Teoria dos problemas completos na classe NP

Armando Matos
Nelma Moreira

Departamento de Ciência de Computadores
Faculdade de Ciências, Universidade do Porto
email: {acm,nam}@ncc.up.pt

Conteúdo

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

Capítulo 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:

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

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 TM 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. Restrições semelhantes se impõem para funções de complexidade de espaço.

1.1 Classe P

Seja $M = (S, \gamma, \Sigma, s_0, \delta, F)$ 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* 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.

Recorde que, se uma máquina de Turing M decidir uma linguagem L, então $F = \{s_y, s_n\}$. Dada uma função $f(n)$, define-se a classe de complexidade $TIME(f(n)$ por:

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

[1] Se a computação não parar $T_M(x) = \infty$. Nesta secção vamos considerar apenas máquinas de Turing que param sempre.

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

$$\mathbf{P} \quad = \quad \bigcup_{k \geq 1} TIME(n^k)$$

1.2 Classe NP

Seja $N = (S, \gamma, \Sigma, s_0, \Delta, F)$ uma máquina de Turing não- determinística (NDTM), onde

$$\Delta \subseteq (S \times \Gamma) \times (S \times \Gamma \times \{l, r\})$$

Recorde que um passo de computação entre duas configurações é definido por uma relação $xsy@ >> M > x's'y'$ 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, $@ >> N >^n$) com dados $x \in \Sigma^\star$ 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 $L = L(N)$.

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,

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

Teorema 1.2.1 *Se $L \in NTIME(f(n))$ então existe $c > 1$ tal que $L \in TIME(c^{f(n)})$.*

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

$$\mathbf{NP} \quad = \quad \bigcup_{k \geq 1} NTIME(n^k)$$

Corolário 1.2.1 $\mathbf{P} \subseteq \mathbf{NP}$

Capítulo 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 <- 1;
V[i+1] <-  X;
enquanto V[k] <> X faça k <-  k+1;
se k <> n então escreva('sim') senão escreva('não')
fim.
```

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?

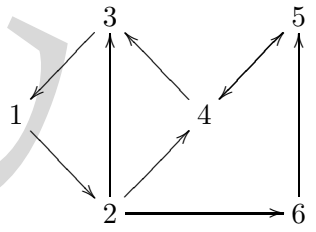
   Resposta: Não se conhece nenhum algoritmo polinomial que resolva este problema. Um algoritmo que corre em tempo exponencial no número de vértices é o seguinte:

```
para i <- 1 até n faça marca[i] <- falso;
k <-  0;
se procura(1) então escreva (' sim') senão escreva('não');
fim.
 bool procura(i)
{
      k <- k+1;
      marca[i]<- verdade;
      para todos (i,j) em E faça {
             se j=1 e k=n então retorna verdade
             se não marca[j] então se procura(j)
                                 então retorna verdade
      }
      k <- k-1; marca[i]<- falso; retorna falso
}
```

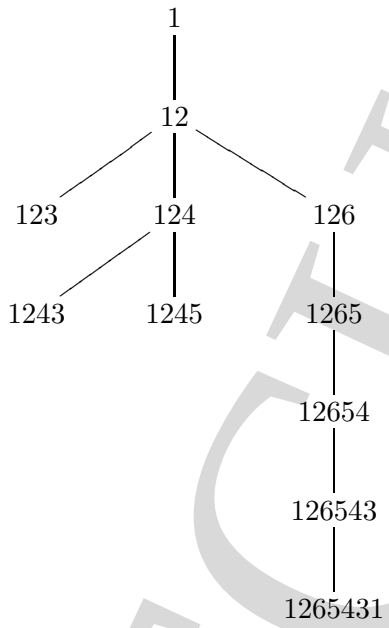
Exemplo 1 *Seja* $G = (V, E)$ *onde*

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

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 2.0.1 Escreva um programa para cada um dos problemas 2 e 3 e obtenha o tempo de execução para vários valores dos parâmetros. Comente 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.

	Valores de n			
Complexidade temporal	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 se conhecem 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?

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 $S' \cap S_2 \neq \emptyset$

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

(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 $b \in V'$.

(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 ?

(3DM) Dado um conjunto $M \subseteq W \times Y \times Z$, onde $|W| = |Y| = |Z| = q$ 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 $\Phi(u) \neq \Phi(v)$?

2.1 Problemas de decisão

Vamos começar por analisar um tipo particular de problemas: os problemas de decisão.

Um problema de decisão $\mathtt{\Pi}$ é um problema que apenas tem duas soluções possíveis: ou "sim"ou "não". O conjunto das suas instâncias $\mathtt{D_\Pi}$ é divido em dois subconjuntos $\mathtt{S_\Pi}$ e $\mathtt{N_\Pi}$, 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 $\mathtt{\Pi}$ em $\Sigma^\star$. A *linguagem associada a* $\mathtt{\Pi}$ para a codificação c é:

$$L(\mathtt{\Pi}, c) = \{x \in \Sigma^\star \mid x = c(\mathtt{I}) \text{ onde } \mathtt{I} \text{ é uma instância para a qual a resposta de } \mathtt{\Pi} \text{ é "sim", } \mathtt{I} \in \mathtt{S_\Pi}\}$$

Resolver o problema $\mathtt{\Pi}$ equivale a reconhecer $L(\mathtt{\Pi}, c)$ 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 $L(\mathtt{\Pi}, c')$ e $L(\mathtt{\Pi}, c)$ 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 $\mathtt{\Pi}$ associámos, independentemente da

codificação, uma função comprimento $comp : \mathtt{D}_\Pi \longrightarrow \mathbb{N}$ nas instâncias $\mathtt{I}$ de Π que é *relacionada polinomialmente*[1] com o comprimento da sequência de símbolos de qualquer codificação razoável de $\mathtt{I}$. Isto é, para qualquer codificação c de Π existem das funções polinomiais p e p' tal que se $\mathtt{I} \in \mathtt{D}_\Pi$ e $x = c(\mathtt{I})$ então $comp(\mathtt{I}) \leq p(|x|)$ e $|x| \leq p'(comp(\mathtt{I}))$. Por exemplo, no problema de decisão do "caixeiro viajante", **(TSP)**, (ver página 14) o comprimento duma instância genérica $\mathtt{I}$ é:

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

Exercício 2.1.1 Escolha uma codificação para o problema TSP e verifique 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 $\Sigma = \{0, 1\}$. 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 6.1.3 da secção 6.1.

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:

(Clique 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)

[1] Lembre-se que, para um dado problema, apenas estamos interessados na existência de algoritmos polinomiais para o resolver.

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 2.1.2 *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. Considere, 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: *Determine uma atribuição de valores para U que satisfaça simultaneamente todas as cláusulas de C?*

Mostre que se existir um algoritmo polinomial `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 `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'<- C;
para  todos u[i] em U  faça {
   C'' <- obtido de C' substituindo 0 por todas as ocorrências de
                u[i] e 1 por todas as ocorrências de ~u[i];
   se SAT(C'') ="sim" então { t(u[i])<-0; C'<- C'' }
                      senão { t(u[i])<- 1;
                              C'<- obtido  de  C'  substituindo  1
                                   por  todas  as ocorrências de u[i]
                                   e 0 por todas as ocorrências de ~u[i]}
}
fim.
```

