\leq_m^p$ HPN
2. Mostra que HPN $\leq_m^p$ HPE

Exercício 2.8.2 Colorir grafos
Considera os seguintes problemas de decisão a que chamamos $C(n)$ (onde, para cada problema n é fixo):

(n-COLOR)

Instância: Um grafo dirigido $G = (V, E)$.

Questão: É possível colorir G com n cores, isto é, existe uma função

$$f : V \longrightarrow \{1, \cdots, n\}$$

tal que sempre que $(a, b) \in E$ é $f(a) \neq f(b)$?

1. Mostra que $C(1)$ e $C(2)$ pertencem à classe P.
2. Supondo que se sabe que $C(3)$ é NP-completo mostra que $C(4)$ também é NP-completo. Um método consiste em mostrar que $C(4)$ é NP e que $C(3) \leq_m^p C(4)$.
3. Supondo que se sabe que $C(3)$ é NP-completo mostra que $C(n)$ é NP-completo para todo o $n \geq 3$.

Exercício 2.8.3 Acabas de demonstrar que um determinado problema Π é NP-completo. Qual o significado *prático* dessa descoberta? Podemos concluir que todos os algoritmos para resolver Π são quase inúteis uma vez que vão demorar um tempo exponencial?

Exercício 2.8.4 *Caminhos entre 2 vértices de um grafo*
Considera um grafo não dirigido $G(V, E)$ e dois vértices fixos a e b de V.

1. Mostra que o número de caminhos simples e distintos entre a e b pode ser exponencial no tamanho do grafo — por exemplo em $max(|V|, |E|)$. Nota: um caminho diz-se *simples* quando não passa mais que uma vez por cada nó ("não tem ciclos"). Dois caminhos são distintos quando têm pelo menos um vértice diferente.

   Sugestão 1: Considera um grafo completo de n nós. *Sugestão 2:* Considera um grafo "quadriculado" de n por n, dois vértices opostos (na diagonal) e os caminhos entre eles que progridem sempre no sentido da diagonal.

2. Considera o problema do caminho mais longo:

   (CML)

   Instância: $G(V, E)$, $a \in V$, $b \in V$, com $a \neq b$ e inteiro k.

   Questão: Existe um caminho simples entre a e b de comprimento maior ou igual a k?

   Mostra que CML é NP-completo. Nota: Existem algoritmos polinomiais como o de Dijkstra e o de Floyd para o problema do caminho mais curto.

3. Classifica os problemas de decisão correspondentes às questões indicadas; a instância é sempre um grafo *dirigido* $G(V, E)$, $a \in V$, $b \in V$ com $a \neq b$ e k (todavia, para os 2 últimos problemas a instância não contêm k). Podes usar os resultados mencionados (ou cuja demonstração é proposta) nas alíneas anteriores; estes resultados são também válidos em grafos dirigidos.

   (a) Todos os caminhos entre a e b têm comprimento maior ou igual a k.

   (b) Todos os caminhos entre a e b têm comprimento menor ou igual a k.

   (c) Todos os caminhos simples entre a e b têm comprimento menor ou igual a k.

   (d) Existe um caminho de comprimento menor que k.

   (e) Existe um caminho de comprimento maior que k.

   (f) Não existe nenhum caminho entre a e b.

   (g) Existem caminhos de comprimento arbitrariamente grande entre a e b?

Exercício 2.8.5 Considera o seguinte problema de decisão:

(IC) *(Implicação de Cláusulas)*

Instância: Conjuntos de cláusulas C_1 e C_2 com variáveis lógicas de um conjunto U.

Questão: C_1 implica C_2? Por outras palavras: C_2 tem o valor lógico *verdade* sempre que C_1 tiver o valor lógico *verdade*?

1. Descreve de forma resumida e utilizando uma linguagem de alto nível um algoritmo determinístico que resolve (IC). O algoritmo é polinomial ou exponencial?

2. A que classe de complexidade pertence IC? Porquê?

3. (PCE)*(Poucas Cláusulas Equivalentes)*

   Instância: Conjunto de cláusulas C, conjunto U de variáveis lógicas e inteiro positivo k.

   Questão: Existe um conjunto C' de k ou menos cláusulas utilizando variáveis de U que seja equivalente a C (isto é, tal que C e C' tenham o mesmo valor lógico qualquer que seja a atribuição de valores lógicos às variáveis de U)?

   Escreve o que descobrires sobre a classe de complexidade a que pertence PEC.

Exercício 2.8.6 Considera uma máquina de Turing M, uma palavra x e um inteiro positivo t. Diz se é possível exprimir em cláusulas (eventualmente dependentes de M, x e t) os seguintes factos por forma que as cláusulas sejam satisfazíveis se e só se os factos forem verdadeiros.

1. A máquina M com os dados x pára em não mais de t passos.

2. A máquina M com os dados x pára (num número qualquer de passos).

3. A máquina M com os dados x não pára.

Exercício 2.8.7 Considera o problema EL da equivalência de 2 expressões lógicas que podem conter variáveis proposicionais, variáveis proposicionais negadas e as conectivas $\vee$ e $\wedge$:

(EL)

Instância: Um conjunto U de variáveis lógicas e 2 expressões lógicas E_1 e E_2 com variáveis em U.

Questão: "E_1 e E_2 são equivalentes?", isto é, têm o mesmo valor lógico para todas as atribuições de valores lógicos às variáveis de U?

Exemplo:

$$U = \{u_1, u_2\},\ E_1 = (u_1 \vee \overline{u_2}) \wedge (\overline{u_1} \vee u_2),\ E_2 = u_1 \wedge \overline{u_1},$$

1. Considera as seguintes afirmações com as quais se pretende mostrar a dificuldade de resolver EL. Comenta-as devidamente.

   (a) EL é NP-completo porque nunca ninguém conseguiu um algoritmo polinomial para o resolver.

   (b) EL é NP porque não existe nenhum algoritmo polinomial para o resolver.

(c) EL é NP-completo porque se conseguissemos resolver EL em tempo polinomial também era possível resolver SAT em tempo polinomial pois uma instância (U, C) de SAT é satisfazível se e só se a instância de EL

$$U',\ E_1 = E_c,\ E_2 = (u_n \wedge \overline{u_n})$$

não tiver solução, onde $U' = U \cup \{u_n\}$, u_n é uma nova variável e E_c é a expressão lógica correspondente ao conjunto C de cláusulas (conjunção de disjunções).

2. Determina a classe de complexidade a que *de facto* pertence EL.

Exercício 2.8.8 Classifica o mais exactamente possível os seguintes problemas. Justifica. Em todos eles é dada uma colecção C de k objectos com tamanhos inteiros $t_1, t_2, \ldots, t_k$ e um tamanho (inteiro) da "mala" t, sendo $t_i \leq t$ para $1 \leq 1 \leq k$. Repara que um mesmo problema Π pode pertencer simultâneamente a várias classes, por exemplo ser decidivel, NP e P; a última caracterização é a mais exacta.

1. é possível arrumar todos os objectos numa mala, isto é, $\sum_{i=1}^{k} t_i \leq t$?
2. Dado (também) m pergunta-se é possível arrumar todos os objectos em não mais de m malas?
3. Dado (também) m pergunta-se: Todas as arrumações exigem pelo menos m malas?
4. Qual o menor número de malas em que é possível arrumar todos os objectos?
5. Dados (também) m e c pergunta-se: todos os subconjuntos de C com c objectos podem ser arrumados em não mais de m malas?
6. Dados (também) m, t_m e c pergunta-se: todos os subconjuntos de C com c objectos cuja soma dos tamanhos é pelo menos t_m podem ser arrumados em não mais de m malas?

Exercício 2.8.9 Considera o problema de decisão do (CLIQUE) que se sabe ser NP–completo.

1. O correspondente problema de optimização é o seguinte: "dado um grafo não dirigido, qual o tamanho do maior *clique* que ele possui?" Mostra que se existisse um algoritmo polinomial para o problema de decisão então também existia um algoritmo polinomial para o de optimização (o recíproco é evidente).
2. Classifica os seguintes problemas de decisão utilizando apenas a informação de que o problema do (CLIQUE) é NP–completo.
   (a) Dado G e k pergunta-se: G tem dois (ou mais) "cliques" disjuntos de tamanho não inferior a k?
   (b) Dado G pergunta-se: G tem um "clique" de ordem 22?

(c) Dado G pergunta-se: não terá G um "clique" de ordem 22?

(d) Dados G_1 e G_2 pergunta-se: existe em G_1 um subgrafo isomorfo a G_2?

(e) Dados G_1 e G_2 pergunta-se: não terá G_1 nenhum subgrafo isomorfo a G_2?

(f) Dado G e k pergunta-se: existe um subconjunto com um máximo de k vértices tal que todo o ramo tem (pelo menos) uma extremidade que é um vértice desse subconjunto?

Exercício 2.8.10 Considere um problema de decisão cuja instância é (A, k) sendo k um inteiro não negativo e cuja pergunta é "$P(A, k)$?" (P é uma proposição). No correspondente problema de optimização pergunta-se "qual o máximo k tal que $P(A, k)$?". Supomos que se verifica a seguinte condição: sempre que existe solução para um determinado valor de k, existe solução para todos os k' tal que $0 \leq k' \leq k$.

1. Mostra que se existir um algoritmo polinomial para o problema de optimização então também existe um algoritmo polinomial para o problema da decisão.

2. Mostra que em condições bastante gerais o inverso (da alínea anterior) também é verdadeiro. Sugestão: considera a possível existência de um majorante de k.

3. Exemplifica os resultados anteriores para o problema do "clique".

Exercício 2.8.11 A Hierarquia cresce
Mostra que, para todo o $k \geq 0$ é

$$\Delta_k^p \subseteq \Sigma_{k+1}^p \subseteq \Delta_{k+1}^p$$

Capítulo 3

Espaço e Teoremas de Hierarquia

Começamos por rever e aprofundar alguns conceitos sobre classes e complexidade em tempo.

3.1 Complexidade em Tempo

Dada uma máquina de Turing M já definimos a sua complexidade temporal e algumas classes de complexidade associadas. O tempo requerido por M em x, $T_M(x)$, é o número de passos da computação de M para dados x, e

$$T_M(n) = max\{t \mid \exists x \in \Sigma^\star,\ |x| = n \text{ e } t = T_M(x)\}$$

é o limite temporal de M (ou seja M opera em tempo limitado por $T_M(n)$). Dada uma função $f(n)$ tempo-construível, a classe de complexidade DTIME($f(n)$) é:

$$\text{DTIME}(f(n)) \quad = \quad \{L \mid L \text{ é decidível por uma MT multi-fita } M \text{ tal que } T_M(n) \text{ é } O(f(n))\}$$

Claro que se $\forall n, f(n) \geq g(n)$, então DTIME($g(n)$) $\subseteq$ DTIME($f(n)$).
Em particular, $\mathsf{P} = \bigcup_{k\geq 1} \text{DTIME}(n^k)$. Podemos ainda definir a classe:

$$\mathsf{EXP} \quad = \quad \bigcup_{k\geq 1} \text{DTIME}(2^{n^k}),$$

e $\mathsf{P} \subseteq \mathsf{EXP}$.

Do mesmo modo, sendo N uma máquina de Turing não-determínistica sendo $T_N(x)$ o número de passos que não é excedido em qualquer caminho de computação com dados x, a complexidade temporal de N é a função:

$$T_N(n) \quad = \quad max\{\{1\} \cup \{m \mid \exists\ x \in L(N),\ |x| = n \text{ e } m = T_N(x)\}\}$$

Com,

$$\text{NTIME}(f(n)) \quad = \quad \{L \mid L \text{ é decidível por uma MTN } N \text{ tal que } T_N(n) \text{ é } O(f(n))\}$$

Pela definição de MTN, DTIME($f(n)$) $\subseteq$ NTIME($f(n)$).

Em particular, $\mathsf{NP} = \bigcup_{k\geq 1} \mathrm{NTIME}(n^k)$. E também podemos definir a classe:

$$\mathsf{NEXP} \;=\; \bigcup_{k\geq 1} \mathrm{NTIME}(2^{n^k}).$$

Temos que:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}.$$

Iremos ver que pelo menos uma destas inclusões tem de ser estrita.

Para melhor estudar as classes de complexidade interessa recordar o que acontece à complexidade em tempo quando as máquinas usadas têm um número restrito de fitas. As seguintes proposições serão usadas nas seccões seguintes. A Proposição 3.1 é de Hartmanis e Stearns, no artigo fundador da complexidade computacional [HS65] e a Proposição 3.2 de Hennie e Streans [HS66].

Proposição 3.1 *Se $L \in \mathrm{DTIME}(T(n))$, então L é aceite em tempo $O(T^2(n))$ por uma máquina de Turing com uma fita.*

Dem. Pelo Exercício 2.1.2. □

Corolário 3.1 *Se $L \in \mathrm{NTIME}(T(n))$, então L é aceite por uma MTN com uma fita com complexidade em tempo $O(T^2(n))$.*

Proposição 3.2 *Se $L \in \mathrm{DTIME}(T(n))$, então L é aceite em tempo $O(T(n)\log T(n))$ por uma máquina de Turing com duas fitas.*

Dem. Seja M_1 um MT com k-fitas tal que $L(M_1) = L$ e M_1 operate em tempo limitado $T(n)$. Seja M_2 uma MT com duas fitas que vai simular M_1. A primeira fita de M_2 tem duas pistas para cada fita de M_1. Vamos considerar apenas o comportamento de duas pistas que correspondem a uma fita de M_1, sendo o comportamento idêntico para as restantes fitas. A segunda fita de M_2 será usada como auxiliar de computação. Uma célula particular da fita 1, B_0, vai conter sempre os símbolos que estão a ser visitados por cada uma das cabeças de M_1. Em vez de se mover os marcadores das cabeças, as fitas irão mover-se de modo a que B_0 tenha sempre o contéudo indicado. Para isso M_2 irá mover a cabeça da fita 1 em direção oposta à da cabeça de de M_1 que está a ser simulada. Assim, M_2 pode simular cada movimento de M_1 considerando só B_0. À direita da célula B_0 estão os blocos B_1, B_2, ... de tamanho exponencialmente crescente, isto é B_i tem tamanho 2^{i-1}. Do mesmo modo, à esquerda de B_0, os blocos B_{-1}, B_{-2}, ... com B_{-i} com tamanho 2^{i-1}. Supõe-se que existem marcadores entre os blocos. Seja a_0 o contéudo da célula visitada pela cabeça de M_1. Os contéudos à direita são $a_1, a_2, \ldots$ e os à esquerda $a_{-1}, a_{-2}, \ldots$. Os valores dos a_i podem mudar quando entrarem em B_0 mas o importante são as suas posições nas pistas da fita 1 de M_2. Inicialmente a pista superior de M_2 para respectiva fita de M_1

supõe-se vazia enquanto a inferior ten $\ldots, a_{-2}, a_{-1}, a_0, a_1, a_2, \ldots$. Estes estão colocados nos blocos $B_-, B_{-1}, B_0, B_1, B_2, \ldots$do seguinte modo: em B_{-3} está $a_{-7}, a_{-6}, a_{-5}, a_{-4}$, em B_{-2} está a_{-3}, a_{-2}, em B_{-1} está a_{-1}, em B_0 está a_0, em B_1 está a_1, em B_2 está a_2, a_3, em B_3 está a_4, a_5, a_6, a_7, etc.

Os dados serão deslocados em relação a B_0 e eventualmente alterados.

Após a simulação de um passo de M_1 será sempre verdade que:

1. Para $i > 0$, ou B_i está cheio (ambas as pistas) e B_{-i} está vazio; ou vice-versa; ou as pistas inferiores de B_i e B_{-i} estão cheias enquanto as superiores estão vazias (isto é, *meio-cheias*)
2. Os conteúdos de B_i ou B_{-i} representam células consecutivas na fita de M_1. Para $i > 0$, a pista superior representa as células à esquerda das da pista inferior. Para $i < 0$, a pista superior representa as células à direita das da pista inferior.
3. Para $i < j$, B_i representa células à esquerda das de B_j
4. B_0 tem só a sua pista inferior preenchida e a sua pista superior está marcada de modo especial

Vamos descrever agora como é simulado um passo de computação de M_1. A isso chama-se uma

Operação B_i. Suponhamos que a fita de M_1 se desloca para a esquerda. Então M_2 vai deslocar os dados correspondentes para a direita. Para isso, M_2 move a cabeça da fita 1 de B_0,onde se encontra sempre antes de um movimento, para a direita até que encontra o primeiro bloco B_i não totalmente cheio. Então, M_2 copia todos os dados de B_0, B_1, $\ldots B_{i-1}$ para a fita 2 e depois guarda-os nas pistas inferiores de $B_1, B_2, \ldots, B_{i-1}$ e ainda na pista inferior de B_i (caso não esteja cheia). Caso a pista inferior de B_i esteja cheia, a pista superior é usada. Em ambos os casos, os dados ocupam completamente os blocos referidos. Notando que $1 + \sum_{l=0}^{i-2} 2^l = 2^{i-1}$, cada conjunto de dados pode ser guardado em tempo proporcional ao tamanho de B_i. Em seguida, em tempo também proporcional ao comprimento de B_i, M_2 pode encontrar B_{-i} (usando a fita 2 para calcular a distância entre B_0 e B_i). Se B_{-i} está completamente cheio M_2 coloca os dados da pista superior de B_{-i} na fita 2. Se B_{-i} está meio-cheio a pista inferior é copiada para a fita 2. O que foi copiado, é colocado nas pistas inferiores de $B_{i-1}, B_{i-2}, \ldots B_0$, que pela regra 1. devem estar vazios. Mais uma vez os dados, têm o espaço exacto para ocupar e podem ser guardados de modo a satisfazer as regras 1., 2. e 3.

Se a cabeça de M_1 se movimentasse para a direita a operação era análoga. Para cada fita de M_1, M_2 tem de executar uma operação B_i no máximo uma vez em cada 2^{i-1} movimentos

de M_1. Isto porque esse é o tempo que demora a encher $B_1, B_2, \ldots, B_{i-1}$ que estão meio-cheios depois de uma operação B_i. Uma operação B_i é executada nunca antes do 2^{i-1}-ésimo movimento de M_1 (isto é, só $\frac{T(n)}{2^{i-1}}$ dos movimentos podem ser correspondentes a B_i).

Assim se M_1 opera em tempo $T(n)$, M_2 só executará operações B_i para $i \leq \log T(n) + 1$. Supondo que m é tal que uma operação B_i leva $m2^i$, se M_1 executa em $T(n)$ então M_2 executa em

$$T_{M_2}(n) = \sum_{i=1}^{\log T(n)+1} m2^i \frac{T(n)}{2^{i-1}} = 2mT(n)\log T(n) = O(T(n)\log T(n)).$$

□

Corolário 3.2 *Se $L \in \text{NTIME}(f(n))$, então L é aceite por uma MTN com duas fitas em tempo $O(f(n)\log f(n))$.*

3.2 Complexidade em Espaço

Seja M uma máquina de Turing multi-fita com uma fita de leitura, uma de escrita e várias leitura/escrita de que pára para todos os dados. O *espaço requerido* por M com dados x, $S_M(x)$ é o número de células das suas fitas de leitura/escrita visitadas durante a computação de M para dados x.

A *complexidade em espaço* de M é a função

$$S_M(n) = \max\{s \mid \exists x, |x| = n \text{ e } s = S_M(x)\}.$$

Como no caso da complexidade temporal podemos ignorar factores multiplicativos e aditivos.

Teorema 3.1 *Se L é uma linguagem aceite por uma máquina com k fitas em espaço limitado por $S(n)$ então para qualquer $c > 0$ é aceite por uma máquina com k fitas em espaço limitado por $cS(n)$.*

Dem. Seja M_1 uma MT que aceita L e opera em espaço limitado por $S(n)$. Podemos construir uma MT M_2 que simula M_1 onde para algum $r > 0$ cada célula de M_2 guarda um símbolo representando o conteúdo de r células adjacentes de M_1. Durante a simulação, o controlo finito de M_2 recorda quais as células de M_1 que estão a ser visitadas. Seja r tal que $rc \geq 2$. M_2 simula M_1 usando no máximo $\lceil S(n)/r \rceil$ células. Sendo $S(n) \geq r$ o número de células visitadas não é mais que $cS(n)$. Se $S(n) < r$ então M_2 pode guardar numa célula o conteúdo de qualquer fita e portanto M_2 só usa uma célula. □

Se $S_M(n)$ é $O(f(n))$, diz-se que M opera em *espaço limitado* por $f(n)$. Em geral supomos que $f(n) \geq 1$ e que $f(n)$ é *espaço-construível*, isto é, que existe uma máquina de Turing que calcula $f(n)$ em espaço limitado por $f(n)$, sendo n o comprimento dos dados x.

A classe de complexidade associada é:

$$\text{DSPACE}(f(n)) \quad = \quad \{L \mid L \text{ é decidível por uma MT multi-fita } M \text{ tal que } S_M(n) \text{ é } O(f(n))\}$$

Definimos

$$\begin{aligned} \mathsf{L} &= \text{DSPACE}(\log n) \\ \mathsf{PSPACE} &= \bigcup_{k\geq 1} \text{DSPACE}(n^k) \\ \mathsf{EXPSPACE} &= \bigcup_{k\geq 1} \text{DSPACE}(2^{n^k}) \end{aligned}$$

Sendo N uma máquina de Turing não-determinística o espaço requerido por M com dados x, $S_N(x)$ não excede o número de células visitadas num qualquer caminho de computação de N. E a *complexidade em espaço* é

$$S_N(n) \quad = \quad max\{\{m \mid \exists\, x \in L(N),\ |x| = n \text{ e } m = S_N(x)\}\}.$$

Analogamente temos,

Corolário 3.3 *Se L é aceite por uma MTN em espaço limitado por $S(n)$, então L é aceite por uma MTN em espaço limitado por $cS(n)$ onde $c > 0$ é uma qualquer constante.*

E,

$$\text{NSPACE}(f(n)) \quad = \quad \{L \mid L \text{ é decidível por uma MTN multi-fita } N \text{ tal que } S_N(n) \text{ é } O(f(n))\}.$$

Temos, também $\text{DSPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$. Definimos,

$$\begin{aligned} \mathsf{NPSPACE} &= \bigcup_{k\geq 1} \text{NSPACE}(n^k) \\ \mathsf{NL} &= \text{NSPACE}(\log n) \end{aligned}$$

Neste caso, o número de fitas de leitura/escrita não é importante.

Proposição 3.3 *Se uma linguagem L é aceite por uma MT com k-fitas de leitura/escrita em espaço limitado por $S(n)$, então é aceite por uma MT de uma fita de leitura/escrita em espaço limitado por $S(n)$.*

Dem. Seja M_1 uma MT com k fitas de leitura/escrita e tal que $L(M_1) = L$ e $S_{M_1}(n) = S(n)$. Podemos construir uma máquina M_2 com uma fita de leitura-escrita que simula as fitas de M_1 em k-pistas. A técnica usada para simular uma MT multi-fita por uma MT de uma fita garante que M_2 não usa mais de $S(n)$ células (Exercício 2.1.2). □

3.3 Relações entre complexidade em tempo e complexidade em espaço

Podemos ter as seguintes relações entre classes de complexidade em espaço e em tempo.

Proposição 3.4 *Seja* $S(n) \geq \log n$. *Então*

a) $\text{DTIME}(T(n)) \subseteq \text{DSPACE}(T(n))$

b) $\text{NTIME}(T(n)) \subseteq \text{NSPACE}(T(n))$

c) $\text{DSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$

d) $\text{NSPACE}(S(n)) \subseteq \text{NTIME}(2^{O(S(n))})$

Dem. Uma MT pode visitar no máximo uma nova célula em cada passo, pelo que o espaço usado não pode ser maior que o tempo de execução. Assim , temos a) e b). Para a mostramos como uma MT $M = (S, \Sigma, \Gamma, s_0, \square, \triangleright, \delta, F)$ com espaço limitado $S(n)$ pode ser modificada para parar depois de $O(2^{S(n)})$ passos e reconhecendo a mesma linguagem. Suponhamos que M tem uma fita de leitura e uma fita de leitura/escrita (já vimos na Proposição 3.3 que múltiplas fitas podem ser simuladas em pistas de uma fita sem perda de espaço, aumentando o tamanho do alfabeto). Supondo que $|\Gamma| = d$ e $|S| = q$. Então, com dados de tamanho n, o número de configurações possíveis é no máximo $qnS(n)d^{S(n)}$: q estados, n posições da cabeça na fita de leitura, $S(n)$ células visitadas (posições da cabeça da fita de leitura/escrita) e $d^{S(n)}$ possíveis conteúdos da fita. Como $S(n) \geq \log n$, este número é no máximo $c^{S(n)}$ para uma constante c suficientemente grande. Qualquer caminho de computação de tamanho maior que este valor, terá de repetir uma configuração, pelo que haverá um caminho menor com o mesmo resultado (aceitação/rejeição). Se existir um caminho de aceitação então existe um de comprimento no máximo $c^{S(n)}$. Podemos contar até $c^{S(n)}$ numa pista separada da fita de leitura/escrita e rejeitar se a computação computação simulada não tiver parado. Esta tarefa não gasta mais espaço se a contagem for feita em base c e o tempo extra é $O(S(n))$ por cada passo simulado da máquina inicial. Temos no total o tempo de $O(S(n)c^{S(n)}) = 2^{O(S(n))}$. Do mesmo modo de demonstra d). □

A demonstração do teorema anterior, resume-se ao seguinte problema. REACHABILITY

Instância: Dado um digrafo $G = (V, E)$, $x, y \in V$ e $K \geq 0$

Questão: Existe uma caminho de x para y de tamanho menor ou igual a K.

É fácil ver que REACHABILITY está em P. No teorema anterior, consideramos o grafo de configurações $G_{M,x} = (C, E)$ onde $|C| = O(c^{S(n)})$, C é o conjunto de todas as configurações e $(\alpha_1, \alpha_2) \in E$, se $\alpha_1 \xrightarrow{M} \alpha_2$.

Pelo Teorema 2.2, $\text{NTIME}(f(n)) \subseteq \text{DTIME}(2^{O(f(n))})$. Temos ainda

Proposição 3.5 *Seja $S(n) \geq \log n$ e espaço-construível. Então*

a) $\text{NTIME}(T(n)) \subseteq \text{DSPACE}(T(n))$

b) $\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$

Dem. Seja N uma MTN com tempo limitado por $T(n)$, i.e $T_N(n) = T(n)$. A nova máquina deterministica MT, M gera sequências de escolhas de N, de inteiros entre 0 e $d-1$ e tamanho $T(n)$. Isto é, para dados x de tamanho n, a podemos fazer uma pesquisa em profundidade na árvore de computação de N, construindo a árvore em cada passo. A máquina M, aceita se for atingida uma configuração de aceitação. Poderia parecer que era necessário espaço $O(T^2(n))$ dado cada configuração necessitar no máximo de $T(n)$ células e a profundidade da árvore ser $T(n)$. No entanto, sendo d o número máximo de escolhas em cada passo, basta guardar uma string d-ária que codifica o caminho desde uma configuração inicial e a configuração corrente. Com essa informação, é possível reconstruir a configuração corrente em espaço $T(n)$ começando pela configuração inicial e simulando a computação de N usando a sequência de escolhas.

Para a segunda parte, sendo $S(n)$ espaço-construível podemos marcar $S(n)$ células. Em seguida, escrevem-se todas as configurações da máquina N que usam não mais de $S(n)$ células. Como visto na Proposição 3.4, o número de configurações é majorado por $c^{S(n)}$, para alguma constante c, e podemos escrever todas as configurações no máximo em $S(n)c^{S(n)}$ passos. Finalmente, podemos marcar todas configurações que são atingíveis da configuração inicial, aceitando se alguma vez marcarmos uma configuração de aceitação. O limite de tempo deste procedimento é $m^{S(n)}$ para m suficientemente grande. □

Vimos já que uma das questões mais importantes na complexidade computational é saber se P = NP. Para a complexidade temporal essa questão foi resolvida em 1970 por Walter Savitch.