2.2 Algoritmos determinísticos

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:

> **P** = {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 2.2.1 *Encontre 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, construa o grafo (dirigido) $G = (V, E)$, onde $V = \{u_1, \ldots, u_n, \bar{u_1}, \ldots, \bar{u_n}\}$ e sendo l_i igual a u_i ou $\bar{u_i}$, $(l_i, l_j) \in E$ se $\{\bar{l_i}, l_j\} \in C$ (isto é, $u_i \Rightarrow l_j$). Mostre 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 <- verdade;
enquanto f faça { /* enquanto existirem vértices
                     por  colorir */
   f <- falso;
   para i<- 1 até n faça     /* pelo menos um vérice
                            é colorido */
    se não colorir(i) então f<-  verdade;
}
escreva('sim');
fim.
   colorir(i){
 se i está colorido retorna verdade
 se i não tem vizinhos coloridos
          então retorna falso /* i tem no maximo n-1
                                 vizinhos */
```

```
                    senão se os vizinhos tiverem cores contraditórias
                           então { escreva('não') ; fim }
                           senão {colorir i com a cor alternativa;
                                  retorna verdade}
        }
```

2.3 Algoritmos não determinísticos

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 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 ser o 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)
se V[i]= X então {escreva('sim'); sucesso}
        senão{ escreva('não'); insucesso}
```

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*[2] o problema, apenas *verifica* se uma possível solução é efectivamente solução.

Exercício 2.3.1 *Para cada um dos seguintes problemas escreva um algoritmo não-determinístico:*

a) Ordenar uma sequência de inteiros positivos.

b) **(HC)** *Existência de ciclo hamiltoniano num grafo (dirigido ou não).*

c) **(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 c e n_0 constantes.

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 e com máquinas de Turing não-determinísticas polinomiais e os problemas correspondentes com linguagens da classe **NP**. Temos:

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

Exercício 2.3.2 *Determine a complexidade temporal de cada um dos algoritmos encontrados no exercício 2.3.1.*

Exercício 2.3.3 *Obtenha 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.

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

Teorema 2.3.1 *Uma linguagem $L \subseteq \Sigma^\star$ pertence a* **NP** *se e só se existe uma linguagem $L_v \subseteq \Sigma^\star \times \Sigma^\star$ que pertence a* **P** *e uma função polinomial p tal que:*

$$L = \{x \mid \exists y(x,y) \in L_v \text{ e } |y| \leq p(|x|)\}$$

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

Dem. ($\Leftarrow$) Suponhamos que $L = \{x \mid \exists y(< x,y > \in L_v \text{ e } |y| \leq p(|x|)\}$ e $L_v \in \mathbf{P}$ e $p(n)$ é polinomial. Então L é decidida pela seguinte NDTM N: Com dados x, N gera uma sequência y de comprimento no máximo $p(|x|)$ e depois usa a TM que decide L_v para testar (verificar) em tempo polinomial se $(x,y) \in L_v$. Se $(x,y) \in L_v$ N aceita x caso contrário rejeita.[3]

($\Rightarrow$) Seja $L \in \mathbf{NP}$. Então existe uma NDTM 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 L_v é definida do seguinte modo: um par $< x,y >$ pertence a L_v se y codifica um caminho duma computação de aceitação de N com dados x. É fácil ver que $L_v \in \mathbf{P}$ e que $x \in L$ se e só se $\exists y(< x,y > \in L_v \text{ e } |y| \leq p(|x|)$. □

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

Exercício 2.3.4 Considere a seguinte definição duma máquina de Turing com "escolhas" (de "adivinhação")(GDTM) (ver [GJ79, Cap. 2.3]): uma GDTM, 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 GDTM para dados $x \in \Sigma^*$ é feita em dois passos:

Escolha Inicialmente os dados x estão escritos na fita a partir da posição 1 até $\mid x \mid$, os restantes em branco, a cabeça de leitura-escrita na posição 1 e a cabeça de escrita a célula -1. 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. Note que este módulo escreveu um qualquer $s \in \Gamma^*$.

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.

Mostre que uma máquina GDTM é equivalente (polinomialmente) a uma NDTM. Sugestão: Considere a demonstração do teorema 2.3.1.

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

Figura 2.1: Relação entre **P** e **NP**

2.4 Relação entre P e NP

Já se viu que $\mathbf{P} \subseteq \mathbf{NP}$.

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.

Será que $\mathbf{NP} \subseteq \mathbf{P}$? Até agora ainda ninguém soube respondera esta pergunta !

Exercício 2.4.1 *Mostrar que se* $\Pi \in \mathbf{NP}$ *então existe um polinómio* $p(n)$ *tal que* Π *pode ser resolvido por um algoritmo determinístico com complexidade* $O(2^{p(n)})$.

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 2.5.1 *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:*

(i) Existe uma máquina de Turing polinomial que calcula f

(ii) 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, $A \propto B$, 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.

Proposição 2.5.1 *Se $L_1 \propto L_2$ e $L_2 \in \mathbf{P}$ então $L_1 \in \mathbf{P}$.*

Proposição 2.5.2 *A relação $\propto$ é transitiva.*

Exercício 2.5.1 *Mostrar as proposições 2.5.1 e 2.5.2.*

Duas linguagens L_1 e L_2 são *polinomialmente equivalentes* se $L_1 \propto L_2$ e $L_2 \propto L_1$. 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 HC $\propto$ 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:

(i) $C = V$

(ii) 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.

(iii) $B = m$

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 $< v_1, \ldots, v_m >$ é um ciclo Hamiltoneano de G. Então, $< v_1, \ldots, v_m >$ é uma volta em $f(G)$ com distância total $m=B$. Inversamente, suponhamos que $< v_1, ..., v_m >$ 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 $\Pi_1 \propto \Pi_2$ dizemos que Π_2 é *pelo menos tão difícil como* Π_1.

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

Exemplo 3 3SAT $\propto$ 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 $K \leq \mid V \mid$ inteiro positivo.*

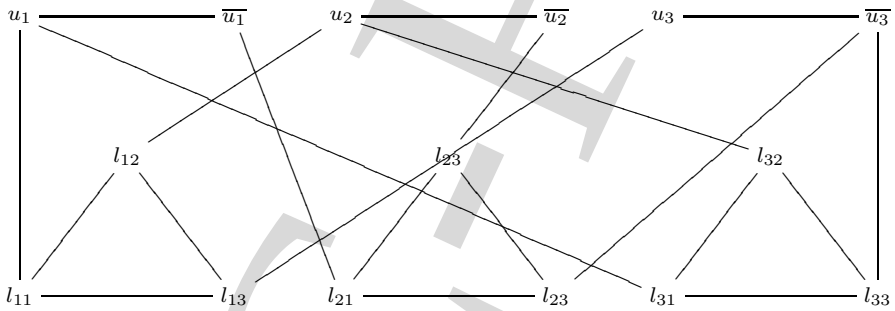
Questão: *Existe um subconjunto $V' \subseteq V$ tal que $\mid V' \mid \leq K$ e para cada ramo $\{u, v\} \in E$, $u \in V'$ ou $v \in V'$? (A V' chama-se* cobertura por vértices *de G.)*

Construção da transformação : *Suponhamos uma instância de* **3SAT**. *Vamos construir uma instância de* **VC** *indicando os conjuntos* V *e* E. *Para cada* $u_i \in U$, $\{u_i, \overline{u_i}\} \in V$ *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* $\mid V \mid = 2n + 3m$ *e* $\mid E \mid = n + 6m$. *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 $V' \subset V$ *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 2.5.2 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 2.5.3 Verificar que V' é uma cobertura por vértices de G e $\mid V' \mid \leq K$.

Capítulo 3

Problemas NP-completos

Definição 3.0.2 *Uma linguagem L é* **NP***-completa se $L \in$* **NP** *e para todo $L' \in$* **NP** *tem-se que $L' \propto L$.*

Em termos de problemas, um problema Π diz-se **NP**-completo se $\Pi \in$ **NP** e para todo $\Pi' \in$ **NP** tem-se que $\Pi' \propto \Pi$.

Pela proposição 2.5.1 os problemas **NP**-completos (representamos por **NPC** a classe destes problemas) 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 $\Pi \in$ **NP** $\setminus$ **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 3.1.

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 3.0.3 *Se L_1 e L_2 pertencem a* **NP***, L_1 é* **NP***-completa e $L_1 \propto L_2$ então L_2 é* **NP***-completa.*

Exercício 3.0.4 Mostrar a proposição 3.0.3.

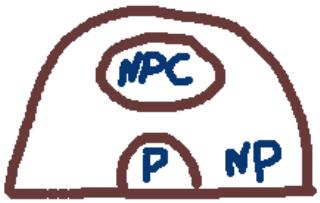


Figura 3.1: Classe **NP**

Cook mostrou que SAT é NP-completo !
Dado que para cada problema Π em **NP** existe uma 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` .

Teorema 3.0.1 **SAT** *é* **NP***-completo.*

Dem. Temos que mostrar:

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

(ii) $\forall \Pi \in \mathbf{NP} \implies \Pi \propto \mathbf{SAT}$

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:

```
para i <- 1 até n faça
    u[i] <- escolha({0,1})
para i <- 1 até m faça
     se c[i] = 0  então insucesso
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, $\mathbf{SAT} \in \mathbf{NP}$.

Seja $L_{SAT} = L[SAT, c]$ a linguagem associada a **SAT** para uma dada codificação c. Temos que mostrar $\forall\, L \in \mathbf{NP},\ L \propto L_{SAT}$. Cada $L \in \mathbf{NP}$ é decidida por uma GDTM polinomial (ver exercício 2.3.4). Seja, então, M uma GDTM polinomial arbitrária, $M = (S, \Sigma, \Gamma, b, s_0, \{s_y, s_n\}, \delta)$ 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, $s_1 = s_y$, $s_2 = s_n$,

$s_3, \ldots, s_r$, onde $r = |S| - 1$, e os elementos de Γ, $a_0 = b$, $a_1, \ldots, a_v$, 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[1]. 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:

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

Grupo	Restrições impostas	Cláusulas nesse Grupo	
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 $x = a_{k_1} \ldots a_{k_n}$.	$\{S[0,0]\}, \{H[0,1]\}, \{A[0,0,0]\},$ $\{A[0,1,k_1]\}, \ldots, \{A[0,n,k_n]\},$ $\{A[0,n+1,0]\}, \ldots, \{A[0,p(n)+1,0]\}$	
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[2]:

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

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

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), $0 \le i < p(n)$, $-p(n) \le j \le p(n)+1$, $0 \le k \le r$ e $0 \le l \le v$ tem-se as seguintes 3 cláusulas[3] :

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

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 NTDM 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 (**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, $Comp[F_L(x)] = O(p(n)^4)$. □

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

Assim, usando a proposição 3.0.3, para mostrar que um problema Π é **NP**-completo basta:

a) Mostrar que $\Pi \in$ **NP**

b) Seleccionar um problema $\Pi' \in$ **NPC**

c) Construir uma transformação F de Π' em Π

d) Mostrar que F é uma redução polinomial

Proposição 3.0.4 3SAT *é* **NP**-*completo.*

Dem. a) **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:

```
para i <- 1 até n faça
    u[i] <- escolha({0,1})
para i <- 1 até m faça
     se c[i] = 0  então insucesso
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**.