Teorema 3.2 (Teorema de Savitch) *Seja $S(n) \geq \log n$. Então*

$$\text{NSPACE}(S(n)) = \text{DSPACE}(S(n)^2).$$

Dem. Seja $M = (S, \Gamma, s_0, \square, \triangleright, \delta, F)$ uma MTN com $S_M(n) = S(n)$. Pela Proposição 3.4, também se tem, $T_M(n) = d^{S(n)}$ para uma constante d suficientemente grande. Uma configuração de M é da forma $\alpha = (s, x_1, y_1, \ldots, x_k, y_k)$ onde $s \in S$ e $x_i y_i \in \Gamma^\star$ o conteúdo da fita i com a cabeça a visitar o símbolo mais à esquerda de y_i. Seja Δ o alfabeto alargado tal que $d = |\Delta|$ e

que para dados x de tamanho n, $\alpha \in \Delta^{S(n)}$. Então, o número de configurações é no máximo $d^{S(n)}$. Para $\alpha, \beta \in \Delta^{S(n)}$, $\alpha \xrightarrow{M}^{\leq k} \beta$ se α e β são configurações legais de M e existem no máximo k movimentos que levam de α a β, de acordo com a relação de transição δ e sem exceder o limite de espaço $S(n)$.

A máquina determinística M_1 codifica uma rotina $\mathsf{SAV}(\alpha, \beta, k)$ e determina se $\alpha \xrightarrow{M}^{\leq k} \beta$. Já vimos, na Proposição 3.4 e pelo Princípio de Dirichlet, que se para algum k', $\alpha \xrightarrow{M}^{\leq k'} \beta$, então existe $k \leq d^{S(n)}$ tal que $\alpha \xrightarrow{M}^{\leq k} \beta$, pelo que nos podemos restringir a inteiros não negativos nesse intervalo. Escrever k na base d requer então espaço limitado por $S(n)$. Para determinar se M aceita é suficiente monstar que $\texttt{start} \xrightarrow{M}^{\leq k} \texttt{accept}$, onde `start` é a configuração inicial e `accept` uma configuração de aceitação (que podemos supor única). A MT M_1 constrói $S(n)$ e depois chama $\mathsf{SAV}(\texttt{start}, \texttt{accept}, d^{S(n)})$. A subrotina é a seguinte

procedure SAV(α, β, k)
  if $k = 0$ **then**
    if $\alpha = \beta$ **then**
      return *True*
    else
      return *False*
    end if
  else if k=1 **then**
    if $\alpha \xrightarrow{M} \beta$ **then**
      return *True*
    else
      return *False*
    end if
  else if $k \geq 2$ **then**
    for all $\gamma \in \Delta^{S(n)}$ **do** ▷ Percorrido por ordem lexicográfica
      if $\mathsf{SAV}(\alpha, \gamma, \lceil \frac{k}{2} \rceil)$ e $\mathsf{SAV}(\gamma, \beta, \lfloor \frac{k}{2} \rfloor)$ **then**
        return *True*
      end if
    end for
  else
    return *False*
  end if
end procedure

Podemos mostrar por indução em k que $\mathsf{SAV}(\alpha, \beta, k)$ retorna *True* se e só se $\alpha \xrightarrow{M}^{\leq k} \beta$. Cada instanciação de SAV requere uma pilha de activação de tamanho $S(n)$ para guardar α, β, k e

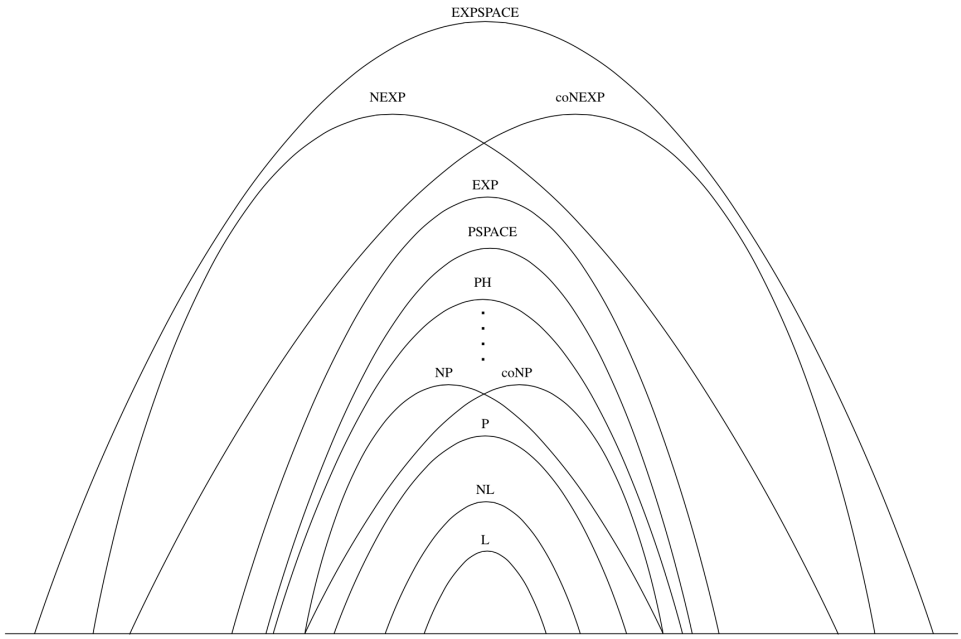


Figura 3.1: Inclusão de Classes.

a variável local γ. Esta informação pode ser guardada na fita em espaço limitado por $S(n)$. A profundidade da recursão (numero de pilhas de activação) é $\log_2 d^{S(n)} = O(S(n))$ (porque em cada chamada recursiva o valor de k é dividido por 2). Portanto o espaço total necessário é $O(S(n)^2)$. $\square$

Corolário 3.4 $\mathsf{PSPACE} = \mathsf{NPSPACE}$.

Temos também que NSPACE($\log n$) $\subseteq$ DSPACE($\log^2 n$) e p.e NSPACE(2^n) $\subseteq$ DSPACE(4^n).

Sabemos que

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP},$$

e que $\mathsf{L} \subsetneq \mathsf{PSPACE}$ e $\mathsf{P} \subsetneq \mathsf{EXP}$.

Para separar classes de complexidade, a técnica utilizada a seguir é a diagonalização, usada para mostrar que existem linguagens indecidíveis. Nestes casos, por uma raciocínio semelhante iremos ver que não poderão existir máquinas que decidam certas linguagens e com uma determinada complexidade.

3.4 Teoremas de Separação e Hierarquias de Complexidade

Será que existe $f(n)$ tal que qualquer linguagem recursiva L pertence a DTIME($f(n)$) ou DSPACE($f(n)$)? Iremos ver que não, e se $f(n)$ for construível pequenas quantidades de tempo ou espaço extra permitem decidir mais linguagens.

Proposição 3.6 *Dada uma função recursiva total $T(n)$ existe uma linguagem recursiva L tal que $L \notin \text{DTIME}(T(n))$ ou $L \notin \text{DSPACE}(T(n))$, respectivamente.*

Dem. Iremos considerar apenas para o tempo, sendo análogo para o espaço. O argumento é baseado em diagonalização. Sendo $T(n)$ recursiva total existe uma MT M que calcula $T(n)$ e para para todos os dados. Vamos construir M' que aceita uma linguagem em $\{0,1\}^\star$ que é recursiva mas não pertence a $\text{DTIME}(T(n))$. Seja $x_i \in \{0,1\}^\star$ a i-ésima string, na ordenação lexicográfica. Podemos ordenar todas as MT's com k-fitas num dado alfabeto, considerando as suas funções de transição como strings binárias. Para isso podemos numerar os estados (0 a $|S|$) e o alfabeto das fitas supondo que o alfabeto de entrada é $\{0,1\}$ (0,1,$\square$, $\triangleright$, X_5, ..., X_m). A codificação em binário pode ser $0 \mapsto 0$, $1 \mapsto 00$, $\square \mapsto 000$, $\triangleright \mapsto 0000$, e $X_i \mapsto 0^i$ para $i \geq 5$). Tendo uma codificação para M podemos ter codificações arbitrariamente grandes delas considerando um número arbitrário de 1's antes da codificação. Podemos então referirmo-nos a M_i como a i-ésima MT multifita. Seja

$$L = \{x_i \mid M_i \text{ não aceita } x_i \text{ em } T(|x_i|) \text{ passos}\}$$

A linguagem L é recursiva porque pode ser reconhecida por uma MT que implemente o seguinte algoritmo: com dados w de tamanho n, simula M com dados n e calcula $T(n)$. Depois determina i tal que $w = x_i$. O inteiro i em binário é a codificação de uma MT multifita M_i (supomos que se i não corresponde a nenhuma codificação então M_i é uma MT que não se move). Simulamos M_i com dados w durante $T(n)$ passos, aceitando se M_i ou parar sem aceitar ou executa mais do que $T(n)$ passos ou não aceita. Podemos então concluir que L é recursiva. Para ver que $L \notin \text{DTIME}(T(n))$ suponhamos que $L = L(M_i)$ e $T_{M_i}(n) = T(n)$. Será que $x_i \in L$? Se sim, M_i aceita x_i em $T(n)$ passos com $n = |x_i|$. Mas por definição de L $x_i \notin L$, o que é uma contradição. Se $x_i \notin L$ então M_i não aceita x_i, e por definição $x_i \in L$. Nova contradição. Ambas as suposições levam a uma contradição pelo que $T_{M_i}(n) = T(n)$ tem de ser falso. $\square$

Por definição temos que se $f'(n) \geq f(n)$, para todo n, então $\text{DTIME}(f(n)) \subseteq \text{DTIME}(f'(n))$. Se $f(n)$ é recursiva total, pela Proposição 3.6 sabemos que existe L linguagem recursiva que não pertence a $\text{DTIME}(f(n))$. Então irá pertencer a $\text{DTIME}(f'(n))$ para algum $f'(n) \geq f(n)$. Então

$$\text{DTIME}(f(n)) \subsetneq \text{DTIME}(f'(n))$$

Temos então uma hierarquia de classes de complexidade determinísticas em tempo. E o mesmo acontece para a complexidade em espaço e para as classes não-determinísticas.

Vamos ver que para certas funções basta um pequeno incremento na taxa de crescimento.

Teorema 3.3 ([SHL65]) *Seja $S(n)$ espaço-construível. Existe $L \in \text{DSPACE}(S(n))$ tal que $L \notin \text{DSPACE}(S'(n))$ para qualquer $S'(n) = o(S(n))$.*

Dem. A demonstração é por diagonalização. Seja $M_0, M_1, \ldots$ uma enumeração de MT's com alfabeto de entrada $\{0,1\}$, baseada numa codificação binária, mas em que é permitido que um qualquer prefixo de 1's para admitir codificações arbitrariamente longas. Temos então que a representação binária de i corresponde a uma máquina M_i e caso essa representação não possa corresponder a uma MT, M_i é uma máquina que para imediatamente com quaisquer dados. Construímos uma máquina M que usa $S(n)$ espaço para qualquer dados x de tamanho n e não coincide pelo menos num x com qualquer MT com complexidade em espaço $S'(n)$. Com dados x de tamanho n, M faz o seguinte:

1. Marca $S(n)$ células na fita: isso pode ser feito porque S é espaço-construível.
2. Simula M_i com dados x, onde a representação em binário de i é x, nunca excedendo o espaço $S(n)$.

Um dos seguintes casos ocorre:

i) Se existe espaço suficiente para completar a simulação e M_i parar, então M faz o oposto: se M_i aceitar, rejeita e vice-versa.

ii) Se M_i não parar então M não para.

iii) Se a simulação requerer mais de $S(n)$ espaço, M para e rejeita.

Temos que $L(M) \in \text{DSPACE}(S(n))$. Todas as MT's são simuladas para alguma string de entrada de tamanho n, para n suficientemente grande. Se $S_{M_i}(n) = o(S(n))$, então para dados x suficientemente grandes tal que x é a representação em binário de i, M simula M_i com dados x e a simulação terá espaço suficiente para terminar. Mesmo que o alfabeto de fita de M_i seja muito grande, por exemplo $t = |\Gamma_i|$, a simulação requere $\lceil \log t \rceil S_{M_i}(n)$. Mas para x suficientemente grande temos $\lceil \log t \rceil S_{M_i}(n) \leq S(n)$. Mas M e M_i diferem com dados x, logo $L(M) \neq L(M_i)$. Pela Proposição 3.4c), podemos supor que M_i pára sempre. Concluímos que M difere em algum valor de entrada de todas as MT's em $o(S(n))$. $\square$

Podemos então concluir que se $S'(n) = o(S(n))$ e $S'(n) \leq S(n)$, para todo o n, então

$$\text{DSPACE}(S'(n)) \subsetneq \text{DSPACE}(S(n)).$$

Contudo se $S'(n) > S(n)$ é possível que $\text{DSPACE}(S'(n))$ e $\text{DSPACE}(S(n))$ tenham ambas linguagens que não estão na outra.

Para a complexidade em tempo um factor extra de $\log T(n)$ é necessário para a simulação.

Teorema 3.4 ([HS65]) *Seja $T(n)$ uma função tempo-construível, em particular $T(n) \geq n$. Existe $L \in \text{DTIME}(T(n))$ tal que $L \notin \text{DTIME}(T'(n))$ para nenhum $T'(n)$ tal que $T'(n) \log T'(n) = o(T(n))$.*

Dem. Seguindo a demonstração do Teorema 3.3, construímos uma MT M tal que $T_M(n) = T(n)$ e que irá simular qualquer MT para algum valor dos dados. M considera os dados de entrada x como a codificação de uma MT M_i onde x é a representação binária de i. M simula M_i com dados x. Pode acontecer que M_i tenha mais fitas que M. Pela Proposição 3.2, apenas duas fitas são necessárias para simular qualquer M_i embora a simulação implique um custo multiplicativo de $\log T_{M_i}(n)$. Do mesmo modo M_i pode ter um alfabeto de fita grande que terá se ser codificado num número fixo de símbolos, então a simulação de $T_{M_i}(n) = T'(n)$ passos de M_i pode requerer para M um tempo $cT'(n) \log T'(n)$, onde c é uma constante que depende de M_i. Para garantir que a simulação de M_i é feita em tempo limitado por $T(n)$, M executa em simultâneo os passos de uma MT que usa exactamente $T(n)$ passos em todos os dados de tamanho n. Após $T(n)$ passos, M para. M aceita x só se a simulação de M_i está completa e M_i rejeita x. A codificação de M_i é como no Teorema 3.3, portanto M_i admite codificações arbitrariamente grandes. Assim, se $T_{M_i}(n) = T'(n)$, existe uma string x suficientemente grande para que

$$cT'(|x|) \log T'(|x|) \leq T(n)$$

e a simulação será completa. Neste caso, $x \in L(M)$ se e só se $x \notin L(M_i)$. Então $L(M) \neq L(M_i)$, para toda a M_i tal que $L(M_i) \in \text{DTIME}(T'(n))$. Assim, $L(M) \in \text{DTIME}(T(n)) \setminus \text{DTIME}(T'(n))$, com $T'(n) \log T'(n) = o(T(n))$. □

Corolário 3.5

1. *Para todo o* $k \geq 1$, $\text{DTIME}(n^k) \subsetneq \text{DTIME}(n^{k+1})$.
2. $\mathsf{P} \subsetneq \mathsf{EXP}$.
3. $\mathsf{L} \subsetneq \mathsf{PSPACE}$.

No entanto existe um *gap* em qualquer hierarquia de complexidade: existe uma função $T(n)$ tal que $\text{DTIME}(T(n)) = \text{DTIME}(2^{T(n)})$. Isto é, $T(n)$ é definida de modo a que nenhuma MT com dados de tamanho n para entre $T(n)$ e $2^{T(n)}$ passos: ou para até $T(n)$ ou para depois de $2^{T(n)}$ passos.

Para as classes de complexidade não-determinísticas a técnica de *padding* permite obter resultados de separação. Vamos ilustrar mostrando que

$$\text{NSPACE}(n^3) \subsetneq \text{NSPACE}(n^4).$$

Suponhamos por contradição que $\text{NSPACE}(n^4) \subseteq \text{NSPACE}(n^3)$. Iremos ver que então $\text{NSPACE}(n^5) \subseteq \text{NSPACE}(n^4)$. Seja M uma MTN que executa em espaço n^5 e $A = L(M)$. Considere-se a linguagem:

$$A' = \{x\#^{|x|^{\frac{5}{4}} - |x|} \mid x \in A\},$$

onde $\#$ não ocorre no alfabeto de M. Seja M' uma MTN que aceita dados da forma $x\#^m$ e

(i) verifica se $m = |x|^{\frac{5}{4}} - |x|$;

(ii) se sim, executa M com dados x, ignorando os #s.

Como M executa em espaço limitado por $|x|^5$ com dados x, M' executa em espaço limitado por:

$$|x|^5 = (|x|^{\frac{5}{4}})^4 = |x\#^{|x|^{\frac{5}{4}}-|x|}|^4,$$

e $A' = L(M') \in \text{NSPACE}(n^4)$. Mas então $A' \in \text{NSPACE}(n^3)$ e seja M'' uma MTN com $S_{M''}(n) = n^3$ e $A' = L(M'')$. Podemos construir uma nova MTN M''' para A que com dados x

(i) concatena a x uma sequência de #s de comprimento $|x|^{\frac{5}{4}} - |x|$;

(ii) executa M'' com a string resultante

Então com dados x, M''' executa em espaço limitado por

$$|x\#^{|x|^{\frac{5}{4}}-|x|}|^3 = (|x|^{\frac{5}{4}})^3 = |x|^{\frac{15}{3}} \leq |x|^4$$

e $A = L(M''')$. Então, podemos concluir que $\text{NSPACE}(n^5) \subseteq \text{NSPACE}(n^4)$. Repetindo o raciocínio, podemos concluir que

$$\begin{aligned} \text{NSPACE}(n^6) &\subseteq \text{NSPACE}(n^5) \\ \text{NSPACE}(n^7) &\subseteq \text{NSPACE}(n^6). \end{aligned}$$

Combinando estas inclusões temos $\text{NSPACE}(n^7) \subseteq \text{NSPACE}(n^3)$. Mas, então

$$\begin{aligned} \text{NSPACE}(n^7) &\subseteq \text{NSPACE}(n^3) \\ &\subseteq \text{DSPACE}(n^6) \text{ pelo asT. Savitch} \\ &\subsetneq \text{DSPACE}(n^7) \\ &\subseteq \text{NSPACE}(n^7), \end{aligned}$$

O que é uma contradição. Logo, $\text{NSPACE}(n^3) \subsetneq \text{NSPACE}(n^4)$. Em geral pode-se mostrar o seguinte teorema,

Teorema 3.5 ([Iba72], [Coo73])

1. *Para todo* $c > 0, r \geq 0$, $\text{NSPACE}(n^r) \subsetneq \text{NSPACE}(n^{r+c})$.
2. *Se* $T(n+1) = o(T'(n))$, *então* $\text{NTIME}(T(n)) \subsetneq \text{NTIME}(T'(n))$.

Podemos concluir que $\mathsf{NP} \subsetneq \mathsf{NEXP}$.

Uma máquina de Turing não determinística calcula uma função $F : \Sigma^\star \longrightarrow \Sigma^\star$, se com dados x cada computação ao retorna a resposta correct $F(x)$ ou termina num estado s_n. Pelo menos uma computação tem de calcular $F(x)$ e se houverem outras computações calculem um valor esse valor tem de ser $F(x)$.

Teorema 3.6 (Immerman-Szelepcsényi)
Para $S(n) \geq \log n$, $\mathrm{NSPACE}(S(n)) = \mathsf{coNSPACE}(S(n))$.

Dem. Suponhamos $S(n)$ espaço-construível. A demonstração baseia-se na ideia seguinte. Suponhamos $L \subseteq \Sigma^\star$ finita e pretende-se um teste não determinístico para $x \in L$. Seja $l = |L|$, conhecido. Então podemos testar se $y \not\in L$: Dado $y \in \Sigma^\star$, adivinhar sucessivamente l elementos distintos e verificando que são todos diferentes de y e em L. Se o teste suceder, $y \notin L$. Nota que o teste pode falhar para algumas "tentativas"mas isso é a essencia do não determinismo.

Seja M uma MTN com espaço limitado por $S(n)$. Vamos construir uma MTN N que aceita o complementar de $L(M)$. Suponhamos que temos uma codificação de M usando um alfabeto Δ, com $d = |\Delta|$ e tal que qualquer configuração com dados x de tamanho n é representado por uma string de $\Delta^{S(n)}$. Podemos supor que M quando quer aceitar, apaga as fitas, move as cabeças para a esquerda e entra num único estado de aceitação: isto garante que existe apenas uma configuração de aceitação $\texttt{accept} \in \Delta^{S(n)}$ para quaisquer dados de entrada de tamanho n. Seja $\texttt{start} \in \Delta^{S(n)}$ a configuração inicial para dados x, $|x| = n$: neste estado as cabeças estão à esquerda e a fita de trabalho vazia.

Se M aceita x, então existe um caminho de computação para a configuração de aceitação de tamanho no máximo $d^{S(n)}$. Para $m \geq 1$, seja

$$A_m = \{\alpha \in \Delta^{S(n)} \mid \texttt{start} \xrightarrow{M}{}^{\leq m} \alpha\}$$

com $A_0 = \{\texttt{start}\}$ e

$$M \text{ aceita } x \Leftrightarrow \texttt{accept} \in A_{d^{S(n)}}$$

A máquina N começa por marcar $S(n)$ células na fita de trabalho. Depois calcula sucessivamente $|A_0|$, $|A_1|$, ..., $|A_{d^{S(n)}}|$. Temos $|A_0| = 1$. Suponhamos que $|A_m|$ foi calculado e está escrito numa pista da fita de N. Como $|A_m| \leq d^{S(n)}$, ocupa no máximo $S(n)$ células. Para calcular $|A_{m+1}|$ escreve em ordem lexicográfica cada $\beta \in \Delta^{S(n)}$. Para cada um determina se $\beta \in A_{m+1}$: se sim incrementa o contador de uma unidade. O valor final do contador é $|A_{m+1}|$. Para testar se $\beta \in A_{m+1}$, adivinha não-deterministicamente os $|A_m|$ elementos de A_m por ordem lexicográfica; verifica se $\alpha \in A_m$ adivinhando um caminho de computação $\texttt{start} \xrightarrow{M}{}^{\leq m} \alpha$ e para cada desses α se $\alpha \xrightarrow{M}{}^{\leq 1} \beta$. Se algum α verificar isso então $\beta \in A_{m+1}$; se nenhum α o fizer então $\beta \notin A_{m+1}$. A computação não determinística garante que embora alguns ramos sejam de rejeição pelo menos um calcula o valor. Após $|A_{d^{S(n)}}|$ ter sido calculado, para testar se $\texttt{accept} \notin A_{d^{S(n)}}$ não deterministicamente, adivinha os $|A_{d^{S(n)}}|$ elementos de $A_{d^{S(n)}}$ em ordem lexicográfica verificando que cada α adivinhado está em $A_{d^{S(n)}}$ adivinhando um caminho de computação $\texttt{start} \xrightarrow{M}{}^{\leq d^{S(n)}} \alpha$ e verificando que cada α é diferente de $\texttt{accept}$.

A MTN N aceita o complemento de $L(M)$ e pode ser construída para executar em espaço limitado por $S(n)$.

□

Mais uma vez o teorema anterior corresponde a determinar a complexidade de um problema de grafos.

Proposição 3.7 *Dado um grafo $G = (V, E)$ e um nó x o número de nós atingíveis de x em G pode ser calculado por uma MTN em espaço limitado por* $\log n$.

Corolário 3.6 $\mathsf{NPSPACE} = \mathsf{coNPSPACE}$.

3.5 Limites do método de diagonalização

No essencial o método de diagonalização consiste na simulação de uma máquina de Turing por outra. A simulação é feita de tal modo que a máquina que simula pode determinar o comportamento da outra máquina e depois comporta-se de modo diferente. Suponhámos que dávamos a estas máquinas o mesmo oráculo. Assim sempre que a máquina simulada consulta o oráculo, o simulador também pode consultar e portanto a simulação pode continuar como inicialmente. Assim qualquer teorema provado usando diagonalização também será válido se as máquinas tiverem o mesmo oráculo.

Em particular, se $\mathsf{P} \neq \mathsf{NP}$ fosse demonstrado usando diagonalização também $\mathsf{P}^C \neq \mathsf{NP}^C$ para qualquer oráculo C.

O teorema seguinte mostra que existe A tal que $\mathsf{P}^A = \mathsf{NP}^A$ pelo que a diagonalização não pode separar as duas classes. Também não permite provar que $\mathsf{P} = \mathsf{NP}$ porque existe um oráculo B tal que $\mathsf{P}^B \neq \mathsf{NP}^B$.

Teorema 3.7

1. *Existe um oráculo recursivo A tal que* $\mathsf{P}^A = \mathsf{NP}^A$.
2. *Existe um oráculo recursivo B tal que* $\mathsf{P}^B \neq \mathsf{NP}^B$.

Dem. O oráculo A é fácil. Basta tomar A o problema QSAT (TQBF) ou outro qualquer problema PSPACE-compelto. Então

$$\mathsf{PSPACE} \subseteq \mathsf{P}^{\mathsf{QSAT}} \subseteq \mathsf{NP}^{\mathsf{QSAT}} \subseteq \mathsf{NPSPACE}^{\mathsf{QSAT}} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{PSPACE}$$

A primeira inclusão verifica-se porque uma linguagem em PSPACE pode-se reduzir polinomialmente a QSAT. A penúltima inclusão, porque em $\mathsf{NPSPACE}$ não necessitamos de consultar o oráculo (isto é, podemos calcular as respostas a QSAT). Conclui-se então que $\mathsf{NP}^{\mathsf{QSAT}} \subseteq \mathsf{P}^{\mathsf{QSAT}}$.

Vamos agora construir o oráculo B. Ideia: B é tal que existe $L_B \in \mathsf{NP}^B$ e garantidamente precisa de pesquisa de força bruta de tal forma que L_B não pode estar em P^B. A construção considera todas as máquinas polinomiais com oráculo e garante que nenhuma decide L_B.

Suponhamos que Σ é um alfabeto com pelo menos dois elementos. Seja $C \in \Sigma^\star$ um oráculo e seja

$$L_C = \{x \in \Sigma^\star \mid \exists y \in C, |y| = |x|\}$$

A linguagem L_C pode ser aceite por uma MTON N com oráculo C que opera do seguinte modo. Com dados x , adivinha uma string y com o mesmo comprimento que x e consulta o oráculo para determinar se $y \in C$. Independentemente da classe do oráculo, é um algoritmo não determinístico polinomial e portanto $L_C \in \mathsf{NP}^C$.

Por diagonalização construí-se um oráculo B tal que $L_B \neq O^B$ para toda a MTO polinomial com oráculo. Isto é $L_B \in \mathsf{NP}^B \setminus \mathsf{P}^B$. A ideia por trás da construção é que uma MTO M polinomial só tem tempo de fazer um número polinomial de perguntas ao oráculo mas existe um número exponencial de strings para cada comprimento n dos dados. Isto permite ajustar o oráculo para incluir ou retirar strings não consideradas por M o que leva a que esteja sempre errada. Seja $M_0, M_1, \ldots$ a lista de todas as MTO's limitadas em tempo polinomial. Suponhamos que cada M_i tem um relógio e um parâmetro c tal que M_i para após n^c passos. Então para cada i, o limite temporal n^c de M_i é reconhecível na descrição de M_i. Construímos um oráculo B como o limite duma sequência de aproximações finitas. Cada aproximação B_k é uma função parcial $f : \Sigma^\star \longrightarrow \{0, 1\}$ com domínio finito tal que

$$B_k(x) = \begin{cases} 1 & \text{se } x \in B \\ 0 & \text{se } x \notin B \\ \perp & \text{caso contrário} \end{cases}$$

Temos que $B_k \sqsubseteq B_{k+1}$ isto é B_{k+1} está definido sempre que B_k está e se ambos estão definidos têm o mesmo valor.