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

c) 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

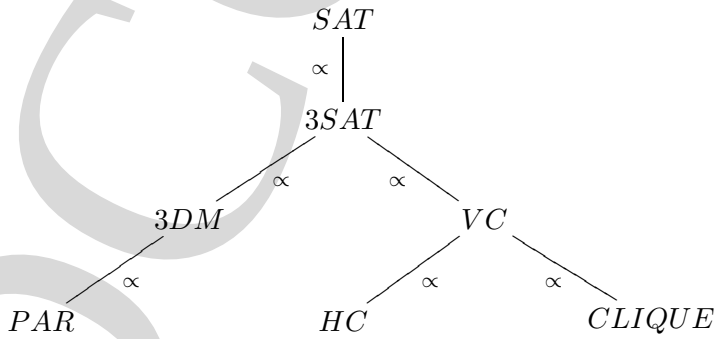
$$\begin{aligned} U' &= U \cup \bigcup_{i=1}^{m} U'_i \\ C' &= \bigcup_{i=1}^{m} C'_i \end{aligned}$$

d) 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 $t'(u_i^j) = 1$ para $1 \leq j \leq m-2$ e $t'(u_i^j) = 0$ 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.

e) 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 **SAT** $\propto$ **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, que já foram referidos anteriormente, e o diagrama seguinte mostra uma sequência de reduções que pode ser usada para a sua demonstração:



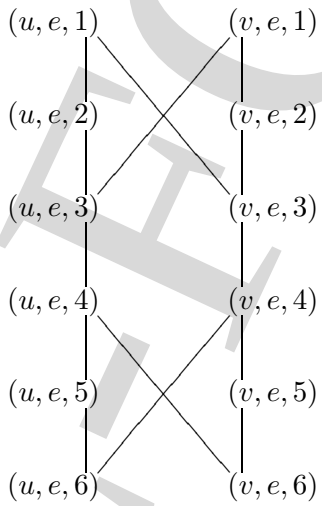
Pelo diagrama e pela redução atrás apresentada podemos concluir que o problema de decisão do "caixeiro viajante"é **NP**-completo: **HC** é **NP**-completo e **HC** $\propto$ **TSP**!

Proposição 3.0.5 **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.

b) Seleccionámos **VC**

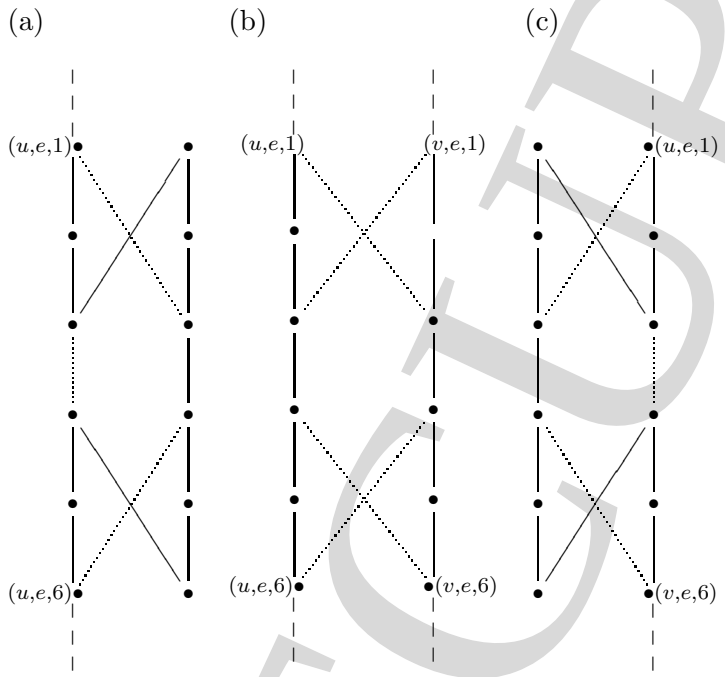
c) Vamos transformar **VC** em **HC**. Seja $G = (V, E)$ e $K \leq \mid V \mid$ uma instância de **VC**. Temos de construir um grafo $G' = (V', E')$ tal que G' tem um ciclo Hamiltoneano se e só se G tem uma cobertura por vértices de tamanho no máximo K. Primeiro, o grafo G' tem K vértices selectores $a_1, a_2, \ldots, a_K$, que serão usados para seleccionar K vértices do conjunto V. Depois, para cada ramo em E, G' contêm 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

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

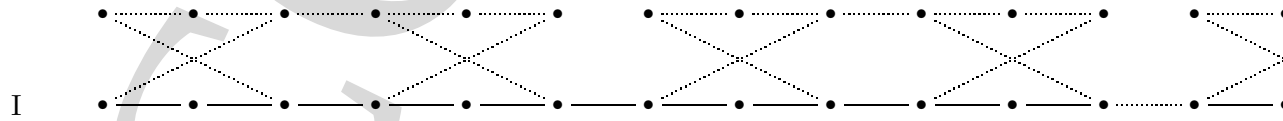
Cada V'_e tem 12 vértices e cada E'_e tem 14 ramos. Na construção final, os únicos vértices desta componente que entraram noutros ramos são os vértices $(u, e, 1)$,$(v, e, 1)$, $(u, e, 6)$ e $(v, e, 6)$. Isto implica que qualquer ciclo Hamiltoneano tem de passar pelos ramos de E'_e de uma das seguintes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

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

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



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:

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

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 $< v_1, v_2, \ldots, v_n >$ onde $n = |V'|$ é um ciclo Hamiltoneano. Considere 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:

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

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

□

Exercício 3.0.5 [⋆] Mostre que **3SAT** $\propto$ **3DM** e **3DM** $\propto$ **PAR**. Sugestão: Consulte [GJ79, Cap 3].

Exercício 3.0.6 Mostrar que os seguintes problemas são **NP**-completos:

a) **CLIQUE**

Sugestão: **VC** $\propto$ **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 $E^c = \{(u, v) : u, v \in V \text{ e } (u, v) \notin E\}$.

b) **4SAT**

Sugestão: **3SAT** $\propto$ **4SAT**

c) **3COLOR**

Sugestão: **3SAT** $\propto$ **3COLOR**

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 4 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 3.0.7 Mostrar por restrição que os seguintes problemas são **NP**-completos:

a) **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 $B = K = 1/2 \sum_{u \in U} s(u)$.

b) **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 $K = |V|$.

c) **(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 $W \cup X \cup Z$.

d) **Conjuntos Disjuntos**

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

Questão: C contém K conjuntos disjuntos?

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 $\mathbf{P} \neq \mathbf{NP}$. 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:

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

b) 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 $\mathbf{P} = \mathbf{NP}$. 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 $\mathbf{NP}-\mathbf{P}-\mathbf{NPC}$ que chamaremos **NPI** – problemas de dificuldade "intermédia". Teoricamente se $\mathbf{P} \neq \mathbf{NP}$ mostra-se que esta classe é não vazia e que existe uma infinidade de classes de equivalência entre **P** e **NP**.

Capítulo 4

Redução de Turing e Problemas NP-hard

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 $\mathtt{I}$, 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). Repare 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 $\mathtt{I} \in D_\Pi$ num conjunto finito $S_\Pi[\mathtt{I}]$ de soluções de $\mathtt{I}$. O problema Π tem solução para a instância $\mathtt{I}$ se $S_\Pi[\mathtt{I}] \neq \emptyset$. Um algoritmo resolve o problema Π , se dado uma instância $\mathtt{I}$ tem como resposta "não" se $S_\Pi[\mathtt{I}] \neq \emptyset$ ou se tem como saída uma solução s pertencente a $S_\Pi[\mathtt{I}]$.

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.

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 tal 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:

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

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

Exercício 4.0.8 Identifique uma linguagem $L \subseteq \Sigma^+$ com uma relação binária em Σ^+. Conclua que um problema de decisão é um problema de procura.

Em termos de problemas, uma redução polinomial de Turing dum problema Π num problema Π' é um algoritmo determinístico $\mathtt{A}$ que resolve Π usando uma subrotina $\mathtt{S}$ 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 $\Pi \propto_t \Pi'$.

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

Definição 4.0.3 *Uma máquina de Turing com oráculo (OTM) corresponde a uma máquina de Turing básica* $O = (S, \Gamma, \Sigma, b, s_0, \delta, F)$ *com as seguintes características adicionais:*

- *possuí uma fita adicional – a fita de oráculo – e uma cabeça se 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* $\delta : S \setminus F \cup \{s_c\} \times \Gamma \times \Sigma \longrightarrow S \times \Gamma \times \Sigma \times \{l, r\} \times \{l, r\}$

Um passo de computação duma OTM 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 4.0.9 Descreva formalmente um passo de computação duma OTM.

Uma OTM 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 4.0.4 *Sejam* R *e* R' *duas relações binárias em* $\Sigma^\star$*. Uma* redução polinomial de Turing *de* R *em* R'*,* $R \propto_t R'$*, é uma OTM* O *tal que para toda a função* $g : \Sigma^\star \longrightarrow \Sigma^\star$ *que realiza* R'*,* O_g *é uma OTM polinomial e a função* f_O^g *calculada por* O_g *realiza* R*.*

Proposição 4.0.6 *A relação* $\propto_t$ *é transitiva.*

Exercício 4.0.10 Mostre a proposição 4.0.6.

Proposição 4.0.7 *Se* Π *e* Π' *são problemas de decisão então:*

$$\Pi \propto \Pi' \implies \Pi \propto_t \Pi'$$

Dem. A transformação F da redução polinomial $\propto$ 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). □

Exercício 4.0.11 Enuncie e demonstre a proposição 4.0.7 em termos de linguagens, relações e máquinas de Turing com oráculo.

Definição 4.0.5 *Uma relação R é* **NP-hard** *se existe uma linguagem $L \in$ **NPC** tal que $L \propto_t R$.*

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

Em particular, um problema de decisão Π é **NP-hard** se $\forall \Pi' \in$ **NP**, $\Pi' \propto_t \Pi$ e é **NP**-completo se além disso $\Pi \in$ **NP**.

Dado um problema de decisão Π o seu complementar designa-se por Π^c e é tal que $S_{\Pi^c} = D_\Pi \setminus S_\Pi$.

Exercício 4.0.12 Mostrar que dado um problema $\Pi \in$ **NP** e o seu problema complementar Π^c, se tem $\Pi \propto_t \Pi^c$ e vice-versa. Explique porque não parece para muitos problemas que se possa ter $\Pi^c \propto \Pi$.

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 $D \propto_t O$ mas também $O \propto_t D$. Se o problema D é **NP**-completo então o problema de optimização O pode ser resolvido em tempo polinomial se e só se **P** = **NP**.