Na construção, o seguinte invariante verifica-se. Para cada B_k qualquer extensão total C de B_k e para $i \leq k$ existe um x tal que

$$M_i^C \text{ aceita } x \quad \Leftrightarrow \quad x \notin L_C \tag{3.1}$$

$$\Leftrightarrow \quad C \text{ não contem elementos de comprimento } |x| \tag{3.2}$$

No estágio 0 temos $B_0 = \perp$. Suponhamos que temos B_k. Consideramos M_k uma MTO tal que $T_{M_k}(n) = n^c$. Seja n maior que o comprimento de todos os elementos do domínio de B_k[1] e suficientemente grande para que $2^n > n^c$.

[1]Isto é possível porque B_k é finito.

- Inicialmente $B_{k+1} = B_k$.
- Simulamos M_k com dados de tamanho n, p.e a^n.
- Sempre que M_k se prepara para consultar o oráculo numa string y, se $B_{k+1}(y) \neq \perp$, devolve-se $B_{k+1}(y)$.
- Se $B_{k+1}(y) = \perp$, define-se $B_{k+1}(y) = 0$, e devolve-se o valor 0.
- A MTO M_k continua a execução e ou aceita ou rejeita

Ajustamos o oráculo de modo a que M_k não possa aceitar L_C. Como M_k corre em tempo $n^c \leq 2^n$, existe pelo menos uma string de tamanho n que nunca foi sujeita ao oráculo por M_k, e portanto B_{k+1} está ainda indefinido. Se M_k aceitou, define-se $B_{k+1}(y) = 0$, para todos os y de tamanho n tal que $B_{k+1}(y) = \perp$. Se M_k rejeita, $B_{k+1}(y) = 1$, Estes ajustes não afectam a computação de M_k com dados a^n, porque nunca foram sujeitos ao oráculo.

Então para qualquer função total C que extende B_{k+1} tem-se que **??** se verifica isto é se $a^n \in L(M_k^C)$ então $a^n \notin L_C$. E se $a^n \notin L(M_k)$ então $a^n \in L_C$. Então $L(M_k^C) \neq L_C$.

Se B for uma extensão total de todos os B_k, $k \geq 0$, então garantidamente $L(M_k^B) \neq L_B$ para qualquer M_k. $\square$

Capítulo 4

Alternância

Alternância generaliza não determinismo e é útil para estabelecer relações entre classes de complexidade e classificar problemas de acordo com a sua complexidade [CKS81]. Sendo outra medida de complexidade permite estabelecer relações entre as complexidades de espaço e de tempo. Relaciona-se também com conjuntos de fórmulas da lógicas e estratégias ganhadores em jogos de tabuleiro.

Num processo não determinístico existem pontos de escolha e segue-se um caminho de escolhas aceitando se existe um caminho que leva à aceitação. Isto pode ser visto como um sistema multiprocessador com um número ilimitado de processadores potenciais. A máquina começa num processo raiz na configuração inicial. Calcula como uma MT normal até chegar a um ponto de escolha não determinística. Nesse ponto bifurca em diversos processos independentes e espera a resposta de um dos processos. Cada processo continua a execução. E assim sucessivamente ao longo da árvore de computação. Se existirem m configurações à profundidade i então haverá m processos paralelos independentes executando ao mesmo tempo no instante i. Quando um processo entra num estado de aceitação envia um 1 ao processo pai e termina. Se rejeita, envia um 0 e também termina. O processo (pai) que está suspenso quando recebe um 1, envia o 1 ao seu pai e termina. Se receber um 0, continua à espera que outro subprocesso termine. Se todos os subprocessos terminam com 0, ele envia um 0 ao seu pai e termina. Os dados de entrada são aceites se 1 for enviado ao processo raiz. Isto este mecanismo corresponde à avaliação lógica de uma disjunção ($\vee$) das mensagens enviadas pelos subprocessos num dado nível (pelo menos um tem de enviar um 1). Assim podemos também considerar o caso em que para se aceitar é necessário que todos subprocessos levem à aceitação o que equivale a avaliar a conjunção ($\wedge$) de mensagens dos subprocessos (ver Figura 4.1). Uma configuração na árvore de computação é $\vee$ ou $\wedge$ consoante o estado é $\vee$ ou $\wedge$. O nome *alternância* tem haver com a alternância de nós $\vee$ e $\wedge$. Podemos mesmo incluir também nós $\neg$ que invertem o valor recebido.

Em termos de classes de complexidade iremos ver que tempo alternado é o mesmo que espaço determinístico e espaço alternado é exponencialmente maior que tempo determinístico

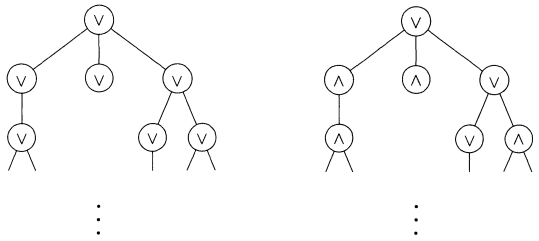


Figura 4.1: Não determinismo e alternância

4.1 Máquinas de Turing Alternadas

Uma máquina de Turing alternada (MTA) é definida como uma MTN $A = (S, \Sigma, \Gamma, s_0, \Delta, \Box, \triangleright, F)$ acrescentando uma função $t : S \longrightarrow \{\vee, \wedge, \neg\}$. Esta função caracteriza o tipo dum estado: estado-$\vee$, estado-$\wedge$ ou estado-$\neg$. Do mesmo modo classificam-se as configurações. Uma configuração-$\neg$ tem de ter exactamente um sucessor. Os estados s_y e s_n podem ser dispensados convencionando que os estados-$\wedge$ sem sucessores são de aceitação e os estados-$\vee$ sem sucessores são de rejeição.

A aceitação de uma MTA é definida em termos de uma etiquetagem indutiva da árvore de computação. Consideram-se duas ordens parciais no conjunto $\{0, 1, \perp\}$, onde $\perp$ representa a falta de informação ou não terminação (usada para computações infinitas):

1. $0 \leq \perp \leq 1$

2. $\perp \sqsubseteq 0$ e $\perp \sqsubseteq 1$ (ordem de informação)

As operações Booleanas estendem-se naturalmente a este conjunto:

$\vee$	1	$\perp$	0
1	1	1	1
$\perp$	1	$\perp$	$\perp$
0	1	$\perp$	0

$\wedge$	1	$\perp$	0
1	1	$\perp$	0
$\perp$	$\perp$	$\perp$	0
0	0	0	0

$\neg$	
1	0
$\perp$	$\perp$
0	0

Em relação à ordem 1, $\vee$ dá o supremo e $\wedge$ o ínfimo. Etiquetamos a árvore de computação com 0, 1 ou $\perp$. Seja C o conjunto de configurações e $\alpha \xrightarrow{A} \beta$, $\alpha, \beta \in C$. Seja $l : C \longrightarrow \{0, 1, \perp\}$ uma etiquetagem. A relação $\sqsubseteq$ estende-se ponto a ponto a etiquetagens ($l \sqsubseteq l' \Leftrightarrow \forall \alpha \in C, l(\alpha) \sqsubseteq l'(\alpha)$. Seja L o conjunto das etiquetagens. A estrutura $(L, \sqsubseteq)$ é uma ordem parcial completa (cpo) com elemento mínimo $\perp(\alpha) = \perp$, para todo $\alpha \in C$. Sabemos assim que todas as cadeias tem supremo e que funções monótonas têm pontos fixos mínimos.

Seja então $\tau : L \longrightarrow L$ a funcional

$$\tau(l)(\alpha) = \begin{cases} \wedge_{\alpha \xrightarrow{A} \beta} l(\beta), \text{se } \alpha \text{ é conf.-}\wedge \\ \vee_{\alpha \xrightarrow{A} \beta} l(\beta), \text{se } \alpha \text{ é conf.-}\vee \\ \neg l(\beta), \text{se } \alpha \text{ é conf.-}\neg \text{ e } \alpha \xrightarrow{A} \beta. \end{cases} \tag{4.1}$$

Então o ponto fixo mínimo existe e é denotado por l_*. É o limite da cadeia $l_0 \sqsubseteq l_1 \sqsubseteq l_2 \sqsubseteq \cdots$ onde $l_0 = \bot$ e $l_{i+1} = \tau(l_i)$.

Definição 4.1 (Aceitação duma MTA) *Uma MTA A aceita $x \in \Sigma^star$ se $l_*(\text{(start)}) = 1$ onde (start) é a configuração inicial com dados x. E rejeita se $l_*(\text{(start)}) = 0$.*

Lema 4.1 *Toda a MTA com negações pode ser simulada por uma MTA sem negações em custo extra de espaço e tempo.*

Dem. Dada uma MTA A seja A' a sua máquina dual (isto é se $t(q) = \wedge$ então $t'(q) = \vee$ e vice-versa. Pode-se mostrar por indução que $l_i(\alpha) = \neg l'_i(\alpha)$ e portanto $l_*(\alpha) = \neg l'_*(\alpha)$. Seja A'' uma MTA cujos estados são a união de disjunta de S e S'. $\square$

4.2 Classes de Complexidade Alternadas

A complexidade em tempo e em espaço para uma MTA é calculado da mesma maneira que para as MTN, considerando o tempo (ou espaço) máximo de qualquer caminho de computação.

As classes de complexidade são:

$$\begin{aligned} \text{ATIME}(f(n)) &= \{L \mid L \text{ é decidível por uma MTA } A \text{ tal que } T_A(n) \text{ é } O(f(n))\} \\ \text{ASPACE}(f(n)) &= \{L \mid L \text{ é decidível por uma MTA } A \text{ tal que } S_A(n) \text{ é } O(f(n))\} \end{aligned}$$

Temos

$$\begin{aligned} \mathsf{AL} &= \text{ASPACE}(\log n) \\ \mathsf{AP} &= \bigcup_{k \geq 1} \text{ATIME}(n^k) = \text{ATIME}(n^{O(1)}) \\ \mathsf{APSPACE} &= \bigcup_{k \geq 1} \text{ASPACE}(n^k) = \text{ASPACE}(n^{O(1)}) \\ \mathsf{AEXP} &= \bigcup_{k \geq 1} \text{ATIME}(2^{n^k}) = \text{ATIME}(2^{n^{O(1)}}) \end{aligned}$$

Exemplo 4.1 *O problema TAUT é* AP. *Para tal considera o seguinte algoritmo alternado. Seja E uma expressão Booleana,*

1. *Seleciona universalmente todas as valorizações de E*

2. *Para cada valorização, avalia E*

3. *Se E avaliar para $True$, aceitar. Senão, rejeitar.*

O primeiro passo gera um ramo para cada valorização e só aceita se todos os ramos aceitarem (computação $\wedge$). O segundo e terceiro verificam uma valorização particular.

Exercício 4.2.1 *Mostra que* $\mathsf{coNP} \subseteq \mathsf{AP}$.

Exemplo 4.2 *Considera o problema seguinte problema.*

(MinE)

Instância: *Uma expressão booleana E com literais num conjunto de variáveis U, constantes V e F e conectivas lógicas $\vee$, $\wedge$, $\neg$ e $\rightarrow$*

Questão: *E é minimal, isto é não existe uma expressão booleana E' com menos símbolos e tal que E' é equivalente a E?*

Seja E uma expressão Booleana e $K \in \mathbb{N}$.

1. *Seleciona universalmente (ramificação $\wedge$) todas as expressões booleanas menores que E.*

2. *Seleciona existencialmente (ramificação $\vee$) uma valorização para E*

3. *Avaliar E e E' para uma valorização.*

4. *Aceitar se as avaliações forem diferentes. Rejeitar se avaliam para o mesmo valor.*

Podemos concluir que MinE está em AP. Podemos também concluir que está em Π_2^{p}. Porquê?

Teorema 4.1 *Seja $T(n) \geq n$ e $S(n) \geq n$ (construíveis).*

i) $\mathrm{ATIME}(T(n)) \subseteq \mathrm{DSPACE}(T(n))$

ii) $\mathrm{DSPACE}(S(n)) \subseteq \mathrm{ATIME}(S(n)^2)$

iii) $\mathrm{ASPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))})$

iv) $\mathrm{DTIME}(T(n)) \subseteq \mathrm{ASPACE}(\log T(n))$

Dem.

i) Semelhante à da Proposição 3.5i). Fazer uma descida em profundidade da árvore de computação de uma MTA limitada em tempo por $T(n)$, calculando as etiquetas Booleanas $l_*(\alpha)$. A posição na árvore de computação da configuração corrente pode ser representada por uma string binária de tamanho no máximo $T(n)$.

ii) É uma versão alternada (paralela) do Teorema de Savitch. Podemos até considerar um resultado mais forte NSPACE($S(n)$) $\subseteq$ ATIME($S(n)^2$). Seja M uma MTN com espaço limitado por $S(n)$ e $\mathsf{PARSAV}(\alpha, \beta, k)$ uma rotina paralela recursiva que determina se $\alpha \xrightarrow{M}^{\leq k} \beta$. Se $k = 0$ verifica se $\alpha = \beta$ e se $k = 1$ verifica se existe uma transição tal que $\alpha \xrightarrow{M} \beta$. Se $k \geq 2$, adivinha não deterministicamente, usando ramificação $\vee$ um $\gamma \in \Delta^{S(n)}$. Isto pode ser feito em tempo $S(n)$ e são gerados $2^{S(n)}$ processos independentes e paralelos, cada um com γ diferente. O processo para um dado γ verifica em paralelo se $\mathsf{PARSAV}(\alpha, \gamma, \lceil \frac{k}{2} \rceil)$ e $\mathsf{PARSAV}(\alpha, \gamma, \lfloor \frac{k}{2} \rfloor)$ usando ramificação $\wedge$. Pode-se verificar que este processo alternado pode ser implementado em tempo $S(n)^2$.

iii) Seja A uma MTA com $S_A(n) = S(n)$. Construir uma MT M'. Podemos escrever na fita todas as configurações que gastam no máximo espaço $S(n)$ (existem $2^{S(n)}$) e calcular l_* indutivamente. Começamos por etiquetar todas as configurações com $\perp$. Isto define a etiquetagem l_0. Suponhamos que já calculamos l_i. Percorre-se a fita calculando $\tau(l_i) = l_{i+1}$. Isto demora $2^{S(n)}$. Quando não houver mudanças atingimos o ponto fixo de τ, logo l_*. Há no máximo $2^{O(S(n))}$ passagens todas demorando $2^{O(S(n))}$. Logo o tempo é $2^{O(S(n))}$.

iv) (Ver mais tarde)

□

Corolário 4.1

$$\begin{aligned} \text{ATIME}(T(n)^{O(1)}) &= \text{DSPACE}(T(n)^{O(1)}) \\ \text{ASPACE}(S(n)) &= \text{DTIME}(2^{S(n)}) \end{aligned}$$

Tendo as inclusões

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{EXPSPACE}$$

podemos concluir que na complexidade alternada à um deslocamento para a direita dado que $\mathsf{AL} = \mathsf{P}$, $\mathsf{AP} = \mathsf{PSPACE}$, $\mathsf{APSPACE} = \mathsf{EXP}$, $\mathsf{AEXP} = \mathsf{EXPSPACE}$, ...

4.3 PSPACE e Completude

O problema seguinte generaliza o problema QSAT_i.

QSAT ou TQBF

Instância: Seja uma fórmula Booleana quantificada ψ

$$Q_1 x_1 Q_2 x_2 Q_3 x_3 \cdots Q_n x_n . \phi(x_1, \ldots, x_n),$$

onde cada $Q_i = \forall$ ou $Q_i = \exists$, e x_i é uma variável Booleana e ϕ é uma expressão Booleana envolvendo os x_i.

Questão: A fórmula ψ é verdade? Isto é, existem valorizações das variáveis Booleanas que a satisfaçam?

Contem SAT como sub-problema considerando só quantificadores existênciais e ϕ em forma normal conjuntiva. Contem TAUT, considerando só quantificadores universais. É equivalente a supor que ϕ está em forma normal conjuntiva e que os Q_i alternam estritamente entre $\exists$ e $\forall$.

Teorema 4.2 *QSAT é* PSPACE*-completo.*

Dem. Alternativamente podemos mostrar que QSAT é AP-completo. Começamos por mostrar que está em AP. Basta considerar que a etiquetagem da árvore de computação, substituindo $\exists$ por $\vee$ e $\forall$ por $\wedge$, corresponde a determinar o valor lógico de uma fórmula Booleana quantificada. Do mesmo modo para mostrar, directamente que está em PSPACE basta considerar o seguinte algoritmo

```
procedure T(ψ)
    if ψ não tem quantificadores then                    ▷ só tem constantes
        if A avaliação de ψ é 1 then
            return True
        else
            return False
        end if
        if ψ = ∃xφ then
            if T(φ[x/0]) ∨ T(φ[x/1]) then
                return True
            else
                return False
            end if
        else if ψ = ∀xφ then
            if T(φ[x/0]) ∧ T(φ[x/1]) then
                return True
            else
                return False
            end if
        end if
    end if
end procedure
```

A profundidade das chamadas recursivas é no máximo igual ao número de variáveis e em cada nível só o valor de uma variável é guardado. o algoritmo T executa em espaço linear.

Falta mostrar que é PSPACE-hard. Seja $L \in \mathsf{PSPACE}$. Podemos supor que $L \in \{0,1\}^\star$. Temos que ter uma redução polinomial de L a QSAT. Seja A uma MTA que reconhece L em tempo limitado por n^c. Podemos assumir que A Não tem estados $\neg$ e que a alternância é estrita introduzindo estados $\vee$ ou $\wedge$. Seja $x = x_1 \dots x_n$. A árvore de computação com dados x tem profundidade no máximo n^c e cada caminho é especificado por uma string binária y de comprimento n^c. Como no Teorema de Cook-Levin podemos construir uma fórmula $\phi(X_1, \dots, X_n, Y_1, \dots, Y_{n^c})$ que vale 1 para uma instanciação de x e y das variáveis X e Y se e só se o caminho de computação de A com dados x especificado por y leva à aceitação. Então A aceita x se e só se é verdadeira a fórmula

$$\exists Y_1 \forall Y_2 \cdots Q_{n^c} Y_{n^c} \phi(x_1, \dots, x_n, Y_1, \dots, Y_{n^k}).$$

A alternância de quantificadores reflecte exactamente a configurações $\vee$ e $\wedge$ na árvore de computação. $\square$

4.3.1 Complexidade de Jogos

Um *jogo perfeito* de 2 pessoas é um grafo $G = (B, M)$ e um nó especial inicial $s \in B$, com $M \subseteq B \times B$ especificando os movimentos legais do jogo. O jogo começa com $s_0 = s$ e os jogadores alternam sendo o primeiro jogador o jogador 1. O jogador 1 escolhe $s_1 \in B$ tal que $(s_0, s_1) \in M$, o jogador 2 escolhe $s_2 \in B$ tal que $(s_1, s_2) \in M$, etc.. Um jogador ganha se o outro não pode jogar. Exemplos de jogos são o Xadrez, Damas, Go, etc. Por exemplo para o Xadrez, $B = \{\text{posições do tabuleiro}\} \times \{b, p\}$, onde b (p) representa as brancas (pretas, respectivamente) e s é a configuração inicial do tabuleiro e b. A relação M codifica as regras do jogo.

Geografia é um jogo de crianças jogado em inglês em que os jogadores alternam em dizer nomes de cidades. O jogador 1 escolhe uma cidade. A partir daí cada jogador tem de dizer o nome duma cidade cujo nome começa com a mesma letra que o nome da cidade anterior termina. Uma cidade só pode ser referida uma vez no máximo.

Os elementos de B são pares (X, Y) onde X é um conjunto de cidades e Y é um conjunto de letras. O estado inicial é conjunto de todos os países$\times$todas as letras. Um movimento $((A, B), (A', B') \in M$ se $A' = A = \{c\}$ para alguma cidade de nome c tal que o seu nome começa por uma letra em B, i.e. $B = \{a\}$ e a é a letra que termina o nome c.

Designamos por $\textsc{move}(x, y)$ a fórmula $(x, y) \in M$, $\textsc{checkmate}(y) = \forall u \neg \textsc{move}(y, u)$ e

$$\textsc{win}(x) \quad = \quad \exists y(\textsc{move}(x, y) \wedge (\textsc{checkmate}(y) \vee \forall z(\textsc{move}(y, z) \rightarrow \textsc{win}(z)))$$

onde $\textsc{win}(x)$ é a menor ponto fixo da função monótona τ definida por:

$$\tau(\phi)(x) \quad = \quad \exists y(\textsc{move}(x, y) \wedge (\textsc{checkmate}(y) \vee \forall z(\textsc{move}(y, z) \rightarrow \phi(z)))$$

Podemos ainda simplificar

$$\textsc{win}(x) \quad = \quad \exists y(\textsc{move}(x, y) \wedge \forall z(\textsc{move}(y, z) \rightarrow \textsc{win}(z)))$$

Diz-se que o jogador que fizer o movimento x tem uma estratégia ganhadora (ou *forced win*). Pretende-se estudar a complexidade de decidir se o jogador 1 ganha. Por que a complexidade é uma análise assimptótica temos de admitir jogos arbitrariamente grandes (por exemplo um tabuleiro de xadrez de dimensão $n \times n$)

O jogo da Geografia pode ser generalizado. Seja GG dado por (C, E, s) onde $E \subseteq C \times C$ e $s \in C$. Em etapas pares, o jogador 1 move um token de s_{2i} para s_{2i+1} adjacente a s_{2i} e em etapas impares o o jogador 2 move um token de s_{2i+1} para s_{2i+2} adjacente a s_{2i+1}. Nenhum jogador pode mover o token para um vértice já visitado. Um jogador ganha forçando o oponente para uma posição da qual não há movimento legal.

GG

Instância: Dado (C, E, s), $E \subseteq C \times C$ e $s \in C$.

Questão: O jogador 1 tem uma estratégia ganhadora?

Teorema 4.3 *GG é* PSPACE*-completo.*

Dem. Seja A uma MTA que opera em tempo polinomial. Marcar o vertice $s_0 = s$ como visitado e iterar a rotina seguinte:

1. Selecionar existencialmente (não-deterministicamente), usando uma ramificação $\vee$ um movimento s_1 para o jogador 1.
2. Mover o token para essa posição e marcar s_1 como visitado.
3. Universalmente (ramificação $\wedge$) considerar todos os possíveis movimentos do jogador 2.
4. Cada novo subprocesso move o token para uma das possíveis posições s_2 e marca como visitada.
5. Selecionar existencialmente (não-deterministicamente) uma posição a partir de s_2, e repetir os passos anteriores alternando os jogadores.
6. Se é o jogador 1 a jogar e não existe um movimento legal seguinte, parar e enviar 0 (insucesso) ao processo pai.
7. Se é o jogador 2 a jogar e não há nenhum movimento seguinte válido então parar e enviar 1 (sucesso).

Cada caminho de computação termina após no máximo $|C|$ passos, porque pelo menos um vértice é marcado como visitado em cada passo.

Isto mostra que GG $\in$ AP, logo em PSPACE.

Para mostrar que é completo reduzimos polinomialmente QSAT a GG. Seja ϕ uma fórmula Booleana quantificada. Queremos construir uma instância de GG tal que o jogador 1 tem

uma estratégia ganhadora se e só se ϕ for verdade. Podemos supor que ϕ tem um prefixo de quantificadores alternados entre $\exists$ e $\forall$ em número par e começados com $\exists$, seguido de uma fórmula em forma normal conjuntiva:

$$\exists x_1 \forall x_2 \cdots \forall x_n c_1 \wedge \cdots \wedge c_m,$$

onde cada $c_i = l_{1_i} \vee \cdots \vee l_{k_i}$ é uma cláusula com l_{j_i} um literal. Construímos uma instância de GG da seguinte forma:

i) Para cada i, $1 \leq i \leq n$ criar quatro vértices $x_i, \overline{x_i}, u_i, v_i$. Temos arcos de u_i para x_i e $\overline{x_i}$ e de x_i e $\overline{x_i}$ para v_i.

ii) Inserimos um arco de v_i para u_{i+1}, $1 \leq i \leq n-1$

iii) Criamos um vértice para cada cláusula c_j e um arco de v_n para $c_1, \ldots, c_m$.

iv) Inserimos um arco de c_j para um literal l se l ocorre em c_j

v) O vértice inicial é u_1.

Temos

$$\begin{aligned} C &= \{x_i, \overline{x_i}, u_i, v_i \mid 1 \leq i \leq n\} \\ E &= \{(u_i, x_i), (u_i, \overline{x_i}), (x_i, v_i), (\overline{x_i}, v_i) \mid 1 \leq i \leq n\} \cup \{(v_i, u_{i+1}) \mid 1 \leq i \leq n-1\} \\ &\quad \cup \{(v_n, c_j) \mid 1 \leq j \leq m\} \cup \{(c_j, l_f) \mid l_f \in c_j\} \\ s &= u_1 \end{aligned}$$

Por exemplo para a fórmula $\exists x_1 \forall x_2 (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$ constrói a correspondente instância de GG. Os primeiros movimentos do jogo são feitos nas partes i) e ii) e escolhem uma valorização para as variáveis. Os valores das variáveis ímpares são jogadas do jogador 1 e as pares do jogador 2. A cada literal escolhido é atribuído o valor 0. Os últimos dois passos são jogados nas partes iii) e iv). Aqui o jogador 2 tem de escolher uma cláusula. O jogador 2 quer fazer a fórmula falsa, portanto se houver uma cláusula falsa, o jogador 2 escolhe essa cláusula. Nesse caso o jogador 1 não tem movimento por que todos os literais atingíveis da cláusula são falsos, o que significa que já foram visitados. Por outro lado, se todas as cláusulas forem verdadeiras, então cada uma tem de conter um literal verdadeiro. Sendo assim, independentemente do que o jogador 2 escolher, o jogador 1 terá um movimento seguinte, a seguir ao qual o jogador estará bloqueado. Então o jogador 1 ganha se e só se a valorização escolhida satisfaz a parte sem quantificadores da fórmula.

□

4.3.2 Relação com a hierarquia polinomial PH

A hierarquia polinomial pode ser definida usando alternância. Começamos por definir MTAs Σ_k e Π_k. Uma MTA-Σ_k não tem negações, para quaisquer dados de entrada faz no máximo k alternâncias de configurações $\vee$ e $\wedge$ ao longo de qualquer caminho, começando com $\vee$. Do mesmo modo se define uma máquina Π_k , começando com $\vee$. Formalmente

Definição 4.2 *Uma MTA-Σ_k é uma MTA tal que em qualquer dados, qualquer computação pode ser dividida em intervalos contíguos tal que*

i) em qualquer intervalo todas as configurações são ou todas $\vee$ ou todas $\wedge$

ii) existem no máximo k intervalos

iii) o primeiro intervalo são configurações $\vee$

Uma MTA-Π_k define-se de modo semelhante só que em iii) são configurações $\wedge$.

Uma máquina Σ_1 é uma MTN e por convenção as máquina Σ_0 e Π_0 são MT's.
Definimos as classes de complexidade,

$$\begin{aligned} A\Sigma_k^p &= \{L(A) \mid A \text{ é uma } MTA - \Sigma_k \text{ tal que } T_A(n) = O(n^{O(1)})\} \\ A\Pi_k^p &= \{L(A) \mid A \text{ é uma } MTA - \Pi_k \text{ tal que } T_A(n) = O(n^{O(1)})\} \end{aligned}$$

Então $A\Sigma_1^p = NP$, $A\Pi_1^p = coNP$ e $A\Sigma_0^p = A\Pi_0^p = P$.

Lema 4.2

$$\begin{aligned} A\Pi_k^p &= coA\Sigma_k^p = \{L^c \mid L \in A\Sigma_k^p\} \\ A\Sigma_k^p \cup A\Pi_k^p &\subseteq A\Sigma_{k+1}^p \cap A\Pi_{k+1}^p \\ \bigcup_k A\Sigma_k^p &= \bigcup_k A\Pi_k^p \subseteq PSPACE \end{aligned}$$

Exercício 4.3.1 *Demonstra o Lema 4.2.*

O teorema seguinte relaciona estas classes com a hierarquia polinomial.