Exemplo 5 *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"$\theta =< c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(k)} >$ de k cidades distintas, $1 \leq k \leq m$.*

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

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

Considerem-se instâncias de **TSE** *e* **TSO** *para C e d comuns. Suponhamos que* $\mathtt{S(C,d,\Theta,B)}$ *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* $\mathtt{S(C,d,<c_1>,B)}$ *para vários valores de B, no máximo $\lceil \log_2 B \rceil$ vezes:*

```
B_MIN <-  m; B_MAX  <- mx
enquanto  B_MAX  -  B_MIN <> 1 faça{
        B<- [(B_MAX    + B_MIN   )/2 ];
        se S(C,d,<c >,B) = "sim" então B_MAX<- B senão B_MIN <- B
}
        B*  <- 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* $< c_1 >$ *e como é estendível, existe* $c_j \in C \setminus \{c_1\}$ *tal que* $< c_1, c_j >$ *é uma volta parcial estendível. Podemos encontrar* c_j *no máximo em* $m-2$ *chamadas a* $\mathtt{S}[\mathtt{C}, \mathtt{d}, < \mathtt{c_1}, \mathtt{c_i} >, \mathtt{B}^\star]$. *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 4.0.6 *Um problema* Π *é* **NP-easy** *se existe um problema* $\Pi' \in$ **NP** *tal que* $\Pi \propto_t \Pi'$.

Exercício 4.0.13 Mostrar que os problemas de optimização (ou de procura de solução) associados com os seguintes problemas **NP**-completos são **NP-easy**:

a) **SAT**

b) **VC**

c) **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 **P** = **NP** !

Capítulo 5

Estrutura das Classes de Problemas

5.1 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". 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 $\Pi \in \mathbf{NP}$, então $\Pi^c \in \mathbf{NP}$. 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) 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

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

Ou em termos de linguagens, dado um alfabeto Σ:

$$\mathbf{coNP} \quad = \quad \{\Sigma^\star \setminus L \mid L \subseteq \Sigma \text{ e } L \in \mathbf{NP}\}$$

Dado que existem muitos problemas em **coNP**que parecem não estar em **NP**, podemos conjecturar que $\mathbf{NP} \neq \mathbf{coNP}$. Note-se que se esta conjectura for verdadeira implica que $\mathbf{P} \neq \mathbf{NP}$. Porquê? Como se viu $\mathbf{P} \in \mathbf{coNP} \cap \mathbf{NP}$. Na figura 5.1 tem-se a relação entre estas classes.

O resultado seguinte relaciona os problemas **NP**-completos com esta conjectura.

Proposição 5.1.1 *Se existe um problema* **NP***-completo* Π *, tal que* $\Pi^c \in \mathbf{NP}$*, então* $\mathbf{NP} = \mathbf{coNP}$*.*

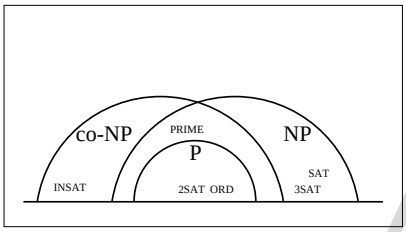


Figura 5.1: **NP**e **coNP**

Figura 5.2: Estrutura de **NP**

Exercício 5.1.1 Mostre a proposição 5.1.1.

Se **P** $\neq$ **NP** e **NP** $\neq$ **coNP** a estrutura para **NP** é a apresentada na figura 5.1.

Exercício 5.1.2 Considere o problema complementar de **3SAT**:

3SAT^c

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?

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

O resultado da proposição 5.1.1 permite ainda encontrar um forte candidato classe **NPI** (ou então a **P**): o problema de determinar se um dado inteiro é composto.

(COMPOSTO)

Instância: Um número inteiro $K > 1$

Questão: Existem inteiros m, $n > 1$ tais que $K = mn$?

O seu problema complementar é

(PRIMOS)

Instância: Um número inteiro K

Questão: K é primo?

Tanto o problema **(COMPOSTO)** como o complementar, o problema (**PRIMOS**) pertencem a **NP**.

Nenhum destes problemas pode ser **NP**-completo a menos que **NP** = **coNP**. Assim, se não existir nenhum algoritmo polinomial que os resolva, pertencem a **NPI**(=**NP**−**P**−**NPC**).

Mais ainda, estes problemas podem ser resolvidos por algoritmos polinomiais, mas que podem falhar com probabilidade arbitrariamente pequena... Nestas circunstâncias podemos conjecturar que ¶ = $NP \cap$ **coNP**! Um problema que "contribuiu"para esta conjectura foi o da "Programação Linear"(LP) que durante muitos anos esteve nas condições do problema dos números compostos mas para o qual já se provou a existência dum algoritmo polinomial...

Exercício 5.1.3 Mostre que (**COMPOSTO**) $\in$ **NP**.

Para uma demonstração que (**PRIMOS**) pertence a **NP**, ver [Pap94, Cap. 10] ou [Pra75].

5.2 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 $\Pi \propto_t^N \Pi'$.

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

Exemplo 6 (EME)

Instância: *Uma expressão booleana E com literais num conjunto de variáveis U, constantes V e F e conectivas lógicas $\vee$, $\wedge$, $\neg$ e $\Longrightarrow$. 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** $\propto_t$ **EME** *(Verifique!). 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 V e F e conectivas lógicas $\vee$, $\wedge$, $\neg$ e $\Longrightarrow$.*

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,* `S[E,U]` *uma subrotina que resolve* **SBE**. *O seguinte algoritmo estabelece a redução,* **EME** $\propto_t^N$ **SBE**:

```
E'<- escolha(B_K)
se S[~((E=>E')/\(E'=>E)),U] = "não" então sucesso
        senão insucesso
```

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

As classes $\mathbf{P^Y}$ e $\mathbf{NP^Y}$ são definidas do modo seguinte:

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

$\mathbf{P^{NP}}$ é a classe dos problemas a que chamamos anteriormente **NP-easy**. O problema **EME** pertence a $\mathbf{NP^{NP}}$.

Exercício 5.2.1 (i) Mostre que o seguinte problema é **coNP**.

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 $\Longrightarrow$.

Questão: E e E' são equivalentes?

(ii) Conclua, novamente, que **EME** $\in \mathbf{NP^{NP}}$ usando a alínea anterior.

Se $\mathbf{P} \neq \mathbf{NP}$ a figura 5.3 mostra a relação entre as classes **P**, **NP**, **coNP**, $\mathbf{P^{NP}}$ e $\mathbf{NP^{NP}}$. 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:

$$\begin{aligned} \mathbf{\Sigma_0^p} &= \mathbf{\Pi_0^p} = \mathbf{\Delta_0^p} = \mathbf{P} \\ \mathbf{\Delta_{k+1}^p} &= \mathbf{P^{\Sigma_k^p}} \\ \mathbf{\Sigma_{k+1}^p} &= \mathbf{NP^{\Sigma_k^p}} \\ \mathbf{\Pi_{k+1}^p} &= \mathbf{co\Sigma_{k+1}^p} \end{aligned}$$

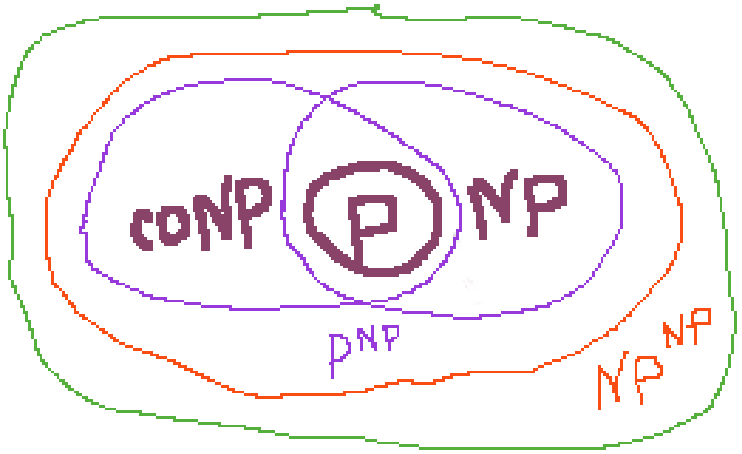


Figura 5.3: Classes **P**, **NP**, **coNP**, $\mathbf{P^{NP}}$ e $\mathbf{NP^{NP}}$

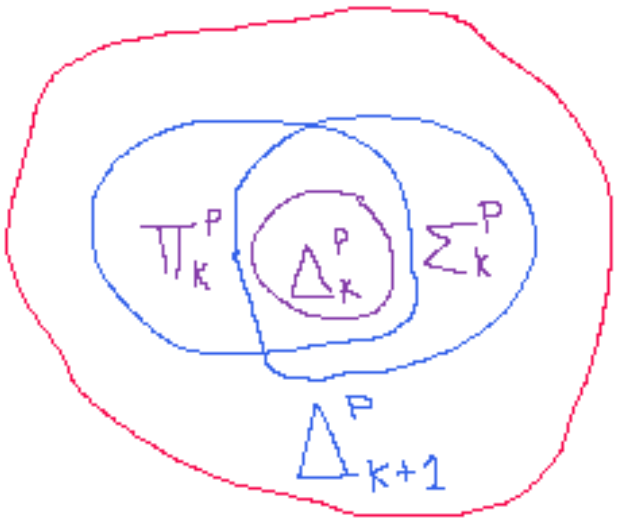


Figura 5.4: Hierarquia Polinomial

Em particular, $\mathbf{\Sigma_1^P = NP^P = NP}$, $\mathbf{\Pi_1^P = coNP}$ e $\mathbf{\Delta_1^P = P}$. Como se viu **EME** pertence a $\mathbf{\Sigma_2^P}$, embora não se saiba se $\mathbf{\Sigma_1^P = \Sigma_2^P}$, ou se existe $k \geq 0$ tal que $\mathbf{\Sigma_k^P \neq \Sigma_{k+1}^P}$. A relação de inclusão entre as várias classes é dada pelo diagrama da figura 5.4.

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 $\Pi \in \mathbf{\Sigma_k^P}$.

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:

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

O teorema seguinte é de [Wra76]:

Teorema 5.2.1 *Seja $L \subseteq \Gamma^\star$ uma linguagem sobre um alfabeto Γ e($|\Gamma| \geq 2$). Para todo $k \geq 1$ $L \in \mathbf{\Sigma_k^P}$ 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 5.2.1 para provar que $\mathbf{EME} \in \mathbf{\Sigma_2^P}$, considere-se a relação R, com $R(x, y_1, y_2)$ se e só se x é $< E, K >$ (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 $E \Leftrightarrow E'$.

As classes $\mathbf{\Pi_k^P}$ também podem ser caracterizadas de modo análogo, bastando no teorema 5.2.1 tocar todos os quantificadores $\exists$ por $\forall$ e vice-versa.

Corolário 5.2.1 *Seja $L \subseteq \Gamma^\star$ uma linguagem sobre um alfabeto Γ e ($|\Gamma| \geq 2$) e $i \geq 1$. $L \in \mathbf{\Sigma_k^P}$ se e só se existe uma relação reconhecida em tempo polinomial tal que a linguagem $\{< x, y > | \, R(x, y)\} \in \mathbf{\Pi_{k-1}^P}$ e existe p tal que $L = \{x \mid \exists y, \ |y| \leq p(|x|) \text{ e } R(x, y)\}$.*

Corolário 5.2.2 *Seja $L \subseteq \Gamma^\star$ uma linguagem sobre um alfabeto Γ e ($|\Gamma| \geq 2$) e $i \geq 1$. $L \in \mathbf{\Pi_k^p}$ se e só se existe uma relação reconhecida em tempo polinomial tal que a linguagem $\{< x, y >|\ R(x,y)\} \in \mathbf{\Sigma_{k-1}^p}$ e existe p tal que $L = \{x \mid \forall y,\ |y| \leq p(|x|)\ e\ R(x,y)\}$.*

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 $\mathbf{\Sigma_k^p}$-completo e portanto torna-se mais fácil provar completitude de outros problemas.

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

Um resultado interessante em relação a esta hierarquia é o seguinte:

Proposição 5.2.1 *Se para algum k, $\mathbf{\Sigma_k^p} = \mathbf{\Pi_k^p}$ então $\mathbf{\Sigma_j^p} = \mathbf{\Pi_j^p} = \mathbf{\Sigma_k^p}$ para todo o $j \geq k$.*

Exercício 5.2.2 Mostrar a proposição 5.2.1. Sugestão: Mostre que $\mathbf{\Sigma_k^p} = \mathbf{\Pi_k^p}$ implica $\mathbf{\Sigma_{k+1}^p} = \mathbf{\Pi_k^p}$.

Isto é, a hierarquia pode colapsar a partir de certa ordem e portanto não ser infinita. Em particular, se $\mathbf{P} = \mathbf{NP}$, então $\mathbf{\Sigma_j^p} = \mathbf{P}$, para todo $j \geq 0$. Se $\mathbf{\Sigma_0^p} \neq \mathbf{\Sigma_k^p}$ para algum $k > 0$, então $\mathbf{P} \neq \mathbf{NP}$. Temos então que qualquer problema da hierarquia só pode ser resolvido em tempo polinomial se e só se $\mathbf{P} = \mathbf{NP}$ (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 $\mathbf{P} \neq \mathbf{NP}$.

Capítulo 6

Exercícios de Revisão sobre Classes de Complexidade

6.0.1 Ordens de Grandeza

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

Definição 6.0.1 *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 6.0.2 *Se* $f(n)$ *é* $o(g(n))$ *então* $f(n)$ *é* $O(g(n))$*.*

Exercício 6.0.3 Mostre o teorema anterior.

Exercício 6.0.4 Mostre as seguintes igualdades:

(i) $f(n) = O(f(n))$

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

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

(iv) $O(O(f(n))) = O(f(n))$

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

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

Exercício 6.0.5 Mostre que:

(i) $a_m n^m + \ldots + a_0$ 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 $h = (2/a_m)max(|a_{m-1}|, \ldots, |a_0|)$