Teorema 4.4 *Para todo o $k \geq 1$, $A\Sigma_k^p = \Sigma_k^p$.*

Dem. Por indução em k. Para $k = 1$ é imediato. $\Sigma_1^p = NP = A\Sigma_1^p$. Suponhamos que $A\Sigma_k^p = \Sigma_k^p$ e vamos mostrar que $A\Sigma_{k+1}^p = \Sigma_{k+1}^p$.

$\supseteq$ Suponhamos que temos A uma MTA-$\mathsf{\Sigma_{k+1}}$ que executa em tempo n^c e $L = L(A)$. Queremos mostrar que $L \in \mathsf{NP}^{\mathsf{\Sigma_k^p}}$. Suponhamos que as configurações de A são codificadas somo strings sobre um alfabeto finito Δ dum modo razoável. Podemos assumir que todas as configurações atingíveis da configuração inicial (`start`) com quaisquer dados x, $|x| = n$, são representadas como strings em Δ^{n^c}. Considere-se

$$D \quad = \quad \{\alpha \in \Delta^{n^c} \mid \alpha\text{configuração-}\wedge, |\alpha| = n^c, \alpha \xrightarrow{\mathsf{\Pi_k}}{}^{n^c} \texttt{accept}\}$$

onde `accept` é uma configuração de aceitação e $\xrightarrow{\mathsf{\Pi_k}}$ indica uma sequência de passos que começam com um intervalo de configurações . Determinar se $\alpha \in D$ pode ser feito por uma MTA-$\mathsf{\Pi_k}$ limitada em tempo polinomial simulando A começando em α. Assim $D \in \mathsf{A\Pi_k^p}$, logo $D^c \in \mathsf{A\Sigma_k^p}$. Mais ainda, A aceita x se e só se existe um caminho de computação conduzindo da configuração inicial através de configurações-$\vee$ até algum $\alpha \in D$. Então, usando a hipótese de indução, L pode ser aceite por uma MTON limitada em tempo polinomial com oráculo D^c que com dados x adivinha um caminho de computação $\texttt{start} \xrightarrow{\vee}{}^{l} \alpha$ e depois consulta o oráculo para verificar se $\alpha \in D$. Se a resposta for *sim* então $\alpha \notin D$.

$\subseteq$ Suponhamos que temos uma MTON O com oráculo $L' \in \mathsf{\Sigma_k^p}$ com $T_O(n) = O(n^c)$ e seja $L = L(O)$. Queremos mostrar que $L \in \mathsf{A\Sigma_{k+1}^p}$. Construímos uma MTA-$\mathsf{\Sigma_{k+1}}$ A que trabalha da seguinte maneira. Com dados x, A começa por simular O com dados x, excepto quando O quer consultar o oráculo para uma dada string y. Nestes casos, A não deterministicamente adivinha uma resposta do oráculo (se y está em L' ou não) e recorda y e a resposta. Continua a simulação ate O chegar a um estado de aceitação ou rejeição o que tem de acontecer em tempo n^c. Se O rejeitar, A também rejeita. Se O aceitar, A tem de verificar que as respostas adivinhadas pelo oráculo estão correctas. Até agora a árvore de computação de A é semelhante à de O, excepto que em cada pergunta ao oráculo A tem uma ramificação ($\vee$) não determinística para as duas possíveis respostas do oráculo. Até este ponto A só executou ramificações existenciais ($\vee$). Em cada folha da árvore de computação, existe um processo com uma lista de perguntas ao oráculo e as respostas adivinhadas que precisam de ser verificadas. As listas podem ser diferentes para caminhos de computação diferentes, porque perguntas posteriores dependem de respostas dadas a perguntas feitas mais cedo, mas todas as listas têm no máximo comprimento n^c. Assim cada processo tem uma lista $y_1, \ldots, y_m$ de perguntas ao oráculo cuja resposta do oráculo foi sim e uma lista $z_1, \ldots, z_l$ para as quais a resposta foi não.

O comprimento total combinado os y_i e dos z_j concatenados não pode exceder n^c porque O teve de os escrever a todos na fita de oráculo. A máquina A tem agora de verificar com uma computação Σ_{k+1} em tempo polinomial que $y_i \in L'$, para $1 \leq i \leq m$ e $z_j \notin L'$, para $1 \leq j \leq l$. Esta computação Σ_{k+1} combinada com a computação Σ_1 já efectuada

é ainda uma computação Σ_{k+1}. Reduzimos o problema a mostrar que para $L' \in \mathsf{\Sigma_k^p}$ a linguagem

$$\{y_1\#\cdots\#y_m\#z_1\#\cdots\#z_l \mid \ y_i \in L', 1 \leq i \leq m, \ z_j \notin L', 1 \leq j \leq l\}$$

está em $\mathsf{\Sigma_{k+1}^p}$.

Cada advinha $y_i \in L'$ pode ser verificada com uma computação Σ_k e cada adivinha $z_j \notin L'$ com uma computação Π_k. Mas não podemos gerar uma ramificação $\wedge$ com todas estas verificações porque teríamos uma computação Π_{k+1}. Uma computação Σ_k para determinar se $y_i \in L'$ é da forma $\vee_1 \wedge_2 \cdots \wedge_k (\exists_1 \forall_2 \cdots \forall_k)$. Mas como temos n variáveis temos que fazer primeiro uma ramificação $\wedge$ $(\forall)$. Por isso, vamos poder tornar $\vee_1$ $(\exists_1)$ desnecessário considerando "skolemização"que permite eliminar quantificadores existenciais transformado variáveis existenciais em funções de variáveis universais (ver abaixo). Seja n^d o limite temporal da $MTA - \Sigma_k$ para L'. Não deterministicamente adivinhar as strings binárias $w_1, \ldots, w_m$ de tamanho n^d. Isto pode ser feito em tempo $mn^d \leq n^{c+d}$. Estas strings irão guiar a primeira etapa de adivinhas existenciais nas computações Σ_k para verificar se $y_i \in L'$ (isto é, $\exists_1$ acima). Fazemos uma ramificação $\wedge$ com $(m + l)$ processos cada um correspondendo a um y_i ou a um z_j. Para já a computação é Σ_2. Para cada pergunta negativa z_j, verificamos se $z_j \notin L'$ usando uma computação Π_k para L'^c. Esta computação combinada com a Σ_2 dá uma computação Σ_{k+1}. Para cada pergunta positiva y_i, simulamos a computação Σ_k para L' mas usamos a adivinha w_i para direcionar o primeiro nível de de ramificações existenciais, e assim fazendo o primeiro nível deterministico. A computação restante é Π_{k-1}, pelo que o total é também uma computação Σ_{k+1}.

A skolemização, consiste na equivalência $(\forall i \in A \exists w \in B \varphi(i, w)) \Leftrightarrow (\exists f : A \longrightarrow B \forall i \in A \varphi(i, f(i)))$. O que também pode ser visto como uma generalização da distributividade da conjunção em relação à disjunção: $(a \vee b) \wedge (c \vee d) = (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d)$. Normalmente isto leva a uma explosão exponencial da fórmula mas aqui não é o caso: $A = \{1, \ldots, m\}$, $B = \{0, 1\}^{n^d}$ com $m \leq n^c$. Logo $|A| \leq n^c$, $|B| = 2^{n^d}$ e $|A \longrightarrow B| = (2^{n^d})^{n^c} = 2^{n^{d+c}}$.

□

Capítulo 5

L e NL

Complexidades de espaço sublineares são possíveis porque as máquinas de Turing têm uma fita de leitura e outra de escrita cujo espaço não é contabilizado.

5.1 Transdutores em espaço log

Antes de formalizarmos melhor este conceito, vamos ver exemplos de problemas/linguagens nas classes $\mathsf{L} = \text{DSPACE}(\log n)$ e $\mathsf{NL} = \text{NSPACE}(\log n)$

Exemplo 5.1 *A linguagem $L = \{a^k b^k \mid k \geq 0\}$ está em L. Habitualmente constrói-se uma MT que alternadamente "corta"um a e um b. Assim a complexidade é linear porque tem de se saber quais os símbolos já cortados e não se pode escrever na fita de leitura. No entanto pode passar a executar em espaço logarítmico se se considerarem dois contadores na fita de trabalho: um conta o número de as e separadamente outro conta o número de bs em binário. Portanto o algoritmo executa em $O(\log n)$.*

Exemplo 5.2 *REACHABILITY* $\in \mathsf{NL}$*, onde* *REACHABILITY*

Instância: *Dado um digrafo $G = (V, E)$, $x, y \in V$ e $K \geq 0$*

Questão: *Existe uma caminho de x para y de tamanho menor ou igual a K.*

Uma MTN limitada em espaço logarítmico começa em x e não deterministicamente adivinha os passos de x para y. A máquina apenas guarda a localização do nó corrente. O nó seguinte é escolhido não deterministicamente entre os que estão ligados ao corrente. Repete-se o procedimento até chegar a y e aceitar ou até terem sido feitos $m = |V|$ passos e rejeitar.

Definição 5.1 *Um* transdutor em espaço logarítmico (ou log)*, TL, é uma máquina de Turing determinística total com escrita limitada em espaço logarítmico, $M = (S, \Sigma, \Gamma, \Delta, \triangleright, \triangleleft, \square, \delta, s_y, s_n, s_o)$. Tem*

- *uma fita de leitura com* 2*-sentidos, inicialmente com os dados*
- *uma fita de leitura/escrita com* 2*-sentidos com* $O(\log n)$*) células, inicialmente com brancos.*
- *uma fita de escrita só com* 1 *sentido (esquerda para a direita), inicialmente com brancos.*

A máquina começa com todas as cabeças posicionadas no extremo esquerdo. Os dados x estão escritos entre os marcadores $\triangleright$ e $\triangleleft$ e a cabeça de leitura nunca passa os marcadores. Funciona como uma MT lendo os símbolos da fita de leitura, escrevendo na fita de trabalho e ocasionalmente entra um estado especial (s_o) que causa a escrita de um símbolo de Δ na fita de escrita e avança uma célula para a direita. Quando parar (o que se assume sempre que acontece) a string de $\Delta^\star$ escrita na fita de escrita é o valor da função calculada pelo transdutor com dados x. Só é contado o espaço da fita de trabalho.

Definição 5.2 *Uma função* $\sigma : \Sigma^\star \longrightarrow \Delta^\star$ *é computável em espaço logarítmico se é calculável por um transdutor* log*.*

O resultado de um transdutor log tem tamanho limitado polinomialmente. Isto é, para qualquer $\sigma : \Sigma^\star \longrightarrow \Delta^\star$ é computável em espaço logarítmico $\exists d \forall x \in \Sigma^\star, |\sigma(x)| \leq |x|^d$. Isto porque um TL só pode emitir um símbolo por cada passo e só pode executar um número de passos polinomial. Caso executasse um número de passos exponencial, 2^n então tinha de repetir alguma configuração e entrava num ciclo infinito, e não parava.

Para codificar um problema noutro podemos definir uma nova relação de redução calculada por um transdutor log.

Definição 5.3 *Seja* $L \subseteq \Sigma^\star$ *e* $L' \subseteq \Delta^\star$*. Dizemos que* $L \leq_m^{\log} L'$*, i.e* L *reduz-se em espaço logarítmico a* L' *se existe uma função computável em espaço logarítmico* $\sigma : \Sigma^\star \longrightarrow \Delta^\star$ *tal que para todo* $x \in \Sigma^\star$*:*

$$x \in L \Leftrightarrow \sigma(x) \in L'.$$

Lema 5.1 *A relação* $\leq_m^{\log}$ *é transitiva.*

Dem. Sejam $L_i \subseteq \Sigma_i^\star$, para $1 \leq i \leq 3$, e $L_1 \leq_m^{\log} L_2$ e $L_2 \leq_m^{\log} L_3$. Temos que mostrar que $L_1 \leq_m^{\log} L_3$. A demonstração não é trivial porque temos de garantir que a composição é calculada em espaço logarítmico. Seja $f : \Sigma_1^\star \longrightarrow \Sigma_2^\star$ e $g : \Sigma_2^\star \longrightarrow \Sigma_3^\star$ as reduções (em espaço) logarítmicas entre L_1 e L_2, e L_2 e L_3, respectivamente. Suponhamos ainda que M e N são transdutores logarítmicos que calculam f e g respectivamente.

Para $x \in \Sigma_1^\star$, sabemos que $|f(x)|$ é polinomial em x mas temos que mostrar que $g(f(x))$ é uma redução logarítmica. Para calcular $g(f(x))$ simula-se a computação de N com dados $f(x)$, mas $f(x)$ é dado a N símbolo a símbolo. Quando N quer ler o i-ésimo símbolo dos seus dados, o número i é dado a uma subrotina que simula M com dados x desde o ínicio,

contando e não escrevendo nenhum símbolo de saída até chegar ao i-ésimo. Só é necessário espaço suficiente para guardar as fitas de N e M e um contador até $|f(x)|$. $\square$

Lema 5.2 *Para $A \in \Sigma^\star$, $A \in \mathsf{L}$ se e só se $A \leq_m^{\log} \{1\}$, onde $\{1\} \subseteq \{0,1\}$.*

Dem. Decidir se $x \in A$ corresponde a uma redução $\sigma : \Sigma^\star \longrightarrow \{0,1\}$ tal que $x \in A$ se e só se $\sigma(x) = 1$. Um algoritmo de decisão em espaço logarítmico para $x \in A$ é uma redução de A a um conjunto de dois elementos. $\square$

Lema 5.3 *Se $A \leq_m^{\log} B$ e $B \in \mathsf{L}$, então $A \in \mathsf{L}$.*

Dem. Pelo Lema $B \leq_m^{\log} \{1\}$ e pela transitividade também $A \leq_m^{\log} \{1\}$. $\square$

Lema 5.4 *Se $A \leq_m^{\log} B$ então $A \leq_m^p B$.*

Dem. Como já vimos um transdutor em espaço logarítmico só pode executar no máximo em tempo polinomial. $\square$

Não se sabe se a relação $\leq_m^{\log}$ é estritamente mais forte que $\leq_m^p$, mas na maior parte dos resultados em que se usou $\leq_m^p$ podemos usar $\leq_m^{\log}$.

5.2 Completude

Uma linguagem $L \in \Sigma^\star$ diz-se $\leq_m^{\log}$-hard para a classe de complexidade $\mathcal{C}$, se $L' \leq_m^{\log} L$, para todo $L' \in \mathcal{C}$. Como anteriormente, L é tão dificil como qualquer problema de decisão em $\mathcal{C}$, porque pode codificar qualquer problema em $\mathcal{C}$. Uma linguagem diz-se $\leq_m^{\log}$-completa para $\mathcal{C}$ se

i) L é $\leq_m^{\log}$-hard para $\mathcal{C}$

ii) $L \in \mathcal{C}$

Teorema 5.1 *REACHABILITY é $\leq_m^{\log}$-completo para* NL.

Dem. Já vimos que REACHABILITY $\in$ NL. Falta ver que é $\leq_m^{\log}$-hard para NL. Seja $A \in$ NL e suponhamos que M é uma MTN limitada em espaço logarítmico. Com dados x, vamos considerar o digrafo das configurações $G_M = (V, E)$ onde $\alpha \in V$ é uma configuração, e $(\alpha, \beta) \in E$ se $\alpha \xrightarrow{M} \beta$. Podemos supor que `start` é a configuração inicial e que existe uma só configuração de aceitação `accept` (fazendo M apagar a fita de trabalho e mover as cabeças para a esquerda antes de aceitar). Então, M aceita x se e só se existe um caminho entre `start` e `accept` em G_M. Cada configuração pode ser representada por um tuplo (s, i, w, y), onde s é o estado cor rente, i a posição da cabeça da fita de leitura e w e y o contéudo da fita de trabalho (com a posição da cabeça no primeiro símbolo de y). Se os dados x tiverem

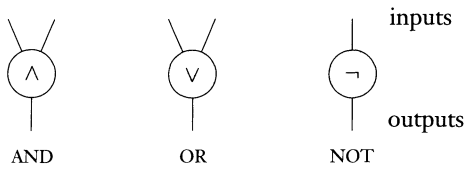


Figura 5.1: Portas lógicas dum circuito

tamanho n toda esta informação pode ser representada por uma string de tamanho $\log n$ sobre um alfabeto Δ de tamanho d. Temos então $V = \Delta^{\log n}$ e $|V| = d^{\log n} = n^{\log d}$. Falta ver que este grafo pode ser produzido por um transdutor log com dados x. O transdutor primeiro escreve o conjunto dos vértices $\Delta^{\log n}$: marca $\log n$ células na fita de trabalho e depois gera as strings de $\Delta^{\log n}$ por ordem lexicográfica escrevendo-as na fita de saída. Para as arestas escreve todos os pares (α, β) em ordem lexicográfica. Para cada par, verifica se codificam uma configuração e se $\alpha \xrightarrow{M} \beta$ com dados x. Para tal tem de ver o i-ésimo símbolo de x, sendo i a posição da cabeça da fita de leitura. Se sim escreve (α, β). Finalmente, escreve as configurações `start` e `accept`. Embora $(V, E, \texttt{start}, \texttt{accept})$ tenha comprimento polinomial, só espaço logarítmico foi usado para o produzir, porque no máximo existem duas configurações de M escritas na fita de trabalho ao mesmo tempo. Então a transformação $x \mapsto (V, E, \texttt{start}, \texttt{accept})$ é uma redução em espaço logarítmico de A a REACHABILITY. □

Corolário 5.1 *REACHABILITY* $\in \mathsf{L}$ *se e só se* $\mathsf{L} = \mathsf{NL}$.

Exercício 5.2.1 *Mostra que* $\mathsf{NL} \subseteq \mathsf{P}$.

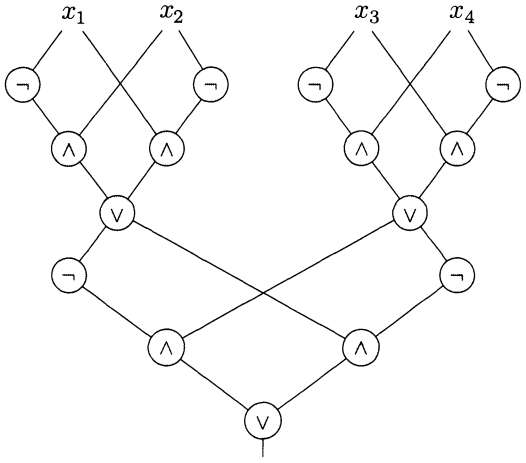
5.3 Circuitos Booleanos

As funções Booleanas $f : \{0,1\}^n \longrightarrow 0,1$ podem ser representadas por circuitos Booleanos que correspondem a digrafos acíclicos onde os nós são operações lógicas e as variáveis (e as constantes) nós com grau de entrada zero. A saída é o nó com grau da saída zero. Em especial podemos considerar circuitos sem variáveis, só com constantes. Se um circuito tiver mais que uma saída corresponde a várias funções Booleanas (uma por cada saída). Na Figura 5.3 estão representadas as portas $\wedge$ e $\vee$ com grau de entrada 2 (*fan-in*) e grau de saída (*fan-out*) 1, e a porta $\neg$ com grau de entrada 1. As arestas (ligações) do grafo (circuito) podem considera-se etiquetadas com o valor 0 ou o valor 1.

O *tamanho* de um circuito C, é o seu número de vértices, e representa-se por $|C|$.

Com dados $x \in \{0,1\}^n$ o valor do circuito $C(x)$ é calculado da maneira natural avaliando as operações lógicas.

Exemplo 5.3 *A função* $\text{paridade}_n : \{0,1\}^n \longrightarrow \{0,1\}$ *tem o valor* 1 *se o número de* 1*s nos dados de entrada* $x \in \{0,1\}^n$ *é ímpar. O circuito seguinte calcula a função para* $n = 4$.



Podemos também representar circuitos por programas só com atribuições (sem ciclos nem outras instruções de controlo) chamados *straight line*.

Definição 5.4 *Um circuito Booleano é um programa com um número finito de atribuições da forma*

$$\begin{aligned} P_i &:= 0, \\ P_i &:= 1, \\ P_i &:= P_j \wedge P_k, \quad j,k \leq i, \\ P_i &:= P_j \vee P_k, \quad j,k \leq i \\ P_i &:= \neg P_j, \quad j, \leq i, \end{aligned}$$

onde cada P_i *aparece do lado esquerdo de uma atribuição exactamente uma vez. Pretende-se calcular o valor* P_n*, onde* n *é o indice máximo. Podemos considerar também instruções da forma* $P_i := x_j$ *ond* x_j *é um dos valores de entrada do circuito.*

Exemplo 5.4 *A função* paridade_2 *tem o seguinte programa com dados* x_1 *e* x_2

$$\begin{aligned} P_1 &:= \neg x_1, \\ P_2 &:= \neg x_2, \\ P_3 &:= P_1 \wedge x_2, \\ P_4 &:= x_1 \wedge P_2, \\ P_5 &:= P_3 \vee P_4. \end{aligned}$$

Exercício 5.3.1 *Desenha o circuito correspondente a este programa:*

$$\begin{aligned} P_1 &:= 1, \\ P_2 &:= 0, \\ P_3 &:= 0, \\ P_4 &:= P_1 \wedge P_2, \\ P_5 &:= \neg P_2, \\ P_6 &:= P_4 \vee P_5, \\ P_7 &:= P_6 \wedge P_3. \end{aligned}$$

Exercício 5.3.2 *Constrói um circuito que permita adicionar dois bits (half-adder) que corresponde a duas funções booleanas (o valor da adição e o transporte). Constrói com esse circuito um que permit adicionar dois inteiros de 3 bits.*

Temos associado o seguinte problema de decisão.

VALOR de um Circuito (CVP)

Instância: Dado um circuito Booleano (com várias entradas uma saída e uma atribuição de valores às entradas)

Questão: Qual o valor do circuito?

Teorema 5.2 *CVP é* $\leq_m^{\log}$ *-completo para* P.

Dem. É evidente que CVP $\in$ P. Seja $A \in$ P e temos de o reduzir a CVP. Seja $M = (S, \Sigma, \Gamma, s_0, \triangleright, \square, \delta, s_y, s_n)$ uma MT de uma fita que aceita A em tempo limitado por n^c. Temos que $\delta : S \times \Gamma \longrightarrow S \times \Gamma \times \{\leftarrow, \rightarrow\}$. Seja Δ o alfabeto de codificação das configurações de M com dados x, $|x| = n$. Podemos considerar as configurações sucessivas de M numa tabela R de dimensão $(n^c+1) \times (n^c+1)$. A i-ésima linha de R é uma string de Δ^{n^c+1} que descreve a i-ésima configuração e a coluna j descreve o que ocorre na célula j ao longo da computação (os símbolos guardados e se a cabeça da fita está ou não na célula, e o estado). Se o i-ésimo movimento for $\delta(s, a) = (s', b, \rightarrow)$ podemos ter as linhas i e $i+1$ seguintes:

$\triangleright$	a	b	a	a	b	a	b	a	a	b	a	b	$\square$	$\square$	$\square$
							s								
$\triangleright$	a	b	a	a	b	b	b	a	a	b	a	b	$\square$	$\square$	$\square$
								s'							

Consideramos os elementos de Δ da forma $\boxed{\begin{array}{c} a \\ \hline \\ \end{array}}$ ou $\boxed{\begin{array}{c} a \\ \hline s \end{array}}$, com $a \in \Gamma$ e $s \in S$.

A tabela R pode ser especificada em termos das condições de consistência local em tabelas $((n^c+1)\times(n^c+1)$ sobre Δ.

Cada condição é uma relação nas entradas da tabela numa vizinhança duma localização (i,j). A conjunção de todas as localizações é suficiente para determinar R univocamente. O circuito contém as seguintes variáveis Booleanas:

$$\begin{aligned} P^a_{i,j} \quad & 0 \le i,j \le n^c, a \in \Gamma \\ Q^s_{i,j} \quad & 0 \le i,j \le n^c, s \in S \end{aligned}$$

onde $P^a_{i,j}$ representa que o símbolo que ocupa a célula j no tempo i é a, e $Q^s_{i,j}$ representa que no tempo i a máquina está no estado s visitando a célula j.

As condições para que a máquina possa estar no estado s visitando a célula j no tempo i ou que o símbolo ocupando a célula j no tempo i é a são dadas a seguir. Para $1 \le i \le n^c$, $0 \le j \le n^c$ e $b \in \Gamma$

$$\begin{aligned} P^b_{i,j} \quad := \quad & \bigvee_{\delta(s,a)=(s',b,d)} (Q^s_{i-1,j} \wedge P^a_{i-1,j}) \\ & \vee (P^b_{i-1,j} \wedge \bigwedge_{s\in Q} \neg Q^s_{i-1,j}) \end{aligned}$$

Para $1 \le i \le n^c$, $0 \le j \le n^c - 1$ e $s \in S$

$$Q^{s'}_{i,j} := \bigvee_{\delta(s,a)=(s',b,\rightarrow)} (Q^s_{i-1,j} \wedge P^a_{i-1,j-1}) \tag{5.1}$$

$$\vee \bigvee_{\delta(s,a)=(s',b,\leftarrow)} (Q^s_{i-1,j+1} \wedge P^a_{i-1,j+1}) \tag{5.2}$$

$$\tag{5.3}$$

para $j = 0$ temos $Q^s_{i,0}$ igual só e a 5.1 e para $j = n^c$, Q^s_{i,n^c} é igual a 5.2. A primeira linha da tabela tem a configuração inicial com os dados $x = a_1 \cdots a_n$, o estado inicial e o resto da fita com brancos.

$$
\begin{aligned}
P^{\triangleright}_{0,0} &:= 1 \\
P^{b}_{0,0} &:= 0, \quad b \in \Gamma \setminus \{\triangleright\} \\
P^{a_j}_{0,j} &:= 1, \quad 1 \leq j \leq n \\
P^{b}_{0,j} &:= 0 \quad b \in \Gamma \setminus \{a_j | 1 \leq j \leq n\} \\
P^{\square}_{0,j} &:= 1 \quad n+1 \leq j \leq n^c \\
P^{b}_{0,j} &:= 0 \quad b \in \Gamma \setminus \{\square, \triangleright\}, n+1 \leq j \leq n^c \\
Q^{s_0}_{0,0} &:= 1 \\
Q^{s_0}_{0,j} &:= 0, \quad 1 \leq j \leq n^c \\
Q^{s}_{0,j} &:= 0, \quad 1 \leq j \leq n^c, s \in S \setminus \{s_0\}
\end{aligned}
$$

Assumindo a configuração habitual de aceitação temos

$$Q^{s_y}_{n^c,0} \vee Q^{s_y}_{n^c,1}$$

determina se M aceita x.

Esta construção pode ser feita em espaço logarítmico. Embora o circuito tenha tamanho polinomial é muito uniforme. As únicas diferenças são os índices i, j que podem ser escritos em espaço logarítmico.

□

Corolário 5.2

(i) $\mathsf{P} = \mathsf{NL}$ *se e só se* $\mathit{CVP} \in \mathsf{NL}$.

(ii) $\mathsf{P} = \mathsf{L}$ *se e só se* $\mathit{CVP} \in \mathsf{L}$.

5.3.1 Famílias de Circuitos e a Classe $\mathsf{P}_{/\mathsf{POLY}}$

Cada circuito só pode ser usado com um número fixo de entradas (um dado comprimento n). Para utilizar circuitos para o reconhecimento de linguagens em $\{0,1\}$ temos de considerar famílias de circuitos $C_0, \ldots, C_n, \ldots$ onde C_i é usado para dados de tamanho i. No entanto isto conduz a uma modelo não -uniforme de computação, dado que para strings de tamanhos diferentes, diferentes circuitos são usados.