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

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

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

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

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

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

Exercício 6.0.6 Mostre que Θ é uma relação de equivalência.

Exercício 6.0.7 Sendo $f(n)$ e $g(n)$ qualquer par de funções a seguir apresentadas, determine se $f(n)$ é $O(g(n))$, $f(n)$ é $\Omega(g(n))$ ou $f(n)$ é $\Theta(g(n))$

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

6.1 Codificações Razoáveis

Definição 6.1.1 *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 6.1.1 Considere 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 6.1.2 Considere o problema de codificar um grafo não dirigido $G = (V, E)$. Dois processos são:

- matriz de adjacências representada por $\mid V \mid^2$ bits

- lista de ramos: a cad vertice associar a lista dos pontos de chegada dos ramos que dele partem

(i) Defina mais rigorosamente as codificações mencionadas

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

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

Exercício 6.1.3 Supondo o alfabeto $\Sigma = \{0, 1, -, [,], (,), , \}$ invente 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 coficações anteriores diga como representar: um conjunto (finito) de objectos, um grafo, uma função finita e um número racional.

6.2 Classes de Complexidade e NP-completitude

Exercício 6.2.1 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 $\propto$ HPN
2. Mostra que HPN $\propto$ HPE

Exercício 6.2.2 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 $f(a) \neq f(b)$?

(i) Mostra que $C(1)$ e $C(2)$ pertencem à classe P.

(ii) 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 $C(3) \propto C(4)$.

(iii) Supondo que se sabe que $C(3)$ é **NP**-completo mostra que $C(n)$ é **NP**-completo para todo o $n \geq 3$.

Exercício 6.2.3 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 6.2.4 *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 6.2.5 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*?

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

(ii) A que classe de complexidade pertence **IC**? Porquê?

(iii) **(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 6.2.6 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 6.2.7 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:

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

(i) 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).

(ii) Determina a classe de complexidade a que *de facto* pertence **EL**.

Exercício 6.2.8 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, ..., 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 6.2.9 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 6.2.10 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 6.2.11 A Hierarquia cresce
Mostra que, para todo o $k \geq 0$ é

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

Bibliografia

[BC94] Daniel P. Bovet and Pierluigi Crescenzi, *Introduction to the theory of complexity*, Prentice Hall, 1994.

[BG89] Josep Díaz Balcázar, José Luis and Joaquim Gabarró, *Structural complexity i*, Springer Verlag, 1989.

[BG90] ———, *Structural complexity ii*, Springer Verlag, 1990.

[Coo71] S. A. Cook, *The complexity of theorem proving procedures*, Proc. 3rd Annual ACM Symposium on Theory of Computing (Association for Computing Machinery, ed.), 1971.

[GJ79] Michael Garey and David Johnson, *Computers and intractability: a guide to the theory of np-completeness*, W.H. Freeman and Company, 1979.

[HU79] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to automata theory, languages and computation*, Addison Wesley, 1979.

[J.S76] L. J.Stockmeyer, *The polynomial-time hierarchy*, Theoretical Computer Science **3** (1976), 1–22.

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

[LP81] Harry R. Lewis and Christos H. Papadimitriou, *Elements of the theory of computation*, Prentice Hall, 1981.

[LV90] Ming Li and Paul M. B. Vitànyi, *Kolmogorov complexity and its applications*, Handbook of Theoretical Computer Science (J. van Leewen, ed.), vol. A, Elsevier Science Publishers, 1990, pp. 188–254.

[Mat95a] Armando B. Matos, *Algoritmos probalísticos – algumas notas*, Tech. report, Faculdade de Ciências da Universidade do Porto, 1995.

[Mat95b] ———, *Entropia, algoritmos e complexidade mínima*, Tech. report, Faculdade de Ciências da Universidade do Porto, 1995.

[Min74] M. Minsky, *Finite and infinite machines*, Addison Wesley, 1974.

[MS72] A. R. Meyer and L. J. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential time*, Proceedings of the 13th Annual Symp. on switching and Automata Theory, IEEE Computer Society, 1972, pp. 125–129.

[Pap94] Christos H. Papadimitriou, *Computational complexity*, Addison Wesley, 1994.

[Pra75] Vaughan Pratt, *Every prime has a succinct certificate*, SIAM J. Computation **4** (1975), no. 3, 214–220.

[Wra76] C. Wrathall, *Complete sets and the polynomial-time hierarchy*, Theoretical Computer Science **3** (1976), 23–33.