Definição 5.5 *Uma familia de circuitos C é uma l ista infinita de circuitos $(C_0, C_1, C_2, \ldots)$ onde C_n tem n variáveis de entrada e uma saída. Um circuito C decide $L \subseteq \{0,1\}^\star$, se para cada string x, $|x| = n$,*

$$x \in L \Leftrightarrow C_n(x) = 1.$$

Seja $T : \mathbb{N} \longrightarrow \mathbb{N}$. Uma familia de circuitos C tem *tamanho* $T(n)$ se $|C_n| \leq T(n)$, para todo $n \geq 0$. Diz-se que $C \in \text{CSIZE}(T(n))$.

Dois circuitos C e C' são equivalentes $C(x) = C'(x)$ para todo o x (que tenha o tamanho dos dados de C e C'). Um circuito é tamanho minimal se não shouver um equivalente de tamanho menor. Uma familia de circuitos $(C_0, C_1, C_2, \ldots)$ é minimal se para circuito C_i é minimal.

A *complexidade por circuitos* de uma linguagem L é o tamanho duma familia de circuitos minimais.

Definimos a seguinte classe de complexidade

$$\begin{aligned} \mathsf{P}_{/\mathsf{POLY}} &= \{L \mid L \text{ é decidível por uma familia de circuitos de tamanho polinomial}\} \\ &= \bigcup_c \text{CSIZE}(n^c) \end{aligned}$$

Exemplo 5.5 *A linguagem $L = \{1^n \mid n \geq 0\}$ pode ser decidida por uma familia de circuitos $(C_i)_i$ em* CSIZE(n)*. Para cada n pode-se considerar um circuito com n entradas e uma árvore binária com portas $\wedge$ sendo $C_n(x)$ correspondente à conjunção de todos os bits de entrada.*

Uma formula proposicional em forma normal conjuntiva corresponde a um circuito especial que pode ser calculada com um circuito de tamanho $n2^n$, para $b = N$ bits de entrada.

Teorema 5.3 $\mathsf{P} \subseteq \mathsf{P}_{/\mathsf{POLY}}$.

Demonstração 5.3.1 *Consequência do Teorema 5.2.*

Contudo a inclusão é estrita, dado que a classe $\mathsf{P}_{/\mathsf{POLY}}$ até contem linguagens indecidíveis. Por exemplo, podemos concluir que todas as linguagens unárias estão em $\mathsf{P}_{/\mathsf{POLY}}$ e existem linguagens unárias indecidiveis

Proposição 5.1 *Seja $L \subseteq \{1^n \mid n \geq 0\}$ então $L \in \mathsf{P}_{/\mathsf{POLY}}$.*

Demonstração 5.3.2 *Construímos uma familia de circuitos C lineares que decide L. Se $1^n \in L$ consideramos o circuito só com portas $\wedge$ que calcula a conjunção de todos os bits de entrada. Senão consideramos um circuito com saída* 0*: pode ser um circuito só com as entradas e a saída e sem ligações.*

Então, $x \in L$ se e só se $C(x) = 1$. Notar que se $1^n \notin L$, L não tem strings de tamanho n (e portanto o circuito tem como saída 0*).*

Seja $L \subseteq \{0,1\}^\star$ uma linguagem indecidível. Então $U \subseteq \{1\}^\star$ tal que

$$U = \{1^n \mid \text{a representação binária de } n \text{ está em } L\},$$

é indecidível também.

Isto mostra que famílias de circuitos arbitrárias não são bom modelo de computação: podemos ter de gastar tempo ilimitado para construir o circuito embora possamos saber que existe.

Definição 5.6 *Uma família de circuitos $(C_n)_n$ é* P*-uniforme para espaço logaritmico se existe uma MT limitada em tempo polinomial que com dados 1^n tem como saída a descrição de um circuito C_n.*

Teorema 5.4 *Uma linguagem L é decidida por uma familia de circuitos* P*-uniformes se e só se $L \in \mathsf{P}$.*

Demonstração 5.4.1 *Se existe uma familia de circuitos C_n construível por uma MT M em tempo polinomial que decide L, podemos construir uma MT M' para L: com dados x, M' executa M com dados $1^{|x|}$ e obtém um circuito $C_{|x|}$ que pode avaliar com dados x. Se $L \in \mathsf{P}$ podemos usar o Teorema .*

Existem as seguintes conjecturas

Conjectura 5.1 *1. Os problemas* NP*-completos não têm circuitos uniformes polinomiais.*

2. Os problemas NP*-completos não têm circuitos polinomiais, uniformes ou não.*

No entanto temos ainda o seguinte

Teorema 5.5 *Se* $\mathsf{NP} \subseteq \mathsf{P}_{/\mathsf{POLY}}$ *então* $\mathsf{PH} = \Sigma_2^{\mathsf{p}}$.

5.3.2 Paralelismo e circuitos

Uma computação paralela pode ser vista como ocorrendo em vários processadores independentes, que podem comunicar instantaneamente. Os modelos mais usados para estudar a complexidade da computação paralela são os PRAMS (*maquinas pararelas de acesso aleatório*) e famílias uniformes de circuitos. Num circuito Booleano, a computação realiza-se concorrentemente em muitas portas e por isso é um bom modelo de paralelismo. A classe de complexidade paralela mais importante é a NC (de *Nick's Class*, Nicholas Pippenger). Situa-se entre L e P e é tão importante para o paralelismo como a classe P para a computação sequencial.

Definição 5.7 *Uma família de circuitos $(C_n)_n$ é uniforme para espaço logarítmico se existe um transdutor T em espaço logarítmico que com dados 1^n tem como saída uma descrição de C_n.*

A *profundidade* de um circuito é o comprimento do maior caminho de um bit de entrada até a uma saída. Se uma família de circuitos C tiver profundidade $f(n)$ dizemos que pertence a CDEPTH($f(n)$).

Definição 5.8 *Uma familia de circuitos Booleanos $(C_0, C_1, \ldots, C_n, \ldots)$ é uma* familia uniforme em espaço logarítmico de circuitos com profundidade polylog e tamanho polinomial *se:*

i) C_n tem n bits de entrada e é composto por portas $\wedge$, $\vee$ e $\neg$;

ii) C_n *tem profundidade no máximo* $(\log n)^{O(1)}$ *(*polylog*)*

iii) C_n *tem no máximo* $n^{O(1)}$ *portas;*

iv) a familia é uniforme em espaço logarítmico.

Define-se a classe

$$\mathsf{NC} = \bigcup_i \mathsf{NC}^i \quad = \quad \{L \mid L \text{ é decidível por uma família de circuitos uniforme em espaço logarítmico com profundidade polylog e tamanho polinomial}\}$$

onde NC^i corresponde à classe en que a profundidade dos circuitos é $O(\log^i n)$, $i \geq 1$.

Exemplo 5.6 (Multiplicação de Matrizes Booleanas) *Considere-se a multiplicação de matrizes Booleanas* $A = (a_{ik})$ *e* $B = (b_{ik})$ *de dimensão* $m \times m$*. O* n*-ésimo circuito tem* $2m^2 = n$ *bits de entrada e* m^2 *bits de saída. Os valores de saída representam a matriz* $C = (c_{ik})$ *também* $m \times m$ *tal que*

$$c_{ik} = \bigvee_{j=1}^{m} (a_{ij} \wedge b_{jk}),$$

para $1 \leq i, k \leq m$*. O circuito* C_n *tem como dados as entradas das matrizes* A *e* B *e saídas as entradas de* C*. O circuito calcula primeiro* $a_{ij} \wedge b_{jk}$ *em paralelo para* $1 \leq i, k \leq m$*. Isto faz-se num passo com* $m^3 = O(n^{\frac{3}{2}})$ *portas. Depois para cada* i *e* k *, são calculadas as disjunções* $\bigvee_{j=1}^{m}(a_{ij} \wedge b_{jk})$*, numa árvore binária com* $m-1$ *portas* $\vee$ *e profundidade* $O(\log m)$*. No total o tamanho do circuito é* $O(n^{\frac{3}{2}})$ *e profundidade* $O(\log n)$*. A familia de circuitos é uniforme em espaço logarítmico dada a sua simplicidade e os subcircuitos são todos muito parecidos apenas dependendo dos indices* i *e* k *que se escritos em binário ocupam espaço logarítmico.*

Exercício 5.3.3 *Descreve com precisão o transdutor construído no Exemplo 5.6.*

Exemplo 5.7 (Fecho transitivo e reflexivo) *Considere-se o cálculo do fecho reflexivo e transitivo duma relação binária* R *num conjunto de* m *elementos e dada pela matriz de adjacências. A relação* $R^\star$*, fecho transitivo e reflexivo de* R*, contem os pares* (u, v) *tal que existe um caminho* R *de tamanho 0 ou mais de* u *a* v*:*

$$R^\star \quad = \quad \bigvee_{i \leq 0} R^i \quad = \quad \bigvee_{i=0}^{n} R^i \quad = \quad (R \vee I)^{n-1},$$

onde $I = R^0$ *é a matriz identidade* $m \times m$*. Notar que* $(R^i)_{uv} = 1$ *se e só se existe um caminho* $(u = v_0, v_1), \ldots, (v_{i-1}, v_i = v)$ *tal que* $(v_j, v_{j+1}) \in R$*,* $0 \leq j \leq i$*. O circuito para calcular* $R^\star$ *tem* $m^2 = n$ *entradas, uma para cada entrada da matriz de adjacências de* R*. O circuito calcula primeiro* $R \vee I$ *e depois repetidamente calcula os quadrados em* $\lceil \log m \rceil$ *vezes para obter* $R^\star$*:*

$$(R \vee I)^2,\ (R \vee I)^4,\ (R \vee I)^8, \ldots\ .$$

Cada quadrado corresponde a um circuito de profundidade $\log m$ *e tamanho polinomial como visto no exemplo anterior,* $O(m^3)$*. No total a profundidade é* $(\log m)^2$*. Temos então que* C_n *tem tamanho* $O(n^{\frac{5}{2}})$ *e profundidade* $O((\log n)^2)$*. Mais uma vez a família é uniforme em espaço logarítmico.*

Exercício 5.3.4 *Descreve com precisão o transdutor construído no Exemplo 5.7.*

5.3.3 Relação com classes de complexidade de tempo e espaço

Vamos considerar uma família de classes de complexidade para MTA's que simultaneamente consideram o tempo, espaço e alternâncias.

Definição 5.9 *A classe* $\mathrm{STA}(S(n), T(n), A(n))$ *é a classe de linguagens aceites por MTA's que são simultaneamente limitadas em espaço por* $S(n)$*, em tempo por* $T(n)$ *e que fazem no máximo* $A(n)$ *alternâncias em dados de tamanho* n*. Na descrição duma classe um* $*$ *indica que nenhum limite é imposto nessa complexidade. Escrevemos um* Σ *ou um* Π *na terceira posição para indicar se a alternância começa com um* $\vee$ *ou um* $\wedge$*, respectivamente.*

Temos

$$\begin{aligned}
\mathsf{L} &= \mathrm{STA}(\log n, *, 0),\\
\mathsf{NL} &= \mathrm{STA}(\log n, *, \Sigma 1),\\
\mathsf{P} &= \mathrm{STA}(\log n, *, *) = \mathrm{STA}(*, n^{O(1)}, 0),\\
\mathsf{NP} &= \mathrm{STA}(*, n^{O(1)}, \Sigma 1),\\
\Sigma_k^p &= \mathrm{STA}(*, n^{O(1)}, \Sigma k),\\
\Pi_k^p &= \mathrm{STA}(*, n^{O(1)}, \Pi k),\\
\mathsf{PSPACE} &= \mathrm{STA}(*, n^{O(1)}, *) = \mathrm{STA}(n^{O(1)}, *, 0).
\end{aligned}$$

Teorema 5.6 *[Ruz81]*

$$\mathsf{NC} = \mathrm{STA}(\log n, *, (\log n)^{O(1)}).$$

Dem.

$\subseteq$ Temos que construir um transdutor T log que com dados 1^n produz o circuito C_n. Como C_n tem um número de portas polinomial, podemos nomear cada porta com uma string de comprimento $O(\log n)$. Reservamos os nomes 1, 2, ..., n em binário para os nomes das entradas do circuito C_n. Com dados 1^n o transdutor deve produzir:

1. uma enumeração dos nomes de todas as portas do circuito;
2. para cada porta c, uma etiqueta indicando qual o tipo da porta: $\wedge$, $\wedge$, $\neg$, ou uma entrada.

3. a lista das ligações (c, d) no circuito, onde c e d são nomes legais de portas enumeradas antes;
4. para uma das portas c, uma etiqueta indicando que é a porta de saída.

Supomos que as portas $\wedge$ e $\vee$ têm exactamente duas entradas (fan-in 2), as portas $\neg$ exactamente uma e as entradas não têm nenhuma. Circuitos con *fan-in* maiores podem ser simulados com um aumento de profundidade $O(\log n)$ e no máximo o dobro do tamanho. Podemos ainda supor que só existem portas $\neg$ possivelmente imediatamente a seguir às entradas.

Desenhamos uma MTA N limitada em espaço logarítmico que simula esta família de circuitos. Com dados x, $|x| = n$, N usa T para produzir C_n e depois avalia $C_n(x)$. Primeiro N, executa T para encontrar o nome da porta de saída e o seu tipo, e escreve-os na fita de trabalho. Suponhamos que algum processo de N tem o nome e o tipo de uma porta d escritos na fita de trabalho.

- Se d é uma porta $\wedge$ ou $\vee$, começa T desde o início, enumerando as ligações até encontrar duas ligações (c, d) e (c', d) incidentes em d. Quando encontrar faz uma ramificação $\wedge$ (universal) ou $\vee$ (existencial) de acordo com o tipo de d. Cada um dos dois subprocessos considera c ou c' e repete
- Se d é uma entrada do circuito $1 \leq d \leq n$, N aceita ou rejeita consoante d-ésimo bit dos dados x é 1 ou 0, respectivamente.
- se d é uma porta $\neg$, N executa M para procurar uma entrada c tal que (c, d) seja a única ligação que chega a d. Aceita ou rejeita consoante o c-ésimo bit de x é 0 ou 1, respectivamente.

A máquina N não necessita de mais do que espaço logarítmico para cada uma das tarefas, porque apenas necessita em cada instante de guardar o nome da porta corrente c que está a visitar. O número de alternâncias que N faz está limitado pela profundidade do circuito que está a simular e que é $O(\log n)^{O(1)}$.

$\supseteq$ Suponhamos que N é uma MTA limitada em espaço logarítmico que faz no máximo $(\log n)^c$ alternâncias com dados de tamanho n. Temos de construir uma família uniforme de circuitos em espaço logarítmico de profundidade polylog e tamanho polinomial e que simule N. Podemos supor que N executa em tempo polinomial. Codificando as configurações de N com strings dum alfabeto Δ, uma configuração atingível da configuração inicial com dados de tamanho n pode ser codificada por uma string de tamanho $O(logn)$ e existem n^c configurações desse tipo, para algum $c > 0$.

Para dados $x \in \{0,1\}^\star$, podemos representar a relação $\xrightarrow{N}$ por uma matriz Booleana R_x de dimensão $n^c \times n^c$. As linhas e as colunas de R_x são indexadas por configurações e temos que

$$R_x(\alpha, \beta) = 1 \Leftrightarrow \alpha \xrightarrow{N} \beta.$$

O circuito C_n calcula em primeiro lugar as entradas de R_x. As codificações dos pares de configurações (α, β) podem ser usados como nomes para as portas e assim obter uma familia uniforme de circuitos em espaço logarítmico. Uma máquina limitada em espaço logarítmico pode comparar α com β e determinar se $\alpha \xrightarrow{N} \beta$, e em caso afirmativo gerar um circuito trivial que tenha como saída o valor de $R_x(\alpha, \beta)$. Isto depende também do símbolo dos dados x que está a ser visitado em α, seja i, e portanto o circuito tem de aceder ao i-ésimo valor de entrada de C_n. A profundidade do circuito construído até agora é 1. Depois de calcular R_x, C_n constrói as matrizes para as seguintes relações:

$$\begin{aligned} S_x &= \{(\alpha, \beta) \mid R_x(\alpha, \beta) = 1 \text{ e } t(\alpha) = t(\beta)\} \\ T_x &= \{(\alpha, \beta) \mid R_x(\alpha, \beta) = 1 \text{ e } t(\alpha) \neq t(\beta)\} \end{aligned}$$

e ainda $S_x^\star$ o fecho reflexivo e transitivo de S_x e $S_x^\star T_x$. Um par $(\alpha, \beta) \in S_x^\star$ se existe um caminho de computação em N de comprimento zero ou maior que com dados x vai de α a β, $\alpha \xrightarrow{N}{}^{l} \beta$, tal que todas as configurações ao longo do caminho têm o mesmo tipo. Analogamente, um par $(\alpha, \beta) \in S_x^\star T_x$, se existe um caminho entre α e β em que todas as configurações excepto β têm o mesmo tipo.

As matrizes para S_x e T_x podem ser obtidas efectuando a conjunção componente a componente da matriz R_x com a matriz da relação

$$\{(\alpha, \beta) \mid t(\alpha) = t(\beta)\},$$

e da seu complementar, respectivamente.

As matrizes de $S_x^\star$ e $S_x^\star T_x$ podem ser calculadas usando as construções dos exemplos 5.6 e 5.7.

Podemos imaginar a árvore de computação de N, com dados x, $|x| = n$, dividida em $(\log n)^c$ niveis. Em cada nível, ou todas as configurações são existênciais ou universais. Numeremos os níveis 0, 1, 2, ... da base para o topo (com o topo a configuração inicial).

- uma configuração-$\vee$ (existêncial) α no nível $i+1$ é de aceitação se existe uma configuração β no nível i tal que $S_x^\star T_x(\alpha, \beta) = 1$ e β é uma configuração de aceitação.
- uma configuração-$\wedge$ (universal) α no nível $j+1$ é de aceitação se para todas as configurações- $\vee$ (existênciais) β do nível j tal que $S_x^\star T_x(\alpha, \beta) = 1$, β é uma configuração de aceitação.

O circuito calcula uma sequência de vectores Booleanos $b_0, b_1, \ldots$ cada com comprimento n^c e indexado por configurações, tal que

$$b_i(\alpha) = 1 \quad \Leftrightarrow \quad \alpha \text{ é uma configuração de aceitação do nível } i.$$

Para obter os níveis existenciais $i+1$, o circuito calcula produto de uma matriz por um vector b_i:

$$b_{i+1} \quad = \quad S_x^\star T_x b_i.$$

Para obter os níveis universais $i+1$, o circuito calcula

$$b_{i+1} \quad = \quad \neg S_x^\star T_x \neg b_i.$$

onde $\neg$ é aplicado a um vector componente a componente. Podemos supor que b_0 é o vector zero e o nível 0 é existencial. Recordando que uma configuração de aceitação é uma configuração-$\wedge$ sem sucessores e uma de rejeição é uma configuração-$\vee$ sem sucessores. Por exemplo se uma configuração-$\wedge$ no nível 1 é de aceitação se não existe um caminho de computação que leve a uma configuração-$\vee$, e neste caso $b_1(\alpha) = 1$. Por indução pode-se mostrar, que b_i estão bem definidos. O valor de saída é $b_{(\log n)^c}(\texttt{start})$, onde `start` é a configuração inicial.

Há no máximo $(\log n)^c$ níveis e cada nível requer profundidade $\log n$ e tamanho polinomial para os produtos entre matrizes e entre uma matriz e um vector e restantes operações Booleanas. Em cada nível o circuito é uniforme e pode ser gerado por um transdutor em espaço logarítmico.

□

Corolário 5.3 $\mathsf{NL} \subseteq \mathsf{NC} \subseteq \mathsf{P}$.

Exercício 5.3.5 *Mostra que* $\mathsf{NC}^1 \subseteq \mathsf{L}$.

Exercício 5.3.6 *Mostra que* $\mathsf{NL} \subseteq \mathsf{NC}^2$. *Sugestão: calcular o fecho transitivo do grafo de configurações duma MT limitada em espaço logarítmico.*

Capítulo 6

Algoritmos aleatorizados

Existem muitos problemas com algoritmos aleatorizados ou probabilísticos eficientes melhores que os algoritmos determinísticos equivalentes. Um algoritmo aleatorizado existe um passo onde é escolhida aleatoriamente uma sequência de valores $a_1, \ldots, a_l$ que podem ser considerados bits correspondentes a um lançamento de uma moeda onde 0 (coroa) tem probabilidade $\frac{1}{2}$ e 1 tem a mesma probabilidade (cara). Muitos algoritmos têm só *erros num dos casos*, i.e se os dados x pertencem à linguagem, o algoritmo aceita com grande probabilidade, mas se x não pertence à linguagem então o algoritmo rejeita sempre.

6.1 Noções básicas de probabilidades discreta

Sejam $A_i \in \mathcal{A}$, $i \in \{1, \ldots, n\}$ conjuntos disjuntos,

$$\Pr(\bigcup \mathcal{A}) = \sum_{i=1}^{n} \Pr(A_i)$$

Sendo X uma variável aleatória discreta o valor esperado $E(X)$ é

$$E(X) = \sum_{n} n \Pr(X = n)$$

Por exemplo considerando o lançamento de uma moeda , temos a variável X tal que

$$X \quad = \quad \begin{cases} 1, \text{ se sai cara,} \\ 0, \text{ se sai coroa,} \end{cases}$$

Então $E(X) = \frac{1}{2}$, que é o valor esperado de "caras"num lançamento. Sendo $f(X)$ uma função de variável aleatória X,

$$E(f(X)) = \sum_{n} \Pr(f(X) = n) = \sum_{m} f(n) \Pr(X = m)$$

O valor esperado E é linear

$$E(X_1 + \cdots + X_n) = E(X_1) + \cdots + E(X_n)$$

E por exemplo, corresponde ao valor esperado do número de caras em n lançamentos duma moeda.

A probabilidade condicional $\Pr(A \mid B)$ é a probabilidade do acontecimento A ocorrer dado que o acontecimento B ocorre,

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)},$$

e está indefinida se $\Pr(B) = 0$. O valor esperado condicional $E(X \mid B)$ é o valor esperado da variável X dado que o acontecimento B ocorre

$$E(X \mid B) = \sum_n n \Pr(X = n \mid B)$$

Se B também for uma variável aleatória Y então $E(X|Y = m)$ é uma função de variável aleatória real m. Temos que $E(X \mid Y)$ é uma variável aleatória tal que

$$\Pr(E(X \mid Y) = n) = \sum_{E(X|Y=m)=n} \Pr(Y = m)$$

E $E(E(X|Y)) = E(X)$.

Um conjunto de acontecimentos $\mathcal{A}$ é independente se para qualquer $\mathcal{B} \subseteq \mathcal{A}$,

$$\Pr(\bigcap \mathcal{B}) = \prod_{A \in \mathcal{B}} \Pr(A)$$

E é dois a dois independente se para todo $A, B \in \mathcal{A}$

$$\Pr(A \cap B) = \Pr(A) \Pr(B).$$

Por exemplo, a probabilidade de dois lançamentos duma moeda serem cara é $\frac{1}{4}$.

Finalmente temos geralmente

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)$$

que generaliza dada um conjunto $\mathcal{A}$ de conjuntos ,para o principio da inclusão-exclusão

$$\Pr(\bigcup A) = \sum_{A \in \mathcal{A}} \Pr(A) - \sum_{\mathcal{B} \subseteq \mathcal{A}, |\mathcal{B}|=2} \Pr(\bigcap \mathcal{B}) - \cdots + / - \Pr(\bigcap \mathcal{A}).$$

6.2 Máquinas de Turing Probabilísticas

Vamos considerar máquinas de Turing determinísticas mas em que certos pontos a computação pode fazer o lançamento de uma moeda e fazer tomar uma decisão binária consoante o resultado. A probabilidade de aceitar é a probabilidade do seu caminho de computação, dirigido pelos resultados dos lançamentos, leve a um estado de aceitação.

Definição 6.1 *Uma máquina de Turing probabilistica (PTM) é uma MT deterministica*

$$M = (S, \Sigma, \Gamma, \Box, \rhd, \delta, s_0, s_y, s_n),$$

com uma fita extra de leitura (semi-infinita) que contem uma string binária chamados bits aleatórios *e normalmente representada por y. Com dados x, a computação é definida como para uma MT e o resultado,* aceitação *ou* rejeição *é designado por $M(x, y)$.*

Dado uma PTM M definimos $T_M(n)$ e $S_M(n)$ de modo análogo excepto que consideramos quaisquer dados x de tamanho n e qualquer string de bits aleatórios. Dizemos que M é limitada em tempo por $T(n)$ se $T_M(n) = O(T(n))$ e o mesmo para o espaço.

Neste modelo, a probabilidade de um acontecimento é medida em relação à distribuição uniforme no espaço de todas as sequências de bits aleatórias. Esta é a medida que resultaria se uma moeda fosse lançada um número infinito de vezes, sendo o i-ésimo bit o que correspondia ao i-ésimo lançamento.

Como se consideram computações limitadas no tempo, a máquina só pode consultar um número finito de bits. Sendo M uma PTM limitada em tempo por $T(n)$ então a probabilidade de aceitar os dados x, $|x| = n$ é

$$\Pr_y(M(x, y) \text{ aceita}) = \frac{|\{y \in \{0,1\}^k \mid M(x, y) \text{ aceita}\}|}{2^k},$$

onde $\Pr_y(A)$ é a probabilidade do acontecimento A com uma string de bits y escolhida uniforme e aleatoriamente entre as strings de comprimento k, com $k > |T(n)|$[1]. Em geral a aleatoridade pode ser vista como um recurso como o tempo ou o espaço.

Sendo $T : \mathbb{N} \longrightarrow \mathbb{N}$, podemos definir a classe RTIME($T(n)$) como o conjunto das linguagens $L \in \{0,1\}^\star$ tal que uma PTM M decide L em tempo limitado por $T(n)$ se para todo o x:

- $x \in L$, então $\Pr_y(M(x, y) \text{ aceita}) \geq \frac{3}{4}$;
- $x \notin L$, então $\Pr_y(M(x, y) \text{ aceita}) = 0$.

Em vez de $\frac{3}{4}$ pode ser qualquer valor $\frac{1}{2} + \varepsilon < 1$, com $\varepsilon > 0$.

Define-se a classe $\mathsf{RP} = \bigcup_{c>0} \text{RTIME}(n^c) = \text{RTIME}(n^{O(1)})$, "tempo aleatorizado polinomial". Esta classe corresponde a algoritmos com *erros unilaterais*. Temos $\mathsf{coRP} = \{L \mid L^c \in \mathsf{RP}\}$.

[1] Qualquer comprimento serve desde que seja suficientemente grande.

A classe de problemas com algoritmos que erram bilateralmente, BPTIME($T(n)$), é definida como o conjunto das linguagens $L \in \{0,1\}^\star$ tal que uma PTM decide L em tempo limitado por $T(n)$ se:

- $x \in L$, então $\Pr_y(M(x,y) \text{ aceita}) \geq \frac{3}{4}$;
- $x \notin L$, então $\Pr_y(M(x,y) \text{ aceita}) \leq \frac{1}{4}$,

onde em vez de $\frac{1}{4}$, pode ser qualquer valor $0 < \frac{1}{2} - \varepsilon$, com $\varepsilon > 0$.

Equivalentemente $L \in \text{BPTIME}(T(n))$ se

$$\Pr_y(M(x,y) \text{ errar em decidir } x \in L) \leq \frac{1}{4}.$$

Definimos a classe $\mathsf{BPP} = \bigcup_{c>0} \text{BPTIME}(n^c) = \text{BPTIME}(n^{O(1)})$ ("bounded error probabilistic polynomial")

Podemos considerar uma definição alternativa da classe BPP usando máquinas de Turing determinísticas onde as sequências de bits aleatórios necessários em cada passo são dadas como dados adicionais.

Definição 6.2 *Uma linguagem $L \subseteq \{0,1\}^\star$ está em BPP se existe uma MT M limitada em tempo polinomial e um polinómio $p : \mathbb{N} \longrightarrow \mathbb{N}$ tal que para todo o $x \in \{0,1\}^\star$,*

$$\Pr_{y \in_R \{0,1\}^{p(|x|)}}(M(x,y) \textit{ errar em decidir } x \in L) \quad \leq \quad \frac{1}{4}.$$

i.e. onde $\Pr_{y \in_R A}(E)$ é a probabilidade do acontecimento E considerando todas as strings y escolhidas uniforme e aleatóriamente do conjunto A.

Exercício 6.2.1 *Mostra que $\mathsf{BPP} \subseteq \mathsf{EXP}$. Sugestão: usa a Definição 6.2 e verifica que em tempo $2^{n^{O(1)}}$ pode-se enumerar todas a escolhas aleatórias.*

Temos as seguintes inclusões $\mathsf{P} \subseteq \mathsf{RP} \subseteq \mathsf{NP}$ e $\mathsf{RP} \subseteq \mathsf{BPP}$ (que não sabemos se são próprias). Podemos também ver que BPP é fechado para a complementação. E também $\mathsf{RP} \cap \mathsf{coRP} \subseteq \mathsf{BPP}$.

Exercício 6.2.2 *Mostra a inclusão anterior. A classe $\mathsf{RP} \cap \mathsf{coRP} = \mathsf{ZPP}$ das linguagens com* zero erros*. Explica porquê.*

Podemos ainda considerar a classe PP das linguagens aceites por MTNs polinomiais que aceitam se e só se pelo menos $\frac{1}{2}$ das possíveis computações aceitam (por maioria)[2]; e rejeitam caso contrário. Temos que $\mathsf{PP} = \mathsf{coPP}$ e $\mathsf{BPP} \subseteq \mathsf{PP}$. A estrutura destas classes é apresentada na Figura 6.1, onde os tracejados correspondem a classes semânticas, isto é que não são caracterizadas por um modelo de computação sintáctico (como P ou NP).

Exercício 6.2.3 *Mostra que $\mathsf{BPP} \subseteq \mathsf{P}_{/\mathsf{POLY}}$.*

[2]Corresponde a usar exactamente $\frac{1}{2}$ na definição de BPP

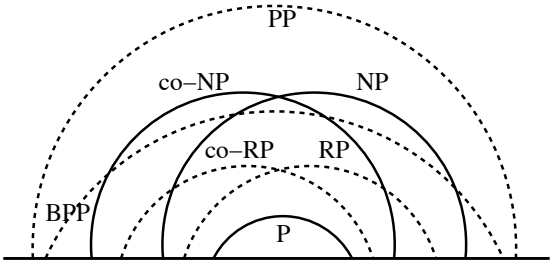


Figura 6.1: Estrutura das classes aleatorizadas

6.3 Exemplos de algoritmos aleatorizados

6.3.1 Procura da mediana

Dado um conjunto de números $\{a_1, \ldots, a_n\}$, a mediana é um valor x tal que

$$\begin{aligned} |\{a_i \mid a_i \leq x\}| &\geq \lfloor \frac{1}{2} \rfloor \\ |\{a_i \mid a_i \geq x\}| &\geq \lfloor \frac{1}{2} \rfloor \end{aligned}$$

Um algoritmo trivial é ordenar os valores e considerar o $\lfloor \frac{1}{2} \rfloor$-ésimo menor. Mas a complexidade em tempo é $O(n \log n)$. Existem outros algoritmos lineares mas são complicados. O algoritmo seguinte é aleatorizado e determina não mediana mas o k-ésimo menor número de um conjunto de n elementos. Supomos uma rotina RANDOM(S) que escolhe um elemento do conjunto S.

procedure FINDKTHELEM($k, a_1, \ldots, a_n$)
  $i \leftarrow$ RANDOM($\{1, \ldots, n\}$)
  $x = a_i$
  $m = 0$
  for all $y \in \{a_1, \ldots, a_n\}$ **do**
    if $y \leq x$ **then**
      $m \leftarrow m + 1$
    end if
  end for
  if $m = k$ **then**
    return x
  else if $m > k$ **then**
    $S \leftarrow \{a_i \mid a_i \leq x\}$
    FINDKTHELEM(k, S)
  else

$S \leftarrow \{a_i \mid a_i > x\}$
FINDKTHELEM$(k - m, S)$
end if
end procedure

Proposição 6.1 *Para* FINDKTHELEM*$(k, a_1, \ldots, a_n)$, seja $T(k, a_1, \ldots, a_n)$ o número esperado de passos que o algoritmo executa. Seja $T(n) = \max T(k, a_1, \ldots, a_n)$, então $T(n) = O(n)$.*

Exercício 6.3.1 *Mostra a proposição anterior. Sugestão: Supondo que todas as instruções não recursivas podem ser executadas em tempo linear, cn para algum $c > 0$, prova por indução que $T(n) \leq 10cn$.*

6.3.2 Testes probabilísticos com polinómios

Considera o seguinte problema

Teste de Polinómio Identidade (PIT)

Instância: Dada uma função polinomial muli-variável $p(x_1, \ldots, x_n)$ com coeficientes inteiros.

Questão: Determinar se p é identicamente zero.

Podemos assumir que p é dado por um programa *straight-line* com operações $+$, $\cdot$ e operações sobre escalares, ou equivalentemente um circuito aritmético (onde as operações lógicas são substituídas por aritméticas. As origens são x_1, $\ldots$, x_n e a saída é o valor. Por exemplo,

$$\begin{aligned} P_1 &:= x_1 - x_2, \\ P_2 &:= x_1 + x_2, \\ P_3 &:= P_1 \cdot P_2. \end{aligned}$$

Podemos determinar se $p(x_1, \ldots, x_n) = 0$ multiplicando os factores até ficar numa soma de termos e determinar se todos os termos se cancelam. Mas isto conduz a um algoritmo exponencial. Nota que um circuito muito compacto pode representar polinómios com um grande número de termos. Por exemplo o polinómio $\prod_{i+1}^{n}(1 + x_i)$ pode ser calculado com um circuito de tamanho $2n$ mas tem 2^n termos. Em alternativa, podemos avaliar p para valores a_1, $\ldots$, a_n escolhidos aleatoriamente de um conjunto suficientemente grande. Se p for 0 então $p(a_1, \ldots, a_n) = 0$. Senão, $p(x_1, \ldots, x_n) \neq 0$, com grande probabilidade. Isto porque o conjunto de valores a_1, $\ldots$, a_n tal que $p(a_1, \ldots, a_n) = 0$. é esparso. Este resultado é independente do corpo sobre o qual se considera o polinómio.

Teorema 6.1 (Schwartz-Zippel) *Seja $p(x_1, \ldots, x_n)$ um polinómio não nulo de grau d multivariável com coeficientes num corpo $\mathbb{F}$. Para qualquer $S \subseteq \mathbb{F}$, se p é avaliado num elemento $(s_1, \ldots, s_n)$ escolhido uniforme e aleatoriamente de S^n, então*

$$\Pr(p(s_1, \ldots, s_n) = 0) \leq \frac{d}{|S|}.$$

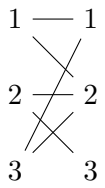
Se tivermos um circuito aritmético podemos considerar que ele define um polinómio, sendo as origens os valores de entrada. Um circuito de tamanho m tem no máximo m multiplicações e portanto define um polinómio de grau no máximo 2^m. Então teríamos o seguinte algoritmo:

for all $i \in \{1, \ldots, n\}$ **do**
  $x_i \leftarrow$ RANDOM$(\{1, \ldots, 10 \cdot 2^m\})$
end for
$y \leftarrow C(x_1, \ldots, x_n)$
if $y = 0$ **then**
  return sim
else
  return não
end if

Se o polinómio definido por C for identicamente zero então o algoritmo retornaria "sim". Pelo Teorema 6.1, caso contrário o algoritmo rejeita com probabilidade pelo menos $1 - \frac{2^m}{10 \cdot 2^m} = \frac{9}{10}$. Contudo, este algoritmo tem um problema. Como grau do polinómio pode ser muito alto (2^m) o resultado y e os valores intermédios podem ser muito grandes, até $(10 \cdot 2^m)^{2^m}$. Para resolver o problema, consideramos artimética modular e avaliamos o circuito, dados $a_1, \ldots, a_n$ módulo um número primo k escolhido aleatóriamente entre $\{1, \ldots, 2^{2m}\}$. Obtemos então $y(\mod k)$. Claramente se $y = 0$, então também o é $y(\mod k)$. Caso contrário, podemos garantir que se $y \neq 0$, então com uma probabilidade pelo menos $\delta = \frac{1}{4m}$, k não divide y (e podemos repetir $O(\frac{1}{\delta})$ vezes e só a aceitar se de todas as vezes o resultado for sim).

Emparelhamento perfeito num grafo bipartido

Seja $G = (V_1 \cup V_2, E)$, com $E \subseteq V_1 \times V_2$ um grafo bipartido e $|V_1| = |V_2| = n$. Um *emparelhamento perfeito* (*Perfect Matching*, PM) para G é um subconjunto $M \subseteq E$ tal que qualquer vértice de V aparece exactamente uma vez em M. Existem algoritmos polinomiais para este problema , mas não se sabe se está em NC. Contudo, existe um algoritmo aleatorizado em NC. Podemos ver M como uma permutação $\pi : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$, tal que $\pi(i) = j$ se e só se $(i, j) \in M$. Podemos atribuir a cada aresta $(i, j) \in E$ uma variável x_{ij} e considerar a matriz de adjacências de G, X, mas com x_{ij} em vez de 1 (se $(i, j) \in E$). Para o grafo bipartido seguinte,



A matriz de adjacências é X=$\begin{bmatrix} x_{11} & x_{12} & 0 \\ 0 & x_{22} & x_{23} \\ x_{31} & x_{32} & 0 \end{bmatrix}$. Se calcularmos o determinante obtemos $\det X = x_{12}x_{23}x_{31} - x_{11}x_{23}x_{32}$. Isto é o determinante é um polinómio de grau n nas variáveis x_{ij}, tendo um termo para cada emparelhamento perfeito, e nenhum desses termos se calcelam. Os emparelhamentos correspodem aos seguintes grafos:

Em geral o determinante duma matriz X é

$$\det X = \sum_{\pi \in S_n} (-1)^{sgn(\pi)} \prod_{i=1}^{n} x_{i,\pi(i)}$$

onde S_n é o conjunto de todas as permutações de $\{1, \ldots, n\}$ e $sgn(\pi)$ é a paridade do número de transposições em π (pares (i, j) tal que $i < j$ mas $\pi(i) > \pi(j)$. Temos então que G tem um emparelhamento perfeito se e só se $\det X$ não é um polinómio identicamente zero. Deterministicamente isto é difícil de testar porque o determinante pode ser um polinómio muito grande. Mas o determinante de uma matriz de inteiros pode ser calculado em NC usando o algoritmo de Csanky. Assim podemos ter um testes em RNC para determinar se $\det X \equiv 0$: podemos atribuir a x_{ij} elementos aleatórios escolhidos de um corpo finito (por exemplo, $\mathbb{Z}_p$ para um primo $p \geq 2n$) e perguntar se o determinante calculado para esses valores é 0. Isto acontece com a probabilidade 1 se $\det X = 0$ e com probabilidade no máximo $\frac{n}{2n} = \frac{1}{2}$ caso contrário. Tendo um teste para a existencia do PM podemos encontrar um apagando arestas uma por uma e testando a existência sem cada aresta.

Teste probabilístico para decidir se duas árvores dirigidas (não ordenadas) são isomorfas

Suponhamos que a altura das árvores é h. Associamos a cada vértice v um polinomial f_v nas variáveis $x_0, \ldots, x_h$ indutivamente do seguinte modo: Para cada folha $f_v = x_0$. Para cada nó interno v com altura k e filhos $v_1, \ldots, v_m$, $f_v = (x_k - f_{v_1})(x_k - f_{v_2}) \cdots (x_k - f_{v_m})$. O grau de f_v é igual ao número de folhas na subárvore com raiz em v. Dado a unicidade da factorização, pode ser

mostrado que duas árvores são isomorfas se e só se os polinómios associados às raízes das árvores são iguais. Temos assim um teste probabilístico eficiente para árvores (não ordenadas): testar se a diferença desses polinómios é identicamente zero, por avaliação em valores aleatórios.

6.3.3 Testes de primalidade

O Teorema de Fermat fornece uma condição necessária para que um número inteiro seja primo, isto é, se p é primo então para todo $1 \leq a < p$,

$$a^p \equiv a \mod p$$

Dado um inteiro n, podemos então dizer que a (tal que n não divide a) é uma testemunha de composicionalidade de n se $a^n \not\equiv a \mod n$. Contudo, são conhecidos números compostos para os quais o teste falha, p.e 561. Os números compostos n para os quais para todo $a < n$, $a^n \equiv a(\mod n)$ são chamados números de Carmichael. E sabemos que o seu conjunto é infinito.

Teorema 6.2 *Se p for um primo ímpar, então a equação*

$$x^2 \equiv 1 \mod p$$

tem somente duas soluções, $x \equiv 1$ e $x \equiv -1$.

Isto é se encontrarmos uma solução para a equação diferente daqueles valores então podemos concluir que n não é primo.

Teorema 6.3 *Seja p um primo ímpar e a tal que p não divide a. Podemos escrever $p - 1 = 2^k d$, onde d é ímpar. Então um das seguintes proposições é verdadeira:*

1. $a^d \equiv 1 \mod p$
2. *alguma das seguintes potências de a deve ser congruente com -1 módulo p: $a^d, a^{2d}, \ldots, a^{2^k d}$.*

Definição 6.3 (testemunha de Miller-Rabin) *Seja n um ímpar e $n - 1 = 2^k d$ com d ímpar. Um inteiro a, tal que $(a, n) = 1$ diz-se testemunha de Miller-Rabin da composicionalidade de n se:*

1. $a^d \not\equiv 1(\mod p)$
2. $a^{2^i d} \not\equiv -1 \mod p$, *para todo* $0 \leq i \leq k$.

procedure MR(a, n)
  $n - 1 = 2^k d$
  $a \leftarrow a^d \mod n$
  if $a \equiv 1 \mod n$ **then**
    return não

```
    end if
    for all i ∈ {0, ..., k − 1} do
        if a ≡ −1 mod n then
            return não
        else
            a ← a^2 mod n
        end if
    end for
    return sim
end procedure
```

Se o algoritmo retorna "sim", o número não é primo e se retornar "não"o número poderá ser primo. Dado n ímpar e $1 \leq a \leq n-1$ escolhido aleatoriamente, a probabilidade de retornar "não" é $\frac{1}{2}$ (isto é de não conseguir determinar que é composto). Se n é composto pelo menos $\frac{3}{4}$ dos $1 \leq a \leq n-1$ são testemunhas MR. Mas se o número for primo consegue sempre detectar.Assim, problema COMPOSTO está em RP, logo o PRIMO é coRP.

6.4 BPP e PH

Repetindo ensaios em computações de RP ou BPP podemos fazer com que a probabilidade de erro diminua exponencialmente.

Lema 6.1 (Amplificação) *Se $L \in$ RP então para qualquer polinómio n^d existe uma PTM M limitada em tempo polinomial tal que para todos os dados x de tamanho n*

i) se $x \in L$, $\Pr_y(M(x,y)$ aceita$) \geq 1 - 2^{-n^d}$;

ii) se $x \neq L$, $\Pr_y(M(x,y)$ aceita$) = 0$;

e se $L \in$ BPP,

$$\Pr_y(M(x,y) \text{ errar em decidir } x \in L) \leq 2^{-n^d}.$$

Dem. Para RP seja M uma PTM tal que $T_M(n) = n^c$, para $c > 0$, e tal que

- se $x \in L$, $\Pr_y(M(x,y)$ aceita$) \geq \frac{3}{4}$;
- se $x \notin L$, $\Pr_y(M(x,y)$ aceita$) = 0$;

Construímos uma PTM N que com dados x executa n^d vezes, M com dados x, usando em cada ensaio um bloco de bits novo e aceitando se qualquer ensaio aceitar. Mas M usa no máximo n^c bits aleatórios e portanto N é limitada em tempo por n^{c+d} e usa no máximo n^{c+d} bits aleatórios. Se $x \notin L$, M rejeita sempre e N também rejeita.

Se $x \in L$, a probabilidade de N errar é

$$\begin{aligned}
\Pr_{y_1,\ldots,y_{n^d}}(N(x,y_1,\ldots,y_{n^d}) \text{ rejeita}) &= \prod_{i=1}^{n^d} \Pr_{y_i}(M(x,y_i) \text{ rejeita}) \\
&= \Pr_y(M(x,y) \text{ rejeita})^{n^d} \\
&\leq 4^{-n^d}
\end{aligned}$$

Para BPP, a construção de N é igual excepto que para decidir se aceita ou rejeita, N faz n^{d+1} ensaios e escolhe a maioria dos resultados. A probabilidade de erro é a probabilidade de no máximo metade dos n^{d+1} resultados estarem corretos:

$$\begin{aligned}
\sum_{k=0}^{\frac{n^{d+1}}{2}} \binom{n^{d+1}}{k} (\frac{3}{4})^k (\frac{1}{4})^{n^{d+1}-k} &\leq 4^{-n^{d+1}} 3^{\frac{n^{d+1}}{2}} \sum_{k=0}^{\frac{n^{d+1}}{2}} \binom{n^{d+1}}{k} \\
&= 4^{-n^{d+1}} 3^{\frac{n^{d+1}}{2}} 2^{n^{d+1}-1} \\
&\leq (\frac{3}{4})^{n^{d+1}/2} \\
&\leq 2^{-\frac{n^{d+1}}{6}} \\
&\leq 2^{-n^d}
\end{aligned}$$

para n suficientemente grande. □

Teorema 6.4 $\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$.

Dem. Como BPP é fechado para o complemento basta provar que $\mathsf{BPP} \subseteq \Sigma_2^p$. Pelo Lema da Ampliação e usando a definição 6.2, existe $c > 1$ e uma MT M (deterministica) com $T_M(n) = n^c$ tal que para todos os dados x

- se $x \in L$, então $\Pr_{y \in_R \{0,1\}^{n^c}}(M(x,y) \text{ aceita}) \geq 1 - 2^{-n}$;
- se $x \notin L$, $\Pr_{y \in_R \{0,1\}^{n^c}}(M(x,y) \text{ aceita}) \leq 2^{-n}$;

Fixemos x com $|x| = n$ e seja $m = n^c$. Definimos

$$\begin{aligned}
A_x &= \{y \in \{0,1\}^m \mid M(x,y) \text{ aceita}\} \\
R_x &= \{y \in \{0,1\}^m \mid M(x,y) \text{ rejeita}\} = \{0,1\}^m \setminus A_x
\end{aligned}$$

Então para $x \in L$

$$\begin{aligned}
|A_x| &\geq 2^m - 2^{m-n} \\
|R_x| &\leq 2^{m-n}
\end{aligned}$$

e para $x \notin L$,

$$\begin{aligned} |R_x| &\geq 2^m - 2^{m-n} \\ |A_x| &\leq 2^{m-n} \end{aligned}$$

Facto $x \in L$ se e só se existem m strings $z_1, \ldots, z_m$ cada de tamanho m e tal que

$$\bigcup_{j=1}^{m} \{y \oplus z_j \mid y \in A_x\} = \{0,1\}^m.$$

onde $\oplus$ é o ou-exclusivo bit a bit (ou adição modulo 2).

A ideia é que se $x \in L$, então o conjunto A_x é tão grande que fazendo pequenas permutações da forma $y \mapsto y \oplus z$, conseguimos ter todos os elementos de $\{0,1\}^m$. E se $x \notin L$, então A_x é tão pequeno que nenhum conjunto de permutações pode fazer o mesmo.

Usando este facto, podemos mostrar que $L \in \Sigma_2^p$. Para determinar se $x \in L$,

i) adivinhar $z_1, \ldots, z_m$ usando uma ramificação existencial

ii) gerar todos os w de tamanho m usando uma ramificação universal

iii) verificar que $w \in \{y \oplus z_j \mid 1 \leq j \leq m,\ y \in A_x\}$ ou equivalentemente que $\{w \oplus z_j \mid 1 \leq j \leq m\} \cap A_x \neq \emptyset$, executando $M(x, w \oplus z_j)$ para todos $1 \leq j \leq m$ (o que pode ser feito de modo determinístico).

Temos então,

$$\exists z_1, \ldots z_m \in \{0,1\}^m \forall w \in \{0,1\}^m \bigvee_{j=1}^{m} M(x, w \oplus z_j) \text{ aceita.}$$

Vamos agora provar que o facto acima se verifica. Embora se possa mostrar usando o método probabilístico, podemos apenas considerar um argumento de contagem. Suponhamos $x \notin L$, então $|A_x| \leq 2^{m-n}$. Seja um conjunto de escolhas $Z = \{z_1, \ldots, z_m\}$, temos

$$\begin{aligned} |\bigcup_{j=1}^{m} \{y \oplus z_j \mid y \in A_x\}| &\leq \sum_{j=1}^{m} |\{y \oplus z_j \mid y \in A_x\}| \\ &= \sum_{j=1}^{m} |A_x| \\ &\leq m 2^{m-n} \\ &< 2^m, \end{aligned}$$

para n suficientemente grande. Logo,

$$\bigcup_{j=1}^{m} \{y \oplus z_j \mid y \in A_x\} \neq \{0,1\}^m.$$

Supondo que $x \in L$, então $|R_x| \leq 2^{m-n}$. Dizemos que Z é *mau* se para algum w,

$$\{w \oplus z_j \mid 1 \leq j \leq m\} \quad \subseteq \quad R_x,$$

e *bom* caso contrário. Queremos mostrar que existe um Z bom. Mas cada Z mau é determinado por um subconjunto de R_x de tamanho m, dos quais existem no máximo $(2^{m-n})^m$ e uma string $w \in \{0,1\}^m$ das quais há 2^m. Então, um limite superior do número de conjuntos Z maus é

$$(2^{m-n})^m 2^m = 2^{m^2-mn+m} < \binom{2^m}{m},$$

para n suficientemente grande, sendo o último valor o número de conjuntos $Z \in \{0,1\}^m$ com $|Z| = m$. Donde, tem de haver um conjunto bom. $\square$

Não se conhecem problemas completos para BPP dado a definição desta classe ser semântica e não sintáctica como as classes NP e P. Isto é não se pode construir uma máquina de Turing que aceite com uma certa probabilidade e portanto não se pode simular computações como se faz para as MT ou MTN.

Capítulo 7

Sistemas Interactivos de Prova

Os sistemas interactivos de prova correspondem a uma contraparte probabilística da classe NP. Correspondem a protocolos interactivos entre dois agentes, onde um pretende fornecer informação ao outro e outro pretende convencer-se com grande probalididade que a informação está correcta. São especialmente úteis em criptografia, autenticação de mensagens e em algoritmos de aproximação.

Enquanto a classe NP pode ser vista como a classe de problemas com provas pequenas a classe IP vai ser a classe de problemas/teoremas com provas interactivas eficientes. No caso de linguagens em NP podemos considerar que existe um demonstrador que encontra provas de pertença a uma linguagem e um verificador polinomial que as verifica No caso do problema SAT, o demonstrador envia uma atribuição de valores às variáveis e o verificador verifica se a fórmula é satisfeita. No entanto, já vimos que parece mais difícil convencer o verificador (polinomial) que uma fórmula não é satisfazível. Vamos ver que adicionando aleatoriedade e interactividade podemos ter modelos mais potentes.

7.1 Classe IP

Definição 7.1 (Sistema de Prova Interactivo) *Um* sistema de prova interactivo *é constituído por duas máquinas de Turing determinísticas MT independentes: o demonstrador (P) e o verificador (V). As máquinas partilham uma fita de leitura e uma fita de comunicação de leitura e escrita. Cada uma tem uma fita de trabalho (leitura/escrita) privada. Além disso*

- *V tem acesso a uma string privada de bits aleatórios*
- *V executa em tempo polinomial*
- *P não tem limitações de tempo ou espaço, mas tem de parar para todos os dados e só pode escrever strings de tamanho polinomial (no tamanho dos dados) na fita de comunicação.*

Na definição anterior, P pode ser visto como um oráculo que calcula uma função.

O protocolo de (P, V) é o seguinte: as duas máquinas alternam. Quando é a vez de V, V executa em tempo polinomial, podendo aceder à string the bits aleatórios para fazer uma escolha probabilística. Se escrever uma mensagem para P, na fita de comunicação, entra num estado que causa o controlo a ser transferido para P. Nesse caso P pode correr um tempo arbitrário, mas finalmente terá de escrever uma mensagem para V o que faz que o controlo passe para V. O controlo pode alternar entre V e P (voltas) um número polinomial de vezes até V decidir aceitar ou rejeitar.

P pode ser visto como tentando convencer V que os dados de entradas verificam uma propriedade. E, V tentar verificar que a informação dada por P está correcta. Se os dados verificam a propriedade então deve ser possível a P convencer V que isso é verdade com alta probabilidade. Se V aceitar é porque está convencido que a propriedade é verdade. Suponhamos que se substituí P por um P' desonesto. Suponhamos que V executa a sua parte do protocolo como antes, mas que P' pode ter um comportamento diferente de P e pode tentar convencer V que a propriedade se verifica mesmo que não seja verdade. O verificador V devia ser capaz de detectar este comportamento desonesto com grande probabilidade e rejeitar

Definição 7.2 *Um protocolo* (P, V) *é um* sistema interactivo de prova*para uma linguagem* $L \subseteq \Sigma^\star$ *se para* $x \in \Sigma^\star$,

i) Se $x \in L$, $Pr_y((P, V)$ *aceita* $x) \geq \frac{3}{4}$

ii) Se $x \notin L$, *para qualquer* P', $Pr_y((P', V)$ *aceita* $x) \leq \frac{1}{4}$,

onde a probabilidade é em relação a bits aleatórios y escolhidos uniforme e aleatoriamente entre todas as strings aleatórias que excedem o limite polinomial de V, *i.e* $|y| \geq n^c$, *para algum* c.

A classe de complexidade IP é definida pelo conjunto de linguagens que têm um sistema de prova interactivo. Como para as classes aleatorizadas BPP e RP, as constantes $\frac{3}{4}$ e $\frac{1}{4}$ podem ser substituídas por valores $1 - \varepsilon$ e ε, com $\varepsilon > 0$.

Exemplo 7.1 *Vamos ver que* $SAT \in$ IP *e portanto qualquer problema em* NP. *Para* SAT *existe um protocolo com uma volta e que não usa bits aleatórios. O demonstrador quer convencer o verificador que uma fórmula proposicional é satisfazível. O demonstrador envia o verificador uma valorização que satisfaz a fórmula. Se ela existir o demonstrador não tem limites de complexidade e pode procurar exaustivamente por uma atribuição que satisfaça a fórmula e enviá-la a* V. *A valorização tem tamanho polinomial no tamanho da fórmula. Se não encontrar nenhuma valorização, pode enviar qualquer uma. O verificador* V *avalia a fórmula para a valorização recebida. E se a fórmula for verdade então* V *aceita, senão rejeita. Se a fórmula for satisfazível então* (P, V) *aceita com probabilidade* 1. *Por outro lado, se a fórmula não for satisfazível, nenhum demonstrador desonesto* P' *pode convencer* V *do contrário, i.e. não pode dar uma atribuição de valores que torne a fórmula verdade. Neste caso* (P', V) *aceita com a probabilidade* 0.

Exercício 7.1.1 Mostrar que $\mathsf{S}AT^c \in \mathsf{IP}$. Sugestão: considerar $\#SAT$.

Exemplo 7.2 *O problema GI do isomorfismo de grafos está em* NP *mas não se sabe se é* NP*-completo ou não . Também não se sabe se é* coNP*. Consideremos o problema complementar GI^c: não isomorfismo de grafos, i.e*

GI^c

Instância: *Dados dois grafos G e H*

Questão: *Determinar se não existe nenhum isomorfismo entre G e H.*

Vamos construir um protocolo para GI^c. Os dados são as codificações de dois grafos G e H com n vértices. O seguinte procedimento é executado k vezes:

1)
- *O verificador V escolhe aleatoriamente uma permutação de $\{1, 2, \ldots, \}$. Isto requere $O(n \log n)$ bits.*
- *Aplica a permutação aos vértices de G e de H e obtém G' e H'.*
- *Então V lança uma moeda. Se sair cara, envia G' para P; se for coroa envia H' para P.*

2) O demonstrador P verifica se o G que está na sua fita de leitura e o grafo que foi enviado por V são isomorfos, procurando exaustivamente um isomorfismo e comunicando-o a V se o encontrar ou indicando que não existe.

3) V toma a acção seguinte baseada no lançamento da moeda que fez anteriormente e na resposta do demonstrador seguindo a seguinte tabela:

Moeda	*Resposta P*	*Acção*
(G') cara	*isomorfo*	*continua*
cara	*não-isomorfo*	*rejeita*
(H') coroa	*isomorfo*	*rejeita*
coroa	*não-isomorfo*	*continua*

Se o protocolo executar k vezes V aceita, convencido com grande probabilidade que G e H não são isomorfos.

Suponhamos que G e H não são isomorfos. Como P é honesto responde isomorfo *se lhe foi passada uma permutação de G e* não-isomorfo*, caso contrário. Então, as linhas* 2 *e* 3 *da tabela nunca ocorrem. O protocolo executa a rotina k vezes e V aceita com a probabilidade* 1*.*

Por outro lado, suponhamos que G e H são isomorfos. Então V envia ou G' ou H', mas P (ou outro P') não consegue ver a diferença. Um demonstrador não honesto tem de dizer que não são isomorfos (caso V tenha lançado coroa) e que são isomorfos quando V lança cara. Mas não podem saber qual destes eventos ocorreu. Assim a probabilidade de acidentalmente escolher a resposta correcta é 2^{-k}.

7.2 IP e outras classes de complexidade

Temos as seguintes relações da classe IP com classes de complexidade temporais.

Teorema 7.1 coNP $\subseteq$ IP.

Teorema 7.2 IP = PSPACE.

7.2.1 PSPACE $\subseteq$ IP

O resultado que PSPACE $\subseteq$ IP é de Adi Shamir [Sha92] e consiste em mostrar que um problema PSPACE-completo está em IP, nomeadamente TQBF $\in$ IP. Dada uma fórmula Booleana quantificada ψ, se ψ é verdade então o demonstrador P pode convencer o verificador V desse facto com grande probabilidade se ψ for falsa então nenhum demonstrador P' pode convencer V que ψ é verdade com mais do que probabilidade negligenciável.

A prova é constituída por vários passos:

1. Mostrar como transformar uma fórmula Booleana quantificada numa forma especial *simples*
2. Transformar uma fórmula Booleana simples ψ numa expressão aritmética A substituindo os operadores Booleanos por operadores aritméticos. A expressão aritmética A representa um valor não nulo se e só se a fórmula Booleana ψ é verdade. Este passo chama-se *aritmetização*.
3. Reduzir o problema a determinar se uma expressão aritmética A é zero módulo um primo p suficientemente grande.
4. Descrever um protocolo para o demonstrador convencer o verificador com grande probabilidade de que a expressão A se anula módulo p. O protocolo tem várias etapas e é indutivo na estrutura de A.

Passo 1

Definição 7.3 *Uma fórmula Booleana quantificada é* simples *se estiver em forma normal negativa (negações só de variáveis) e para cada subfórmula $\exists x C(x)$ ou $\forall x C(x)$ qualquer ocorrência livre de x em $C(x)$ ocorre no âmbito de no máximo um quantificador universal $\forall y$ em $C(x)$. Por outras palavras, existe no máximo um $\forall y$ entre qualquer ocorrência duma variável e a sua quantificação.*

Exemplo 7.3 *Considera as seguintes fórmulas:*

$$\forall x_1 \exists x_2 \exists x_3 [(x_1 \vee x_2) \wedge (\forall x_4 (x_1 \wedge x_3 \wedge x_4)] \tag{7.1}$$

$$\forall x_1 \forall x_2 [(x_1 \wedge x_2) \wedge \forall x_3 (\neg x_1 \wedge x_3)] \tag{7.2}$$

A primeira fórmula 7.1 é simples mas a segunda fórmula não, dado que a segunda ocorrência de x_1 está separada de seu quantificadore tanto

Lema 7.1 *Qualquer formula Booleana quantificada de tamanho n é equivalente a uma simples cujo tamanho é polinomial em n.*

Dem. Podemos mover as negações para dentro usando as Leis de De Morgan:

$$\begin{aligned} \neg(\phi \wedge \psi) &= \neg\phi \vee \neg\psi \\ \neg(\phi \vee \psi) &= \neg\phi \wedge \neg\psi \\ \neg\exists x\psi &= \forall x\neg\psi \\ \neg\forall x\psi &= \exists x\neg\psi \end{aligned}$$

Para cada subfórmula da forma

$$\forall x C(x, y_1, \ldots, y_n) \tag{7.3}$$

onde $y_1, \ldots, y_n$ são livres em (7.3), introduzir novas variáveis $y'_1, \ldots, y'_n$ e substituir (7.3) por

$$\forall x \exists y'_1 \ldots \exists y'_n \bigwedge_{i=1}^{n} ((y_i \wedge y'_i) \vee (\neg y_i \wedge \neg y'_i)) \wedge C(x, y'_1, \ldots, y'_n) \tag{7.4}$$

e efectuar esta transformação de fora para dentro, i.e começando pelas subfórmulas maiores. Isto é para cada quantificador universal, introduz-se variáveis y_i^l que correspondem ao l-ésima nome da variável i e indica-se que x_i^l e x_i^{l-1} são equivalentes, para $l > 1$.

Por definição a fórmula resultante é simples e contem um número de variáveis quadrático (comparado com a fórmula original). □

No exemplo 7.1, a fórmula resultante da segunda fórmula seria

$$\forall x_1 \forall x_2 \exists x'_1((x_1 \wedge x'_1) \vee (\neg x_1 \wedge \neg x'_1)) \wedge (x'_1 \wedge x_2) \wedge \forall x_3(\exists x''_1((x'_1 \wedge x''_1) \vee (\neg x'_1 \wedge \neg x''_1))(\neg x''_1 \wedge x_3)]$$

Passo 2 (Aritmetização) O problema de decidir se uma fórmula Booleana quantificada é verdade ou falsa pode ser expresso usando uma expressão aritmética. Seja ψ uma fórmula Booleana quantificada simples. Transformamos ψ para uma expressão aritmética A (denominada *forma aritmética* de ψ) do seguinte modo:

- Mantemos o nome das variáveis (literais positivos) mas agora assumimos que o seu âmbito é $\mathbb{Z}$.
- Substituímos cada literal negativo $\neg x$ por $1 - x$
- Substituímos $\wedge$ por $\cdot$
- Substituímos $\vee$ por $+$

- Substituímos $\exists x$ por $\sum_{x\in\{0,1\}}$; sendo que $\sum_{x\in\{0,1\}} C(x)$ é $C(0)+C(1)$.
- Substituímos $\exists x$ por $\prod_{x\in\{0,1\}}$; sendo que $\prod_{x\in\{0,1\}} C(x)$ é $C(0)\cdot C(1)$.

Exemplo 7.4 *Considera a fórmula Booleana quantificada*

$$\psi \;=\; \forall x_1 \exists x_2 [(x_1 \wedge x_2) \vee \exists x_3 (\neg x_2 \wedge x_3)].$$

A sua aritmetização é

$$A \;=\; \prod_{x_1\in\{0,1\}} \sum_{x_2\in\{0,1\}} (x_1 \cdot x_2) + \sum_{x_3\in\{0,1\}} (1-x_2)\cdot x_3$$

que pode ser avaliada para o inteiro 2*. Sendo não nulo, isso indica que a fórmula original é verdade.*

Lema 7.2 *Uma fórmula Booleana quantificada ψ é verdade se e só se a sua forma aritmética A é não nula.*

Exercício 7.2.1 *Demonstra o Lema 7.2 por indução na estrutura de ψ.*

Passo 3 Se ψ é verdade o valor de A pode ser muito grande. No entanto podemos estimar um limite superior.

Lema 7.3 *Seja ψ uma fórmula Booleana quantificada de tamanho n. Então, o valor da sua forma aritmética A não pode exceder $O(2^{2^n})$.*

Dem. Para cada subfórmula ψ' de ψ define-se $v(\psi')$ como o valor máximo da forma aritmética de ψ' sob qualquer substituição das variáveis por valores em $\{0,1\}$. Temos que as seguintes inequações:

1. $v(x_i) = v(\neg x_i) = 1$.
2. $v(\psi' \vee \psi'') \leq v(\psi') + v(\psi'')$.
3. $v(\psi' \wedge \psi'') \leq v(\psi') \cdot v(\psi'')$.
4. $v(\exists x \psi') \leq 2v(\psi')$.
5. $v(\forall x \psi') \leq v(\psi')^2$.

É fácil de verificar que $O(2^{2^n})$ satisfaz todas as inequações. Notar que este valor não pode ser melhorado significativamente, dado que para

$$\psi \;=\; \forall x_1 \forall x_2 \cdots \forall x_{n-1} \exists x_n (x_n \vee \neg x_n)$$

temos

$$A = \prod_{x_1} \prod_{x_2} \cdots \prod_{x_n} 2 = 2^{2^n}.$$

□

Assim, como um verificador polinomial não pode tratar com números tão grandes temos que os reduzir módulo um primo pequeno.

Lema 7.4 *Seja ψ uma fórmula Booleana quantificada de tamanho n. Então existe um primo p que em binário tem tamanho polinomial em n e tal que $A \not\equiv 0 \mod p$ se e só se ψ é verdade.*

Dem. Suponhamos que A é não nula. Se fosse nula modulo todos os primos (de tamanho polinomial), pelo Teorema do resto Chinês também é zero módulo o seu produto. Mas pelo Teorema dos números primos esse produto é $\Omega(2^{2^{n^d}})$, para qualquer constante d, o que contraria o facto de A ser não zero e menor que $O(2^{2^n})$. Por outro lado, se ψ for falsa então $A = 0$ módulo qualquer primo. □

Exercício 7.2.2 *Completa a demonstração anterior, usando o facto de que $n > 30$, o produto de primos menores que n é maior que 2^n.*

Passo 4 (protocolo) A tarefa do demonstrador fica então reduzida a fornecer um primo p (com n^d bits) e um valor não nulo $a \in \mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ e convencer o verificador que:

- p é primo
- a expressão aritmética $A - a$ anula-se módulo p

O demonstrador P pode escolher um primo e enviar ao verificador V juntamente com um certificado de primalidade. Isto pode ser feito numa etapa, dado que o problema PRIMO está em NP.

Em seguida o demonstrador P tem de convencer o verificador que uma expressão aritmética A anula-se módulo p (com n^d bits). Suponhamos ainda que $p \gg n$. Para uma expressão $\sum_x B(x)$ ou $\prod_x B(x)$ a subexpressão $B(x)$ reduz-se módulo p a um polinómio em x com coeficientes em $\mathbb{Z}_p$

Exemplo 7.5 *Considera*

$$\psi \quad = \quad \forall x_1(\neg x_1 \vee \exists x_2 \forall x_3((x_1 \wedge x_2) \vee x_3),$$

Então,

$$A \quad = \quad \prod_{x_1}\left[(1-x_1) + \sum_{x_2}\prod_{x_3}((x_1 \cdot x_2) + x_3)\right],$$

temos

$$\begin{aligned}(1-x_1) + \prod_{x_3}((x_1 \cdot 1 + x_3) + (x_1 \cdot 0 + x_3)) &= \\ (1-x_1) + ((x_1 \cdot 1 + 0) + (x_1 \cdot 0 + 0))((x_1 \cdot 1 + 1) + (x_1 \cdot 0 + 1)) &= \\ (1-x_1) + x_1(x_1+2) &= x_1^2 + x_1 + 1\end{aligned}$$

O polinómio que representa $B(x)$ pode ter um grau que é exponencial em n. Considera,

$$\psi = \forall x_1 \forall x_2 \cdots \forall x_n (x_1 \vee \cdots \vee x_n),$$

cuja forma aritmética é

$$A = \prod_{x_1} \prod_{x_2} \cdots \prod_{x_n} (x_1 + \cdots + x_n).$$

O polinómio correspondente $b(x_1)$ é o produto de 2^{n-1} termos da forma $(x_1 + c)$ para $0 \leq c \leq n$ e que tem grau 2^{n-1}. No entanto para fórmulas Booleanas quantificadas simples temos que

Lema 7.5 *Se $\sum_x D(x)$ ou $\prod_x D(x)$ é uma forma aritmética derivada de uma fórmula Booleana quantificada simples ψ , então $D(x)$ é equivalente módulo p a um polinómio $d(x)$ de grau linear no tamanho de ψ e com coeficientes em $\mathbb{Z}_p$.*

Dem. Cada ocorrência de x em $D(x)$ está no âmbito de no máximo um $\prod_y$. Substituímos cada subexpressão $\prod_y E(x, y)$ por $E(x, 0) \cdot E(x, 1)$. Como todas estas subexpressões são disjuntas, isto no máximo duplica o grau do polinómio. As somas podem alterar os valores dos coeficientes mas não o grau do polinómio. Assim, a duplicação so pode ocorrer no máximo uma vez. □

Exercício 7.2.3 *Mostra que introduzindo variáveis novas o grau do polinómio no lema anterior pode ser* 3.

O seguinte lema será usado para mostrar que o protocolo é sistema de prova interactivo.

Lema 7.6 *Seja $d(x) \in \mathbb{Z}_p[x]$ um polinómio de grau no máximo n. A probabilidade de que d se anule num elemnto escolhido aleatóriamente em $\mathbb{Z}_p$ é pelo máximo $\frac{n}{p}$.*

Dem. O polinómio $d(x)$ tem no máximo n raízes em $\mathbb{Z}_p$. □

O protocolo para provar que $A \not\equiv 0 \mod p$ é o seguinte. O demonstrador P envia ao verificador V o valor a de $A(\mod p)$ e justifica a sua afirmação considerando sucessivamente subexpressões de A mais pequenas.

Suponhamos que o demonstrador P quer convencer o verificador que a expressão A se anula módulo p Nota que se A for um polinómio totalmente instanciado o seu valor a_1 pode ser calculado imediatamente por V e comparado com a. Se e só se $a_1 = a$, V para e aceita. Senão para e rejeita.

Em geral, A é da forma:

$$B\left(\prod_{z_1} D_1(z_1), \sum_{z_2} D_2(z_2), \ldots, \prod_{z_m} D_m(z_m)\right),$$

onde $\prod_{z_i} D_i(z_i)$ e $\prod_{z_i} D_i(z_i)$ são subexpressões de B da forma respectiva, i.e $B(y_1, \ldots, y_m)$ não tem ocorrências de $\prod_z$ ou $\sum_z$. Pelo Lema 7.5 cada $D_i(z_i)$ é equivalente a um polinómio $d_i(z_i) \in \mathbb{Z}_p[z_i]$ de grau baixo. O demonstrador envia a V

$$d_1(z), \ldots, d_m(z) \in \mathbb{Z}_p[z],$$

afirmando que

$$D_i(z) \equiv d_i(z) \mod p,\ 1 \leq i \leq m. \tag{7.5}$$

O verificador V calcula directamente

$$B(\prod_{z_1} d_1(z_1), \sum_{z_2} d_2(z_2), \ldots, \prod_{z_m} d_m(z_m) \equiv 0 \mod p)$$

Mas o demonstrador tem ainda de convencer o verificador que (7.5) é verdade. O verificador toma elementos aleatórios $a_i \in \mathbb{Z}_p$ e pergunta ao demonstrador para verificar

$$D_i(a_i) \equiv d_i(a_i) \mod p,\ 1 \leq i \leq m, \tag{7.6}$$

ou seja

$$D_i(a_i) - d_i(a_i) \equiv 0 \mod p,\ 1 \leq i \leq m, \tag{7.7}$$

Estámos de novo no início do Passo 4, com uma expressão estritamente mais simples.

Exercício 7.2.4 *Simula o protocolo anterior para a fórmula dada no Exemplo 7.5.*

Finalmente temos,

Teorema 7.3 *1. Se ψ é verdade e P honesto, V aceita com probabilidade* 1

2. Se ψ é falsa, V aceita com probabilidade negligenciável.

Dem. 1. Neste caso se P for honesto pode justificar todos os valores e V acaba por aceitar.

2. Um demonstrador desonesto P' indicou valores incorrectos de a e tem de fornecer polinómios incorrectos d_i. Usando o Lema 7.6, se um polinómio tiver grau t, ele só coincide com o polinómio correcto no máximo em t valores dos p valores de $\mathbb{Z}_p$. Sendo t um polinómio em e p exponencial no tamanho de ψ, é negligenciável a probabilidade de um d incorrecto gerar um valor correcto num ponto aleatório a_i escolhido por V. Como resultado, um P' desonesto é forçado a dar valores incorrectos para subexpressões cada vez menores até ser exposto com grande probabilidade quando V avaliar a subexpressão final.

□

7.2.2 IP $\subseteq$ PSPACE

Suponhamos que temos $L \in \mathsf{IP}$ e queremos mostrar que $L \in \mathsf{PSPACE}$. Seja (P, V) um sistema interactivo de prova para L. Sabemos que o protocolo, tem vários etapas. O verificador V executa em tempo polinomial, e envia uma mensagem m_1 a P e transfere o controlo para P. P pode executar o tempo que quiser mas depois envia uma mensagem l_1 a V de tamanho polinomial e V fica com o controlo. O número de etapas é polinomial, seja N e no fim V indica na sua última mensagem se aceita ($m_N = 1$) ou se rejeita $m_N = 0$. Vamos considerar que $N = n^c$ é o limite temporal de todo o protocolo (excluindo a execução de P), para dados de tamanho n e algum $c > 0$. Isto é

- $|m_i| \leq N$ e $|l_i| \leq N$
- o número de etapas é N
- V pode usar no máximo N bits aleatórios

Suponhamos que $P \in \mathsf{PSPACE}$. Neste caso é simples decidir que $L \in \mathsf{PSPACE}$. Com dados x, $|x| = n$ podemos considerar em sequência todas as strings de bits aleatórios de tamanho N e simular o protocolo inteiro para cada uma, contando o número de vezes que V aceita. Aceitamos x se esse número é pelo menos 2^{N-1}, o que garante uma probabilidade de aceitar de pelo menos $\frac{1}{2}$. Pela definição de sistema interactivo de prova isso ocorre se e só se $x \in L$. Toda a computação pode ser feita em PSPACE.

Suponhamos agora que P executa com uma complexidade temporal arbitrária e apenas sabemos que é determinístico, pára sempre e as suas mensagens são polinomiais. P pode ser visto como uma função que considera a história de mensagens $m_1, \ldots, m_k$, recebidas previamente de V, e os dados x e produz uma nova mensagem,

$$l_k = P(x, m_1, \ldots, m_k)$$

que envia a V. Como existe um protocolo que satisfaz as condições i) e ii), podemos considerar um demonstrador que escolhe as suas mensagens de modo a maximizar a probabilidade de V aceitar. As condições i) e ii), mantêm-se válidas se se usar esta estratégia optimizada P_{opt} que podemos mostrar que pode ser calculada em PSPACE.

Vamos assumir que todas as mensagens do demonstrador são de 1 bit[1]. Então o demonstrador pode ser visto como um oráculo que sempre que questionado numa string

$$x\#m_1\#\cdots\#m_k,$$

retorna um bit

$$l_k = \begin{cases} 1 & \text{se } x\#m_1\#\cdots\#m_k \text{ está na linguagem do oráculo;} \\ 0 & \text{caso contrário.} \end{cases}$$

[1] corresponde a enviar mensagens bit a bit para o verificador

Por questões técnicas também se assume que cabeça da fita de bits aleatórios só se move para a direita (i.e cada bit é lido no máximo uma vez por V). Caso V precise do bit, pode guardá-lo na fita de trabalho.

O protocolo (P_{opt}, V) pode ser descrito por uma árvore de computação T. Os vértices de T são etiquetados pelas configurações que descrevem:

- o estado de V,
- o contéudo e posições da cabeça da fita de leitura e de trabalho de V,
- a fita de comunicação.

A configuração não inclui o conteúdo da fita de bits aleatórios de V. Uma pergunta à fita de bits aleatórios é modelada por uma ramificação binária na árvore T e determinada pelo valor do bit lido (nessa chamada). Estas ramificações são denominadas *aleatórias*. A árvore T tem outro tipo de ramificação binária: a ramificação de *oráculo*, que correspondem às chamadas ao oráculo que modela as mensagens de um bit do demonstrador ao verificador.

A profundidade de T é no máximo N e as etiquetas de cada vértice são descritas por strings de tamanho N sobre um alfabeto finito Δ. Para cada estratégia P, as ramificações de oráculo estão completamente determinadas por P. Isto permite que se corte os ramos de oráculo e obtermos uma árvore só com ramificações aleatórias T_P.

A probabilidade de V aceitar dado P é a soma das probabilidades de todos os caminho de T_P que levam à aceitação, i.e $m_N = 1$.

A probabilidade de um caminho é o produto das probabilidades dos arcos ao longo desse caminho, onde a probabilidade de um arco que saí de um vértice de ramificação aleatória é $\frac{1}{2}$ e de qualquer outro vértice é 1.

Com dados x, P_{opt} deve calcular a resposta óptima l_i a cada pergunta de oráculo feita por V que maximiza a probabilidade de V aceitar.

O demonstrador P_{opt} tem conhecimento do programa de V mas não dos bits aleatórios, excepto por informações possivelmente contidas nas mensagens m_i. As mensagens m_i são variáveis aleatórias no conjunto das strings de tamanho N e dependem das ramificações aleatórias anteriores da árvore, dos dados x e das respostas anteriores do demonstrador.

Seja $Pr_{opt}(E)$ a probabilidade do evento E ocorrer, assumindo que o demonstrador se comporta sempre optimamente. Isto é, que as mensagens do demonstrador $l_1, \ldots, l_N$ são escolhidas para maximizar a probabilidade de aceitar.

Seja $Pr_{opt}(E \mid F)$ a probabilidade condicionada do evento E ocorrer, dado o evento F ocorrer assumindo que o demonstrador se comporta sempre optimamente. Temos,

$$Pr_{opt}(E \mid F) = \begin{cases} \frac{Pr_{opt}(E \cap F)}{Pr_{opt}(F)} & \text{se} Pr_{opt} \neq 0 \\ \text{indefinido} & \text{caso contrário.} \end{cases} \tag{7.8}$$

Sendo $F = \bigcup_i F_i$ e F_i eventos disjuntos temos que

$$\begin{aligned} Pr_{opt}(E \mid F) &= \sum_i Pr_{opt}(E \mid F_i) \cdot Pr_{opt}(F_i \mid F) \\ &\leq \max_i Pr_{opt}(E \mid F_i). \end{aligned}$$

A condição de aceitação é o evento

$$m_N = 1.$$

Para $y_1, \ldots, y_i \in \{0,1\}^N$ e $z_1, \ldots, z_i \in \{0,1\}$ sejam R_i e S_i os eventos

$$\begin{aligned} R_i &= \bigwedge_{j=1}^{i} m_j = y_j, \\ S_i &= \bigwedge_{j=1}^{i} l_j = z_j. \end{aligned}$$

Nota que R_i e S_i são função dos y_j e dos z_j, respectivamente. Temos

$$\begin{aligned} Pr_{opt}(m_N = 1 \mid R_{i-1} \wedge S_{i-1}) &= \sum_{y_i \in \{0,1\}^N} Pr_{opt}(m_N = 1 \mid R_i \wedge S_{i-1}) \cdot Pr_{opt}(R_i \mid R_{i-1} \wedge S_{i-1}) \quad (7.9) \\ &= \sum_{y_i \in \{0,1\}^N} \max_{z_i \in \{0,1\}} Pr_{opt}(m_N = 1 \mid R_i \wedge S_i) \cdot Pr_{opt}(R_i \mid R_{i-1} \wedge S_{i-1}) \end{aligned}$$

A última igualdade corresponde à maximização da probabilidade de aceitação.

Os valores

$$Pr_{opt}(R_i \mid R_{i-1} \wedge S_{i-1}) = Pr_{opt}(m_i = y_i \mid R_{i-1} \wedge S_{i-1})$$

podem ser calculados em PSPACE simulando a computação de V em todas as string de bits de comprimento N, fornecendo os $z_1, \ldots, z_{i-1}$ como respostas às chamadas ao oráculo, ignorando todas as computações que não gerem $y_1, \ldots, y_{i-1}$ por esta ordem e calculando a fração das computações restantes para as quais V gera $m_i = y_i$. Com este procedimento PSPACE os valores de

$$Pr_{opt}(m_N = 1 \mid R_i \wedge S_i) \quad (7.10)$$

podem ser calculados fazendo uma pesquisa em profundidade na árvore de computação e usando 7.9. A probabilidade de aceitação é

$$Pr_{opt}(m_N = 1) = Pr_{opt}(m_N = 1 \mid R_0 \wedge S_0). \quad (7.11)$$

Esta computação pode ser feita em PSPACE. O demonstrador pode calcular a sua resposta óptima em PSPACE, calculando (7.10) para $z_i \in \{0,1\}$ e escolhendo para l_i o valor que dá o máximo.

Capítulo 8

Aproximação de problemas de optimização e PCP

Num problema de optimização, cada solução tem um custo positivo associado e pretende-se obter a solução óptima. Se o problema for de maximização pretende-se o custo máximo possível, se for de minimização, o custo mínimo possível.

Como exemplo podemos considerar o seguinte problema de optimização associado ao KNAPSACK. Pretende-se maximizar o valor colocado no *knapsack* sujeito a que o tamanho total dos objectos não exceda a sua capacidade.

KNAPSACK-O

Instância: Dado um conjunto finito $U = \{u_1, \ldots, u_n\}$, uma função *tamanho* $s : U \longrightarrow \mathbb{Z}^+$, uma função *peso* $v : U \longrightarrow \mathbb{Z}^+$ e uma peso máximo $M \geq \max\{s(u) \mid u \in U\}$.

Questão: Encontrar um subconjunto $U' \subseteq U$ tal que $\sum_{u \in U'} s(u) \leq M$ e $\sum_{u \in U'} v(u)$ é máximo.

Formalmente um problema de optimização Π é constituído por:

- um conjunto de instâncias $I \in D_\Pi$;
- para cada instância $I \in D_\Pi$, um conjunto finito $S_\Pi(I)$ de soluções admissíveis para I; e
- uma função de custo (ou valor) m_Π que atribui a cada instância $I \in D_\Pi$ e cada $s \in S_\Pi(I)$ um valor racional positivo $m_\Pi(I, s)$

Se Π é de maximização o valor óptimo é

$$OPT(I) = \max_{s \in S_\Pi(I)} m_\Pi(I, s)$$

e se Π for de minimização,

$$OPT(I) = \min_{s \in S_\Pi(I)} m_\Pi(I, s).$$

Uma *solução óptima* é uma solução $s \in S_{\Pi}(I)$ tal que $m_{\Pi}(I, s) = OPT(s)$.

Para resolver um problema de optimização Π que é NP-hard uma das soluções é desistir da solução óptima e procurar um algoritmo polinomial A que dado $I \in D_{\Pi}$ calcule $s \in S_{\Pi}(I)$, tal que $A(I) = m_{\Pi}(I, s)$ se aproxime de $OPT(I)$. Um tal algoritmo diz-se de aproximação, mas pretende-se que a aproximação seja a menos de um factor garantido do óptimo, i.e. com uma determinada razão de aproximação.

Em geral, os métodos de aproximação são específicos para cada problema NP-hard e as reduções polinomiais em geral não ajudam a relacionar algoritmos de aproximação [GJ79]. Mas, a menos que P = NP podemos obter resultados sobre majorantes e minorantes das razões de aproximação e determinar resultados de não aproximação, i.e. casos que as heurísticas são tão difíceis como os algoritmos exactos [GJ79, Vaz01]. Novas caracterizações da classe NP em termos de protocolos interactivos permitiram grandes avanços em determinar quando as aproximações são tão como determinar os valores óptimos.

Definição 8.1 *Seja Π um problema de optimização e $\rho : \mathbb{N} \longrightarrow \mathbb{Q}^+$, com $\rho(n) \geq 1$, $\forall n \in \mathbb{N}$. Um* algoritmo de aproximação (ou aproximado) *A para um problema Π tem uma* razão de aproximação *$\rho(n)$ se para qualquer instância $I \in D_{\Pi}$ e $|I| = n$, o custo $A(I)$ da solução produzida pelo algoritmo está dentro dum factor $\rho(n)$ do custo $OPT(I)$ da solução óptima i.e*

$$\max\left(\frac{A(I)}{OPT(I)}, \frac{OPT(I)}{A(I)}\right) \leq \rho(n)$$

- *para* problemas de maximização $0 < A(I) \leq OPT(I)$ *e a razão $\frac{OPT(I)}{A(I)}$ indica o factor pelo qual o custo da solução óptima é maior do que a encontrada;*
- *para* problemas de minimização $0 < OPT(I) \leq A(I)$ *e a razão $\frac{A(I)}{OPT(I)}$ indica o factor pelo qual o custo da solução aproximada é maior do que a óptima.*

E ainda A executa em tempo polinomial em n. Se ρ não depender de n, usamos só ρ. Se $\rho = 1$ temos o óptimo. Por vezes para problemas de maximização usa-se $\rho < 1$.

Notar que para se obter a razão de aproximação de um algoritmo não se pode usar o custo/valor da solução óptima (dado isso ser tão difícil como resolver o problema de optimização em causa). Em alternativa utiliza-se um limite inferior do valor óptimo que possa ser calculado em tempo polinomial.

Exemplo 8.1 *Seja o problema de minimização seguinte:*

Cobertura por vértices VCO

Instância: *Dado um grafo $G = (V, E)$*

Questão: *Encontrar uma cobertura por vértices de tamanho mínimo.*

Considere-se o seguinte algoritmo aproximado que calcula um emparelhamento maximal.

procedure APPROX-VCO*(V, E)*
  $C \leftarrow 0$
  $E' \leftarrow E$
  while $E' \neq \emptyset$ ***do***
    $(u, v) \leftarrow$ *aresta de* E'
    $C \leftarrow C \cup \{u, v\}$
    remover de E' *todas as arestas incidentes em* u *ou* v
  end while
  return C
end procedure

O algoritmo APPROX-VCO é algoritmo polinomial aproximado VCO para com razão 2*. Primeiro é imediato que o algoritmo executa em* $O(V + E)$*, usando lista de adjacências. Também é fácil ver que está correcto, isto é o valor calculado* C *é uma cobertura por vértices. Falta ver que a cobertura* C *é no máximo o dobro da óptima. Seja* A *o conjunto de arestas escolhidas pelo algoritmo. Para cobrir os vértices de* A *qualquer cobertura tem de ter pelo menos um vértice de cada aresta de* A*. Por outro lado, nenhuma aresta de* A *partilha os mesmos vértices e nenhuma aresta de* A *é coberta pelo mesmo vértice em* OPT*,. Temos então que* $|OPT| \geq |A|$*. E cada aresta escolhida não tem os vértices ainda em* C*, i.e.* $|C| = 2|A|$*. Vem que* $|C| \leq 2|OPT|$*. Nota que este valor é atingido considerando grafos bipartidos completos,* $K_{n,n}$*. Por outro lado, qualquer algoritmo que considere como limite inferior o tamanho de um emparelhamento maximal, não pode ter uma razão de aproximação melhor. Isto porque para grafos completos* K_n *o tamanho de um emparelhamento maximal é* $\frac{n-1}{2}$ *e o tamanho de uma cobertura óptima é* $n - 1$*.*

Para alguns problemas de optimização, não só existem algoritmos de aproximação com uma determinada razão, mas também admitem algoritmos com razões de aproximação arbitrárias de modo podemos obter uma solução tão próxima da óptima quanto o desejado. Neste caso a razão é também um pâramtero do algoritmo.

Definição 8.2 *Um* esquema de aproximação em tempo polinomial *(PTAS) para um problema de optimização* Π *é algoritmo que para além da instância* I *têm um parâmetro* $\varepsilon > 0$*, sendo garantido que a solução aproximada tem uma razão de aproximação* $1 + \varepsilon$ *e que o tempo de execução é polinomial em* $n = |I|$*. Dizemos que um esquema de aproximação é totalmente polinomial (FPTAS) se o tempo de execução for polinomial em* $\frac{1}{\varepsilon}$ *e em* n*.*

Para o problema KNAPSACK-O não só podemos ter algoritmos de aproximação de uma razão fixa mas também esquemas de aproximação em tempo polinomial que podem ser totais.

Uma algoritmo simples de aproximação pode ser obtido do seguinte modo. Primeiro ordenar por ordem decrescente os objectos de $U = \{u_1, \ldots, u_n\}$ pela razão entre o valor e tamanho que

$\frac{v(u_1)}{s(u_1)} \geq \cdots \geq \frac{v(u_n)}{s(u_n)}$. Depois colocar sucessivamente em U' os u_i tal que $\sum_{u \in U'} s(u) \leq M - s(u_i)$. No fim compara-se o valor obtido com o valor máximo dos objectos e retorna-se o melhor. Este algoritmo tem uma razão de aproximação 2.

Exercício 8.0.1 *Analiza e verifica a afirmação anterior.*

Teorema 8.1 *Existe um PTAS para* KNAPSACK-O *de razão de aproximação* $1+\varepsilon$*, para qualquer* $\varepsilon > 0$.

Dem. Seja $V = \max_{u \in U} v(u)$. Então, nV 'r um limite superior para o peso máximo de qualquer solução. Para $i \in \{1, \ldots, n\}$ e $p \in \{1, \ldots, nV\}$ seja $U_{i,p}$ um subconjunto de $\{u_1, \ldots, u_i\}$ cujo peso total é exactamente p e cujo tamanho seja mínimo. Seja $A(i,p) = \sum_{u \in S_{i,p}} s(u)$ o tamanho do conjunto $S_{i,p}$, com $A(i,p) = \infty$ se tal conjunto não existir. O valor de $A(1,p)$ é conhecido para todo o $p \in \{1, \ldots, nV\}$. Os restantes valores podem ser calculados indutivamente do modo seguinte:

$$A(i+1,p) \quad = \quad \begin{cases} \min(A(i,p), A(i, p - v(u_{i+1})) + s(u_{i+1})) \\ A(i,p) \qquad\qquad\qquad\qquad\qquad\qquad \text{otherwise .} \end{cases}$$

Escolhe-se o valor $\max\{p \mid A(n,p) \leq M\}$. O algoritmo executa em $O(n^2V)$. Notar contudo que o algoritmo não é polinomial em $|I|$ dado V poder ser muito grande. Contudo se os valores de v forem pequenos o algoritmo pode ser polinomial. Assim podemos optimizar o tempo de execução se ignorarmos alguns bits menos significativos dos valores de v. Seja b o número de bits a ignorar. Temos então $v'(u_i) = 2^b \lfloor \frac{v(u_i)}{2^b} \rfloor$. Usando o algoritmo anterior obtemos U' em tempo $O(\frac{n^2V}{2^b})$. Vamos ver qual é a razão de aproximação em função de ε (e assim determinar b). Temos que para todo $u \in U$,

$$0 \leq v(u) - 2^b v'(u) \leq 2^b.$$

Sendo O o conjunto que corresponde á solução óptima temos

$$\sum_{i \in U'} v(u) \geq 2^b \sum_{i \in O} v'(u) \geq \sum_{i \in O} v(u) - n2^b.$$

Seja b tal que $2^b = \frac{V\varepsilon}{(1+\varepsilon)n}$. Como a solução optima $\sum_{i \in O} v(u) \geq V$, obtemos um algoritmo com razão de aproximação $1+\varepsilon$ e que executa em tempo polinomial de n e $\frac{1}{\varepsilon}$(verifica). □

No entanto existem problemas para os quais se pode provar que não existe um esquema de aproximação polinomial a menos que P = NP. Um desses casos é o problema do caixeiro viajante (geral). Seja,

(TSO) Dado um conjunto de cidades e as distâncias entre elas existe uma permutação das cidades que minimize a distância total?

Instância: $C = \{c_1, c_2, \ldots c_n\}$ e $d : C \times C \longrightarrow \mathbb{N}$ distância

Questão: Qual é a permutação $\pi : C \longrightarrow C$ que minimiza

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

Teorema 8.2 *Se* $\mathsf{P} \neq \mathsf{NP}$*, então para nenhuma constante* $\rho \geq 1$ *existe um algoritmo aproximado polinomial de razão* ρ *para o problema* TSO*.*

Dem. Por contradição. Suponhamos que para um $\rho \geq 1$, existe um algoritmo aproximado A com razão ρ e $\rho \in \mathbb{N}$. Vamos ver que A pode resolver polinomialmente o problema do ciclo Hamiltoneano, i.e. (HC) Existência de ciclo hamiltoniano

Instância: Seja $G = (V, E)$ um grafo dirigido.

Questão: Existe um ciclo $< (X_1, X_2)(X_2, X_3) \ldots (X_{n-1}, X_n)(X_n, X_1) >$, $X_i \in V$, $(X_i, X_{i+1}) \in E$ e e $n = |V|$, tal que todos os X_i são diferentes?

o que é um absurdo se $\mathsf{P} \neq \mathsf{NP}$. Seja $G = (V, E)$ uma instância de (HC). Transformámos G na instância de TSO seguinte:

- Seja $G' = (V', E')$ o grafo completo em V
- Considere-se

$$d(u, v) = \begin{cases} 1 & (u, v) \in E \\ \rho|V| + 1 & \text{caso contrário.} \end{cases}$$

É fácil ver que (G', d) é uma instância de TSO e pode ser obtida em tempo polinomial em $|V|$ e $|E|$. Se G tem um ciclo Hamiltoneano H então, cada aresta em H tem d igual a 1, e (G', d) tem uma permutação de custo $|V|$.Senão, qualquer volta em G' usa uma aresta que não está em E. Mas tal volta terá um custo pelo menos

$$\begin{aligned} (\rho|V| + 1) + (|V| - 1) &= \rho|V| + |V| \\ &> \rho|V|. \end{aligned}$$

Isto é, existe uma diferença de pelo menos $\rho|V|$ entre uma volta que é um ciclo Hamiltoneano em G ($|V|$) e um custo de qualquer outra volta ($\rho|V| + |V|$). Se aplicarmos A a (G', d), é garantido que obtemos uma volta com custo não maior que ρ vezes o óptimo. Então se G tem um ciclo Hamiltoneano, A retorna essa volta. Caso contrário, A retorna uma volta de custo maior que $\rho|V|$. $\square$

A demonstração do teorema anterior é um exemplo duma técnica geral para mostrar que não existe um esquema polinomial de aproximação de razão ρ para um dado problema. Suponhamos que dado um problema Π NP-hard podemos obter um problema de minimização Π' em P tal que as instâncias "sim"de Π correspondem a instâncias de Π' com um valor no máximo k (para algum k) e as instâncias "não "de Π correspondem a instâncias de Π' com um valor maior que ρk. Então a menos que $\mathsf{P} = \mathsf{NP}$ não existe nenhum algoritmo aproximado com razão ρ para o problema Π' que execute em tempo polinomial. Na realidade para a maior parte dos problemas de optimização NP-hard não existe um FPTAS [Vaz01].

Iremos ver que existem problemas para os quais a não aproximação é ainda mais difícil.

8.1 PCP: Provas verificáveis probabilísticamente

Uma linguagem $L \in \Sigma^\star$ tem um *prova verificável probabilísticamente* (*Probabilistic Checkable Proof*) se existe um protocolo interactivo (P, V) tal que V executa em tempo polinomial $O(n^c)$ e

i) se $x \in L$ então $Pr_y((P, V) \text{ aceita } x) = 1$

ii) se $x \neq L$ então para qualquer P', $Pr_y((P, V) \text{ aceita } x) < \frac{1}{2}$

onde y escolhido uniforme e aleatoriamente e $|y| \geq n^c$, para algum c. E $\frac{1}{2}$ pode ser substituído por $\varepsilon \geq 0$.

Em cada passo, podemos supor que P é função de $(x, m_1, \ldots, m_k)$ onde m_i são as mensagens anteriores de V e que a mensagem de P, l_{k+1} é um bit, Podemos ver P como um oráculo que codifica uma string binária da prova que $x \in L$ e as perguntas de V extraem esses bits. Dizemos que (P, V) é $(r(n), q(n))$-*limitado* se para dados de tamanho n o verificador usa $r(n)$ bits aleatórios e faz no máximo $q(n)$ perguntas ao oráculo ao longo de qualquer caminho de computação.

Definição 8.3 *A classe* $\mathsf{PCP}(r(n), q(n))$ *é o conjunto de linguagens com protocolos* $(O(r(n)), O(q(n)))$-*limitados.*

Podemos ver algumas relações das classes PCP com outras classes de complexidade.

$\mathsf{NP} = \bigcup_{c \geq 0} \mathsf{PCP}(0, n^c)$ porque o verificador pode ler deterministicamente a testemunha polinomial de que $x \in L$

$\mathsf{NP} = \bigcup_{c \geq 0} \mathsf{PCP}(\log n, n^c)$ Com P fixo, podemos enumerar as strings aleatórias de tamanho logarítmico do verificador e calcular a probabilidade de aceitar deterministicamente.

$\mathsf{NEXP} = \bigcup_c \mathsf{PCP}(n^c, n^c)$ ver [BFL91].

$\mathsf{NP} \subseteq \bigcup_i \mathsf{PCP}(\log n^i, \log n^i)$ ver[BFLS91].

O teorema seguinte foi o culminar do trabalho de diversos autores e é um resultado surpreendente e que permitiu uma nova caracterização da classe NP.

Teorema 8.3 ([ALM+98]) *Teorema* PCP

$$\mathsf{NP} = \mathsf{PCP}(\log n, 1).$$

Se $L \in \mathsf{NP}$ então L tem um protocolo interactivo que usa no máximo $O(\log n)$ bits aleatórios e pergunta um número constante de bits da prova independentemente do tamanho dos dados. Uma prova mais simples do teorema PCP é a de Dinur [Din05].

8.1.1 PCP e a dificuldade de aproximação

Os resultado de dificuldade de aproximação (ou não aproximação) dividem-se em três classes principais:

- Problemas não aproximáveis com razão $(1+\varepsilon)$ (constante), para $\varepsilon > 0$. Como exemplo iremos ver MAX-3SAT e MAX-3ESAT (já vimos TSP-O). E também VC-O.
- Problemas não aproximáveis com razão $O(\log n)$.
- Problemas não aproximáveis com razão n^{ε}, $\varepsilon > 0$. Iremos ver MAX-CliQUE. Mas também MAX-INDEP-SET e COLOR.

Começamos por considerar as seguintes variantes de optimização do problema SAT, onde se pretende encontrar subconjuntos de cláusulas satisfazíveis.

MAX-3SAT

Instância: Seja $C = \{c_1, c_2, \ldots, c_m\}$ um conjunto de cláusulas com literais num conjunto finito $U = \{u_1, u_2, \ldots, u_n\}$ onde cada c_i tem exactamente três literais

Questão: Encontrar uma atribuição de verdade que maximize o número de cláusulas satisfeitas.

e MAX-E3SAT

Instância: Seja $C = \{c_1, c_2, \ldots, c_m\}$ um conjunto de cláusulas com literais num conjunto finito $U = \{u_1, u_2, \ldots, u_n\}$ onde cada c_i tem exactamente três variáveis distintas

Questão: Encontrar uma atribuição de verdade que maximize o número de cláusulas satisfeitas.

Começamos por mostrar que podemos aproximar MAX-E3SAT com razão 8/7. Notar que se em cada cláusula houver variáveis repetidas o teorema não é válido. Por exemplo, para

$$(x \vee y \vee y) \wedge (x \vee \overline{y} \vee \overline{y}) \wedge (\overline{x} \vee y \vee y) \wedge (\overline{x} \vee \overline{y} \vee \overline{y})$$

não é possível satisfazer mais que $\frac{3}{4}$ das cláusulas.

Teorema 8.4 *Existe um algoritmo polinomial que dado uma conjunto de cláusulas de 3SAT com n variáveis e m cláusulas cada uma com 3 variáveis distintas que encontra uma valorização que satistaz $\frac{7m}{8}$ das cláusulas.*

Dem. Escolher aleatóriamente uma atribuição de verdade $r_1, \ldots r_n$ para as variáveis $u_1, \ldots, u_n$ lançando independentemente uma moeda para cada variável. Sejam S_i e S as variáveis aleatórias

$$\begin{aligned} S_i &= \begin{cases} 1, & r_1, \ldots r_n \text{ satisfaz } i \\ 0, & \text{caso contrário} \end{cases} \\ S &= S_1 + \cdots + S_m \end{aligned}$$

onde S é o número de cláusulas safisfeitas. Temos que $E(S_i) = Pr(S_i = 1) = 1 - \frac{1}{8} = \frac{7}{8}$, logo $E(S) = \frac{7m}{8}$. Então existe uma atribuição de verdade que satisfaz pelo menos $\frac{7}{8}$ das cláusulas. Para as obter podemos usar um algoritmo goloso. Atribuímos valores de verdade a $u_1, \ldots, u_n$ por esta ordem. Seja $a_1, \ldots, a_{k-1}$ uma atribuição parcial de $u_1, \ldots, u_{k-1}$. Calcula-se o número esperado de cláusulas safisfeitas se $u_k = 0$ e os restantes valores escolhidos aleatoriamente, E o mesmo para $u_k = 1$. Seja a_k o valor de x_k que dá o máximo. Seja o evento $E_k = \bigwedge_{i=1}^{k} r_i = a_i$. Então $E(S \mid E_k)$ é o número esperado de cláusulas satisfeitas atribuindo $a_1, \ldots, a_k$ a $u_1, \ldots, u_k$ e o restante aleatório. Mostramos que

$$E(S \mid E_{k-1} \wedge r_k = a_k) \geq E(S \mid E_{k-1} \wedge r_k = \overline{a}_k)$$

dado termos que:

$$\begin{aligned} E(S \mid E_{k-1}) &= E(S \mid E_{k-1} \wedge r_k = a_k).Pr(r_k = a_k \mid E_{k-1}) \\ &\quad + E(S \mid E_{k-1} \wedge r_k = \overline{a}_k).Pr(r_k = \overline{a}_k \mid E_{k-1}) \\ &\leq E(S \mid E_k). \end{aligned}$$

Então, $E(S \mid E_n) \geq E(S|E_0) = E(S) = \frac{7m}{8}$. $\square$

Será que MAX-E3SAT tem um esquema de aproximação em tempo polinomial?

Teorema 8.5 *A menos que* P = NP*, não existe nenhum esquema polinomial de aproximação de razão $(\frac{8}{7} + \varepsilon)$ para MAX-E3SAT.*

Considere-se o problema de optimização associado ao problema CLIQUE.

(MAX-CLIQUE)

Instância: Um grafo $G = (V, E)$ não dirigido

Questão: Determinar o maior clique em G.

Teorema 8.6 ([FGL+96]) *Existe um algoritmo aproximado de razão ρ, para algum ρ, para MAX-CLIQUE se e só se* $\mathsf{P} = \mathsf{NP}$.

Dem. Se $\mathsf{P} = \mathsf{NP}$, MAX-CLIQUE pode ser resolvido em tempo polinomial. Suponhamos que existe um algoritmo aproximado de razão ρ. Seja $L \in \mathsf{NP}$, $L \in \mathsf{PCP}(\log n, 1)$. Então existe um protocolo PCP (P, V) para L que , com dados de tamanho n, usa $c \log n$ bits aleatórios e k perguntas ao oráculo. Pela amplificação, podemos assumir que a probabilidade de aceitar se $x \neq L$ é estritamente menor que $\frac{1}{\rho}$. Dado x, cada $y \in \{0,1\}^{c \log n}$ e $a \in \{0,1\}^k$ determina unicamente as perguntas $z_1, \ldots, z_k$ a P. Temos que z_1 é determinada apenas por y; z_2 por a_1 e y, etc. Seja $G = (V, E)$ um grafo com

$$\begin{aligned} V &= \{(y,a) \in \{0,1\}^{c \log n} \times \{0,1\}^k \mid \\ &\quad \text{o caminho determinado por } (y,a) \text{ aceita } x\} \\ E &= \{((y,a),(y',a')) \mid \text{se os vértices são consistentes}\} \end{aligned}$$

onde dizemos que (y, a) e (y', a') são consistentes se$a_i = a'_j \implies z_i = z'_j$. Pode-se mostrar que se $a \neq b$ então (y, a) e (y, b) não são consistentes.

Lema 8.1 ([FGL+96]) *Seja $\omega(G)$ o tamanho do clique máximo de G. Então se P maximal,*

$$Pr_y((P,V) aceita\ x) n^c = \omega(G)$$

.

Temos então que se $x \in L$ então o clique máximo de G tem tamanho n^c. Se $x \neq L$, o clique máximo de G tem tamanho estritamente menor que $\frac{n^c}{\rho}$. Podemos então ter o seguinte algoritmo de aproximação para decidir se $x \in L$;

- se $x \in L$ o algoritmo retorna um clique de tamanho pelo menos $\frac{n^c}{\rho}$.
- se $x \notin L$, o algoritmo retorna um clique de tamanho estritamente menor.

□

Exercício 8.1.1 *Provar o lema anterior considerando as duas inequações.*

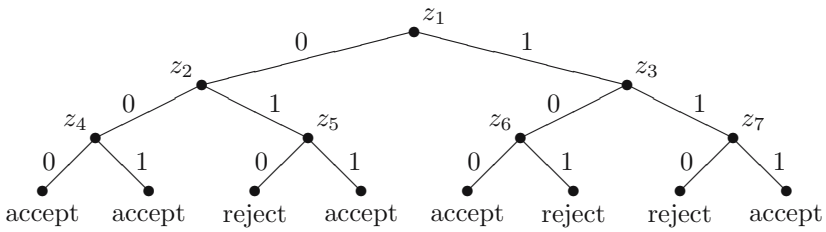
Teorema 8.7 *Existe um esquema de aproximação em tempo polinomial para MAX-3SAT se e só se* $\mathsf{P} = \mathsf{NP}$.

Dem. Seja $L \in \mathsf{NP}$. Como $\mathsf{NP} \subseteq \mathsf{PCP}(\log n, 1)$ existe um protocolo PCP (P, V) para L tal que dado x, $|x| = n$, usa no máximo $c \log n$ bits aleatórios e k perguntas ao oráculo. Uma redução $L \leq_m^p$ 3SAT produz um conjunto de cláusulas C_x tal que C_x é satisfeita se e só se $x \in L$. Mas por causa do protocolo temos uma redução mais forte

i $x \in L$, C_x satisfazível

ii $x \notin L$, nenhuma atribuição de verdade satisfaz mais que $1 - \varepsilon$ cláusulas com $\varepsilon = \frac{1}{(k-2)(2^{k+1})}$

Se conseguissemos aproximar MAX-3SAT com uma razão arbitrária podíamos distinguir (i) de (ii). O protocolo (P, V) para L determina uma árvore de computação T para V contendo ramos aleatórios e de oráculo. Existem só n^c strings aleatórias com $c \log n$ bits. Para cada desses y, seja T_y a subárvore de T obtida determinizando os ramos com y. A árvore T_y só tem ramos de oráculo. Ao longo de um caminho de T_y no máximo existem k perguntas. Logo 2^k caminhos. Mas só há $2^k - 1$ perguntas distintas (porque causa da dependência). Seja $z_1, \ldots, z_m$ a lista de perguntas em T_y. Associando uma variável Booleana a cada z_i obtemos um conjunto de cláusulas que descreve as respostas que levam à aceitação. Por exemplo:



Considerando os caminhos que levam a rejeição

$$(\overline{z}_1 \wedge z_2 \wedge \overline{z}_5) \vee (z_1 \wedge \overline{z}_3 \wedge z_6) \vee (z_1 \wedge z_3 \wedge \overline{z}_7).$$

Negando e aplicando as leis de De Morgan temos a fórmula das respostas que levam à aceitação

$$(z_1 \vee \overline{z}_2 \vee z_5) \wedge (\overline{z}_1 \wedge z_3 \vee \overline{z}_6) \wedge (\overline{z}_1 \vee \overline{z}_3 \vee z_7).$$

Em geral, as cláusulas podem ser reduzidas a três literais usando a seguinte regra:

$$(w_1 \vee \cdots \vee w_k) \implies$$
$$(w_1 \vee w_2 \vee x_1) \wedge (\overline{x}_1 \vee w_3 \vee x_2) \wedge \cdots \wedge (\overline{x}_{k-3} \vee w_{k-1} \vee w_k)$$

Seja C_y o conjunto resultante. C_y tem no máximo $(k-2)2^k$ cláusulas Seja C_x a conjunção de todas as C_y, $y \in \{0,1\}^{n^c}$. Temos que C_x tem no máximo $n^c(k-2)2^k$ cláusulas. Se $x \in L$, (P, V) aceita x com probabilidade 1. Para todos os y, T_y aceita e a atribuição de verdade correspondente satisfaz C_y. Logo C_x, também é satisfeita. Se $x \notin L$, nenhum (P', V) aceita x com probabilidade maior que $\frac{1}{2}$. Então, com perguntas determinadas por P', pelo menos $\frac{1}{2}$ das árvores T_y rejeitam e C_y não é satisfeita. Para tal, pelo menos uma cláusula de C_y não é satisfeita. O que é mais que uma fração

$$\varepsilon = \frac{1}{(k-2)2^{k+1}}$$

de todas as cláusulas de C_x.

Se MAX-3SAT tivesse um algoritmo de aproximação com razão $\frac{1}{1-\varepsilon}$ então podíamos decidir a pertença a L deterministicamente e em tempo polinomial:

- Se $x \in L$ o algoritmo aproximado daria uma atribuição satisfazendo pelo menos uma fração maior que $1 - \varepsilon$ das cláusulas.
- Se $x \notin L$ a maior fração de cláusulas que podia ser simultaneamente satisfeita é no máximo $1 - \varepsilon$.

□

Bibliografia

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: a Modern approach*. Cambridge, 2009.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[BC94] Daniel P. Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall, 1994.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31. ACM, 1991.

[Cha87] C. J. Chaitin. *Algorithmic Information Theory*. Cambridge University Press, 1987.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In Association for Computing Machinery, editor, *Proc. 3rd Annual ACM Symposium on Theory of Computing*, 1971.

[Coo73] Stephen A. Cook. An observation on time-storage trade off. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 29–33. ACM, 1973.

[Dew89] A. K. Dewdley. *The Turing Omnibus – 61 Excursions in Computer Science*. Computer Science Press, 1989.

[Din05] Irit Dinur. The PCP theorem by gap amplification. *Electronic Colloquium on Computational Complexity (ECCC)*, (046), 2005.

[FGL+96] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

[GJ79] Michael Garey and David Johnson. *Computers and Intractability: a guide to the theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[HS65] J. Hartmanis and R. Stearns. On the complexity complexity of some algorithms. *Trans. Amer. Math. Soc.*, (117):285–306, 1965.

[HS66] F. Hennie and R. Stearns. Two-tape simulation of multitape Turing machines. *J. ACM*, (13):533–546, 1966.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[Iba72] Oscar H. Ibarra. A note concerning nondeterministic tape complexities. *J. ACM*, 19(4):608–612, 1972.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Tatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Lad75] R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, (22):155–171, 1975.

[Lev73] Leonid Levin. Universal sequential search problems. *Problems of Information Transmission*, (9):265–266, 1973. translated from Problemy Peredachi Informatsii (Russian)), 9, 1973.

[LP81] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, 1981.

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th Annual Symp. on switching and Automata Theory*, pages 125–129. IEEE Computer Society, 1972.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[Ruz81] Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[SHL65] R. E. Stearns, J. Hartmanis, and P. M. Lewis. Hierarchies of memory limited computations. In *Switching Circuit Theory and Logical Design, 1965. SWCT 1965. Sixth Annual Symposium on*, pages 179–190, Oct 1965.

[Vaz01] Vijay V. Vazirani. *Approximation Algoritms*. Springer, 2001.

[Wra76] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1976.